Anwendungssystem einer Kaufhauskette

Hausarbeit im Bereich Datenbanksysteme

Bremen, 12. Februar 2004

Andrea Darabos
Leobener Str. 4.
28359 Bremen
Email: darabos_andrea@hotmail.com
Matrikelnr.: 1877456

Inhaltsverzeichnis

Inhaltsverzeichnis 2
Anwendungsbeschreibung 3
ER-Diagramm...4
Erklärung von Entities und Kardinalitäten 5
  Entity-Typs...5
  Relationships 6
Semantische Integrität, Integritätsregeln 8
Integritätsbedingungen in dem konzeptionellen Schema 8
Das Relationale Schema 11
Weitere Integritätsbedingungen 15
  Übersetzung der Integritätsbedingungen 15
  Referentielle Integritätsbedingungen...16
Beispiel-Zustand 17
Anfragen 21
  Anfragen im textuellen Form..24
  Anfragen im Relationale Algebra dargestellt 25
  Anfragen und Deskription im DATALOG 28
  Anfragen im Bereichskalkül..30
  Anfragen im Tupelkalkül 31
  Anfragen der SQL-Sprache 32
Sichten im SQL..35
  Die universale Sicht: W-UNIV 37
Literaturverzeichnis..40

Anwendungsbeschreibung

Eine Gemischtwarenhauskette braucht ein EDV-System, das viele Daten nach Standorten getrennt aber auch für die Kette als Ganzes verwaltet. Zunächst gilt es, die angebotenen Waren, die sich in verschiedene Warengruppen gliedern, in das System aufzunehmen. Das Warenangebot kann von Standort zu Standort unterschiedlich sein. Dies ist zu einem gewissen Grad von den regionalen Zulieferern abhängig, die die Kaufhäuser mit Waren versorgen.

Die zu der Kette gehörigen Kaufhäuser befinden sich an verschiedenen Standorten und sind in Abteilungen untergliedert. Jeder Abteilung sind eine oder mehrere Warengruppen zugeordnet.
Ausserdem hat jede Abteilung feste Mitarbeiter, die unterschiedliche Positionen bekleiden und dementsprechend unterschiedlich viel verdienen.

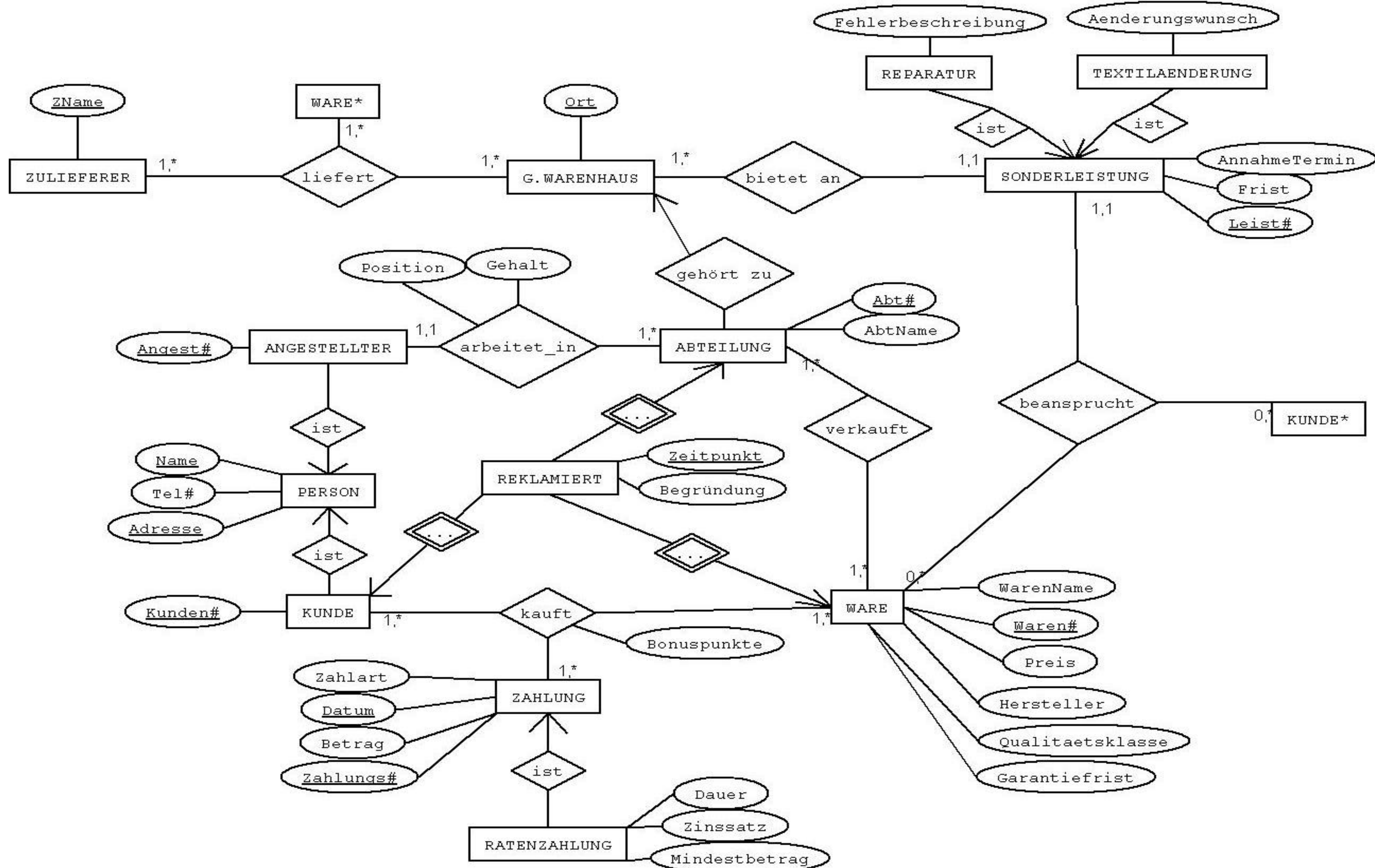
Jede Ware in der Warenhauskette hat eigene Kennzeichen, eine eigene Warennummer, die überall das gleiche ist. Neben diesen verwalte ich noch den Name, den Preis, den Hersteller, die Qualitätsklasse und die Garantiefrist für jeder Ware.

Zusätzlich zu ihren Waren bietet die Kaufhauskette Sonderleistungen, wie z. B. Textiländerungen an. Ein Kunde erhält, wenn er eine etwas aufwändigere Sonderleistung in Anspruch nimmt, einen Abholschein. Zu einem bestimmten Termin kann der Kunde seine Ware mit diesem Schein wieder in Empfang nehmen.

Sollte ein Kunde mit der erstandenen Ware nicht zufrieden sein, kann er sie (im Rahmen der Garantiefrist) reklamieren. Nachdem bestätigt ist, dass es sich um eine berechtigte Reklamation handelt, bekommt der Kunde entweder ein neues Exemplar der reklamierten Ware oder einen dem Warenwert entsprechenden Gutschein. Diesen Gutschein kann der Kunde in einer der Filialen der Warenhauskette einlösen.

Bevor jedoch eine Ware reklamiert, geändert oder einfach nur mit nach Hause genommen werden kann, muss sie gekauft werden. Bezahlen kann man entweder in bar oder per Kreditkarte. Wenn der Kunde es wünscht, kann auch eine Ratenzahlung vereinbart werden. In diesem Fall zahlt er 12, 23 oder 36 monatliche Raten, deren Summe einen gewissen Prozentsatz über dem eigentlichen Kaufpreis der betreffenden Ware liegt. Ausserdem erhält der Kunde – vorausgesetzt er nimmt an dem Bonusprogramm der Warenhauskette teil – relativ zum Rechnungsbetrag Punkte. Hat ein Kunde eine entsprechend hohe Punktzahl erreicht, kann er seine Punkte gegen eine Prämie eintauschen.

ER-Diagramm



Erklärung von Entities und Kardinalitäten

Laut dem ER-Diagramm besteht meine Datenbankmodell aus den folgenden Entity-Typs und Relationships.

Entity-Typs

ZULIEFERER(ZName)

GWARENHAUS (Ort)

Es gibt verschiedene Zulieferer, die für mehrere Filialen der Warenhauskette besorgen. Die Warenhauskette ist so ausgebaut, dass in jeder Stadt nur eine Filiale gibt, also mit dem Ort Attribut kann ich eine Filiale eindeutig identifizieren.
Ein Zulieferer liefert verschiedene Waren, so für diese 3-er Beziehung stelle ich die Kardinalitäten zu 1,*. (Wegen optimierung, lasse ich solche Fallen weg, wenn zB.ein Zulieferer keine Waren liefert.)

