Studienarbeit

VL Informationssyteme

Dennis Nikolay 2544912
Mark Schuegraf 2543591
Moritz Beck 2544065

Tutor: Nils Vossebein

Inhaltsverzeichnis

1. Projektbeschreibung

Ziel unseres Projektes ist die Entwicklung eines zeitbezogenen DBMS, welches dem als Beispiel herangezogenen Autohersteller „Happy Car“ bei der Abarbeitung von Aufträgen behilflich seien soll. Dieser verfügt über ein Verwaltungsgebäude, eine Logistikabteilung, Werke, die angeforderte Fahrzeuge assemblieren, und mehrere werkseigene Teile-Lager sowie einem zentralem Autolager. Diese Einheiten verfügen allesamt über eigene Mitarbeiter, die im Fall eines Problems verantwortlich sind. Weiterhin gibt es externe Zulieferer, die Fahrzeugteile produzieren, sowie Großhändler und Einzelkunden, die beim Unternehmen Fahrzeuge bestellen.

Die Aufgabe des DBMS ist nun diese Organisationseinheiten so zu verbinden, dass der Zeitverbrauch des gesamten Herstellungsprozesses nahezu optimal ist. Insbesondere haben alle Organisationseinheiten zeitliche Verzögerungen, die von dem Auftrag und den Entscheidungen des DBMS abhängig sind. Weiterhin können durch Komplikationen in der Teilelieferung sowie der Assemblierung Verspätungen entstehen. Das möchten wir im Folgenden konkretisieren, indem wir den Herstellungsablauf eines einzelnen Auftrags darlegen.

Zunächst geht in der Verwaltung eine Bestellung eines Kunden ein. Aus Organisationsgründen darf ein Kunde außerdem pro Auftrag nur eine Art Fahrzeug anfordern. Ein Verwaltungsmitarbeiter fügt diese Bestellung dann ins DBMS ein.

Sobald der Auftrag in einem Werk eingeht, wird dieser in eine Auftragsliste eingereiht. Innerhalb dieser Liste wird der voraussichtlich nächste Zeitpunkt, an dem das Werk wieder zur Verfügung steht, als Bearbeitungsverzögerung berechnet. Diese spielt eine heuristische Rolle in der Verteilung der Aufträge auf die Werke. Durch das DBMS wird zum ersten Auftrag in dieser Liste die Verfügbarkeit benötigter Teile geprüft. Werke produzieren aus Kostengründen nie weniger als eine bestimmte Stückzahl Fahrzeuge.

Dieses prüft zunächst, ob die bestellten Fahrzeuge bereits im zentralen Autolager zur Auslieferung vorliegen. Ansonsten wird der Auftrag demjenigen Werk zugeordnet, in dem am zeitnahesten produziert werden kann. Im Anschluss wird dem Kunden der Bestellungspreis und eine voraussichtliche Lieferzeit genannt, die pauschal berechnet werden. Abweichungen der voraussichtlichen von der erst später bekannten tatsächlichen Lieferzeit werden am Ende analysiert.

Die werkseigenen Lagerhallen werden so verwaltet, dass immer ein gewisser Mindestbestand an Autoteilen vorhanden ist, der die Bearbeitung der Mehrzahl der Aufträge erlaubt.

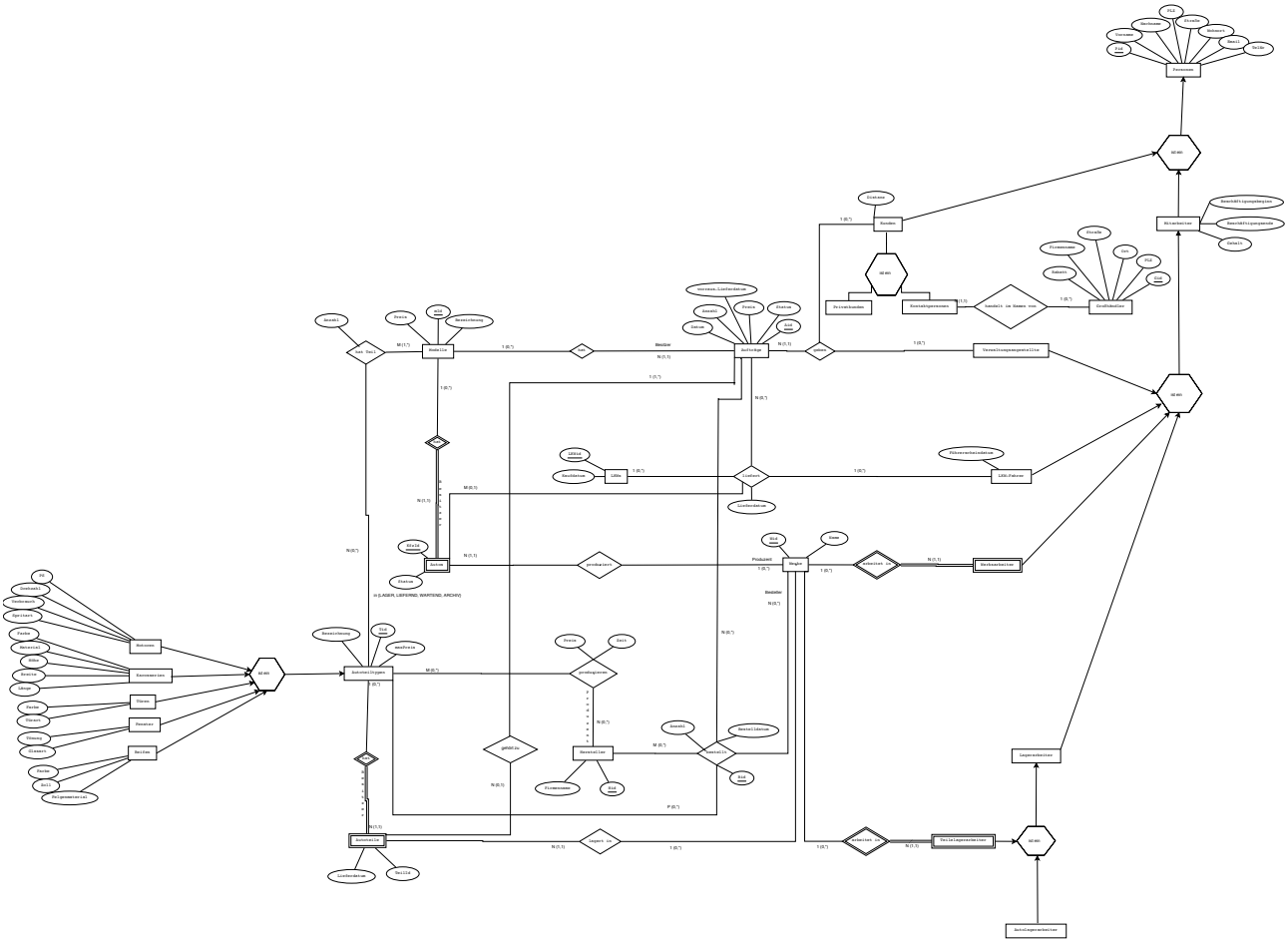
Stellt das DBMS nun aber fest, dass der Lagerbestand unzureichend ist, wird vom Lager aus an einen den Anforderungen genügenden Zulieferer eine Autoteilbestellung entsandt. Nun wartet das Werk auf die Ankunft der Teile im Lager, die zum Wareneingang von einem Lagermitarbeiter eingetragen wird und von Lieferkomplikationen abhängt.

Das Werk benötigt nun eine aufgrund der Fließbandtechnik konstante Montagezeit, sofern keine Probleme eintreten. Zuständige Mitarbeiter vermerken Anfang und Ende des Assemblierungsprozesses im DBMS.

Danach werden diese Fahrzeuge im Rahmen einer Testfahrt ins zentrale Autolager gefahren. Von dort wird die im Auftrag spezifizierte Fahrzeuganzahl an den entsprechenden Kunden ausgeliefert, wobei Überschüsse bei der Abarbeitung zukünftiger Bestellungen verwendet werden können. Hierfür wählt die Logistik einen verfügbaren LKW aus der Fahrzeugflotte aus. Ein Autolagermitarbeiter scannt ausgehende Fahrzeuge. Dieser konkrete Auftrag ist dann abgearbeitet und wird ins Archiv verschoben.

2. E/R Modell des Szenarios

2.1. Unser Modell



2.2. Unklare Zusammenhänge

Aufträge spielen in unserem Modell eine hervorgehobene Rolle, da diese in jeder Produktionsphase involviert sind und nach Bestellungsabschluss archiviert werden.

Der Entitätstyp “Autos” beschreibt nicht etwa das zentrale Autolager, sondern die Gesamtheit aller Fahrzeuge inklusive der bereits ausgelieferten archivierten. Das zentrale Autolager wird später über eine Sicht auf “Autos” modelliert, die nur diejenigen Fahrzeuge anzeigt, die aktuell im Lager sind. Hierfür haben wir uns entschieden, weil es nur ein Autolager gibt und aus diesem Grund ein Entitätstyp “Autolager” überflüssig ist. Damit gilt für Autolagerarbeiter, dass diese nur im zentralen Autolager arbeiten können und deswegen auch kein Beziehungstyp “arbeiten in” spezifiziert werden muss.

Der Teilelagerbestand wird durch den Entitätstyp “Autoteile” repräsentiert, der alle sich in Teilelagern befindenden Teile umfasst.

2.3. Änderungen

Wir haben folgende Änderungen vorgenommen:

1) Der Entitätstyp “Teilelager” wurde entfernt, da ein Teilelager vollständig durch die Attribute des entsprechenden Werks gegeben ist. Es ergibt daher aus Datenredundanzgründen keinen Sinn eine eigene Tabelle für Teilelager anzulegen. Fortan umfasst der Begriff Werk auch das zugehörige Teilelager.

2) Autos sind nun ein schwacher Entitätstyp, da diese in ihrer Existenz von der ihres Modells abhängen. Selbiges gilt nun für Autoteile und ihren Typ, Teilelagerarbeiter und ihr Werk sowie Werksarbeiter und ihr Werk.

3) Statt Großhändlern werden nun ihre Kontaktpersonen von der Generalisierung „Kunden“ umfasst, da dies einen einfacheren Umgang mit IDs ermöglicht.

4) Der Beziehungstyp “hat Teil” zwischen den Entitätstypen “Werke” (ehemals Teilelager) und “Autoteile” wurde in “lagert in” umbenannt.

5) Autoteile und Bestellungen haben nun eine Auftrags ID (AID) um eine Zuordnung zu einem Werksauftrag herstellen zu können. Insbesondere wird darüber ein Trigger zur Fortsetzung auf Teile wartender Werksaufträge implementiert.

6) Bestellungen haben nun eine Anzahl, da es sinnvoller ist Einzelbestellungen zusammenzufassen.

7) Modellteile haben nun ebenfalls eine Anzahl damit Mehrauftreten eines Teils innerhalb Modellen möglich sind.

8) Der Beziehungstyp “produzieren” hat nun das zusätzliches Attribut “Zeit”, welches die durchschnittliche Herstellungszeit beschreibt. Damit kann das voraussichtliche Lieferdatum besser approximiert werden.

9) Aufträge haben nun ein Attribut “Status”, das die Abarbeitung des Auftrags sowie die anschliessende Archivierung vereinfacht.

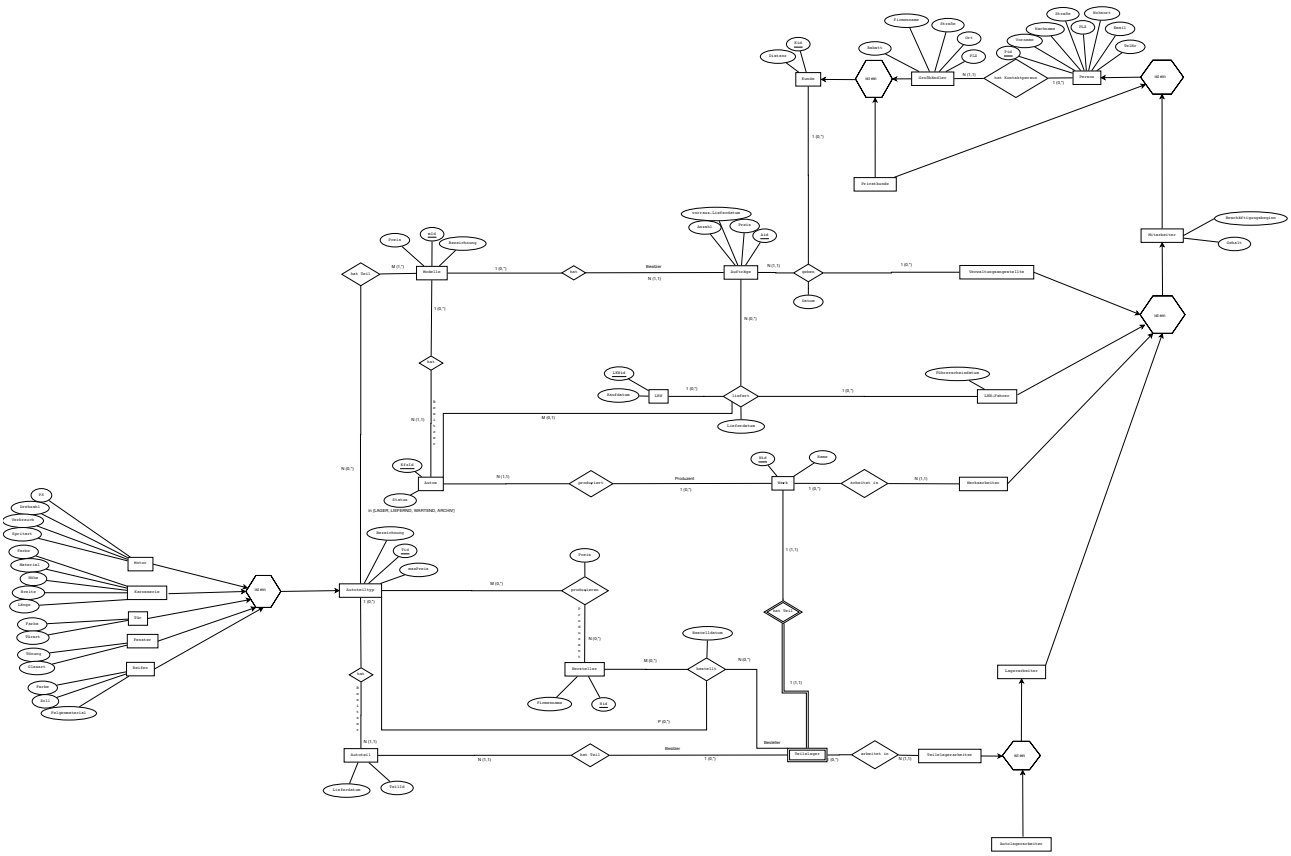
10) Mitarbeiter verfügen nun über ein Attribut “Beschäftigungsende”, welches die Archivierung ehemaliger Mitarbeiter ermöglicht.

11) Großhändler dürfen nun mehrere Kontaktpersonen haben.

12) Zwischen “Autoteilen” und “Aufträge” besteht nun eine Beziehung “gehört zu”, die die Reservierung von Autoteilen innerhalb der Teilelager zur Abarbeitung eines bestimmten Auftrags ermöglicht.

13) “Aufträge” nehmen nun auch am Beziehungstyp “bestellt” teil, damit Bestellungen ihrem Auftrag zugeordnet werden können.