Für jede Ware nehme ich die folgende wichtige Attributen an:

WARE (WarenName, Waren#, Preis, Hersteller, Qualitätsklasse, Garantiefrist)

Die Waren sind durch Waren# eindeutig identifiziert. Die Garantiefrist spielt eine Rolle später in Reklamation.

ABTEILUNG (Abt#, AbtName)

Jeder Abteilung gehört zu irgendeiner Filiale der Kette und hat eine identifizierende Abteilungsnummer. In dieser Nummer ist immer der Ort der Filiale codiert.

PERSON (Name, Adresse, Tel#)

ANGESTELLTER (Name, Adresse, Tel#, Angest#)

Jeder Angestellter hat eine identifizierende Angestellternummer und arbeitet an genau einer Abteilung. Jeder Abteilung hat mindestens einen Angestellter, der in einem bestimmten Position arbeitet und eine bestimmte Summe verdient.

KUNDE (Name, Adresse, Tel#, Kunden#)

Die Kunden der Warenhauskette haben auch eine identifizierende Kundennummer, die ich später für Verwaltung der Zahlungen, Reklamationen und Sonderleistungen benutzen werde.

REKLAMIERT (Zeitpunkt, Begründung)

Reklamiert ist einen Entity-Typ, dessen Schlüssel an einer totale funktionale Beziehung geteiligt ist, deshalb nehme ich es als abhängigen Entity-Typ. Es heisst, dass zu einer

Reklamation an einer bestimmte Zeitpunkt gehört immer genau einen Kunde, eine Ware und eine Abteilung. Für Reklamation nehme ich ein Attribut als Begründung auch, um die Verwaltung zu erleichtern.

SONDERLEISTUNG (Leist#, AnnahmeTermin, Frist)

Für jeder Sonderleistung gibt es eine identifizierende Leistungsnummer, eine Annahmetermin und eine Frist, bis wann die Leistung beendet sein muss.
Eine Leistungsnummer gehört immer nur zu einen Kunde und zu eine Ware. Also jede Ware wird separiert behandelt.
Aber mehrere Kunden können Sonderleistungen beanspruchen und auch für mehrere Waren. Es kann auch sein, dass einige Kunden noch nie Sonderleistungen beansprochen haben (und auch, einige Waren, die noch nie repariert oder geändert müssen sollten).

Die zwei verschiedene Sorten von Sonderleistungen sind die folgenden:

REPARATUR (Leist#, AnnahmeTermin, Frist, Fehlerbeschreibung)

TEXTILAENDERUNG (Leist#, AnnahmeTermin, Frist, Aenderungswunsch)

Für jeder Zahlung gibt es auch eine identifizierende Zahlungs#, die an einem Zahlungsdatum erfolgt. Für dasselbe Zahlungsnummer können mehrere Waren gehören, also eine Kunde kann gleichzeitig für mehrere Waren bezahlen. Aber für Datenbankzustände erlaube ich hier auch nur 1,* Kardinalitäten, also redundante Daten , zB. Kunde ohne Zahlungs# lasse ich weg.

ZAHLUNG (Zahlart, ZahlungsDatum, Betrag, Zahlungs#)

Es gibt eine spezielle Sorte von Zahlungen, die Ratenzahlungen. Diese haben eine mit 0 beginnende, identifizierende Zahlungsnummer und neben der Attributen von Zahlung Entity, Ratenzahlungen haben auch eine Dauer, einen Zinssatz und einen Mindestbetrag. Mehrere Bedingungen werde ich im Teil `Integritätsbedingungen` definieren.

RATENZAHLUNG (Zahlart, ZahlungsDatum, Betrag, Zahlungs#, Dauer, Zinssatz,
Mindestbetrag)

Relationships

liefert (WARE, ZULIEFERER, G.WARENHAUS)

Das liefert Relationship ist ein 3-er Relationship zwischen mehrere Wahren, mehrere Zulieferer und mehrere Filialen. Die Einschränkung war hier nur, dass ich keine alleinstehende Waren, Zulieferer und Warenhäuser erlaube (Kardinalitäten von 1 zu *).

bietet_an (GWARENHAUS, SONDERLEISTUNG)

Eine Filiale bietet verschiedene, viele Sonderleistungen an, aber eine bestimmte Sonderleistung is immer an einem bestimmten Ort befindlich.

abgehört_zu (GWARENHAUS, ABTEILUNG)

Eine Abteilung mit einer bestimmten Abteilungsnummer gehört immer zu genau einer Filiale.

arbeitet_in (ANGESTELLTER, ABTEILUNG, Position, Gehalt)

An einer Abteilung können verschiedene Angestellten in verschiedenen Position und mit verschiedenen Gehälter arbeiten. Aber einem Angestellter ist nur erlaubt an genau einer Abteilung zu arbeiten.

verkauft (ABTEILUNG, WARE)

Eine Abteilung verkauft viele, verschiedene Waren, und auch, eine bestimmte Ware kann in mehrere Städten und Abteilungen verkauft werden.

kauft (KUNDE, WARE, ZAHLUNG, Bonuspunkte)

Eine Kunde kann mehrere Waren gleichzeitig kaufen, aber zu einer Zahlungs# und Datum gehört immer einen bestimmten Kunde und eine Ware. (kauft Relation ist in 1.NF)
Die redundante Daten lasse ich weg mit Wahl 1,* als Kardinalitäten.
Für jeder einzeilne gekaufte Ware bekommt aber der Kunde Bonuspunkte, die sie sammeln und später abkaufen kann.

beansprucht (KUNDE, SONDERLEISTUNG, WARE)

Hier kann ich aber 0,* als Kardinalität erlauben, da es sicher mehrere Waren gibt die noch nie repariert oder geändert werden mussten.Es gilt auch für einige Kunden, die noch nie eine Sonderleistung beansprucht haben. Aber zu einer bestimmte Sonderleistung (und Leistungsnummer) muss immer genau einen Kunde und eine Ware gehören.

ist (REPARATUR, SONDERLEISTUNG)

ist (TEXTILAENDERUNG, SONDERLEISTUNG)

ist (KUNDE, PERSON)

ist (ANGESTELLTER, PERSON)

ist (RATENZAHLUNG, ZAHLUNG)

Semantische Integrität, Integritätsregeln

Es ist sehr wichtig, in einer Datenbank die logische Korrektheit von den Daten gegen Änderungen zu bewahren. Für diese Konsistenz werden die Integritätsregeln sorgen, die der DB-Programmierer immer sehr genau von der Auftrag-geber formulieren lassen muss.

Ein Integritätsregel besteht immer aus:

- eine Integritätsbedingung
- eine Menge von Objekt(typ)en, auf die sich die Bedingung bezieht
- ein Auslöser, wann die Bedingung zu überprüfen ist
- eine Reaktion, falls die Bedingung verletzt ist

Es gibt verschiedene Sorten von Integritätsbedingungen.
Statische IBen betreffen immer genau ein DB-Zustand, als die dynamische Integritätsbedingungen bezihen sich auf mehrere Zustände, Zustandsübergänge.

Diese Hausarbeit befasst sich nur mit statische Integritätsbedingungen, zwar bei normalen DB-Entwurf würden die dynamische IBen auch wichtig sein.

Integritätsbedingungen in dem konzeptionellen Schema

In dem ER Diagramm von der Kaufhaus-Datenbank sind schon bereits Integritätsbedingungen enthalten, entsprechend in der Form von Schlüsselattributen und Kardinalitäten.

In folgendem Kapitel wird die Wahl von diesen erklärt und noch additionelle Einschränkungen beschrieben.

1. Die WAREn sind durch Waren# eindeutig definiert, dass heisst wenn zwei Waren mit dasselbe Warennumer existieren, dann sind die Waren auch gleich.

Ähnliche Bedingungen gelten für die Entity-Typs ANGESTELLTER, KUNDE und ABTEILUNG, SONDERLEISTUNG, ZAHLUNG:

Für alle ANGESTELLTER gilt, dass wenn zwei Angestellter dasselbe Angest# haben, dann sind alle andere Attributen von der Angestellter auch gleich.

Für alle KUNDEn gilt, dass wenn zwei Kunden dasselbe Kunden# haben, dann sind die Kunden gleich.

Für alle ABTEILUNG gilt, dass die Abt# identifizierend ist. Es existieren also zwei Abteilungen mit dasselbe Abteilungsnummern, dann sind die Abteilungen selbst gleich, also auch Abteilungsnamen gleich.

Für alle SONDERLEISTUNG gilt, dass wenn zwei Sonderleistungen dasselbe Leistungsnummer haben, dann müssen sie in anderen Attributen auch übereinstimmen.

Für alle ZAHLUNGEN gilt, dass wenn das ZahlungsDatum und auch die Zahlungsnummer von zwei Zahlungen gleich sind, dann müssen auch ihre anderen Attributen übereinstimmen.

2. Für alle Ratenzahlung muss der Zinssatz zwischen 0 und 100 Prozent liegen und der Mindestbetrag muss kleiner als der gesamten Betrag sein.

3. Es kann eine Ratenzahlung mit 12, 24 oder 36 monatliche Raten (Dauer) vereinbart werden, also es muss gelten, dass für alle Ratenzahlung, den ZahlungsDatums müssen nach der erste Zahlungsdatum, und innerhalb der (erste Z.Datum + Dauer) Schranke liegen.

4. Jeder Zahlung gehört zu einem Kunde, aber ein Kunde kann gleichzeitig für mehrere Waren bezahlen.
Um die verschiedene Zahlungen besser definieren zu können, führe ich eine Zahlungs# als Attribut ein und geben zur RATENZAHLUNG-Entities eine Zahlungs#, die immer mit '0' beginnt ein.
Alle andere Zahlungen (Bar oder Kreditkarte) haben eine verschiedene Zahlungs# ;
Für Zahlungen mit Bargeld die Zahlungs# immer beginnt mit 1, und für Zahlungen mit Kreditkarte beginnt die Zahlungs# immer mit 2.
Die Zahlungen mit Gutschein sind gleich gehandelt, wie Zahlungen in bar, nur, es gibt kein Rückgeld zu einer Zahlung mit Gutschein.

Es ist auch wichtig, dass zu einer Zahlungs# kann nur ein ZahlungsDatum gehören, d.h. es existieren keine 2 ZAHLUNG-Entities, dessen Zahlungs# gleich ist, aber dessen ZahlungsDatum-Attribut verschieden sind.
Diese Bedingung ist so gelöst, dass in dem Code von der Zahlungs# ist das aktuelle Datum immer enthalten.

5. Änlicherweise die Leistungsnummer ist ein Code, der für jeder Datum verschieden ist, und wenn der Kunde einen Reparatur beansprucht, dann beginnt dieser Code mit 1, sonst für Textilaenderung beginnt der Code mit 2.

6. Für alle reklamiert Entity muss gelten, dass es existiert in dem DB eine kauft Relationship so dass der Kunde der reklamiert, ist dasselbe der die reklamierte Ware gekauft hat und das Datum der Zahlung dieser Ware war früher als die Zeitpunkt der Reklamation.

7. Es gibt ein Garantiefrist für jede Ware. Das bedeutet, dass das Warenhaus nur innerhalb der Garantiefrist Sonderleistungen anbietet, und akzeptiert auch Reklamationen nur in diesem Zeitraum. Ich nehme nur die akzeptierte Reklamationen in die Datenbank auf.

Für alle Reklamiert Entitiy muss in der DB eine kauft Relationship existieren, so dass zu dem reklamierenden Kunde und zu der reklamierten Ware gibt es ein Zahlungsdatum, dass die Zeitpunkt der Reklamation innerhalb der (Zahlungsdatum + Garantiefrist der Ware) Schranke liegt.

Für alle beansprucht Relation muss in der Datenbank eine kauft Relationship existieren, so dass zu dasselben Kunde und zu der reparierenden Ware gibt es ein Zahlungsdatum, dass der Annahmetermin der Sonderleistung innerhalb der (Zahlungsdatum + Garantiefrist der Ware) Schranke liegt.

8. Wenn jemand eine Sonderleistung beansprucht, dann muss für diese Sonderleistung unbedingt gelten, dass die Frist dieser Sonderleistung dem Annahmetermin folgt und nicht vica versa.

9. Die Kaufhäuser bieten Sonderleistungen an, aber nur für eigene Waren. Also nur in der Kaufhauskette gekaufte Waren können repariert oder geändert werden. Deshalb muss auch das ZahlungsDatum dieser Ware dem AnnahmeTermin vorangehen.

Für alle beanspruchtes Relationship muss gelten, dass es existiert in dem DB ein kauft Relationship so dass der Kunde der die Sonderleistung beansprucht, ist dasselbe der die reparierende Ware gekauft hat, und das Datum der Zahlung dieser Ware früher als deren AnnahmeTermin war.

10. Gleichzeitig kann einer Kunde mehrere Waren zur Reparatur bringen, aber jede Ware wird allein behandelt, und jede Ware wird eine identifizierende Leist# bekommen.

Das Relationale Schema

Laut dem obigen ersten Kapitel, wo ich den Datenbankmodell mit Entities und Relationships aufgebaut habe, muss ich jetzt ein Relationales Schema komponieren.
Die oben beschriebene Integritätsbedingungen muss dieses Schema schon enthalten.

a, **Übersetzung den Entities**:

ZULIEFERER(ZName)

GWARENHAUS(Ort)

ABTEILUNG (Ort, Abt#, AbtName)

Das Relationenschema ABTEILUNG wird aus den Attributen von dem ABTEILUNG Entity und aus dem funktionalen Beziehung „gehört zu" gebildet, die noch weitere Attributen mitbringt. Das Ort-Attribut wird als normales Attribut übernommen.

PERSON (Name, Adresse, Tel#)

Wegen der Ist- Beziehung enthalten die zugehörigen Relationenschemata KUNDE und ANGESTELLTER alle Attribute des Entity-Typs. Ich kann wieder Primärschlüssel wählen.
Unsere Wahl ist die identifizierende Kunden# und Angest#.

KUNDE (Name, Adresse, Tel#, Kunden#)

ANGESTELLTER (Name, Adresse, Tel#, Angest#)

WARE (Abt#, WarenName, Waren#, Preis, Hersteller, Qualitätsklasse, Garantiefrist)

Da verkauft eigentlich eine gehört zu Relationship ist.

SONDERLEISTUNG (Leist#, AnnahmeTermin, Frist)

REPARATUR (Leist#, AnnahmeTermin, Frist, Fehlerbeschreibung)

TEXTILAENDERUNG (Leist#, AnnahmeTermin, Frist, Aenderungswunsch)

ZAHLUNG (Zahlart, ZahlungsDatum, Betrag, Zahlungs#)

RATENZAHLUNG (Zahlart, ZahlungsDatum, Betrag, Zahlungs#, Dauer, Zinssatz, Mindestbetrag)

REKLAMIERT (Zeitpunkt, Begründung)

b, **Übersetzung der Relationships**

bietet_an (Ort, Leist#)

Beide ehemalige Schlüsselattribute von den Entity-Typs GWARENHAUS und SONDERLEISTUNG werden übernommen und bilden den neuen Primärschlüssel im Relationenschema.

abgehört_zu (Ort, AbtName, Abt#)

arbeitet_in (Angest#, Abt#, AbtName, Position, Gehalt)

verkauft (Abt#, Waren#, WarenName, Preis, Hersteller, Qualitätsklasse, Garantiefrist)

Zwei Beziehungen mit eigenem Schlüssel-attributen:

kauft (Kunden#, WarenName, Waren#, Zahlungs#, ZahlungsDatum, Bonuspunkte)

reklamiert (Kunden#, WarenName, Waren#, Abt#, AbtName, Zeitpunkt, Begründung)

beansprucht (Kunden#, WarenName, Waren#, Leist#)

liefert (Zname, Waren#, Ort)

Die Ist-Beziehungen werden nicht als neue Relationen übernommen da ich sie schon bei den Punkt a, in Anspruch genommen habe. (Bei Spezialisierung enthalten die spezialisierende Entities auch den Schlüsseln von den Ober-Entities.)

c**, Optimierung von dem DB-Schema**

Man merkt sofort, dass diese Relationen-DB-Schema noch redundant ist, also gegebenenfalls muss man Relationen streichen und/oder zusammenfassen, und Attribute bzw. Relationen umbenennen. Aber bei der Vereinfachung von den Shemata muss ich noch weitere Integritätsbedingungen einführen.

1. Ich streiche den Relation ZULIEFERER(ZName), da es seine Attribute Teilmenge von den Attributen dem Relation liefertWaren sind (ZName, Ort, LieferTermin). Das kann ich machen,da folgendes gilt:

Jeder Zulieferer in dem DB liefert mindenstens eine Sorte von Waren, also Attribute Ort, und LieferTermin hat.

Dann kann ich einfach den Relation liefertWaren zu LIEFERUNGEN umbenennen.