Unser ursprüngliches Modell zur Referenz:



3. Relationales Modell

3.1. Liste der Relationen

[Personen] := {[PID: int, Vorname: string, Nachname: string, PLZ: string, Straße: string, Wohnort: string, Email: string, Tel.: string]}

[Mitarbeiter] := {[Beschäftigungsbegin: date, Beschäftigungsende: date, Gehalt: currency, PID: int -> Personen]}

[Werksarbeiter] := {[PID: int -> Mitarbeiter, WID: int -> Werke]}

[LKW-Fahrer] := {[PID: int -> Mitarbeiter, Führerscheindatum: date]}

[Verwaltungsangestellte] := {[PID: int -> Mitarbeiter]}

[Teilelagerarbeiter] := {[PID: int -> Mitarbeiter, WID: int -> Teilelager]}

[Autolagerarbeiter] := {[PID: int -> Mitarbeiter]}

[Lagerarbeiter] := {[PID: int -> Mitarbeiter]}

[Kunden] := {[Distanz: int, PID: int -> Personen]}

[Privatkunden] := {[PID: int -> Kunden]}

[Kontaktpersonen] := {[PID: int -> Kunden, GID: int -> Großhändler]}

[Großhändler] := {[GID: int, Rabatt: int, Firmenname: string, Straße: string, Ort: string, PLZ: string]}

[Aufträge] := {[AID: int, Preis: currency, Voraussichtliches_Lieferdatum: date, Modell-ID: int -> Modelle, Anzahl: int, Datum: date, KundenID: int -> Kunden, MitarbeiterID: int -> Verwaltungsangestellte, Status: string]}

[Modelle] := {[Preis: currency, ModellID: int, Bezeichnung: string]}

[Autoteiltypen] := {[TeiletypID: int, maxPreis: currency, Bezeichnung: string]}

[Modellteile] := {[Modell-ID: int -> Modelle, TeiletypID: int -> Autoteiltypen, Anzahl: int]}

[Autos] := {[KFZ-ID: int, Status: string, Modell-ID: int -> Modelle, produziertVon: int - > Werke]}

[LKWs] := [{ LKW-ID: int, Kaufdatum: date }]

[liefert] := [{ LKW-ID: int -> LKWs, Modell_ID: int -> Autos, KFZ-ID: int -> Autos, MID: int -> LKW-Fahrer, AID: int -> Aufträge, Lieferdatum: date }]

[Hersteller] := [{ HID: int, Firmennamen: string }]

[produzieren] := [{ TeiltypID: int -> Autoteiltypen, HID: int -> Hersteller, Zeit: int }]

[bestellt] := [{ BID: int HID: int -> Hersteller, WID: int -> Werke, TeiltypID: int -> Autoteiltypen, Bestelldatum: date, Anzahl: int, AID: int -> Aufträge }]

[Autoteile] := [{ TeileID: int, TeilTypID: int -> Autoteiltypen, lagertIn: int -> Teilelager, Lieferdatum: date, AID: int -> Aufträge }]

[Motoren] := [{ TeiltypID: int -> Autoteiltypen, Ps: int, Drehzahl: int, Verbrauch: int, Spritart: string}]

[Karosserien] := [{ TeiltypID: int -> Autoteiltypen, Farbe: string, Material: string, Höhe: int, Breite: int, Länge: int }]

[Türen] := [{ TeiltypID: int -> Autoteiltypen, Farbe: string, Türart: string }]

[Fenster] := [{ TeiltypID: int -> Autoteiltypen, Tönung: string, Glasart: string }]

[Werke] := [{ WID: int, Name: string, }]

[Werksaufträge] := [{ WID: int -> Werke, AID: int -> Aufträge, Status string }]

[Reifen] := [{ TeiltypID: int -> Autoteiltypen, Farbe: string, Zoll: int, Felgenmaterial: string }]

Anmerkungen:

- Die Relation Werksaufträge modelliert den Beziehungstypen “produziert” im Sinne einer Tabelle sich aktuell in Produktion befindender Fahrzeuge der Werke.
- Zur Auflösung jeglicher Generalisierungen haben wir vertikale Partitionierung verwendet, um Datenredundanz zu vermeiden.
- Der LKW-Fahrer in der Beziehung “liefert” ist eindeutig gegeben durch LKW-ID, KFZ-ID, Modell_ID und AID.
- Die Zeit der Relation “produzieren” wird in Tagen angegeben.

3.2. Zusätzliche Einschränkungen und Integritätsbedingungen

- Postleitzahlen und Emails sollten zumindest rudimentär auf Gültigkeit überprüft werden.
- Rabatte müssen einen Wert zwischen null und hundert annehmen.
- Der Beschäftigungsbeginn eines Mitarbeiters sowie das Kaufdatum eines LKWs muss in der Vergangenheit liegen.
- Die Distanz eines Kunden, die Autoanzahl bei einem Auftrag, PS, Drehzahl und Verbrauch eines Motors, Zeiten, Preise und Gehälter müssen positiv sein.
- Werksarbeiter müssen einem Werk zugeordnet sein bzw. eine WID haben.
- Lagerarbeiter müssen eine WID (Werks ID) haben, damit ihnen eindeutig ein Werk und somit Lager zugeordnet werden kann.

- Das Führerscheindatum eines LKW-Fahrers muss mindestens drei Jahre in der Vergangenheit liegen.
- Voraussichtliche Lieferdaten von Aufträgen müssen beim Einfügen in der Zukunft liegen.
- Der Status eines Autos muss einen der folgenden Werte annehmen: {lagernd, liefernd, wartend, archiviert}.
- Analog gilt für die Türart einer Tür, dass sie in {Standard, Kofferraum, Schiebetür, Flügeltür} liegen darf.
- Die bei der Erstellung eines neuen Modells benötigten Autoteiltypen müssen vorher bereits eingeführt worden sein.
- Beim Einfügen in die Tabelle Modellteile muss sichergestellt werden, dass teilnehmende Autoteiltypen sowie das entsprechende Modell bereits vorhanden sind.
- Kontaktpersonen müssen genau einem Großhändler zugeordnet sein.
- Die AuftragsID einer Bestellung ist initial NULL, beinhaltet jedoch die AID, falls beim Eingang des Auftrags Teile vom Werk bestellt worden sind. Dies gilt auch für produzierte Autoteile.
- Das Beschäftigungsende der Relation Mitarbeiter ist initial NULL
- Für jeden Autoteiltyp eines Modells muss es mindestens einen Hersteller geben.
- Teiletypen und Modelle dürfen nur gelöscht werden, sofern sie noch nicht Teil eines Auftrags waren

4. Datenbankschema

4.1. SQL

4.1.1. Tabellenerzeugung (daten.sql, dataBasic.sql für umfangreichere Daten)

```
CREATE TABLE Personen (
      PID serial,
      Vorname varchar NOT NULL,
      Nachname varchar NOT NULL,
      PLZ varchar(10) NOT NULL,
      Straße varchar(50) NOT NULL,
      Wohnort varchar(50) NOT NULL,
      Email varchar(50) NOT NULL,
      TelNr string NOT NULL,

      CONSTRAINT personenPK PRIMARY KEY (PID),

      CONSTRAINT validEmail CHECK (Email LIKE '%_@__%.__%'),
      CONSTRAINT validPLZ CHECK (PLZ LIKE '_____')
);

CREATE TABLE Werke (
      WID serial,
      Name varchar NOT NULL,

      CONSTRAINT werkePK PRIMARY KEY (WID)
);

CREATE TABLE Mitarbeiter (
      PID integer,
      Beschäftigungsbeginn date NOT NULL,
      Gehalt numeric(10,2) NOT NULL,
      Beschäftigungsende date DEFAULT NULL,

      FOREIGN KEY (PID) REFERENCES Personen,

      CONSTRAINT mitarbeiterPK PRIMARY KEY (PID),
      CONSTRAINT notSlave CHECK (Gehalt>0)
);

CREATE TABLE Werksarbeiter (
      PID integer,
      WID integer REFERENCES Werke,

      FOREIGN KEY (PID) REFERENCES Mitarbeiter,

      CONSTRAINT werksarbeiterPK PRIMARY KEY (PID)
);


CREATE TABLE LKW_Fahrer (
      PID integer,
      Führerscheindatum date NOT NULL,

      FOREIGN KEY (PID) REFERENCES Mitarbeiter,

      CONSTRAINT lkwFahrerPK PRIMARY KEY (PID)
);


CREATE TABLE Verwaltungsangestellte (
      PID integer,

      FOREIGN KEY (PID) REFERENCES Mitarbeiter,

      CONSTRAINT verwaltungsangestelltePK PRIMARY KEY (PID)
);
```

```
CREATE TABLE Lagerarbeiter (
      PID integer,

      FOREIGN KEY (PID) REFERENCES Mitarbeiter,

      CONSTRAINT lagerarbeiterPK PRIMARY KEY (PID)
);

CREATE TABLE Teilelagerarbeiter (
      PID integer,
      WID integer NOT NULL,

      FOREIGN KEY (WID) REFERENCES Werke,
      FOREIGN KEY (PID) REFERENCES Lagerarbeiter,

      CONSTRAINT teilelagerarbeiterPK PRIMARY KEY (PID)
);

CREATE TABLE Autolagerarbeiter (
      PID integer,

      FOREIGN KEY (PID) REFERENCES Lagerarbeiter,

      CONSTRAINT autolagerarbeiterPK PRIMARY KEY (PID)
);

CREATE TABLE Großhändler (
      GID serial,
      Firmenname varchar(50) NOT NULL,
      Straße varchar(50) NOT NULL,
      PLZ varchar (10) NOT NULL,
      Ort varchar(50) NOT NULL,
      Rabatt integer,

      CONSTRAINT validPLZ CHECK (PLZ LIKE '_____'),
      CONSTRAINT validDiscount CHECK (rabatt>=0 AND rabatt<=100),
      CONSTRAINT großhändlerPK PRIMARY KEY (GID)
);

CREATE TABLE Modelle (
      Modell_ID serial,
      Preis numeric(10,2) NOT NULL,
      Bezeichnung varchar NOT NULL,

      CONSTRAINT modellePK PRIMARY KEY (Modell_ID),

      CONSTRAINT validPrice CHECK(Preis>0)
);

CREATE TABLE Kunden (
      PID integer,
      Distanz integer NOT NULL,

      FOREIGN KEY (PID) REFERENCES Personen,

      CONSTRAINT kundenPK PRIMARY KEY (PID),

      CONSTRAINT validDistance CHECK(Distanz>0)
);

CREATE TABLE Privatkunden (
      PID integer,

      FOREIGN KEY (PID) REFERENCES Kunden,

      CONSTRAINT privatkundenPK PRIMARY KEY (PID)
);
```

```
CREATE TABLE Kontaktpersonen (
      PID integer,
      GID integer,

      FOREIGN KEY (GID) REFERENCES Großhändler,
      FOREIGN KEY (PID) REFERENCES Kunden,

      CONSTRAINT kontaktpersonenPK PRIMARY KEY (PID)
);

CREATE DOMAIN Auftragsstatus AS varchar(14)
      CHECK (VALUE ~ 'WARTEND' OR VALUE~'IN_BEARBEITUNG' OR VALUE~'ARCHIVIERT');

CREATE TABLE Aufträge (
      AID serial,
      Preis numeric(10,2) DEFAULT 0,
      Vorraussichtliches_Lieferdatum date,
      Modell_ID integer NOT NULL,
      Anzahl integer NOT NULL,
      Datum date DEFAULT now(),
      KundenID integer NOT NULL,
      MitarbeiterID integer NOT NULL,
      Status Auftragsstatus DEFAULT 'WARTEND',


      FOREIGN KEY (Modell_ID) REFERENCES Modelle,
      FOREIGN KEY (KundenID) REFERENCES Kunden,
      FOREIGN KEY (MitarbeiterID) REFERENCES Mitarbeiter,

      CONSTRAINT aufträgePK PRIMARY KEY (AID),

      CONSTRAINT validPriceAndCount CHECK(Preis>=0 AND Anzahl>0)
);

CREATE TABLE Werksaufträge (
      WID integer,
      AID integer,
      Status Auftragsstatus DEFAULT 'WARTEND',
      Herstellungsbeginn date DEFAULT NULL,
      Herstellungsende date DEFAULT NULL,


      FOREIGN KEY (WID) REFERENCES Werke,
      FOREIGN KEY (AID) REFERENCES Aufträge,

      CONSTRAINT werksaufträgePK PRIMARY KEY (AID)
);

CREATE TABLE Autoteiltypen (
      TeiletypID serial,
      maxPreis numeric(10,2) NOT NULL,
      Bezeichnung varchar NOT NULL,

      CONSTRAINT autoteiltypenPK PRIMARY KEY (TeiletypID),

      CONSTRAINT validMaxPrice CHECK (maxPreis>0)
);

CREATE TABLE Modellteile (
      Modell_ID integer,
      TeiletypID integer,
      Anzahl integer,

      FOREIGN KEY (Modell_ID) REFERENCES Modelle,
      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,

      CONSTRAINT modellteilePK PRIMARY KEY (Modell_ID, TeiletypID)
);
```

```
CREATE DOMAIN Autostatus AS varchar(10) CHECK (VALUE~'LAGERND' OR
VALUE~'LIEFERND' OR VALUE~'ARCHIVIERT' OR VALUE~'WARTEND');

CREATE TABLE Autos (
      KFZ_ID serial,
      Modell_ID integer,
      Status Autostatus DEFAULT 'LAGERND',
      produziertVon integer NOT NULL,

      FOREIGN KEY (Modell_ID) REFERENCES Modelle,
      FOREIGN KEY (produziertVon) REFERENCES Werke,

      CONSTRAINT autosPK PRIMARY KEY (KFZ_ID, Modell_ID)
);

CREATE TABLE LKWs (
      LKW_ID serial,
      Kaufdatum date NOT NULL,

      CONSTRAINT lkwsPK PRIMARY KEY (LKW_ID)
);

-- Lieferdatum null = noch nicht ausgeliefert
CREATE TABLE liefert (
      -- Bei löschen soll in ausgeführten
      -- Lieferungen NULL bei LKW stehen
      LKW_ID integer NOT NULL,
      KFZ_ID integer,
      Modell_ID integer,
      MID integer,
      AID integer,
      Lieferdatum date,

      FOREIGN KEY (KFZ_ID, Modell_ID) REFERENCES Autos,
      FOREIGN KEY (MID) REFERENCES Mitarbeiter,
      FOREIGN KEY (LKW_ID) REFERENCES LKWs ON DELETE SET NULL,
      FOREIGN KEY (AID) REFERENCES Aufträge,

      CONSTRAINT liefertPK PRIMARY KEY (KFZ_ID, MID, AID)
);

CREATE TABLE Hersteller (
      HID serial,
      Firmenname varchar NOT NULL,

      CONSTRAINT herstellerPK PRIMARY KEY (HID)
);

CREATE TABLE produzieren (
      TeiletypID integer,
      HID integer,
      Preis numeric(10,2) NOT NULL,
      Zeit integer NOT NULL, -- represents days

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,
      FOREIGN KEY (HID) REFERENCES Hersteller,

      CONSTRAINT produzierenPK PRIMARY KEY (TeiletypID, HID),
      CONSTRAINT validTime CHECK (Zeit>0),
      CONSTRAINT validPrice CHECK (Preis>0)
);

CREATE DOMAIN Bestellungsstatus AS varchar(10) CHECK (VALUE~'ARCHIVIERT' OR
VALUE~'BESTELLT');
```

```
CREATE TABLE bestellt (
      BID SERIAL,
      HID integer,
      WID integer,
      TeiletypID integer,
      Anzahl integer,
      Bestelldatum date,
      AID integer,
      Status Bestellungsstatus DEFAULT 'BESTELLT',

      FOREIGN KEY (HID) REFERENCES Hersteller,
      FOREIGN KEY (WID) REFERENCES Werke,
      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,
      FOREIGN KEY (AID) REFERENCES Aufträge,

      CONSTRAINT bestelltPK PRIMARY KEY (BID)
);

--Auftrag NULL bedeutet, dass das Teil verfügbar ist, da es keinem Auftrag
zugeordnet ist
CREATE TABLE Autoteile (
      TeileID serial,
      TeiletypID integer,
      lagert_in integer,
      Lieferdatum date,
      AID integer DEFAULT NULL,

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,
      FOREIGN KEY (lagert_in) REFERENCES Werke,
      FOREIGN KEY (AID) REFERENCES Aufträge,

      CONSTRAINT autoteilePK PRIMARY KEY (TeileID)
);

CREATE TABLE Motoren (
      TeiletypID integer,
      PS integer NOT NULL,
      Drehzahl integer NOT NULL,
      Verbrauch integer NOT NULL,
      Spritart varchar NOT NULL,

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,

      CONSTRAINT motorenPK PRIMARY KEY (TeiletypID),

      CONSTRAINT validData CHECK (PS>0 AND Drehzahl>0 AND Verbrauch>0)
);

CREATE TABLE Karosserien (
      TeiletypID integer,
      Farbe varchar NOT NULL,
      Material varchar NOT NULL,
      Höhe integer NOT NULL,
      Breite integer NOT NULL,
      Länge integer NOT NULL,

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,

      CONSTRAINT karosserienPK PRIMARY KEY (TeiletypID),

      CONSTRAINT validData CHECK (Höhe>0 AND Breite>0 AND Länge>0)
);

CREATE DOMAIN Türart AS varchar(10) CHECK (VALUE~'FLÜGELTÜR' OR
VALUE~'KOFFERRAUM' OR VALUE~'SCHIEBETÜR' OR VALUE~'STANDARD' );

CREATE TABLE Türen (
      TeiletypID integer,
      Farbe varchar NOT NULL,
      Türart varchar,

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,
```

```
      CONSTRAINT türenPK PRIMARY KEY (TeiletypID)
);

CREATE TABLE Fenster (
      TeiletypID integer,
      Tönung varchar NOT NULL,
      Glasart varchar NOT NULL,

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,

      CONSTRAINT fensterPK PRIMARY KEY (TeiletypID)
);

CREATE TABLE Reifen (
      TeiletypID integer,
      Farbe varchar NOT NULL,
      Zoll integer NOT NULL,
      Felgenmaterial varchar NOT NULL,

      FOREIGN KEY (TeiletypID) REFERENCES Autoteiltypen,

      CONSTRAINT reifenPK PRIMARY KEY (TeiletypID),

      CONSTRAINT validData CHECK (Zoll>0)
);
```