2. Wir brauchen auch der Relation abgehört_zu nicht, weil alle Attribute von diesem im Relation ABTEILUNG enthalten sind. Also ich kann einfach abgehört_zu weglassen.

3. Ich kann GWARENHAUS (Ort) weglassen und den Relation ABTEILUNG zu KAUFHAUS umbenennen.
Begründung:
Jede Kaufhaus hat mindestens eine Abteilung.
Es ist sinnvoll und keine wirkliche Einschränkung, weil ohne Abteilungen kein Kaufhaus existieren kann.

4. Ich kann WARENGRUPPE auch streichen, da es im Relation verkauft enthalten ist. Begründung:

Jede Warengruppe muss zu irgendeinem Abteilung gehören, Warengruppen können nicht allein in der DB stehen.

5. Den Relation wgehört_zu kann ich weglassen, da seine Attribute im WARE enthalten sind.

6. Den Relation PERSON braucht man auch nicht mehr, da alle Attributen von diesen im KUNDE und im ANGESTELLTER enthalten sind, und die Vereinigung von KUNDE und ANGESTELLTER gibt immer PERSON zurück.

Weitere Redundanzen könnte man durch die Normalformen und Transformationen verrigern.

So mein **vereinfachtes DB-Schema** ist die Folgende:

KAUFHAUS (Ort, Abt#, AbtName)

KUNDE (Name, Adresse, Tel#, Kunden#)

ANGESTELLTER (Name, Adresse, Tel#, Angest#)

WARE (WarenName, Waren#, Preis, Hersteller, Qualitätsklasse, Garantiefrist)

SONDERLEISTUNG (Leist#, AnnahmeTermin, Frist)

REPARATUR (Leist#, AnnahmeTermin, Frist, Fehlerbeschreibung)

TEXTILAENDERUNG (Leist#, AnnahmeTermin, Frist, Aenderungswunsch)

ZAHLUNG (Zahlart, ZahlungsDatum, Betrag, Zahlungs#)

RATENZAHLUNG (Zahlart, ZahlungsDatum, Betrag, Zahlungs#, Dauer, Zinssatz, Mindestbetrag)

bietet_an (Ort, Leist#)

arbeitet_in (Angest#, Abt#, AbtName, Position, Gehalt)

verkauft (Abt#, AbtName, Waren#)

kauft (Kunden#, WarenName, Waren#, Zahlungs#, ZahlungsDatum, Bonuspunkte)

reklamiert (Kunden#, WarenName, Waren#, Abt#, AbtName, Zeitpunkt, Begründung)

beansprucht (Kunden#, WarenName, Waren#, Leist#)

liefert (Zname, Waren#, Ort)

Weitere Integritätsbedingungen

Übersetzung der Integritätsbedingungen

1. Schlüsselattribute von den Entities im ER-Modell sind Schlüsselattribute in meinem neuen Relationenmodell geblieben, also diese Bedingung bleibt erfüllt.

2. Bei Ratenzahlungen muss der Zinssatz zwischen 0 und 100 Prozent liegen. Dass heisst mit RA formuliert:

   $\sigma_{\text{Zinssatz} < 0}$ (RATENZAHLUNG) = ∅
   und
   $\sigma_{\text{Zinssatz} > 100}$ (RATENZAHLUNG) = ∅

   Bemerkung: ∅ bedeutet hier eine leere Menge.

3. Es kann eine Ratenzahlung mit 12, 24 oder 36 monatliche Raten (Dauer) vereinbart werden. Ich formuliere diese Bedingung wieder mit Termen von Relationalen Algebra:

   $\sigma_{\text{Dauer} \neq 12 \text{ and Dauer} \neq 24 \text{ and Dauer} \neq 36}$(RATENZAHLUNG) = ∅.

4. bleibt so wie definiert.

5. Die Zeitpunkt der Reklamation sollte das ZahlungsDatum der Ware folgen.

   $\sigma_{\text{ZahlungsDatum} > \text{Zeitpunkt}}$ (reklamiert * kauft) = ∅.

6. Waren können nur innerhalb der Garantiefrist geändert oder reklamiert werden.

   $\sigma_{(\text{ZahlungsDatum} + \text{Garantiefrist}) > \text{AnnahmeTermin}}$ (SONDERLEISTUNG * WARE * kauft) = ∅

   $\sigma_{(\text{ZahlungsDatum} + \text{Garantiefrist}) > \text{Zeitpunkt}}$ (reklamiert* WARE * kauft) = ∅

7. Für Sonderleistungen muss gelten, dass die Frist dem Annahmetermin folgt und nicht umgekehrt.

   $\sigma_{\text{Frist} < \text{Annahmetermin}}$ (SONDERLEISTUNG) = ∅.

8. Die Zeitpunkt des AnnahmeTermins sollte das ZahlungsDatum der Ware folgen, da das Kaufhaus bietet Sonderleistungen für eigene Waren an.

   $\sigma_{\text{ZahlungsDatum} > \text{Annahmetermin}}$ (SONDERLEISTUNG * kauft) = ∅.

Referentielle Integritätsbedingungen

Bei der Übersetzung von Relationstypen fehlen noch solche Integritätsbedingungen, die sicherstellen, dass in den Relationenschemata nur Objekte vorkommen, die auch in den Entitytyp-Schemata vorhanden sind.

Diese Erscheinung kommt davon, dass das ER-Modell und das Relationenmodell ein bisschen unterschiedlich in der Ausdrucksfähigkeit sind.

1. Z.B. im ER-Modell war „bietet_an” eine Beziehung, und Leist# gehörte zu einer SONDERLEISTUNG und Ort zu einem GWARENHAUS.
   In meinem Relationen Schema muss ich sicherstellen, dass in dem Relation „bietet an”, die Attribute gleich mit den Attribute im ER sind.
   Also IB mit Relationalen Algebra formuliert:

   $p_{Ort}\ (\text{bietet_an}) \subseteq p_{Ort}\ (\text{GWARENHAUS})$
   $p_{Leist\#}\ (\text{bietet_an}) \subseteq p_{Leist\#}\ (\text{SONDERLEISTUNG})$

2. Bei dem Relation „arbeitet_in”:

   $p_{Angest\#}\ (\text{arbeitet_in}) \subseteq p_{Angest\#}\ (\text{ANGESTELLTER})$

3. Bei dem Relation „verkauft” :

   $p_{Abtt\#}\ (\text{verkauft}) \subseteq p_{Abt\#}\ (\text{ABTEILUNG})$
   $p_{Waren\#}\ (\text{verkauft}) \subseteq p_{Waren\#}\ (\text{WARE})$

4. Bei dem Relation „kauft” :

   $p_{Kunden\#}\ (\text{kauft}) \subseteq p_{Kunden\#}\ (\text{KUNDE})$
   $p_{Waren\#}\ (\text{kauft}) \subseteq p_{Waren\#}\ (\text{WARE})$
   $p_{Zahlungs\#,ZahlungsDatum}\ (\text{kauft}) \subseteq p_{Zahlungs\#,ZahlungsDatum}\ (\text{ZAHLUNG})$

5. Bei dem Relation „reklamiert” :

   $p_{Kunden\#}\ (\text{reklamiert}) \subseteq p_{Kunden\#}\ (\text{KUNDE})$
   $p_{Waren\#}\ (\text{reklamiert}) \subseteq p_{Waren\#}\ (\text{WARE})$
   $p_{Abt\#}\ (\text{reklamiert}) \subseteq p_{Abt\#}\ (\text{ABTEILUNG})$

6. Und bei dem Relation „beansprucht” :

   $p_{Leist\#}\ (\text{beansprucht}) \subseteq p_{Leist\#}\ (\text{SONDERLEISTUNG})$

Beispiel-Zustand

In diesem Kapitel nehme ich fiktive Tupeln als Beispiel-Zustand auf. Alle diese Relationen erfüllen aber die bisherig formulierte Integritätsbedingungen.

Ich stelle die Beispiel-Zustand im tabellarischen Form vor.

LIEFERT	ZName	Ort	LieferTermin
	Foodtrans	Bremen	010204
	Foodtrans	Köln	020404
	W&Co	Bremen	080204
	Rynart	Berlin	250304
	Rynart	Köln	200704

KAUFHAUS	Ort	Abt#	AbtName
	Bremen	280	Elektronische Waren
	Bremen	290	Musikwaren
	Bremen	211	Schuhen
	Köln	331	Musikwaren
	Berlin	121	Lebensmittel
	Berlin	131	Kinderkleidung

Abt# ist unterschiedlich für verschiedene Orten, da es immer als erste nummer den Ortcode enthält (ähnlich wie der Postcode und die zugehörige Stadt).