4.1.2. Funktionen und funktionale Trigger

Wir haben es uns zum Ziel gemacht die Abarbeitung eines Auftrags so weit wie möglich zu automatisieren. Hierfür bedienen wir uns an einer Reihe funktionaler Trigger, die anhand gewisser Heuristiken (in Form von Funktionen) einfache Aufgaben wie beispielsweise die Auftragsverteilung auf Werke, Nachbestellung fehlender Teile und Vergabe eines Lieferauftrags kompetent erledigen.

```
--Lieferungen können nicht gelöscht werden, stattdessen werden sie archiviert.
--@param $1 - Die KFZ_ID des Fahrzeugs, das geliefert wurde
--@param $2 - Die Modell_ID des Fahrzeugs
--@param $3 - Die ID des LKW-Fahrers
--@param $4 - Die ID des Auftrags.
CREATE FUNCTION finishedDelivery(integer, integer, integer, integer) RETURNS
boolean AS
      $$ BEGIN
            UPDATE Autos SET Status='ARCHIVIERT'
                  WHERE kfz_id=$1 AND modell_id=$2;
            UPDATE liefert SET Lieferdatum=CURRENT_DATE
                  WHERE KFZ_ID=$1 AND MID=$3 AND AID=$4;
            UPDATE Aufträge SET Status='ARCHIVIERT' WHERE AID=$4;
            RETURN true;
      END; $$ LANGUAGE plpgsql;

CREATE RULE setOnDeliveryFinished AS ON DELETE TO liefert DO INSTEAD SELECT
finishedDelivery(OLD.kfz_id, OLD.modell_id, OLD.MID, OLD.AID);
```

```
--Startet in einem Werk den nächsten Auftrag, falls möglich.
--@param $1 - Das Werk, in dem geprüft werden soll
CREATE FUNCTION checkAvailableWork(integer) RETURNS boolean AS
      $$ DECLARE
            auftrag integer;
            countNeeded integer;
            available boolean;
            notmissing boolean;
      BEGIN
      IF(EXISTS (SELECT 1 FROM Werksaufträge
                  WHERE WID=$1 AND Status='IN_BEARBEITUNG') )
      THEN RETURN false;
      END IF;
      FOR auftrag IN SELECT AID FROM Werksaufträge WHERE WID=$1
      LOOP
            countNeeded := (SELECT Anzahl FROM Aufträge WHERE AID=auftrag);
            available:=(NOT EXISTS (SELECT 1 FROM
                  -- Wähle Autoteile, die diesem Auftrag zugeordnet sind
                  ((SELECT TeiletypID, count(*)
                        FROM Autoteile
                        WHERE lagert_in = $1
                        AND AID=auftrag GROUP BY TeiletypID) AS tmp1
                   RIGHT OUTER JOIN
                  --Anzahl benötigter Teile um NEW.Anzahl Autos herzustellen
                  (SELECT TeiletypID, (Anzahl * countNeeded) AS teileNeeded
                        FROM ModellTeile
                        WHERE Modell_ID=(SELECT Modell_ID FROM
                        Aufträge WHERE AID = auftrag)) AS tmp2
                  ON tmp1.TeiletypID=tmp2.TeiletypID)
                  WHERE teileNeeded>count OR count IS NULL LIMIT 1));
            --Sind genügend Teile im Lager?
            IF (available) THEN
                  IF(NOT EXISTS (SELECT * FROM Werksaufträge WHERE
                        Status='IN_BEARBEITUNG' AND WID=$1))
                  THEN UPDATE Werksaufträge SET Status='IN_BEARBEITUNG' WHERE
                        WID=$1 AND AID=auftrag;
                       UPDATE Werksaufträge SET Herstellungsbeginn=CURRENT_DATE
                        WHERE WID=$1 AND AID=auftrag;
                       RETURN true;
                  END IF;
            END IF;
            notmissing:=(NOT EXISTS (SELECT 1 FROM
                  -- Wähle Autoteile, die keinem Auftrag zugeordnet sind
                  -- (AID ist NULL)
                  ((SELECT TeiletypID, count(*) FROM Autoteile
                        WHERE lagert_in = $1
                        AND AID IS NULL GROUP BY TeiletypID) AS tmp1
                  RIGHT OUTER JOIN
```

```
                --Anzahl benötigter Teile um NEW.Anzahl Autos herzustellen
                (SELECT TeiletypID, (Anzahl * countNeeded) AS teileNeeded
                      FROM ModellTeile
                      WHERE Modell_ID=(SELECT Modell_ID
                                       FROM Aufträge
                                       WHERE AID =auftrag)) AS tmp2
                ON tmp1.TeiletypID=tmp2.TeiletypID)
                WHERE teileNeeded>count OR count IS NULL LIMIT 1));
          IF (notmissing) THEN
                IF(NOT EXISTS (SELECT * FROM Werksaufträge
                      WHERE Status='IN_BEARBEITUNG' AND WID=$1))
                THEN UPDATE Werksaufträge SET Status='IN_BEARBEITUNG'
                      WHERE WID=$1 AND AID=auftrag;
                     UPDATE Werksaufträge SET Herstellungsbeginn=CURRENT_DATE
                      WHERE WID=$1 AND AID=auftrag;
                     RETURN true;
                END IF;
          END IF;
    END LOOP;
    RETURN false;
    END; $$ LANGUAGE plpgsql;


--Sobald Teile ankommen muss geprüft werden, ob ein Auftrag zum Ausführen bereit
ist.
CREATE FUNCTION carPartsArrived() RETURNS TRIGGER AS
    $$ DECLARE
          teilid integer;
          countNeeded integer;
          available boolean;
          counter integer;
    BEGIN
    counter:=OLD.Anzahl;
    IF (OLD.Status='ARCHIVIERT' OR NEW.Status!='ARCHIVIERT')
    THEN RETURN NEW;
    END IF;
    LOOP
    EXIT WHEN counter=0;
          INSERT INTO Autoteile (TeiletypID, lagert_in, Lieferdatum, AID)
                VALUES (OLD.TeiletypID, OLD.WID, now(), OLD.AID);
          UPDATE Autoteile SET Lieferdatum=CURRENT_DATE
                WHERE TeileID=lastVal();
          counter=counter-1;
    END LOOP;
    PERFORM checkAvailableWork(OLD.WID);
    RETURN NEW;
    END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER setOnCarPartsArrived AFTER UPDATE ON bestellt FOR EACH ROW EXECUTE
PROCEDURE carPartsArrived();


--Wählt aus bei welchem Hersteller ein Teil bestellt werden soll.
--@param $1 - Die ID des Teiletyps, der bestellt werden soll.
CREATE FUNCTION getBestManufacturer(integer) RETURNS integer AS
      $$ BEGIN
      RETURN
      (SELECT HID
            FROM  (SELECT * FROM produzieren WHERE TeiletypID=$1
                  ORDER BY Zeit ASC
                  FETCH FIRST 3 ROWS ONLY) AS tmp
            ORDER BY Preis ASC
            FETCH FIRST 1 ROWS ONLY);
      END; $$ LANGUAGE plpgsql;


-- Wenn ein neuer Auftrag im Werk ankommt muss das Lager geprüft werden,
eventuell Teile bestellt werden und eine Produktion kann unter Umständen schon
beginnen.
CREATE FUNCTION insertInJobs() RETURNS TRIGGER AS
$$
      DECLARE
            missing boolean;
            neededParts integer;
            part integer;
            countNeeded integer;
      BEGIN
      countNeeded := (SELECT Anzahl FROM Aufträge WHERE AID=NEW.AID);
      missing:=(EXISTS (SELECT 1 FROM
            -- Wähle Autoteile, die keinem Auftrag zugeordnet sind
            -- d.h. AID ist NULL
            ((SELECT TeiletypID, count(*)
                  FROM Autoteile
                  WHERE lagert_in = NEW.WID AND AID IS NULL
                  GROUP BY TeiletypID) AS tmp1
            RIGHT OUTER JOIN
            --Anzahl benötigter Teile um NEW.Anzahl Autos herzustellen
            (SELECT TeiletypID, (Anzahl * countNeeded) AS teileNeeded
                  FROM ModellTeile
                  WHERE Modell_ID=(SELECT Modell_ID
                                    FROM Aufträge
                                    WHERE AID = NEW.AID)) AS tmp2
            ON tmp1.TeiletypID=tmp2.TeiletypID)
            WHERE teileNeeded>count OR count IS NULL LIMIT 1));
```

```
--Sind genügend Teile im Lager?
IF (missing) THEN --NEIN
	--Iteriere über benötigte Teile
	FOR part IN (SELECT TeiletypID
			FROM Modellteile
			WHERE Modell_ID=(SELECT Modell_ID
						FROM Aufträge
						WHERE AID=NEW.AID))
	LOOP
		--Anzahl wieoft Teil gebraucht wird.
		neededParts := (SELECT Anzahl*
			(SELECT Anzahl FROM Aufträge WHERE AID=NEW.AID)
		FROM ModellTeile
			WHERE Modell_ID=(SELECT Modell_ID
						FROM Aufträge
						WHERE AID=NEW.AID)
				AND TeiletypID = part);
		IF(neededParts > 0) THEN
			--Bestelle die Teile
			INSERT INTO bestellt (HID, WID, TeiletypID,
				Anzahl, Bestelldatum, AID)
				VALUES (getBestManufacturer(part),
				NEW.WID, part, neededParts,
			now(), NEW.AID);
		END IF;
	END LOOP;
ELSE --JA
	--Iteriere über benötigte Teile
	FOR part IN (SELECT TeiletypID
			FROM Modellteile
			WHERE Modell_ID=(SELECT Modell_ID
						FROM Aufträge
						WHERE AID=NEW.AID))
	LOOP
		--Anzahl wieoft Teil gebraucht wird.
		neededParts := (SELECT Anzahl*
			(SELECT Anzahl FROM Aufträge WHERE AID=NEW.AID)
			FROM ModellTeile
			WHERE Modell_ID=(SELECT Modell_ID
						FROM Aufträge
						WHERE AID=NEW.AID)
				AND TeiletypID=part);
		IF(neededParts > 0) THEN
			--Bestelle die Teile
			INSERT INTO bestellt (HID, WID, TeiletypID,
				Anzahl, Bestelldatum, AID)
```

```
                                VALUES (getBestManufacturer(part),
                                NEW.WID, part, neededParts,
                                now(), NULL);
                        --Setze bei genau einem Teil des benötigten Typs
                        --den Auftragsstatus
                        UPDATE Autoteile SET AID=NEW.AID
                                FROM (SELECT TeileID
                                        FROM Autoteile
                                        WHERE TeiletypID=part
                                                AND AID IS NULL
                                        ORDER BY ID ASC LIMIT neededParts)
                                                AS tmp3;
                END IF;
        END LOOP;
        -- Es sind alle Teile da, wird irgendein auftrag gerade
        -- aufgeführt? Wenn nein dann führe diesen hier aus.
        IF(NOT EXISTS (SELECT *
                        FROM Werksaufträge
                        WHERE Status='IN_BEARBEITUNG' AND WID=NEW.WID))
        THEN UPDATE Werksaufträge SET Status='IN_BEARBEITUNG'
                WHERE WID=NEW.WID AND AID=NEW.AID;
             UPDATE Werksaufträge SET Herstellungsbeginn=CURRENT_DATE
                WHERE WID=NEW.WID AND AID=NEW.AID;
        END IF;
END IF;
RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER onInsertWerksaufträge AFTER INSERT ON Werksaufträge FOR EACH ROW
EXECUTE PROCEDURE insertInJobs();



--Schätzt die Werksauslastung ab, wird benötigt um Lieferzeiten zu
--prognostizieren und Aufträge auf Werke zu verteilen.
--@param $1 - Die Id des Werkes, das abgeschätzt werden soll.
CREATE OR REPLACE FUNCTION getWerksauslastung(integer) RETURNS interval AS
        $$ DECLARE
                auftrag integer;
                expectedTime numeric(20,2);
                liefer date;
                switchCounter integer;
        BEGIN
        switchCounter:=0;
        expectedTime=0;
        FOR auftrag IN (SELECT Werksaufträge.AID
                        FROM Werksaufträge
                        WHERE WID=$1 AND Status='WARTEND')
```

```
      LOOP
            switchCounter:=switchCounter +1;
            liefer:=(SELECT Vorraussichtliches_Lieferdatum
                        FROM Aufträge
                        WHERE AID=auftrag);
            expectedTime = expectedTime + (liefer - CURRENT_DATE)+0.8;
      END LOOP;
      expectedTime=expectedTime + switchCounter;
      IF(EXISTS(SELECT 1
                  FROM Werksaufträge
                  WHERE WID=$1 AND Status='IN_BEARBEITUNG'))
      THEN expectedTime=expectedTime + 0.35;
      END IF;
      RETURN CEIL(expectedTime) * interval '1 days';
      END; $$ LANGUAGE plpgsql;



--Schätzt die Fahrzeit nach Distanz ab.
--@param $1 - Die Distanz in KM, die zu fahren ist.
CREATE OR REPLACE FUNCTION getTimeForDistance(integer) RETURNS interval AS
      $$ BEGIN
      -- 50km/h
      RETURN CEIL(($1/50)/24)*interval '1 days';
      END;
      $$ LANGUAGE plpgsql;



-- Überprüft, ob bestellte Autos bereits im Lager verfügbar sind.
--@param $1 - Die Modell_ID, der gewünschten Fahrzeuge
--@param $2 - Die Anzahl der gewünschten Fahrzeuge
CREATE OR REPLACE FUNCTION checkCarStock(integer, integer) RETURNS boolean AS
      $$
      DECLARE
            counting integer;
      BEGIN
      counting = (SELECT count(*) AS Anzahl
                  FROM Autos
                  WHERE Modell_ID = $1 AND Status = 'LAGERND');
      RETURN counting >= $2;
END; $$ LANGUAGE plpgsql;
```

```
--Prüft, ob ein LKW zur Lieferung verfügbar ist.
CREATE OR REPLACE FUNCTION checkLkwAvailable() RETURNS integer AS
      $$ BEGIN
            RETURN
            ((SELECT LKW_ID FROM LKWs
              EXCEPT
              SELECT LKW_ID FROM liefert WHERE Lieferdatum IS NULL)
              ORDER BY LKW_ID ASC
              FETCH FIRST 1 ROWS ONLY);
      END; $$ LANGUAGE plpgsql;


-- Prüft, ob ein Fahrer zur Lieferung verfügbar ist.
CREATE OR REPLACE FUNCTION checkDriverAvailable() RETURNS integer AS
      $$ BEGIN
      RETURN ((SELECT PID FROM LKW_Fahrer
               EXCEPT
               SELECT MID AS PID FROM liefert WHERE Lieferdatum IS NULL)
               ORDER BY PID ASC
               FETCH FIRST 1 ROWS ONLY);
      END; $$ LANGUAGE plpgsql;



--Sobald ein neuer Auftrag ankommt, muss geprüft werden, ob die Autos schon im
Lager verfügbar sind, ansonsten wird der Produktionsauftrag einem Werk mit
geringer Auslastung zugewiesen.
CREATE OR REPLACE FUNCTION insertInOrders() RETURNS TRIGGER AS
      $$ DECLARE
            orderInStock boolean;
            counter integer;
            cars integer;
            driver integer;
            lkw integer;
            distance integer;
            werk integer;
            minwerktime interval;
            minwerkid integer;
      BEGIN
      minwerkid:=(SELECT WID FROM Werksaufträge LIMIT 1);
      IF(minwerkid IS NULL)
      THEN minwerkid:=(SELECT WID FROM Werke LIMIT 1);
      END IF;
      minwerktime:=getWerksauslastung(minwerkid);
      orderInStock := checkCarStock(NEW.Modell_ID, NEW.Anzahl);
      driver := checkDriverAvailable();
      lkw := checkLkwAvailable();
      distance := (SELECT Distanz
```

```
            FROM Kunden
            WHERE PID = (SELECT KundenID
                        FROM Aufträge
                        WHERE AID=NEW.AID));
counter := (SELECT Anzahl FROM Aufträge WHERE AID=NEW.AID);
IF orderInStock THEN
-- Autos sind schon im Autolager
      UPDATE Aufträge SET Vorraussichtliches_Lieferdatum=now()
            +getTimeForDistance(distance) WHERE AID=NEW.AID;
      IF (lkw IS NULL OR driver IS NULL) THEN
      --NEIN, dann Lagere Autos vorübergehend
            UPDATE Autos SET Status='WARTEND'
                  WHERE Modell_ID = NEW.Modell_ID
                        AND KFZ_ID IN (SELECT KFZ_ID
                                    FROM Autos
                                    WHERE Modell_ID=NEW.Modell_ID);
      ELSE --JA, dann liefere sofort.
            FOR cars IN (SELECT KFZ_ID
                        FROM Autos
                        WHERE Modell_ID = NEW.Modell_ID
                              AND Status = 'LAGERND')
            LOOP
            EXIT WHEN counter = 0;
                  INSERT INTO liefert (LKW_ID, KFZ_ID, Modell_ID, MID,
                        AID, Lieferdatum)
                  VALUES (lkw, cars, NEW.Modell_ID, driver, NEW.AID,
                        NULL);
                  counter=counter-1;
            END LOOP;
      END IF;
ELSE  --NEIN, dann produziere
      FOR werk IN (SELECT WID FROM Werke)
      LOOP
            IF (minwerktime>=getWerksauslastung(werk)) THEN
                  minwerktime=getWerksauslastung(werk);
                  minwerkid=werk;
            END IF;
      END LOOP;
      UPDATE Aufträge SET Vorraussichtliches_Lieferdatum=now()+minwerktime
                  +getTimeForDistance(distance)+
                  (CEIL(counter*1.5/24)*interval '1 days')
            WHERE AID=NEW.AID;
      INSERT INTO Werksaufträge (WID, AID) VALUES (minwerkid, NEW.AID);
END IF;
RETURN NULL;
END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER onInsertAufträge AFTER INSERT ON Aufträge FOR EACH ROW EXECUTE
PROCEDURE insertInOrders();



--Bei Einchecken eines fertigen Auftrags werden die Autos eingefügt.
CREATE FUNCTION finishedJob() RETURNS TRIGGER AS
      $$ DECLARE
            counter integer;
            modellid integer;
            werk integer;
            kfzid integer;
            lkw integer;
            fahrer integer;
      BEGIN
      lkw=checkLkwAvailable();
      fahrer=checkDriverAvailable();
      --Setze Startpunkt in Aufträge
      IF(OLD.Status!='IN_BEARBEITUNG' AND NEW.status='IN_BEARBEITUNG')
      THEN UPDATE Aufträge SET Status = 'IN_BEARBEITUNG'
            WHERE NEW.Aid = Aufträge.AID;
      END IF;
      --Es wird nur etwas getan, falls die Änderung auch von beliebigem Status zu
      --ARCHIVIERT war
      IF (OLD.Status='ARCHIVIERT' OR NEW.Status!='ARCHIVIERT') THEN
            RETURN NULL;
      END IF;
      modellid=(SELECT Modell_ID FROM Aufträge WHERE AID=OLD.AID);
      werk=(SELECT WID FROM Werksaufträge WHERE AID=OLD.AID);
      counter=(SELECT Anzahl FROM Aufträge WHERE AID=OLD.AID);
      UPDATE Werksaufträge SET Herstellungsende=CURRENT_DATE
            WHERE WID=OLD.WID AND AID=OLD.AID;
      --Gibt es einen freien LKW und Fahrer?
      IF (lkw IS NULL OR fahrer IS NULL)
      THEN --NEIN, dann Lagere Autos vorübergehend
            LOOP
            EXIT WHEN counter=0;
                  INSERT INTO Autos (Modell_ID, Status, produziertVon)
                        VALUES (modellid, 'WARTEND', werk);
                  counter=counter-1;
            END LOOP;
      ELSE --JA, dann liefere sofort.
            LOOP
            EXIT WHEN counter=0;
                  INSERT INTO Autos (Modell_ID, Status, produziertVon)
                        VALUES (modellid, 'LAGERND', werk);
                  kfzid=(SELECT max(KFZ_ID) FROM Autos);
```

```
                    INSERT INTO liefert (LKW_ID, KFZ_ID, Modell_ID, MID, AID,
                                                Lieferdatum)
                          VALUES (lkw, kfzid, modellid, fahrer, OLD.AID, NULL);
                    counter=counter-1;
             END LOOP;
             DELETE FROM Autoteile WHERE AID=OLD.AID;
      END IF;
      RETURN NEW;
      END;
      $$ LANGUAGE plpgsql;

CREATE TRIGGER onFinishedJob AFTER UPDATE ON Werksaufträge FOR EACH ROW EXECUTE
PROCEDURE finishedJob();



--Berechnet den Preis zu einer Bestellung eines Kunden.
CREATE FUNCTION calculatePrice() RETURNS TRIGGER AS
      $$ DECLARE
             var_rabatt integer;
             price numeric(10,2);
      BEGIN
      var_rabatt := 0;
      -- Kunde ist Kontaktperson / Großhändler
      IF (EXISTS (SELECT 1 FROM Kontaktpersonen WHERE PID = NEW.KundenID))
      THEN var_rabatt := (SELECT Rabatt
                          FROM ( SELECT *
                                 FROM Aufträge
                                 JOIN Kontaktpersonen
                                 ON NEW.KundenID = Kontaktpersonen.PID
                                 ) AS tmp1
                          JOIN Großhändler
                          ON tmp1.GID = Großhändler.GID LIMIT 1);
      END IF;
      price := (100-var_rabatt) * (SELECT Preis
                                        FROM Modelle
                                        WHERE NEW.Modell_ID = Modelle.Modell_ID)
             * NEW.Anzahl/100;
      UPDATE Aufträge SET Preis = price WHERE Aufträge.AID = NEW.AID;
      RETURN NEW;
      END; $$ LANGUAGE plpgsql;

CREATE TRIGGER calculatePrice AFTER INSERT ON Aufträge FOR EACH ROW EXECUTE
PROCEDURE calculatePrice();
```

4.2. Beispieldaten

Um die Konsistenz unseres Systems zu gewährleisten haben wir nicht für alle Tabellen Beispieldaten erzeugt. Die Vervollständigung der Daten geschieht nach dem Einfügen durch unsere funktionalen Trigger. Diese sind vor dem Einfügen der Daten zu definieren (siehe Abgabe funktionaleTrigger.sql).

```
INSERT INTO Personen (Vorname, Nachname, PLZ, Straße, Wohnort, Email, TelNr) VALUES
    ('Peter', 'Müller', '66740', 'Hauptstr.1', 'Saarlouis',
          'Peter.Müller@web.de', '0683100000'),
    ('Hans Adolf', 'Bibelroy', '11111', 'Einsstr. 11', 'Einsstadt',
          'Hans_Adolf@somepage.com', '11111111111'),
    ('Sandra', 'Schmidt', '12487', 'Abram-Joffe-Straße 3', 'Berlin',
          'S.Schmidt@joadkhgfa89.to', '017200011100'),
    ('Hermann', 'Buchmann', '01222', 'Straße 42', 'Stadt', 'buchmann@verwaltung.de',
          '028409374'),
    ('Gus', 'Thomer', '73733', 'Einkaufsstr. 55', 'Schopstadt', 'Gus@thomer.de',
          '065115635'),
    ('Petra', 'Müller', '66740', 'Hauptstr. 1', 'Saarlouis', 'Petra.Müller@web.de',
          '0683100000'),
    ('Hildegard', 'Bibelroy', '11111', 'Einsstr.11', 'Einsstadt',
          'HildegardB@somepage.com', '11111111111'),
    ('Yildiz', 'Tilbe', '12487', 'Hauptstr. 200', 'Berlin', 'YildizTilbe@ghioapi.de',
          '017200011100'),
    ('Hermann', 'Buchmannklon', '01222', 'Straße 43', 'Stadt',
          'buchmannklon@verwaltung.de', '028409275'),
    ('Alexandra', 'Dünsch', '73732', 'Einkaufsstr. 155', 'Klossstadt',
          'Alexandra.dünsch@gmx.net', '049846433');

INSERT INTO Werke (Name) VALUES
    ('Gutes Werken am See');

INSERT INTO Mitarbeiter VALUES
    (1, '2011-11-11', 2000.00),
    (2, '2012-12-12',1800.00),
    (3, '1999-01-01', 1200.00),
    (4, '1999-02-02', 3400.40),
    (5, '2012-12-13',1600.00),
    (6, '2003-04-04', 800.21);

INSERT INTO Werksarbeiter VALUES
    (1, 1);

INSERT INTO LKW_Fahrer VALUES
    (2, '2009-01-01');

INSERT INTO Verwaltungsangestellte VALUES
    (3);

INSERT INTO Lagerarbeiter VALUES
    (4),(6);

INSERT INTO Teilelagerarbeiter VALUES
    (4, 1);

INSERT INTO Autolagerarbeiter VALUES
    (6);

INSERT INTO Großhändler (Firmenname, Straße, PLZ, Ort, Rabatt) VALUES
    ('Lensen und Partner GmbH', 'Hauptstraße 1', '66551', 'Saarhausen', 0),
    ('Kadetten und Twingos GmbH', 'Rue de Kartoffel 17', '44251', 'Köpfern', 0);

INSERT INTO Modelle (Preis, Bezeichnung) VALUES
    (12000.99, 'Happy Hippo Car - Der Verkaufsschlager'),
    (13999, 'Twilight 500'),
    (9999.99, 'Karre');

INSERT INTO Kunden VALUES
    (7,100),(8, 300), (9, 104), (10, 304);
```

```
INSERT INTO Privatkunden VALUES
    (7),(8);

INSERT INTO Kontaktpersonen VALUES
    (9, 1),
    (10, 2);

INSERT INTO Autoteiltypen (maxPreis, Bezeichnung) VALUES
    (100, 'Flügeltür Rot'),
    (100, 'Tür Blau'),
    (50, 'Fenster Tönung Blau'),
    (500, 'Karosserie Blau'),
    (100, 'Reifen klein'),
    (2000, 'Motor schwach');

INSERT INTO Modellteile VALUES
    (1, 1, 1), (1, 2, 1), (1, 6, 1),
    (2, 3, 1), (2, 4, 1), (2, 6, 1),
    (3, 5, 1), (3, 6, 1), (3, 1, 1);

INSERT INTO LKWs (Kaufdatum) VALUES
    ('2010-01-10'),
    ('2011-02-01');

INSERT INTO Hersteller (Firmenname) VALUES
    ('Teilezurichter - Profis und Azubis'),
    ('Katalysatoren 4 Life'),
    ('Teile gegen Bares'),
    ('Items, Gadgets and more');

INSERT INTO produzieren VALUES
    (1, 1, 50.50, 2),
    (2, 2, 50.50, 3),
    (3, 3, 60.0, 4),
    (4, 4, 60.0, 1),
    (5, 1, 50.50, 2),
    (6, 2, 50.50, 1),
    (1, 3, 60.0, 4),
    (2, 4, 60.0, 2),
    (3, 1, 50.50, 4),
    (4, 2, 50.50, 3);

INSERT INTO Autoteile (TeiletypID, lagert_in, Lieferdatum) VALUES
-- (TeiletypID, lagert_in, lieferdate)
    (1, 1, '2013-01-12'),
    (2, 1, '2013-01-12'),
    (3, 1, '2013-01-12'),
    (4, 1, '2013-01-12'),
    (5, 1, '2013-01-12'),
    (6, 1, '2013-01-12'),
    (1, 1, '2013-01-12'),
    (2, 1, '2013-01-12'),
    (3, 1, '2013-01-12'),
    (4, 1, '2013-01-12');

INSERT INTO Motoren VALUES
    (6, 80, 7000, 5, 'Diesel');

INSERT INTO Karosserien VALUES
    (4, 'Blau', 'Blech', 80, 150, 200);

INSERT INTO Türen VALUES
    (1, 'Rot', 'FLÜGELTÜR');

INSERT INTO Fenster VALUES
    (3, 'Blau', 'Sicherheitsglas');

INSERT INTO Reifen VALUES
    (5, 'Blau', 17, 'Chrom');

INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID) VALUES
    (1, 2, 7, 3);
```