ANGESTELLTER	Name	Adresse	Tel#	Angest#
	Weiss	Brake	234123	145643
	Gross	Oldenburg	541167	121340
	Denz	Berlin	314567	145211
	Klein	Köln	123453	213443
	Burg	Berlin	312223	156678
	Blau	Dresden	413512	313342

Arbeitet_in	Angest#	Abt#	AbtName	Position	Gehalt
	313342	211	Schuhen	Manager	2500
	145643	280	Elektr.Waren	Kassierer	1300
	213443	331	Musikwaren	Putzfrau	800
	121340	290	Musikwaren	Kassierer	1200
	145211	131	Kinderkleid.	Abt.leiter	2500
	156678	121	Lebensmittel	Kassierer	1200

Verkauft	Abt#	AbtName	Waren#
	290	Musikwaren	9981
	290	Kleider	9981
	290	Musikwaren	9099
	121	Lebensmittel	0069
	131	Kinderkleid	1412
	131	Kinderkleid	1665
	280	Elektr.W.	3321
	280	Elektr.W.	2232
	331	Musikwaren	9982

WARE	Waren#	WName	Preis	Herst.	Qual.Klasse	Garantiefrist
	9981	MadonnaCD	20	UK	1	0
	9099	MI (DVD)	20	USA	1	0
	3321	S&C Tel	40	USA	2	1 Jahr
	2232	Waschgerät	300	EU	1	5 Jahren
	1412	Rock	20	UK	2	0
	1665	Jacke	30	Spanien	1	0
	9982	JazzCD	20	EU	1	0

KUNDE	Name	Adresse	Tel#	Kunden#
	Müller	Bremen	254321	675466
	Klein	Hamburg	312343	543342
	Gross	Stuttgart	145632	343221
	Weiss	Köln	234564	122432
	Grün	Köln	345677	876546
	Grün	Berlin	234543	789394

Kauft	Kunden#	Waren#	WarenName	Zahlungs#	ZDatum	Bonuspunkte
	675466	9981	MadonnaCD	2030104122	030404	10
	675466	2232	Waschgerät	0300503001	300503	70
	789394	1412	Rock	1210303344	210303	20
	122432	3321	S&C Tel	2110603111	110603	30
	343221	1665	Jacke	2170104332	170104	50
	343221	9099	MI (DVD)	2200104103	200104	10

ZAHLUNG	Zahlart	Zahlungs#	ZDatum	Betrag
	Karte	2030104122	030404	20
	Ratenz	0300503001	300503	300
	Bar	1210303344	210303	20
	Karte	2110603111	110603	40
	Karte	2170104332	170104	30

RATENZAHLUNG	Zart	Z#	ZDat	Betrag	Dauer	Zinssatz	M..betrag
	R	0300503001	300503	300	6	5	50

Reklamiert	K#	W#	WarenName	Abt#	AbtName	Zeitpunkt	Begründung
	675466	2232	Waschgerät	280	Elektr.Waren	120104	kaputtgeg.
	789394	1412	Rock	131	Kinderkleid.	240303	eingegangen

Bietet_an	Ort	Leist#
	Bremen	1030403
	Bremen	1020104
	Bremen	2210104
	Köln	2140203
	Köln	2040603
	Berlin	2220104

SONDERLEISTUNG	Leist#	AnnahmeTermin	Frist
	10304031	030403	080403
	10201043	020104	120104
	22101042	210104	250104
	21402033	140203	200203
	22101043	210104	130104
	22201041	220104	290104

REPARATUR	Leist#	AnnahmeTermin	Frist	Fehlerbeschreibung
	10304031	030403	080403	schleudert nicht

TEXTILAENDERUNG	Leist#	AnnahmeTermin	Frist	Aenderungswunsch
	22201041	220104	290104	verkürzen
	22101043	210104	130104	anpassen

Beansprucht	Kunden#	Waren#	WarenName	Leist#
	675466	2232	Waschgerät	13002031
	789394	1412	Rock	22201041
	343221	1665	Jacke	22101043
	122432	5433	Uhr	10405035

Ich habe absichtlich 3 Fehler in der oberen DB-Zustand gemacht. Schauen wir wieder uns die Zustand an!

1. Bei Sonderleistung-Tupeln steckt die folgende Fehler:

Leist#	AnnahmeTermin	Frist
22101043	210104	130104

Es musste ein Verschreibungsfehler sein, dass die Frist früher als der AnnahmeTermin ist. Die Integritätsbedingung Nummer 9 ist damit verletzt.

Wir korrigieren diese folgenderweise:

22101043	210104	230104

2. Die Folgende Zeile verletzt die Integritätsbedingung 10. Im Warenhaus können die Kunden nur in der Warenhauskette gekaufte Waren eine Sonderleistung beansprechen. In unserer DB-Zustand existiert aber keine solche Waren# im kauft – Relation. Es bedeutet dann: diese Ware war nicht in der Warenhauskette gekauft.

Kunden#	Waren#	WarenName	Leist#
122432	5433	Uhr	10405035

Wir erlauben solche Fälle im korrekter Zustand nicht, deshalb lassen wir dieser Tupel weg.

3. Der dritte Fehler ist das einfachste. Die Schlüsselbedingungen sind im verkauft-Relation verletzt.

verkauft	Abt#	AbtName	Waren#
	290	Musikwaren	9981
	290	Kleider	9981

Beim Korrigieren muss man entscheiden, ob die Abteilung mit Nr 290 eine Musikwarenabteilung oder eine Kleiderabteilung ist. Dafür schauen wir uns der KAUFHAUS Relation an, und –angenommen, dass es korrekt ist – wir können entscheiden: Die Abteilung mit Abteilungsnummer 290 ist eine Musikwarenabteilung. Wir sollen also der schlechte Tupel einfach aus der Zustand löschen.

Jetzt sehen wir schon, wie wichtig die Integritätsbedingungen sind, und wie pünktlich sie definiert werden müssen. Bei Benutzung einer Datenbank muss immer ein Konsistenzüberwachung im Hintergrund laufen.

Anfragen

In diesem Kapitel werde ich einige Anfragen mit verschiedenen Sprachmitteln formulieren.
Zuerst werde ich die Anfragen im Text beschreiben und dann diese mit Termen von der Relationale Algebra darstellen. Dann wandele ich einige von diesen Anfragen ins Terme von Bereichskalkül und Tupelkalkül um, und formuliere einige auch mit Hilfe von DATALOG Sprache. Am Ende werden Anfragen mit Hilfe von SQL formuliert.

Um alle diese machen zu können, muss ich die angewandten Sprachmitteln erklären und die verschiedenen Sprachmächtigkeiten beschreiben.

Relationale Algebra

Relationale Algebra ist eine Algebra, wo man Terme für Anfragen benutzen kann und wenn man diese Terme für die aktuelle Zustand der Relationalen DB auswertet, dann bekommt man als Ergebnis wieder Relationen.

τ ($RelName_1$, $RelName_2$ …,$RelName_k$) : DB-Zustand -> $\mathcal{R}$

wobei τ ein relationaler Term ist.

Relationaler Terme sind Operationen oder korrekte Zusammensetzungen von mehrere Operationen.

Die 6 (5 und halb) Grundoperationen der RA sind die folgende:

- Vereinigung $R \cup S$
- Differenz R - S
- Kartesisches Produkt R X S
- Projektion $\pi_{\underline{A}}$ (R) wo $\underline{A}=(A_{i_1}, A_{i_2}, \dots A_{i_k})$ wo $1<=i_1 \dots i_k <=$ rang(R)=n
- Selektion σ_f (R) wo f ist eine atomare Formel
- Umbenennung $d_{C<-Ai}$ (R) wo C not in $\{A_1,\dots A_k)$

Wo R, S sind zwei Relationen aus der zugehörigen Relationen-Schemata, je mit Attributen $R(A_1, A_2, \dots, A_n)$ und $S(B_1, B_2, \dots, B_m)$

Alle andere Operationen von der RA kann man aus den Grundoperationen ableiten.
Wir werden später auch sehen, dass RA hat die selbe Sprachmächtigkeit als Bereichskalkül (BK-Safe), Tupelkalkül(TK-Safe) und SQL(-Kern).

DATALOG

DATALOG ist sowohl eine Data Definition Language und auch eine Data Manipulation Language, d.h. man kann sie für Anfragen und auch für Datenbankdefinition benutzen. Ich werde später Beispiele für beide Fällen zeigen.