4.3. Konsistenztrigger

```
--1. Die Werksid bei Werksarbeiter und Teilelagerarbeiter
-- darf nicht NULL sein bei Update oder Insert
CREATE FUNCTION checkWerksid() RETURNS TRIGGER AS
    $$ BEGIN
           IF(NEW.wid IS NULL) THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Wid bei Insert/Update NULL';
           END IF;
           RETURN NEW;
    END; $$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validWerksarbeiter
AFTER INSERT OR UPDATE ON Werksarbeiter INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE checkWerksid();

CREATE CONSTRAINT TRIGGER validTeilelagerarbeiter
AFTER INSERT OR UPDATE ON Teilelagerarbeiter INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE checkWerksid();


--2. LKW-Fahrer müssen den Führerschein mindestens 3 Jahre besitzen.
CREATE FUNCTION checkLicenseDate() RETURNS TRIGGER AS
    $$ BEGIN
           IF(now() - NEW.führerscheindatum <= interval '3 years')
           THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Führerschein noch nicht lange genug';
           END IF;
           RETURN NEW;
    END; $$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validLkwFahrer
AFTER INSERT OR UPDATE ON LKW_Fahrer INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE checkLicenseDate();


--3. Beschäftigungsbeginn muss in der Vergangenheit liegen
CREATE FUNCTION checkBeginn() RETURNS TRIGGER AS
    $$ BEGIN
           IF(NEW.beschäftigungsbeginn > now())
           THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Beschäftigungsbeginn in der Zukunft';
           END IF;
           RETURN NEW;
    END; $$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validWerksarbeiter
AFTER INSERT OR UPDATE ON Mitarbeiter INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE checkBeginn();


--4. Wenn ein Auto auf ein LKW geladen wird, muss der Status auf LIEFERND gesetzt
werden.
CREATE FUNCTION newDelivery() RETURNS TRIGGER AS
    $$ BEGIN
           UPDATE Autos SET Status='LIEFERND'
                  WHERE kfz_id=NEW.kfz_id AND modell_id=NEW.modell_id;
           RETURN NEW;
    END;$$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validDelivery
AFTER INSERT OR UPDATE ON liefert INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE newDelivery();
```

```
--5. Beim Einfügen eines neuen Auftrags muss der Mitarbeiter ein
Verwaltungsangestellter sein.
CREATE FUNCTION newOrderCheckWorker() RETURNS TRIGGER AS
    $$ BEGIN
           IF(NOT EXISTS(SELECT 1 FROM Verwaltungsangestellte
                  WHERE PID=NEW.MitarbeiterID))
           THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Der Mitarbeiter mit ID % ist kein
                               Verwaltunsangestellter', NEW.MitarbeiterID;
           END IF;
           RETURN NEW;
           END; $$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validInsertInLiefert
AFTER INSERT ON Aufträge INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE newOrderCheckWorker();

--6. Beim Einfügen einer neuen Lieferung, muss der Mitarbeiter ein LKW-Fahrer sein.
CREATE FUNCTION newDeliveryCheckWorker() RETURNS TRIGGER AS
    $$ BEGIN
           IF(NOT EXISTS (SELECT 1 FROM LKW_Fahrer WHERE PID=NEW.MID))
           THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Der Mitarbeiter mit ID % ist kein
                                     LKW_Fahrer',NEW.MID;
           END IF;
           RETURN NEW;
    END; $$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validInsertInDelivery
AFTER INSERT ON liefert INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE newDeliveryCheckWorker();

--7. Beim Erstellen/Ändern eines neuen Modells muss es auch Hersteller für alle
benötigten Teile geben
CREATE FUNCTION changeOnModell() RETURNS TRIGGER AS
    $$ BEGIN
           IF(EXISTS(SELECT 1 FROM Modellteile WHERE TeiletypID IN (SELECT TeiletypID
                  FROM Autoteiltypen EXCEPT (SELECT TeiletypID FROM
                         Autoteiltypen JOIN produzieren USING (TeiletypID)))))
           THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Nicht für alle benötigten
                                     Teile ist ein Hersteller verfügbar';
           END IF;
           RETURN NEW;
    END; $$ LANGUAGE plpgsql;
CREATE CONSTRAINT TRIGGER validChangeInModellTeile
AFTER INSERT OR UPDATE ON Modellteile INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE changeOnModell();

--8. Wenn ein Hersteller ein Teil löschen will, darf er nicht der einzige sein, der
das Teil produziert.
CREATE FUNCTION changeOnOffer() RETURNS TRIGGER AS
    $$ BEGIN
           IF(NEW.Preis>(SELECT maxPreis FROM Autoteiletypen
                         WHERE TeiletypID=NEW.TeiletypID)) THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Ein Preis von % für das Teil mit der ID % ist zu
                                     teuer',NEW.Preis,NEW.TeiletypID;
           END IF;
           IF(NOT EXISTS(SELECT 1 FROM produzieren
                  WHERE TeiletypID=OLD.TeiletypID AND HID!=OLD.HID))
           THEN
                  --ROLLBACK TRANSACTION;
                  RAISE EXCEPTION 'Zur Zeit kann die Produktion des Teils mit ID %
                               nicht eingestellt werden.', OLD.TeiletypID;
           END IF; RETURN OLD;
    END; $$ LANGUAGE plpgsql;

CREATE CONSTRAINT TRIGGER validChangeInProduction AFTER DELETE ON produzieren
INITIALLY DEFERRED FOR EACH ROW EXECUTE PROCEDURE changeOnOffer();
```

5. Normalisierung

Tabelle *liefert*:
Da es sich um einen mehrstelligen Beziehungstypen mit vier teilnehmenden Entitätstypen handelt, kommen jeweils die Mengen der Primärschlüssel von drei dieser Entitätstypen als Kandidatenschlüssel in Frage. Die Abbildungen jeder dieser Mengen auf den jeweiligen verbleibenden Primärschlüssel stellen die einzigen nicht trivialen, funktionalen Abhängigkeiten dar. Für jede dieser FDs gilt sowohl, dass die Determinate ein Superschlüssel ist (da jede oben genannte Menge einen Kandidatenschlüssel darstellt) als auch, dass die Bildmenge sich aus genau einem Attribut eines anderen Kandidatenschlüssels zusammensetzt und somit nur aus Primattributen besteht. Damit ist die Definition der BCNF erfüllt.

Tabelle *Teilelagerarbeiter*:
Die einzige nicht triviale, funktionale Abhängigkeit besteht zwischen der PersonenID *PID* und der WerksID *WID*. Da die Menge {PID} einziger Kandidatenschlüssel der Tabelle ist, ist somit die BCNF erfüllt.

Tabelle *Werksaufträge*:
Die Menge der nicht trivialen, funktionalen Abhängigkeiten dieser Tabelle setzt sich aus der Abbildung des einzigen Kandidatenschlüssels *AID* auf alle anderen Attribute, sowie jede von dieser Abbildung umfasste andere Abbildung zusammen. Daher ist auch diese Tabelle in BCNF, weil die Determinante dieser FDs den einzigen Kandidatenschlüssel repräsentiert.

Tabelle *produzieren*:
Analog lässt sich auch die Erfüllung der BCNF Definition auf dieser Tabelle nachweisen, da es wieder genau einen Kandidatenschlüssel {*TeiletypID, HID*} gibt, der die anderen Attribute bestimmt.

Tabelle M*otoren*:
Erneut ist die Menge der FDs wie oben beschrieben konstruierbar. Somit ist wieder die Kandidatenschlüssel Eigenschaft der Determinanten entscheidend. Diese ist auch hier gegeben.