Warum ist diese Sprache so beliebt?
DATALOG ist eine regelbasierte, deskriptive Sprache, die auch Rekursion handeln kann, was im RA nicht geht. Man benutzt diese Sprache oft für deduktive Datenbanken,d.h. man speichert nur den Basisrelationen und die anderen, die man mit Rekursion ableiten kann, muss man nicht speichern.

Bereichskalkül, Tupelkalkül

Kalkülen sind formale, logische Sprachen zur Formulierung von Aussagen. Im Bereich von Relationenmodelle sind 2 Kalküle bereits verwendet: der Bereichskalkül und der Tupelkalkül.
Die Anfragen haben im allgemeinem die folgende Form:

$\{ \mathbf{f}(\underline{\mathbf{x}}) \mid \mathbf{p}(\underline{\mathbf{x}}) \}$

wo $\underline{x}$ bezeichnet eine Menge von freien Variablen, über die die Bedingung ausgewertet wird.
$\underline{x}$ = {x1: D1, x2: D2, ...,xn : Dn} jeder xi Variable mit Datentyp.

Die Funktion **f** bedeutet eine Ergebnisfunktion über den freien Variablen $\underline{x}$, und **p** ist ein Selektionspredikat, mit anderen Wörter eine Einschränkung oder Qualifikation.

Die Ergebnis von einer Anfrage wird wie folgendes bestimmt:
Zuerst bestimme alle Belegungen der freien Variablen $\underline{x}$, für die das Selektionspredikat pwahr ist. Dann wende die Funktion f auf die durch diesen Belegungen gegebenen Werten an.

Ausdruksfähigkeit von BK, TK:

Es ist sehr einfach mit Hilfe von BK- oder TK-Anfragen solche Fragen stellen, die ein unendliches Ergebnis liefern. Das ist dann sehr gefährlich, da man nie das ganze Ergebnis zB. an der Bildschirm ausdrucken kann...
Deshalb vermeide ich diese unsichere Anfragen, wie zB.

{ x,y | R(x,y) } berechnet das Komplement einer endlichen Relation R.

Durch Ausschliessung dieser unsicheren Anfragen konstruiere ich BK- und TK-Safe Sprachen, wo alle Anfragen für jeden Datenbankzustand s(R) ein endliches Ergebnis liefern.

Satz:
1. Relationenalgebra
2. BK-Safe und
3. TK-Safe
sind in ihrer Ausdrucksfähigkeit äquivalent.

Der Bereichskalkül ist dadurch gekennzeichnet, dass Variablen Werte elementarer Datentypen (Datentypbereiche) annehmen. Deshalb nennt man auch diesen Kalkül Domainkalkül (domain calculus).

In dem Tupelkalkül hingegen variieren Variablen über Tupelwerte (entsprechend den Zeilen einer Relation).

Ich werde der Syntax und der Semantik der Sprachen später, bei den Anfragen erklären.

SQL

SQL ist eine alltäglich benutzte, Kalkül-basierte Sprache, wo man Anfragen mit Hilfe von Unteranfragen und select-from-where Blöcke formulieren kann.

SQL-Kern ist *streng relational vollständig*, also mit SQL kann man jeden Term in RA durch einem Ausdruck der SQL-Sprache realisieren, so dass diese dasselbe Anfrage formulieren. Auch umgekehrt gilt, dass SQL-Anfragen kann man immer als RA-Terme übersetzen lassen.

Für die zweite Richtung gilt generall:

Select T
From R1,R2
Where F

Wo:
T: belieblige Terme
R1,R2 belieblige Relationen (Relationenliste)
F: Einschränkung, belieblige Terme

Diese SQL-Anfrage bedeutet dasselbe wie die folgende RA-Term:

$\pi_T (\sigma_F (R1 X R2))$
Natürlich kann man nicht nur zwei Relationen, sondern eine Relationenliste nehmen, und dann Kartesische Produkt von dieser Liste nehmen.

Es gilt auch, dass jeder sichere Ausdruck des Tupelkalküls (auch Bereichkalküls) durch eine SQL-Anfrage äquivalent dargestellt werden kann. Diese Eigenschaft, dass SQL immer nur endliche Ergebnisse liefert, kommt von der Sprachmittel der Sprache SQL (from R[i] wo R[i] eine endliche Relationliste ist).

Anfragen im textuellen Form

1. In welchen Städte hat die Warenhauskette Filialen? (Ort)

2. Welche Abteilungen hat das Gemischtwarenhaus in Bremen? (Abt#, AbtName)

3. Welche Zulieferern liefern für das Warenhaus in Berlin?

4. Welche Angestellter arbeiten in anderer Stadt als sie wohnen?

5. Welche Angestellter haben eine Gehalt grösser als 2000 Euro, und in welcher Stadt arbeiten sie?

6. Welche Waren in der Filiale in Köln stammen aus Spanien?

7. Welche Waren haben eine Garantiefrist, also welche Waren kann man reklamieren? (select * from Ware where Garantiefrist is <> 0)

8. Welche Waren haben Qualitätsklasse 2,(und sind deshalb billiger?) (W#,Wname, Preis)

9. Welche Kunden haben im Jahr 2003 reklamiert?

10. Welche Kunden haben mit Ratenzahlung bezahlt?

11. Welche Reparaturen hat das Kaufhaus Köln im 2003 gemacht?

12. Welche Kunden warten noch auf ihre Waren bei Reparatur oder Textiländerung, angenommen, dass heute 24. 01. 2004 ist?

    Anfragen für SQL:

13. Welche Lieferanten welche Waren liefern günstiger als alle Lieferanten die diese Waren liefern?

14. Welche Waren waren noch nie reklamiert?

    Gruppierung und Aggregation:

15. Anzahl from Lieferanten

16. Welche Kunde hat schon mehr als 3mal reklamiert?

17. Anzahl der Sonderleistungen je Ort, wobei Anzahl > 600 ist.

18. Preissume von verkaufte Waren je Ware (welche Waren sind am populärsten?)

19. Waren, die von mehr als einem Lieferant geliefert werden

Anfragen im Relationale Algebra dargestellt

1. In welchen Städte hat die Warenhauskette Filialen? (Ort)

   Ich möchte wissen, welche Ort-Attributeder KAUFHAUS Relation hat. Das Ergebnis liefert die folgende Anfrage:

   p_{Ort} (KAUFHAUS)

   In meiner Beispiel-Zustand bekomme ich:

R	Ort
	Bremen
	Köln
	Berlin

2. Welche Abteilungen hat das Gemischtwarenhaus in Bremen? (Abt#, AbtName)

   $p_{\text{Abt\#, AbtName}} (\sigma_{\text{Ort = 'Bremen'}}$ (KAUFHAUS))

R	Abt#	AbtName
	280	Elektronische Waren
	290	Musikwaren

3. Welche Zulieferern liefern für das Warenhaus in Berlin?

   $p_{\text{ZName}} (\sigma_{\text{Ort = 'Berlin'}}$ (liefert))

R	Zname
	Rynart

4. Welche Angestellter arbeiten in anderer Stadt als sie wohnen? (Angest#, Name)

   $(\sigma_{\text{ANGESTELLTER.Adresse <> KAUFHAUS.Ort}}$ (ANGESTELLTER * arbeitet_in * KAUFHAUS))

   Diese Anfrage ergibt solche Tupeln, in denen die Adresse Attributen ungleich zu den Ort Attributen sind.
   Wir sollen noch aus diesen mit einer Projektion die für uns gewünschte Angest# und Name auswählen.

   $p_{\text{Angest\#, Name}} (\sigma_{\text{ANGESTELLTER.Adresse <> KAUFHAUS.Ort}}$ (ANGESTELLTER * arbeitet_in * KAUFHAUS))

R	Angest#	Name
	145643	Weiss
	121340	Gross
	313342	Blau

5. Welche Angestellter haben eine Gehalt grössergleich 2000 Euro, und in welcher Stadt arbeiten sie? (Angest#, Name, Ort)

   $p_{\text{Angest\#, Name, KAUFHAUS.Ort}} (\sigma_{\text{Gehalt} <= 2000} (\text{ANGESTELLTER * arbeitet_in * KAUFHAUS}))$

   Diese Anfrage kann ich aber noch optimieren, wenn ich die Selektion innerhalb der natürlicher Verbund schiebe:

   $p_{\text{Angest\#, Name, KAUFHAUS.Ort}} (\sigma_{\text{Gehalt} <= 2000} (\text{arbeitet_in}) * \text{KAUFHAUS} * \text{Angestellter})$

R	Angest#	Name	Ort
	113342	Blau	Dresden
	145211	Denz	Berlin

6. Welche Waren in der Filiale in Köln stammen aus Spanien? (Waren#, WarenName)

   Ich schreibe sofort die optimale Anfrage auf:

   $p_{\text{Waren\#, WarenName}} (\sigma_{\text{Herst = `Spanien`}} (\text{WARE}) * \sigma_{\text{Ort = `Köln`}} (\text{KAUFHAUS}) * \text{verkauft})$

   In unserer Zustand ist das Ergebnis {} leer, da wir für besseres Verständnis nur wenige Waren aufgenommen haben.