6. Logische Datenunabhängigkeit

```
-- FORMAT: Je eine Sicht gefolgt von den auf sie bezogenen Regeln

-- Bietet Kundenname, Modellname, Vor. Lieferdatum, Anzahl, Preis und Auftragsnummer
-- aller Aufträge (offen wie archiviert) einer Schnittstelle an
CREATE OR REPLACE VIEW Kundensicht AS
       WITH Kundeninfo AS
              (SELECT Vorname, Nachname, Aufträge.KundenID AS KundenID, Aufträge.Status
                     AS Auftragsstatus, Aufträge.AID AS Auftrag1
              FROM Aufträge
                     JOIN Personen ON Aufträge.KundenID = Personen.PID),
              Modellname AS
              (SELECT Bezeichnung, Aufträge.AID AS Auftrag2
              FROM Aufträge
                     JOIN Modelle ON Aufträge.Modell_ID = Modelle.Modell_ID)

       SELECT Vorname, Nachname, tmp.KundenID AS KundenID, Bezeichnung AS Modell,
              Vorraussichtliches_Lieferdatum, Datum AS Auftrag_erteilt_am, Anzahl, Preis,
              AID AS Auftragsnummer, Auftragsstatus
       FROM (SELECT * FROM Kundeninfo JOIN Modellname ON Auftrag1 = Auftrag2) AS tmp
                     JOIN Aufträge on AID = Auftrag1;


-- Diese Sicht erlaubt die Einsicht offener Aufträge
CREATE OR REPLACE VIEW offene_Aufträge AS
       WITH offeneAufträge AS
       (SELECT *
       FROM (SELECT Aufträge.AID, KundenID, Vorraussichtliches_Lieferdatum
              FROM Aufträge
                     JOIN Kunden ON Aufträge.KundenID = Kunden.PID
                     WHERE Aufträge.status = 'WARTEND')
              AS auftragT
              JOIN Personen ON auftragT.KundenID = Personen.PID)

       SELECT AID AS "Auftragsnr.", vorraussichtliches_lieferdatum AS "Vorrauss.
                     Lieferung" , Vorname AS "Kundenvorname", Nachname AS "Kundenname",
                     TelNr AS "Tel."
       FROM offeneAufträge;


-- Erlaubt einem Hersteller sein Teileangebot einzusehen und ggf. anzupassen
CREATE OR REPLACE VIEW Herstellerangebot AS
       WITH Herstellerteile AS
              (SELECT Hersteller.HID AS HID, Firmenname, TeiletypID, Bezeichnung, Preis,
                     Zeit FROM Hersteller
                     JOIN produzieren ON Hersteller.HID = produzieren.HID
                     JOIN Autoteiltypen USING (TeiletypID))

       SELECT Firmenname, HID AS HerstellerID, TeiletypID, Bezeichnung, Preis, Zeit
       FROM Herstellerteile;


CREATE OR REPLACE RULE manufacturerCommandI AS ON INSERT TO Herstellerangebot
DO INSTEAD NOTHING;

CREATE OR REPLACE RULE manufacturerCommandU AS ON UPDATE TO Herstellerangebot
DO INSTEAD NOTHING;


CREATE OR REPLACE RULE manufacturerNewPartInInventory
AS ON INSERT TO Herstellerangebot
WHERE EXISTS (SELECT * FROM Autoteiltypen
              WHERE Autoteiltypen.TeiletypID = NEW.TeiletypID)
DO INSERT INTO produzieren VALUES (NEW.TeiletypID, NEW.HerstellerID, NEW.Preis,
                                   NEW.Zeit);

CREATE OR REPLACE RULE manufacturerPartUpdatedInInventory
AS ON UPDATE TO Herstellerangebot
```

```
-- TeiletypIDs dürfen von Herstellern nicht geändert werden.
-- Möchte ein Hersteller ein Teil komplett durch ein neues ersetzen muss er stattdessen
löschen und dann einfügen.
WHERE NEW.TeiletypID = OLD.TeiletypID
DO ALSO UPDATE produzieren SET HID = NEW.HerstellerID, Preis = NEW.Preis, Zeit = NEW.Zeit
       WHERE TeiletypID = OLD.TeiletypID AND HID = OLD.HerstellerID;

CREATE OR REPLACE RULE manufacturerDeletePartInInventory
AS ON DELETE TO Herstellerangebot
DO INSTEAD DELETE FROM produzieren
                     WHERE TeiletypID = OLD.TeiletypID
                     AND HID = OLD.HerstellerID
                     AND Preis = OLD.Preis
                     AND Zeit = OLD.Zeit;


-- Zeigt die aktuellen Bestellungen einem Hersteller an (nicht modifizierbar)
CREATE OR REPLACE VIEW Herstellerbestellungen AS
       SELECT Firmenname, Hersteller.HID AS HerstellerID, BID AS Bestellungsnummer,
              Status AS Bestellungsstatus,bestellt.TeiletypID AS TeiletypID,
              Bezeichnung AS Teilbezeichnung
       FROM Hersteller
              JOIN bestellt ON Hersteller.HID = bestellt.HID
              JOIN Autoteiltypen ON bestellt.TeiletypID = Autoteiltypen.TeiletypID;


-- Ermöglicht einem Teilelagerarbeiter das Einscannen eingegangener Teilelieferungen
CREATE OR REPLACE VIEW Teilelagerarbeitersicht AS
       SELECT BID AS BestellungsID FROM bestellt;

CREATE OR REPLACE RULE scanIn
AS ON UPDATE TO Teilelagerarbeitersicht
DO ALSO UPDATE bestellt SET Status = 'ARCHIVIERT'
       WHERE BID = OLD.BestellungsID AND Status != 'ARCHIVIERT';


-- Mittels dieser Sicht kann ein Autolagerarbeiter den aktuellen Autobestand einsehen
CREATE OR REPLACE VIEW Autolagerarbeitersicht AS
       SELECT KFZ_ID AS Fahrgestellnummer, Bezeichnung AS Modell, Status, Name as Werk
       FROM Autos JOIN Modelle ON Autos.Modell_ID = Modelle.Modell_ID
              JOIN Werke ON Autos.produziertVon = Werke.WID
       WHERE Status = 'LAGERND';


-- Die Schnittstelle über die ein Werksarbeiter Zugriff auf die Werksaufträge hat
-- Diese führen einen Scan durch sobald alle Autos
-- eines Werksauftrags produziert wurden um das Produktionsende zu vermerken
-- Es ist ihnen nicht gestattet Werk- oder Auftragsnummer zu ändern.
CREATE OR REPLACE VIEW Werksarbeitersicht AS
       SELECT WID AS Werknummer, AID AS Auftragsnummer, Status
       FROM Werksaufträge;

CREATE OR REPLACE RULE factoryU AS ON UPDATE TO Werksarbeitersicht
DO INSTEAD NOTHING;

CREATE OR REPLACE RULE factoryScan AS ON UPDATE TO Werksarbeitersicht
WHERE OLD.Werknummer = NEW.Werknummer AND OLD.Auftragsnummer = NEW.Auftragsnummer
DO ALSO UPDATE Werksaufträge SET Status = 'ARCHIVIERT'
WHERE WID = OLD.Werknummer AND AID = OLD.Auftragsnummer AND Status != 'ARCHIVIERT';

-- Die Informationen, die ein LKW Fahrer zur Auslieferung benötigt
CREATE OR REPLACE VIEW LKW_FahrerSicht AS
       SELECT Vorname AS Fahrervorname, Nachname AS Fahrernachname, MID AS FahrerID,
liefert.LKW_ID AS LKW_Nummer, Autos.KFZ_ID AS Fahrgestellnummer_Ware,
       liefert.AID AS Auftragsnummer, Aufträge.Status AS Auftragsstatus
       FROM liefert JOIN Personen ON liefert.MID = Personen.PID JOIN Autos ON
liefert.KFZ_ID = Autos.KFZ_ID
              JOIN Aufträge ON liefert.AID = Aufträge.AID;

-- LKW Fahrer dürfen nur den Status auf 'ARCHIVIERT' setzten
CREATE OR REPLACE RULE driverCommandU AS ON UPDATE TO LKW_FahrerSicht
DO INSTEAD NOTHING;
```

```
-- Archiviert Auftrag in liefert
-- (durch den ON DELETE DO INSTEAD UPDATE Trigger auf liefert)
-- sobald die bestellten Autos beim Kunden ausgeliefert werden
CREATE OR REPLACE RULE scanDeliveryDate AS ON UPDATE TO LKW_FahrerSicht
WHERE OLD.Fahrervorname = NEW.Fahrervorname
AND OLD.Fahrernachname = NEW.Fahrernachname
AND OLD.LKW_Nummer = NEW.LKW_Nummer
AND OLD.Fahrgestellnummer_Ware = NEW.Fahrgestellnummer_Ware
AND OLD.Auftragsnummer = NEW.Auftragsnummer
AND OLD.FahrerID = NEW.FahrerID
DO ALSO DELETE FROM liefert
WHERE liefert.Lieferdatum IS NULL
AND liefert.KFZ_ID = OLD.Fahrgestellnummer_Ware
AND liefert.AID = OLD.Auftragsnummer
AND liefert.MID = OLD.FahrerID;


-- Erlaubt das Hinzufügen von Großhändlern
CREATE OR REPLACE VIEW Großhändlersicht AS
       SELECT GID AS GroßhändlerID, Firmenname, Straße, PLZ, Ort, Rabatt
       FROM Großhändler;

CREATE OR REPLACE RULE insertGH AS ON INSERT TO Großhändlersicht
DO INSTEAD INSERT INTO Großhändler(Firmenname, Straße, PLZ, Ort, Rabatt) VALUES
(NEW.Firmenname, NEW.Straße, NEW.PLZ, NEW.Ort, NEW.Rabatt);

CREATE OR REPLACE RULE updateGH AS ON UPDATE TO Großhändlersicht
DO INSTEAD UPDATE Großhändler SET Firmenname = NEW.Firmenname, Straße = NEW.Straße, PLZ =
NEW.PLZ, Ort = NEW.Ort, Rabatt = NEW.Rabatt
WHERE GID = OLD.GroßhändlerID;

-- Nur Großhändler, die noch keine Kontaktperson haben können gelöscht werden.
CREATE OR REPLACE RULE deleteGH AS ON DELETE TO Großhändlersicht
DO INSTEAD DELETE FROM Großhändler
WHERE GID = OLD.GroßhändlerID;



-- Sicht für Verwaltungsangestellte, die Aufträge entgegen nehmen
CREATE OR REPLACE VIEW VerwaltungAuftragssicht AS
       WITH Kundeninfo AS
              (SELECT Vorname, Nachname, Aufträge.AID AS Auftrag1
              FROM Aufträge
                     JOIN Personen ON Aufträge.KundenID = Personen.PID),
       Modellname AS
              (SELECT Bezeichnung, Aufträge.AID AS Auftrag2
              FROM Aufträge
                     JOIN Modelle ON Aufträge.Modell_ID = Modelle.Modell_ID)

       SELECT Vorname, Nachname, KundenID, Bezeichnung AS Modell, Aufträge.Modell_ID AS
              Modell_ID, Vorraussichtliches_Lieferdatum, Datum AS Auftrag_erteilt_am,
              Anzahl, Preis, AID AS Auftragsnummer, MitarbeiterID
       FROM (SELECT * FROM Kundeninfo JOIN Modellname ON Auftrag1 = Auftrag2) AS tmp
              JOIN Aufträge on AID = Auftrag1;

CREATE OR REPLACE RULE insertNewJob AS ON INSERT TO VerwaltungAuftragssicht
DO INSTEAD INSERT INTO Aufträge(Modell_ID, Anzahl, KundenID, MitarbeiterID)
VALUES (NEW.Modell_ID, NEW.Anzahl, NEW.KundenID, NEW.MitarbeiterID);

CREATE OR REPLACE RULE updateJob AS ON UPDATE TO VerwaltungAuftragssicht
DO INSTEAD UPDATE Aufträge SET Modell_ID = NEW.Modell_ID, Anzahl = NEW.Anzahl, KundenID =
       NEW.KundenID, MitarbeiterID = NEW.MitarbeiterID WHERE AID = OLD.Auftragsnummer;

CREATE OR REPLACE RULE deleteJob AS ON DELETE TO VerwaltungAuftragssicht
DO INSTEAD DELETE FROM Aufträge WHERE AID = OLD.Auftragsnummer;
```

```
-- Sicht für Verwaltungsangestellte, die Kunden ins System aufnehmen
-- sowie ihre Daten anpassen
-- Kunden können aus Nachverfolgungsgründen nicht gelöscht werden,
-- außer sie haben noch keinen Auftrag aufgegeben
-- GroßhändlerID ist 0 bei Privatkunden
CREATE OR REPLACE VIEW VerwaltungKundenaufnahme AS
       SELECT PID, Vorname, Nachname, (varchar '') AS Kundenart, Distanz, PLZ, Straße,
              Wohnort, Email, TelNR, (integer '0') AS GroßhändlerID
       FROM Kunden JOIN Personen USING (PID);

CREATE OR REPLACE RULE "_RETURN"
AS ON SELECT TO VerwaltungKundenaufnahme
DO INSTEAD SELECT PID, Vorname, Nachname,
       (CASE
              WHEN PID  IN (SELECT PID FROM Privatkunden) THEN
                     varchar 'PRIVATKUNDE'
              WHEN PID  IN (SELECT PID FROM Kontaktpersonen) THEN
                     varchar 'KONTAKTPERSON'
              ELSE varchar 'Kein Kunde oder noch nicht in Kunden Tabelle.'
       END) AS Kundenart, Distanz, PLZ, Straße, Wohnort, Email, TelNR, GID AS
                     GroßhändlerID
FROM Kunden JOIN Personen USING (PID) FULL OUTER JOIN Kontaktpersonen USING (PID);

CREATE OR REPLACE FUNCTION insertNewCustomer() RETURNS TRIGGER AS
       $$
       DECLARE
       thisID integer;
       BEGIN
       INSERT INTO Kunden (Distanz) VALUES (NEW.Distanz);
       thisID=lastval();
       CASE NEW.Kundenart
              WHEN 'PRIVATKUNDE' THEN
                     INSERT INTO Privatkunden VALUES (thisID);
              WHEN 'KONTAKTPERSON' THEN
                     INSERT INTO Kontaktpersonen VALUES (thisID, NEW.GroßhändlerID);
              ELSE RAISE EXCEPTION 'Ungültige Kundenart: %', NEW.Kundenart;
       END CASE;
       RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER onInsertNewCustomer INSTEAD OF INSERT ON VerwaltungKundenaufnahme FOR EACH
ROW EXECUTE PROCEDURE insertNewCustomer();

CREATE OR REPLACE FUNCTION updateCustomer() RETURNS TRIGGER AS
       $$ BEGIN
       IF OLD.PID != NEW.PID THEN RAISE EXCEPTION 'ID of a customer cannot be changed.';
       END IF;
       UPDATE Kunden SET Distanz = NEW.Distanz WHERE PID = NEW.PID;
       IF OLD.Kundenart = NEW.Kundenart THEN
              CASE NEW.Kundenart
                     WHEN 'PRIVATKUNDE' THEN -- nichts zu updaten
                     WHEN 'KONTAKTPERSON' THEN
                            UPDATE Kontaktpersonen SET GID = NEW.GID;
                     ELSE RAISE EXCEPTION 'Ungültige Kundenart: %', NEW.Kundenart;
              END CASE;
       ELSE
              CASE OLD.Kundenart
                     WHEN 'PRIVATKUNDE' THEN
                            DELETE FROM Privatkunden WHERE PID = OLD.PID;
                     WHEN 'KONTAKTPERSON' THEN
                            DELETE FROM Kontaktpersonen WHERE PID = OLD.PID;
                     ELSE RAISE EXCEPTION 'Some inconsistent state was reached.';
              END CASE;
              CASE NEW.Kundenart
                     WHEN 'PRIVATKUNDE' THEN
                            INSERT INTO Privatkunden VALUES (NEW.PID);
                     WHEN 'KONTAKTPERSON' THEN
                            INSERT INTO Kontaktpersonen VALUES (NEW.PID,
NEW.GroßhändlerID);
                     ELSE RAISE EXCEPTION 'Ungültige Kundenart: %',NEW.Kundenart;
              END CASE;
       END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER onUpdateCustomer INSTEAD OF UPDATE ON VerwaltungKundenaufnahme FOR EACH
ROW EXECUTE PROCEDURE updateCustomer();

CREATE OR REPLACE FUNCTION deleteCustomer() RETURNS TRIGGER AS
	$$
	BEGIN
	CASE OLD.Kundenart
		WHEN 'PRIVATKUNDE' THEN
			DELETE FROM Privatkunden WHERE PID = OLD.PID;
		WHEN 'KONTAKTPERSON' THEN
			DELETE FROM Kontaktpersonen WHERE PID = OLD.PID;
		ELSE RAISE EXCEPTION 'Some inconsistent state was reached.';
	END CASE;
	DELETE FROM Kunden WHERE PID = OLD.PID;
	RETURN OLD;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER onDeleteCustomer INSTEAD OF DELETE ON VerwaltungKundenaufnahme FOR EACH
ROW EXECUTE PROCEDURE deleteCustomer();


-- Sicht auf alle sich in Produktion befindender Aufträge bzw. solcher die in den Werken
momentan assembliert werden
CREATE OR REPLACE VIEW Produktion AS
	SELECT WID, AID, Status FROM Werksaufträge GROUP BY AID HAVING Status =
'IN_BEARBEITUNG';


-- Sicht auf alle Fahrzeuge in der LKW Fahrzeugflotte
-- Updates ergeben keinen Sinn, da das Kaufdatum nie angepasst wird
CREATE OR REPLACE VIEW Fuhrpark AS
	SELECT * FROM LKWs;

CREATE OR REPLACE RULE newVehicle AS ON INSERT TO Fuhrpark
DO INSTEAD INSERT INTO LKWs(Kaufdatum) VALUES (NEW.Kaufdatum);

CREATE OR REPLACE RULE deleteVehicle AS ON DELETE TO Fuhrpark
DO INSTEAD DELETE FROM LKWs WHERE LKW_ID = OLD.LKW_ID;



-- Architektur erlaubt das Bauen neuer Werke sowie Einsicht aktueller Firmengebäude
CREATE OR REPLACE VIEW Architektur AS
	SELECT * FROM Werke;

CREATE OR REPLACE RULE newFactory AS ON INSERT TO Architektur
DO INSTEAD INSERT INTO Werke(Name) VALUES (NEW.Name);


-- Teilelager liefert eine Übersicht über alle aktuell in den Lagern vorhandenen Teilen
CREATE OR REPLACE VIEW Teilelager AS
	SELECT TeileID, TeiletypID, Bezeichnung, lagert_in, Lieferdatum, AID as AuftragsID
	FROM Autoteile JOIN Autoteiltypen USING (TeiletypID);


-- Modellsicht ermöglicht einfügen neuer Modelle
-- Preis und Bezeichnung können angepasst werden
-- Modelle können nicht ohne weiteres gelöscht werden
-- Teile eines Modells müssen über die Modellteilesicht eingefügt werden
CREATE OR REPLACE VIEW Modellsicht AS
	SELECT * FROM Modelle;

CREATE OR REPLACE RULE modelInsert AS ON INSERT TO Modellsicht
DO INSTEAD INSERT INTO Modelle(Preis, Bezeichnung) VALUES (NEW.Preis, NEW.Bezeichnung);

CREATE OR REPLACE RULE modelUpdate AS ON UPDATE TO Modellsicht
DO INSTEAD UPDATE Modelle SET Preis = NEW.Preis, Bezeichnung = NEW.Bezeichnung
	WHERE Modell_ID = OLD.Modell_ID;
```

```
-- Über die Modellteilesicht können neue Teile eines Modells eingefügt oder entfernt,
-- sowie die Anzahl bestehender angepasst werden.
CREATE OR REPLACE VIEW Modellteilesicht AS
       SELECT * FROM Modellteile;

CREATE OR REPLACE RULE modelPartAddition AS ON INSERT TO Modellteilesicht
DO INSTEAD INSERT INTO Modellteile VALUES (NEW.Modell_ID, NEW.TeiletypID, NEW.Anzahl);

CREATE OR REPLACE RULE modelPartAdjust AS ON UPDATE TO Modellteilesicht
DO INSTEAD UPDATE Modellteile SET Anzahl = NEW.Anzahl
WHERE Modell_ID = OLD.Modell_ID AND TeiletypID = OLD.TeiletypID;

CREATE OR REPLACE RULE modelPartDelete AS ON DELETE TO Modellteilesicht
DO INSTEAD DELETE FROM Modellteile WHERE Modell_ID = OLD.Modell_ID AND TeiletypID =
OLD.TeiletypID;


-- Personal liefert einen Überblick über alle momentan im Unternehmen beschäftigten
Mitarbeiter,
-- sowie ihre Spezialisierung und deren Attribute, und dient der Umsetzung des
Personalmanagements
-- Über die Select Regel wird die Spezialisierung automatisch ermittelt und ausgegeben
-- Das Personalmangement darf Mitarbeiter einstellen und entlassen, sowie ihre Daten
anpassen (ausser ihre ID)
CREATE OR REPLACE VIEW Personal AS
       SELECT  Vorname, Nachname, PLZ, Straße, Wohnort, Email, TelNr,
Beschäftigungsbeginn, Gehalt, (date '01-01-2014') AS Führerscheindatum,
       (integer '13') AS arbeitet_in, (varchar '') AS Spezialisierung FROM (Personen JOIN
Mitarbeiter ON Personen.PID = Mitarbeiter.PID)
       WHERE Beschäftigungsende IS NULL;


CREATE OR REPLACE RULE "_RETURN"
AS ON SELECT TO Personal
DO INSTEAD SELECT  Vorname, Nachname, PLZ, Straße, Wohnort, Email, TelNr,
Beschäftigungsbeginn, Gehalt, Führerscheindatum, WID AS arbeitet_in,
       (CASE
              WHEN Mitarbeiter.PID  IN (SELECT PID FROM Teilelagerarbeiter) THEN
                     varchar 'TEILELAGERARBEITER'
              WHEN Mitarbeiter.PID  IN (SELECT PID FROM Autolagerarbeiter) THEN
                     varchar 'AUTOLAGERARBEITER'
              WHEN Mitarbeiter.PID IN (SELECT PID FROM LKW_Fahrer) THEN
                     varchar 'LKW FAHRER'
              WHEN Mitarbeiter.PID IN (SELECT PID FROM Werksarbeiter) THEN
                     varchar 'WERKSARBEITER'
              WHEN Mitarbeiter.PID IN (SELECT PID FROM Verwaltungsangestellte) THEN
                     varchar 'VERWALTUNGSANGESTELLTE'
              WHEN Mitarbeiter.PID IN (SELECT PID FROM Lagerarbeiter
       EXCEPT
              (SELECT PID FROM (Teilelagerarbeiter FULL OUTER JOIN
                     Autolagerarbeiter USING (PID) ))) THEN
              varchar 'LAGERARBEITER'
       ELSE
              varchar 'MITARBEITER'
       END)
 AS Spezialisierung
FROM (Personen RIGHT OUTER JOIN Mitarbeiter USING (PID)
              FULL OUTER JOIN LKW_Fahrer USING (PID)
              FULL OUTER JOIN Teilelagerarbeiter USING (PID));
```

```
CREATE OR REPLACE FUNCTION insertInPersonal() RETURNS TRIGGER AS
	$$
	DECLARE
	thisID integer;
	BEGIN
	INSERT INTO Personen (Vorname, Nachname, PLZ, Straße, Wohnort, Email, TelNr)
VALUES (NEW.Vorname, NEW.Nachname, NEW.PLZ, NEW.Straße, NEW.Wohnort, NEW.Email,
NEW.TelNr);
	thisID=lastval();
	INSERT INTO Mitarbeiter VALUES (thisID, CURRENT_DATE, NEW.Gehalt, NULL);
	CASE NEW.Spezialisierung
		WHEN 'TEILELAGERARBEITER' THEN
			INSERT INTO Lagerarbeiter VALUES (thisID);
			INSERT INTO Teilelagerarbeiter VALUES (thisID, NEW.arbeitet_in);
		WHEN 'AUTOLAGERARBEITER' THEN
			INSERT INTO Lagerarbeiter VALUES (thisID);
			INSERT INTO Autolagerarbeiter VALUES (thisID);
		WHEN 'LKW FAHRER' THEN
			INSERT INTO LKW_Fahrer VALUES (thisID, NEW.Führerscheindatum);
		WHEN 'WERKSARBEITER' THEN
			INSERT INTO Werksarbeiter VALUES (thisID, NEW.arbeitet_in);
		WHEN 'VERWALTUNGSANGESTELLTE' THEN
			INSERT INTO Verwaltungsangestellte VALUES (thisID);
		WHEN 'LAGERARBEITER' THEN
			INSERT INTO Lagerarbeiter VALUES (thisID);
		WHEN 'MITARBEITER' THEN
			RETURN NEW;
		ELSE RAISE EXCEPTION 'Ungültige Spezialisierung: %',NEW.Spezialisierung;
	END CASE;
	RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER onInsertInPersonal INSTEAD OF INSERT ON Personal FOR EACH ROW EXECUTE
PROCEDURE insertInPersonal();

CREATE OR REPLACE FUNCTION updateOnPersonal() RETURNS TRIGGER AS
	$$
	BEGIN
	IF OLD.PID != NEW.PID THEN RAISE EXCEPTION 'ID of a person cannot be changed.';
	END IF;
	IF NEW.Beschäftigungsbeginn IS NOT NULL
	THEN RAISE EXCEPTION 'Data of a former employee cannot be changed. Neither can
		employees be fired via an update operation.';
	END IF;
	UPDATE Personen SET Vorname = NEW.Vorname, Nachname = NEW.Nachname, PLZ = NEW.PLZ,
		Straße = NEW.Straße, Wohnort = NEW.Wohnort, Email = NEW.Email, TelNr =
		NEW.TelNr
		WHERE PID = NEW.PID;

	UPDATE Mitarbeiter SET Beschäftigungsbeginn = NEW.Beschäftigungsbeginn, Gehalt =
		NEW.Gehalt
	WHERE PID = NEW.PID;
	IF OLD.Spezialisierung = NEW.Spezialisierung THEN
		CASE NEW.Spezialisierung
			WHEN 'TEILELAGERARBEITER' THEN
				UPDATE Teilelagerarbeiter SET WID = NEW.arbeitet_in
				WHERE PID = NEW.PID;
			WHEN 'AUTOLAGERARBEITER' THEN -- nichts zu updaten
			WHEN 'LKW FAHRER' THEN
				UPDATE LKW_Fahrer SET Führerscheindatum=NEW.Führerscheindatum
				WHERE PID = NEW.PID;
			WHEN 'WERKSARBEITER' THEN
				UPDATE Werksarbeiter SET WID = NEW.arbeitet_in
				WHERE PID = NEW.PID;
			WHEN 'VERWALTUNGSANGESTELLTE' THEN -- nichts zu updaten
			WHEN 'LAGERARBEITER' THEN -- nichts zu updaten
			WHEN 'MITARBEITER' THEN -- nichts zu updaten
			ELSE RAISE EXCEPTION 'Ungültige Spezialisierung:
%',NEW.Spezialisierung;
		END CASE;
	ELSE
```

```
            CASE OLD.Spezialisierung
                  WHEN 'TEILELAGERARBEITER' THEN
                        DELETE FROM Teilelagerarbeiter WHERE PID = OLD.PID;
                        DELETE FROM Lagerarbeiter WHERE PID = OLD.PID;
                  WHEN 'AUTOLAGERARBEITER' THEN
                        DELETE FROM Autolagerarbeiter WHERE PID = OLD.PID;
                        DELETE FROM Lagerarbeiter WHERE PID = OLD.PID;
                  WHEN 'LKW FAHRER' THEN
                        DELETE FROM LKW_Fahrer WHERE PID = OLD.PID;
                  WHEN 'WERKSARBEITER' THEN
                        DELETE FROM Werksarbeiter WHERE PID = OLD.PID;
                  WHEN 'VERWALTUNGSANGESTELLTE' THEN
                        DELETE FROM Verwaltungsangestellte WHERE PID = OLD.PID;
                  WHEN 'LAGERARBEITER' THEN
                        DELETE FROM Lagerarbeiter WHERE PID = OLD.PID;
                  WHEN 'MITARBEITER' THEN -- nichts zu löschen
                  ELSE RAISE EXCEPTION 'Some inconsistent state was reached.';
            END CASE;
            CASE NEW.Spezialisierung
                  WHEN 'TEILELAGERARBEITER' THEN
                        INSERT INTO Lagerarbeiter VALUES (NEW.PID);
                        INSERT INTO Teilelagerarbeiter VALUES (NEW.PID,
                              NEW.arbeitet_in);
                  WHEN 'AUTOLAGERARBEITER' THEN
                        INSERT INTO Lagerarbeiter VALUES (NEW.PID);
                        INSERT INTO Autolagerarbeiter VALUES (NEW.PID);
                  WHEN 'LKW FAHRER' THEN
                        INSERT INTO LKW_Fahrer VALUES (NEW.PID,
                              NEW.Führerscheindatum);
                  WHEN 'WERKSARBEITER' THEN
                        INSERT INTO Werksarbeiter VALUES (NEW.PID, NEW.arbeitet_in);
                  WHEN 'VERWALTUNGSANGESTELLTE' THEN
                        INSERT INTO Verwaltungsangestellte VALUES (NEW.PID);
                  WHEN 'LAGERARBEITER' THEN
                        INSERT INTO Lagerarbeiter VALUES (NEW.PID);
                  WHEN 'MITARBEITER' THEN
                  RETURN NEW;
                  ELSE RAISE EXCEPTION 'Ungültige Spezialisierung:
                        %',NEW.Spezialisierung;
            END CASE;
      END IF;
      RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER onUpdateOnPersonal INSTEAD OF UPDATE ON Personal FOR EACH ROW EXECUTE
PROCEDURE updateOnPersonal();

-- Löschen bzw. entlassen eines Mitarbeiters kommt einer Archivierung gleich, die durch
die Setzung des Beschäftigungsendes realisiert wird.
CREATE OR REPLACE FUNCTION deleteFromPersonal() RETURNS TRIGGER AS
      $$
      BEGIN
      CASE OLD.Spezialisierung
            WHEN 'TEILELAGERARBEITER' THEN
                  DELETE FROM Teilelagerarbeiter WHERE PID = NEW.PID;
            WHEN 'LKW FAHRER' THEN
                  DELETE FROM LKW_Fahrer WHERE PID = NEW.PID;
            WHEN 'WERKSARBEITER' THEN
                  DELETE FROM Werksarbeiter WHERE PID = NEW.PID;
            WHEN 'VERWALTUNGSANGESTELLTE' THEN
                  DELETE FROM Verwaltungsangestellte WHERE PID = NEW.PID;
            WHEN 'LAGERARBEITER' THEN
                  DELETE FROM Lagerarbeiter WHERE PID = NEW.PID;
            WHEN 'MITARBEITER' THEN -- nichts zu löschen
            ELSE RAISE EXCEPTION 'Some inconsistent state was reached.';
      END CASE;
      UPDATE Mitarbeiter SET Beschäftigungsende = now() WHERE PID = OLD.PID;
      RETURN OLD;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER onDeleteFromPersonal INSTEAD OF DELETE ON Personal FOR EACH ROW EXECUTE
PROCEDURE deleteFromPersonal();
```

```
-- Ingenieure können über diese Sichten neue Teiletypen einführen oder anpassen, aber
nicht löschen,
-- da dadurch Informationen vergangener Aufträge verloren gehen könnten.
-- Die TeiletypID, die beim Einfügen angegeben wird, wird ignoriert, da diese eine Serial
ist.
-- Die Entscheidung fünf Sichten statt nur einer anzulegen beruht auf der sonst hohen
Anzahl von Null Values,
-- da jedes Teil ja nur eine Spezialisierung hat.
CREATE OR REPLACE VIEW IngenieursichtMotoren AS
       SELECT Bezeichnung, TeiletypID, maxPreis, PS, Drehzahl, Verbrauch, Spritart
       FROM Autoteiltypen JOIN Motoren USING (TeiletypID);

CREATE OR REPLACE VIEW IngenieursichtKarosserien AS
       SELECT Bezeichnung, TeiletypID, maxPreis, Farbe, Material, Höhe, Breite, Länge
       FROM Autoteiltypen JOIN Karosserien USING (TeiletypID);

CREATE OR REPLACE VIEW IngenieursichtTüren AS
       SELECT Bezeichnung, TeiletypID, maxPreis, Farbe, Türart
       FROM Autoteiltypen JOIN Türen USING (TeiletypID);

CREATE OR REPLACE VIEW IngenieursichtFenster AS
       SELECT Bezeichnung, TeiletypID, maxPreis, Tönung, Glasart
       FROM Autoteiltypen JOIN Fenster USING (TeiletypID);

CREATE OR REPLACE VIEW IngenieursichtReifen AS
       SELECT Bezeichnung, TeiletypID, maxPreis, Farbe, Zoll, Felgenmaterial
       FROM Autoteiltypen JOIN Reifen USING (TeiletypID);

CREATE OR REPLACE RULE insertMotor AS ON INSERT TO IngenieursichtMotoren
DO INSTEAD (INSERT INTO Autoteiltypen(maxPreis, Bezeichnung) VALUES (NEW.maxPreis,
NEW.Bezeichnung);
       INSERT INTO Motoren VALUES (lastVal(), NEW.PS, NEW.Drehzahl, NEW.Verbrauch,
NEW.Spritart));

CREATE OR REPLACE RULE insertKarosserie AS ON INSERT TO IngenieursichtKarosserien
DO INSTEAD (INSERT INTO Autoteiltypen(maxPreis, Bezeichnung) VALUES (NEW.maxPreis,
NEW.Bezeichnung);
       INSERT INTO Karosserien VALUES (lastVal(), NEW.Farbe, NEW.Material, NEW.Höhe,
NEW.Breite, NEW.Länge));

CREATE OR REPLACE RULE insertTür AS ON INSERT TO IngenieursichtTüren
DO INSTEAD (INSERT INTO Autoteiltypen(maxPreis, Bezeichnung) VALUES (NEW.maxPreis,
NEW.Bezeichnung);
       INSERT INTO Türen VALUES (lastVal(), NEW.Farbe, NEW.Türart));

CREATE OR REPLACE RULE insertFenster AS ON INSERT TO IngenieursichtFenster
DO INSTEAD (INSERT INTO Autoteiltypen(maxPreis, Bezeichnung) VALUES (NEW.maxPreis,
NEW.Bezeichnung);
       INSERT INTO Fenster VALUES (lastVal(), NEW.Tönung, NEW.Glasart));

CREATE OR REPLACE RULE insertReifen AS ON INSERT TO IngenieursichtReifen
DO INSTEAD (INSERT INTO Autoteiltypen(maxPreis, Bezeichnung) VALUES (NEW.maxPreis,
NEW.Bezeichnung);
       INSERT INTO Reifen VALUES (lastVal(), NEW.Farbe, NEW.Zoll, NEW.Felgenmaterial));

CREATE OR REPLACE RULE updateMotor AS ON UPDATE TO IngenieursichtMotoren
DO INSTEAD (UPDATE Autoteiltypen SET maxPreis = NEW.maxPreis, Bezeichnung =
NEW.Bezeichnung WHERE TeiletypID = OLD.TeiletypID;
       UPDATE Motoren SET PS = NEW.PS, Drehzahl = NEW.Drehzahl, Verbrauch =
NEW.Verbrauch, Spritart = NEW.Spritart
       WHERE TeiletypID = NEW.TeiletypID);

CREATE OR REPLACE RULE updateKarosserie AS ON UPDATE TO IngenieursichtKarosserien
DO INSTEAD (UPDATE Autoteiltypen SET maxPreis = NEW.maxPreis, Bezeichnung =
NEW.Bezeichnung WHERE TeiletypID = OLD.TeiletypID;
       UPDATE Karosserien SET Farbe = NEW.Farbe, Material = NEW.Material, Höhe =
NEW.Höhe, Breite = NEW.Breite, Länge = NEW.Länge
       WHERE TeiletypID = NEW.TeiletypID);
```

```
CREATE OR REPLACE RULE updateTür AS ON UPDATE TO IngenieursichtTüren
DO INSTEAD (UPDATE Autoteiltypen SET maxPreis = NEW.maxPreis, Bezeichnung =
NEW.Bezeichnung WHERE TeiletypID = OLD.TeiletypID;
       UPDATE Türen SET Farbe = NEW.Farbe, Türart = NEW.Türart
       WHERE TeiletypID = NEW.TeiletypID);

CREATE OR REPLACE RULE updateFenster AS ON UPDATE TO IngenieursichtFenster
DO INSTEAD (UPDATE Autoteiltypen SET maxPreis = NEW.maxPreis, Bezeichnung =
NEW.Bezeichnung WHERE TeiletypID = OLD.TeiletypID;
       UPDATE Fenster SET Tönung = NEW.Tönung, Glasart = NEW.Glasart
       WHERE TeiletypID = NEW.TeiletypID);

CREATE OR REPLACE RULE updateReifen AS ON UPDATE TO IngenieursichtReifen
DO INSTEAD (UPDATE Autoteiltypen SET maxPreis = NEW.maxPreis, Bezeichnung =
NEW.Bezeichnung WHERE TeiletypID = OLD.TeiletypID;
       UPDATE Reifen SET Farbe = NEW.Farbe, Zoll = NEW.Zoll, Felgenmaterial =
NEW.Felgenmaterial
       WHERE TeiletypID = NEW.TeiletypID);



-- Über folgende Sichten können die Archive eingesehen werden:
CREATE OR REPLACE VIEW archivierteAutos AS
       SELECT * FROM Autos WHERE Status = 'ARCHIVIERT';

CREATE OR REPLACE VIEW archivierteAufträge AS
       SELECT * FROM Aufträge WHERE Status = 'ARCHIVIERT';

CREATE OR REPLACE VIEW archivierteWerksaufträge AS
       SELECT WID, Name AS Werkname, AID, Status, Herstellungsbeginn, Herstellungsende
       FROM Werksaufträge JOIN Werke USING (WID) WHERE Status = 'ARCHIVIERT';

CREATE OR REPLACE VIEW archivierteBestellungen AS
       SELECT * FROM bestellt WHERE Status = 'ARCHIVIERT';

CREATE OR REPLACE VIEW archivierteLieferungen AS
       SELECT * FROM liefert WHERE Lieferdatum IS NOT NULL;


CREATE OR REPLACE VIEW Zeitverzögerungen AS
       Select AID, Vorraussichtliches_Lieferdatum, Datum AS Eingangsdatum,
Herstellungsbeginn, Herstellungsende, Lieferdatum
       FROM Aufträge JOIN Werksaufträge USING (AID) JOIN liefert USING (AID);


-- Views für admins um vollständige logische Datenunabhängigkeit zu gewährleisten
-- Erlaubt einem Datenbankadministrator im Notfall Eingriffe an allen Tabellen
vorzunehmen
-- Bei der Manipulation von Daten durch diese Sichten ist Vorsicht geboten.
CREATE OR REPLACE VIEW admin_Personen AS
       SELECT * FROM Personen;

CREATE OR REPLACE VIEW admin_Werke AS
       SELECT * FROM Werke;

CREATE OR REPLACE VIEW admin_Mitarbeiter AS
       SELECT * FROM Mitarbeiter;

CREATE OR REPLACE VIEW admin_Werksarbeiter AS
       SELECT * FROM Werksarbeiter;

CREATE OR REPLACE VIEW admin_LKW_Fahrer AS
       SELECT * FROM LKW_Fahrer;

CREATE OR REPLACE VIEW admin_Verwaltungsangestellte AS
       SELECT * FROM Verwaltungsangestellte;

CREATE OR REPLACE VIEW admin_Lagerarbeiter AS
       SELECT * FROM Lagerarbeiter;

CREATE OR REPLACE VIEW admin_Teilelagerarbeiter AS
       SELECT * FROM Teilelagerarbeiter;
```

```
CREATE OR REPLACE VIEW admin_Autolagerarbeiter AS
       SELECT * FROM Autolagerarbeiter;

CREATE OR REPLACE VIEW admin_Großhändler AS
       SELECT * FROM Großhändler;

CREATE OR REPLACE VIEW admin_Modelle AS
       SELECT * FROM Modelle;

CREATE OR REPLACE VIEW admin_Kunden AS
       SELECT * FROM Kunden;

CREATE OR REPLACE VIEW admin_Privatkunden AS
       SELECT * FROM Privatkunden;

CREATE OR REPLACE VIEW admin_Kontaktpersonen AS
       SELECT * FROM Kontaktpersonen;

CREATE OR REPLACE VIEW admin_Aufträge AS
       SELECT * FROM Aufträge;

CREATE OR REPLACE VIEW admin_Reifen AS
       SELECT * FROM Reifen;

CREATE OR REPLACE VIEW admin_Fenster AS
       SELECT * FROM Fenster;

CREATE OR REPLACE VIEW admin_Türen AS
       SELECT * FROM Türen;

CREATE OR REPLACE VIEW admin_Karosserien AS
       SELECT * FROM Karosserien;

CREATE OR REPLACE VIEW admin_Motoren AS
       SELECT * FROM Motoren;

CREATE OR REPLACE VIEW admin_Autoteile AS
       SELECT * FROM Autoteile;

CREATE OR REPLACE VIEW admin_Bestellt AS
       SELECT * FROM bestellt;

CREATE OR REPLACE VIEW admin_Produzieren AS
       SELECT * FROM produzieren;

CREATE OR REPLACE VIEW admin_Hersteller AS
       SELECT * FROM Hersteller;

CREATE OR REPLACE VIEW admin_Liefert AS
       SELECT * FROM liefert;

CREATE OR REPLACE VIEW admin_LKWs AS
       SELECT * FROM LKWs;

CREATE OR REPLACE VIEW admin_Autos AS
       SELECT * FROM Autos;

CREATE OR REPLACE VIEW admin_Modellteile AS
       SELECT * FROM Modellteile;

CREATE OR REPLACE VIEW admin_Autoteiltypen AS
       SELECT * FROM Autoteiltypen;

CREATE OR REPLACE VIEW admin_Werksaufträge AS
       SELECT * FROM Werksaufträge;
```