7. Welche Waren haben eine Garantiefrist, also welche Waren kann man reklamieren?

   $p_{\text{Waren\#, WarenName}} (\sigma_{\text{Garantiefrist} <> 0} (\text{WARE}))$

R	Waren#	WarenName
	3321	S&C Tel
	2232	Waschgerät

8. Welche Waren haben Qualitätsklasse 2,(und sind deshalb vermutlich billiger?) (W#,Wname, Preis)

   $p_{\text{Waren\#, WarenName, Preis}}\ (\sigma_{\text{Qualitätsklasse=2}}\ (\text{WARE}))$

R	Waren#	WarenName	Preis
	3321	S&C Tel	40
	1412	Rock	20

9. Welche Kunden haben im Jahr 2003 reklamiert? (Name, Kunden#)

   $p_{\text{Kunden\#, Name}}\ (\sigma_{\text{Zeitpunkt=2003}}\ (\text{reklamiert}) * \text{KUNDE})$

   In unserer Zustand ist das Erlebnis eine leere Menge.

10. Welche Kunden haben mit Ratenzahlung bezahlt?

    $p_{\text{Kunden\#, Name}}\ (\text{kauft} * \text{RATENZAHLUNG} * \text{KUNDE})$

R	Kunden#	Name
	675466	Müller

11. Welche Reparaturen hat das Kaufhaus Köln im 2003 gemacht (Leist#) ?

    $p_{\text{Leist\#}}\ [\sigma_{\text{Ort = `Köln`}}\ (\text{bietet_an}) * (\sigma\ (_{\text{AnnahmeTermin > 010103 and AnnahmeTermin< 311203}})\ \text{REPARATUR})\ ]$

    { } Leeres Ergebnis in unserer Zustand.

12. Welche Kunden warten noch auf ihre Waren bei Reparatur oder Textiländerung, angenommen, dass heute 24. 01. 2004 ist?

    $p_{\text{Kunden\#, Name}}\ \{\ [\sigma_{\text{Frist > 240104}}\ (\text{REPARATUR})\ \text{U}\ (\sigma_{\text{Frist > 240104}}(\text{TEXTILAENDERUNG})] * \text{beansprucht}\ \}$

R	Kunden#	Name
	789394	Grün

Anfragen und Deskription im DATALOG

DATALOG als Data Definition Language

Bei meinem Beispiel kann ich einfach meinen Datenbank-Schema und DB-Zustand mit Hilfe von DATALOG-Fakten beschreiben.

zB.:
DB-Schema:
KAUFHAUS(Ort, Abt#, AbtName).
KUNDE(Name, Adr, Tel#, Kunden#).
...

DB-Zustand:
KAUFHAUS(`Bremen`, `280`, `Elektronische Waren`).
KAUFHAUS(`Bremen`, `290`, `Musikwaren`).
KAUFHAUS(`Köln`, `331`, `Musikwaren`).
KUNDE(`Müller`, `Bremen`, `245321`, `675466`).
KUNDE(`Klein`, `Hamburg`, `312343`, `543342`).

Man kann auch Sichten definieren, wie folgende:

1, Alle Kunden, die Müller heissen:

MÜLLER_KUNDE(Name, Adr, Tel#, Kunden#) :- KUNDE(Name, Adr, Tel#, Kunden#),
Name = `Müller`.

Oder kurz schreiben:

MÜLLER_KUNDE(`Müller`, Adr, Tel#, Kunden#) :- KUNDE(`Müller`,Adr,Tel#, Kunden#).

Im Relationalen Algebra ist diese Anfrage mit einer Selektion ekvivalent, $\sigma_{\text{Name = `Müller`}}$ (KUNDE).

2, Einkauf-Sicht für wichtige Informationen über Zahlungen:

EINKAUF(Kname, K#, Bonusp, Zahlart, Datum, Betrag, Z#, Wname, W#, Preis, Garantiefrist) :- KUNDE(Kname, A, T, K#),
kauft(K#, Wname, W#, Z#, Datum, Bonusp),
WARE(Wname, W#, Preis, H, Q, Garantiefrist),
ZAHLUNG(Zahlart, Datum, Betrag, Z#).

Man merkt sofort, dass diese Regel bildet zuerst der Kartesische Produkt von 4 Relationen und dann macht eine Projektion auf die –für die Sicht wichtige - Attribute.

DATALOG als Anfragesprache

Mit Hilfe von DATALOG kann ich ausser dem Differenz alle Terme den RA formulieren.
Schauen wir jetzt einige von meine Anfragen im DATALOG an.

1. In welchen Städte hat die Warenhauskette Filialen? (Ort)

   KAUFHAUS_ORT(Ort) :- KAUFHAUS(Ort, Abt#, AbtName)

   Diese Regel bedeutet eine Projektion auf Ort von dem KAUFHAUS-Relation.

2. Welche Abteilungen hat das Gemischtwarenhaus in Bremen? (Abt#, AbtName)

   ABT_BREMEN(Abt#, AbtName) :- KAUFHAUS(`Bremen`, Abt#, AbtName).

   Die folgende Regel entspricht diese RA-Anfrage:

   $p_{\text{Abt\#, AbtName}} (\sigma_{\text{Ort = 'Bremen'}} (\text{KAUFHAUS})).$

3. Welche Zulieferern liefern für das Warenhaus in Berlin?

   ZULIEF_BERLIN(Zname) :- liefert(Zname, Waren#, `Berlin`).

   Diese Anfrage ist so wie die letzte, eine Projektion angewendet auf eine Selektion.

   $p_{\text{ZName}} (\sigma_{\text{Ort = 'Berlin'}} (\text{liefert}))$

4. Welche Angestellter arbeiten in anderer Stadt als sie wohnen?

   ANG_PENDLER(Ang#, AngName) : -
   ANGESTELLTER(AngName, Adresse, Tel#, Ang#),
   arbeitet_in(Ang#, Abt#, AbtName, Position, Gehalt),
   KAUFHAUS(Ort, Abt#, AbtName),
   Ort<>Adresse.

   $p_{\text{Angest\#, Name}} (\sigma_{\text{ANGESTELLTER.Adresse <> KAUFHAUS.Ort}} (\text{ANGESTELLTER * arbeitet_in * KAUFHAUS}))$

Man sieht, dass einige Anfragen im DATALOG sehr einfach und logisch formuliert werden können. Ein andere Vorteil ist auch, dass man keine komplizierte Terme und Symbole der RA anzuwenden hat, um die Ausdrücke zu schreiben. Wichtig ist nun, dass die Regeln im DATALOG nur dann funktionieren, wenn man an der rechte Seite nur positive atomare Formeln hat. Deshalb ist die Differenz-Bildung im DATALOG nicht möglich, solche Anfragen hat man im RA oder in anderen Anfragesprachen zu formulieren.

Anfragen im Bereichskalkül

Syntax der Anfragen:

Eine Anfrage im Bereichskalkül hat die folgende Form:

{ x1 [: D1], x2 [D2], ... , xn [Dn] | f }

wo f Formel entspricht eine Einschränkung, und hat freie Variablen (x1,...xn)
in dem ersten Teil sind x1,...xn mit zugehörigen Datentypen D1,...Dn die Ergebnisvariablen

Semantik:

In einem gegebenen Zustand σ bestimmt die Anfrage

{x1, x2, ... , xn | f }

die folgende Relation über den Kreuzprodukt den Datentypbereiche | D1 | X...X | Dn | :

{ (d1, ... , dn) | es gibt eine Belegung ß mit ß(xi) = di (i = 1..n),
so dass [s, ß] |= f gilt }

Wo [s, ß] |= f bedeutet: Formel f gilt im Zustand s unter der Belegung ß.

Jetzt kann ich einige von meinen Anfragen im Bereichskalkül definieren:

1. In welchen Städte hat die Warenhauskette Filialen? (Ort)

{o | ∃ y, z (KAUFHAUS(o, anu, ana)) }

Hier steht o für das Ort Attribut, und alle die anderen Variablen sind gebunden.

2. Welche Abteilungen hat das Gemischtwarenhaus in Bremen? (Abt#, AbtName)

{ anu, ana | ∃ort (KAUFHAUS(ort, anu, ana) ∧ ort = `Bremen` }.