7. Beispielanfragen

7.1. Änderungsoperationen

```
INSERT INTO admin_Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID)
VALUES (1, 3, 7, 3);

INSERT INTO admin_Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID)
VALUES (1, 4, 7, 3);

UPDATE admin_bestellt SET Status='ARCHIVIERT'
WHERE AID=(SELECT currval('Aufträge_aid_seq'));

UPDATE admin_Werksaufträge SET Status='ARCHIVIERT'
WHERE AID=(SELECT currval('Aufträge_aid_seq'));

DELETE FROM admin_liefert
WHERE AID=(SELECT currval('Aufträge_aid_seq'));

UPDATE admin_bestellt SET Status='ARCHIVIERT'
WHERE AID=(SELECT currval('Aufträge_aid_seq'));

UPDATE admin_Werksaufträge SET Status='ARCHIVIERT'
WHERE AID =(SELECT currval('Aufträge_aid_seq'));

DELETE FROM admin_liefert WHERE AID=(SELECT currval('Aufträge_aid_seq'));


-- 1 Transaktion, die einen fertigen Auftrag einfügt.
BEGIN;
INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID)
VALUES (1,5, 7, 3);

UPDATE bestellt SET Status='ARCHIVIERT'
WHERE AID=(SELECT currval('Aufträge_aid_seq'));

UPDATE Werksaufträge SET Status='ARCHIVIERT'
WHERE AID=(SELECT currval('Aufträge_aid_seq'));

DELETE FROM liefert
WHERE AID=(SELECT currval('Aufträge_aid_seq'));
COMMIT;


-- 2 Transaktion, die einen archivierten Auftrag löscht.
BEGIN;
DELETE FROM admin_Autoteile WHERE AID=1;
DELETE FROM admin_bestellt WHERE AID=1;
DELETE FROM admin_liefert WHERE AID=1;
DELETE FROM admin_Werksaufträge WHERE AID=1;
DELETE FROM admin_Aufträge WHERE AID=1;
COMMIT;


-- 3 Transaktion, die einen Großhändler mit Kontaktperson einfügt.
BEGIN;
INSERT INTO admin_Großhändler (Firmenname, Straße, PLZ, Ort, Rabatt)
VALUES ('Fuego Corp', 'IobgabguodeStr. 19', '00000', 'Somewhere', 9);

INSERT INTO admin_Personen (Vorname, Nachname, PLZ, Straße, Wohnort, Email, TelNr)
VALUES ('Thomas', 'Müller', '12345', 'Gortortstr. 1', 'Tuzfy', 'Trererestr@gmx.de',
          '0863493738');

INSERT INTO Kunden VALUES ((SELECT currval('personen_pid_seq')), 40000);

INSERT INTO admin_Kontaktpersonen (PID,GID)
VALUES ((SELECT currval('personen_pid_seq')),currval('großhändler_gid_seq'));
COMMIT;
```

```
-- 4 Transaktion, die ein neues Modell mit Autoteiltypen einfügt.
BEGIN;
DO $$
DECLARE
teil1 integer;
teil2 integer;
teil3 integer;
teil4 integer;
car integer;
BEGIN
INSERT INTO admin_Autoteiltypen (maxpreis, Bezeichnung)
VALUES ('888.99', 'Yufeka');
teil1=currval('autoteiltypen_teiletypid_seq');

INSERT INTO admin_Autoteiltypen (maxpreis, Bezeichnung)
VALUES ('156', 'Oplelj');
teil2=currval('autoteiltypen_teiletypid_seq');

INSERT INTO admin_Autoteiltypen (maxpreis, Bezeichnung)
VALUES ('8897', 'Hungar');
teil3=currval('autoteiltypen_teiletypid_seq');

INSERT INTO admin_Autoteiltypen (maxpreis, Bezeichnung)
VALUES ('81', 'Afhgoae');
teil4=currval('autoteiltypen_teiletypid_seq');

INSERT INTO admin_produzieren VALUES (teil1, 1, '799.99', 5);

INSERT INTO admin_produzieren VALUES (teil2, 1, '99.99', 6);

INSERT INTO admin_produzieren VALUES (teil3, 1, '7889.99', 7);

INSERT INTO admin_produzieren VALUES (teil4, 1, '79.99', 8);


INSERT INTO admin_Motoren VALUES (teil1, 188, 4679,  1, 'Elektro');

INSERT INTO admin_Türen VALUES (teil2, 'Burgunderrot', 'FLÜGELTÜR');

INSERT INTO admin_Reifen VALUES (teil4, 'Schwarz', 14, 'Chrom');

INSERT INTO admin_Karosserien
VALUES (teil3, 'Pink', 'Plastik', 654, 170, 300);

INSERT INTO admin_Modelle (Preis, Bezeichnung)
VALUES (15149.99, 'El Aridnai');
car = currval('modelle_modell_id_seq');

INSERT INTO admin_Modellteile VALUES (car, teil1 , 1);

INSERT INTO admin_Modellteile VALUES (car, teil2, 5);

INSERT INTO admin_Modellteile VALUES (car, teil3, 1);

INSERT INTO admin_Modellteile VALUES (car, teil4, 4);

END; $$ LANGUAGE plpgsql;
COMMIT;

-- 5 Transaktion, die einen Auftrag mit mehrern Modellen in das System einfügt.
BEGIN;
INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID) VALUES (1,6, 7, 3);

INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID) VALUES (2,7, 7, 3);

INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID) VALUES (3,8, 7, 3);

INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, mitarbeiterID) VALUES (4,8, 7, 3);
COMMIT;
```

7.2. Leseoperationen

```
-- 1: Zeige alle offenen Aufträge, die schon mind. 4 Tage offen sind
SELECT *
FROM ((SELECT "Auftragsnr." AS AID
       FROM offene_Aufträge) AS tmp
              JOIN admin_Aufträge
              USING (AID))
WHERE Datum<(now()-interval '4 days');


-- 2: Zeige alle Aufträge von Personen namens Michael Müller
SELECT *
FROM Kundensicht
WHERE Vorname='Michael'
AND Nachname='Müller';

-- 3: Zeige zu einem zufälligen Hersteller alle Teileangebote an.
SELECT *
FROM Herstellerangebot
WHERE HerstellerID = (SELECT HID
                      FROM admin_Hersteller
                      LIMIT 1);


-- 4: Zeige alle Mitarbeiter, die LKW_Fahrer sind
SELECT *
FROM Personal
WHERE Spezialisierung='LKW FAHRER';

-- 5: Zeige alle Motoren, die in einem Modell verbaut sind
SELECT *
FROM IngenieursichtMotoren
WHERE TeiletypID IN (SELECT TeiletypID
                     FROM produzieren);


-- 6: Zeige alle Zeitverzögerten Aufträge an
SELECT *
FROM Zeitverzögerungen
WHERE (vorraussichtliches_Lieferdatum<lieferdatum);


-- 7: Zeige alle Aufträge an, bei denen die Produktion länger als 10 Tage gedauert
hat
SELECT *
FROM Zeitverzögerungen
WHERE (Herstellungsende-Herstellungsbeginn) > 10;


-- 8: Zeige alle Modelle mit Teilen, die schonmal verkauft wurden.
SELECT Modell_id, TeiletypID, ModellteileSicht.anzahl
FROM ModellteileSicht
       JOIN admin_Aufträge
       USING (Modell_ID);


-- 9: Zeige aktuelle Produktion
SELECT *
FROM Produktion;


-- 10: Zeige Kunden
SELECT *
FROM VerwaltungKundenaufnahme;
```

7.3. Analyseoperationen

```
-- 1: Die Anzahl aller bereits verkauften Autos absteigend sortiert.
SELECT Modelle.Modell_ID, Bezeichnung,
          sum(archivierteAufträge.Anzahl) AS "bereits verkauft"
FROM archivierteAufträge
JOIN Modelle
ON Modelle.Modell_ID = archivierteAufträge.Modell_ID
GROUP BY Modelle.Modell_ID
ORDER BY "bereits verkauft" DESC;


-- 2: Alle Autoteiltypen mit Anzahl der Hersteller, Mindestpreis und den zu diesem
Preis produzierenden Hersteller
SELECT tmp1.TeiletypId, Bezeichnung,  Herstelleranzahl, Mindestpreis, Herstellerid,
Firmenname
FROM
(
SELECT TeiletypId, Bezeichnung, count(*) AS Herstelleranzahl, min(Preis) AS
Mindestpreis
FROM herstellerangebot
GROUP BY teiletypid, Bezeichnung
) AS tmp1
JOIN

(SELECT TeiletypID, Herstellerid, Firmenname, Preis
FROM herstellerangebot
) AS tmp2

ON tmp1.teiletypid = tmp2.teiletypid AND mindestpreis = preis
ORDER BY tmp1.Teiletypid;


-- 3: Produktivität der Werke beim Bearbeitung der Werksaufträge hinsichtlich der
gebauten Autos pro Zeit
WITH tmp AS (
SELECT auftragsnummer AS aid, anzahl, modell
FROM kundensicht
)

SELECT WID, avg(tmp2.Effizienz) AS "durchnittl. Werksperformance"
FROM
(
SELECT wid, aid, age(herstellungsende, herstellungsbeginn) AS zeit, anzahl,
    modell, Anzahl / 1 + (SELECT (EXTRACT(epoch FROM age(herstellungsende,
                          herstellungsbeginn))/3600 ):: integer) AS Effizienz
FROM archivierteWerksaufträge
JOIN tmp
USING (aid)
ORDER BY Effizienz DESC
) AS tmp2
GROUP BY WID;


-- 4: Zeigt an, wieviele der LKWs, die ein Fahrer gefahren hat bereits defekt sind.
SELECT MID, count(*)
FROM liefert
GROUP BY MID, LKW_ID
HAVING (LKW_ID IS NULL);

-- 5: Zeigt die Großhändler an, die einen Rabatt haben und was deren teuerste
Bestellung bei uns war.
SELECT count(*) AS "Anzahl Einkäufe", Rabatt, Firmenname, max(Preis) AS "teuerster
Einkauf", GID
FROM ((Kontaktpersonen
    JOIN (SELECT KundenID AS PID, Preis FROM Aufträge) AS tmp USING (PID))
    JOIN Großhändler USING (GID))
GROUP BY GID, Rabatt, Firmenname
HAVING Rabatt>0;
```

7.4. MapReduce

```
--Realisiert die 5. Analyse Abfrage als Map-Reduce Funktion.
CREATE OR REPLACE FUNCTION map(integer) RETURNS TABLE (GID integer,Preis
numeric(10,2)) AS
    $$
    BEGIN
    IF((SELECT Rabatt FROM Großhändler WHERE Großhändler.GID=(SELECT
Kontaktpersonen.GID FROM Kontaktpersonen WHERE PID=$1))>0) THEN
          RETURN QUERY (SELECT Großhändler.GID, Aufträge.Preis FROM ((Kontaktpersonen
JOIN Großhändler USING (GID)) JOIN Aufträge ON KundenID=PID));
    END IF;
    END; $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION reduce(integer, numeric ARRAY) RETURNS TABLE("Anzahl Käufe"
integer, Rabatt integer, Firmenname varchar, "teuerster Einkauf" numeric(10,2), GID
integer) AS
    $$
    DECLARE
    maxPrice numeric(10,2);
    price numeric(10,2);
    counter integer;
    discount integer;
    buisnessname varchar;
    BEGIN
    counter=0;
    maxPrice=-42;
    FOREACH price IN ARRAY $2
    LOOP
          counter=counter+1;
          IF(price>maxPrice)THEN
                maxPrice=price;
          END IF;
    END LOOP;
    buisnessname=(SELECT Großhändler.Firmenname FROM Großhändler WHERE
                Großhändler.GID=$1);
    discount=(SELECT Großhändler.Rabatt FROM Großhändler WHERE Großhändler.GID=$1);
    RETURN QUERY (SELECT counter, discount, buisnessname, maxPrice, $1);
    END; $$ LANGUAGE plpgsql;


--Ausgeführt wird das Statement, indem man erst Map auf eine GID aufruft:
--SELECT * FROM map(1);
--und dann reduce auf einen Ikey(GID) und ein Array von Ivalues aufruft:
--SELECT * FROM reduce(1,  ARRAY[37998.10,1899905.00,18999050.00]);


--Beispieldaten für Map-Reduce:
--INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, MitarbeiterID) VALUES (1, 100, 1,
3);
--INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, MitarbeiterID) VALUES (1, 1000,
1, 3);
--INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, MitarbeiterID) VALUES (1, 10000,
1, 3);
--INSERT INTO Aufträge (Modell_ID, Anzahl, KundenID, MitarbeiterID) VALUES (1, 100000,
1, 3);
```

8. Indexe

```
-- Index Autos
CREATE INDEX IDX_Autos ON Autos (KFZ_ID) WHERE Status = 'ARCHIVIERT';

-- Index Aufträge
CREATE INDEX IDX_Aufträge ON Aufträge (AID) WHERE Status = 'ARCHIVIERT';

-- Index Werksaufträge
CREATE INDEX IDX_Werksaufträge ON Werksaufträge (WID, AID) WHERE Status = 'ARCHIVIERT';

-- Index bestellt
CREATE INDEX IDX_bestellt ON bestellt (BID) WHERE Status = 'ARCHIVIERT';

-- Index liefert
CREATE INDEX IDX_liefert ON liefert (KFZ_ID, MID, AID) WHERE Lieferdatum IS NOT NULL;
```

9. Zusammenfassung

Wir konnten uns allen in der Projektbeschreibung genannten Punkten zuwenden und die angesprochenen Aspekte mittels funktionalen Triggern in einem bis auf unvorhersehbare Faktoren automatischen DBMS vereinen. Insbesondere bedeutet dies, dass unser Datenbanksystem eingehende Aufträge intelligent einem Werk zuordnet, automatisch benötigte Teile nachbestellt, nach der Produktion die Autos in das zentrale Autolager verlegt und abschliessend eine zeitnahe Lieferung in die Wege leitet. Unterbrechungen in Form von Verzögerungen in der Teile- oder Autolieferung sowie bei der Produktion in den Werken werden dem DBMS über Scans mitgeteilt und bewirken die Fortsetzung der Auftragsabarbeitung und eventuell Anpassungen am Ablauf.

Weiterhin versuchten wir finanzielle Aspekte bei der Auftragsabarbeitung zu berücksichtigen. Dies taten wir durch die Aufnahme herstellerabhängiger Autoteilpreise, die Einfluss auf die Auswahl des Herstellers haben. Dadurch wurde ebenfalls die Analyse von Profit ermöglicht. Selbstverständlich spielt die Zeit in unserem System nach wie vor die Hauptrolle. Dennoch wollten wir das Szenario möglichst realitätsnah modellieren.

Schlussendlich haben wir mehr Sichten als für die logische Datenunabhängigkeit erforderlich implementiert, um die Konsistenz unseres Systems bei nicht administratorischen Zugriffen zu gewährleisten. Unsere Unternehmenshierarchie konnten wir so auf das DBMS übertragen und die Entwicklung von Anwendungen für Mitarbeiter erleichtern.

Weitere Anregungen für künftige Weiterentwicklungen unseres Systems sind Einbindung auftragsspezifischer Daten, eine Verbesserung der Entscheidungsheuristiken sowie verbesserte Selbstreinigung der Datenbank in Hinblick auf potenziell große Indexe.