Die obene Anfrage entspricht diese RA-Anfrage:

$p_{\text{Abt\#, AbtName}} (\sigma_{\text{Ort = 'Bremen'}} (\text{KAUFHAUS})).$

4. Welche Angestellter arbeiten in anderer Stadt als sie wohnen? (Angest#, Name)

{ anu, an | ∃adr, tnu, ort, abtnu, abtn, pos, geh (ANGESTELLTER(an, adr, tnu, anu) ∧
arbeitet_in(anu, abtnu, abtn, pos, geh) ∧
KAUFHAUS(ort, abtnu, abtn) ∧ adr <> ort)}

Die obene Anfrage im RA ausgedruckt sieht so aus:
$p_{\text{Angest\#, Name}}$ ($\sigma_{\text{ANGESTELLTER.Adresse <> KAUFHAUS.Ort}}$ (ANGESTELLTER * arbeitet_in * KAUFHAUS))

Anfragen im Tupelkalkül

Ein wichtiger Unterschied von Bereichskalkül ist, dass im TK die Variablen (hier r) sind immer Tupelvariablen, und diese variieren über dem Kreuzproduckt von Datentypen zusammen mit Komponentennamen. Hier r: (A1: D1, A2: D2, ..., An : Dn)

Deshalb einie Belegung von r ist folgendes:

ß (r)= (d1,...dn) $\in$ | D1 | X ... X | Dn |

Ich kann den i-sten Tupelkomponent mit $r.A_i$ erreichen.

Es gilt noch : [s, ß] |= R(r) genau dann, wenn ß (r) $\in$ s (R).

Schauen wir jetzt einige von meinen Anfragen im Tupelkalkül an!

1. In welchen Städte hat die Warenhauskette Filialen? (Ort)

Im RA: p_{Ort} (KAUFHAUS)
Im TK: {r : (Ort) | $\exists$ k: KAUFHAUS (k.Ort = r.Ort) }

8. Welche Waren haben Qualitätsklasse 2,(und sind deshalb vermutlich billiger?) (W#,Wname, Preis)

Ein bisschen komplizierter ist die folgende Anfrage. Im RA ist die Lösung die folgende:

$p_{\text{Waren\#, WarenName, Preis}}$ ($\sigma_{\text{Qualitätsklasse=2}}$ (WARE))

{r : (Waren#, WarenName, Preis) | $\exists$ w : WARE (w.Waren# = r.Waren# $\wedge$
w.WarenName = r.WarenName $\wedge$
w.Preis = r.Preis $\wedge$
w.Qualitätsklasse = 2) }

Anfragen der SQL-Sprache

13. Welche Lieferanten liefern welche Waren günstiger als alle Lieferanten die diese Waren liefern?

```
Select l.Zname, w.WarenName
from   liefert l, WARE w
where w.Preis <= all ( select Preis
                       From  WARE
                       Where Waren# = l.Waren#
```

14. Welche Waren waren noch nie reklamiert?

Mit dieser Anfrage bekommen wir solche Waren, die wahrscheinlich eine gute Qualität haben, da mit ihnen noch keine Problemen hatte.
Ich benutze hier den Kvantor `not exists` um diese Unteranfrage zu formulieren:

```
select w.Waren#, w.WarenName
from  WARE w
where not exists ( select *
                   from  reklamiert r
                   where w.Waren# = r.Waren# )
```

In SQL-Kern gibt es 14 Möglichkeiten Unteranfragen zu formulieren, man kann `any`, `all`, `in` and `exists` with 7 Vergleichsoperatoren kombinieren.

Gruppierung und Aggregation:

15. Anzahl von Zulieferer

Es kann oft vorkommen, dass wir wissen wollen, wieviel Zulieferer wir haben. Dann benutze ich einfach die count ( ) Aggregationsfunktion:

```
select count (distinct ZName)
from  liefert
```

Wenn ich auch für einer bestimmten Stadt (z.B. Bremen) die Anzahl von Zulieferer wissen will, dann formuliere ich es noch im where-clause:

```
select count (distinct l.ZName )
from  liefert l
where l.Ort like ‘%Bremen%’
```

16. Welche Kunde hat mal schon mehr als 3-mal reklamiert?

```
select Kunden#, count ( Waren# )
from reklamiert
group by Kunden#
having count(Waren#) > 3
```

Ich gruppiere die reklamiert Relation nach Kundennummer und dann zähle ich, wieviel eine Kunde mal schon reklamiert hat. Eine Kunde kann dasselbe Ware mehrmals reklamieren, diese Reklamationen zähle ich hier auch ein.

Aber wenn die Frage ist, dass welche Kunde schon mehr als 3 verschiedene Waren reklamiert hat, dann kann ich die Folgende schreiben:

```
Select Kunden#
from  reklamiert r
where exists ( select *
               From reklamiert r1, reklamiert r2, reklamiert r3
               Where r1.KundenName= r2.KundenName and
                     r2.KundenName = r3.KundenName and
                     r.KundenName =  r1.KundenName and
                     r1.Waren# <> r2.Waren# and
                     r2.Waren# <> r3.Waren# and
                     r.Waren# <> r1.Waren#.
```

17. Anzahl der Sonderleistungen je Ort, wobei Anzahl > 600 ist.

Diese Anfrage ist einen Beispiel für sowohl Gruppierung als auch für Aggregationsfunktionen. Es macht die folgendes:
Es nimmt die bietet_an Relation und bildet Gruppen aus den Tupeln so dass in jeder Gruppe haben die Tupeln dasselbe Ort-Attribut.
Dann mit der having-Bedingung filtere ich diese Gruppen, nur solche werden im Ergebnisrelation aufgeschrieben, die mehr als 600 Zeilen haben. (ich kann Zeilen sagen, weil zu einem bestimmten Ort nur genau ein Leistungs# gehört (Ort und Datum ist codiert ins Leist#)). Als Ergebnis bekomme ich eine Relation der für jeder Ort eine Anzahl (>600) von Leistungs#n ergibt.

```
select Ort, count ( Leist# )
from  bietet_an
group by Ort
having count ( Leist# ) > 600
```

18. Preissume von verkaufte Waren je Ware (welche Waren sind am populärsten?)

Ein anderes Beispiel für Gruppierung ist diese Anfrage:

```
Select Waren#, sum ( Preis)
from  WARE
group by Waren#
```

19. Waren, die von mehr als einem Lieferant geliefert werden

Diese Anfrage ist deshalb interessant, weil ich es entweder mit SQL-Kern, oder mit Gruppierung formulieren kann. Dieses gilt nicht für alle Anfragen: es gibt einige, die ich nur mit Hilfe von Gruppierung und Aggregationsfunktionen aussagen kann, also vom Sprachmächtigkeit die erweiterte SQL ist strenger.

```
Select distinct Waren#
from   liefert
group by Waren#
having count(Zname) > 1
```

Ohne Gruppierung dasselbe Anfrage im SQL-Kern formuliert:

```
Select distinct Waren#
from   liefert L1
where exists ( select *
                From  liefert L2
               where  L1.Waren# = L2.Waren#  and
               L1.Zname <> L2.Zname )
```

Sichten im SQL

In SQL gibt es auch die Möglichkeit, verschiedene Sichten zu definieren. Eine Sicht ist eine additionelle Relation, die eine Anfrage enthält.
ZB. Eine Sicht für den Kunden Klein:

```
create view Klein-543342-kauft (W# , Betrag )
as select Waren#, Preis
from kauft, KUNDE
where Kunden# = 543342 and Name = `Klein`
```

Welche Vorteile haben Sichten für uns?

- Sichten unterstützen die logische Datenunabhängigkeit, dass heisst man kann die externe Ebene von der DBSystem verändern, ohne die konzeptionelle und interne Ebene ändern zu müssen. Jeder Sicht definiert eine Benutzergruppe, so kann ein DB sehr mehrseitig benutzt werden.

- Die Sichten ermöglichen auch eine Beschränkung von Zugriffen auf die Daten, da nicht jeder Benutzergruppe muss alle Daten von dem DB sehen können.

- Da die Sichten eigentlich Anfragen sind, sie können andere Anfragen auch vereinfachen.

Zuerst werde ich unseres Beispiel für die dritte Punkt verwenden: wir werden sehen, wie eine Sicht eine Anfrage vereinfachen kann.
Sei eine folgende Anfrage gegeben:

Wieviel hat bisher unser Kunde `Klein` mit Kundennummer : 543342 in der Warenhauskette gekauft?

Dann diese Anfrage mit der Sicht ist so einfach geworden:

```
select sum ( Betrag)
from  Klein-543342-kauft
```

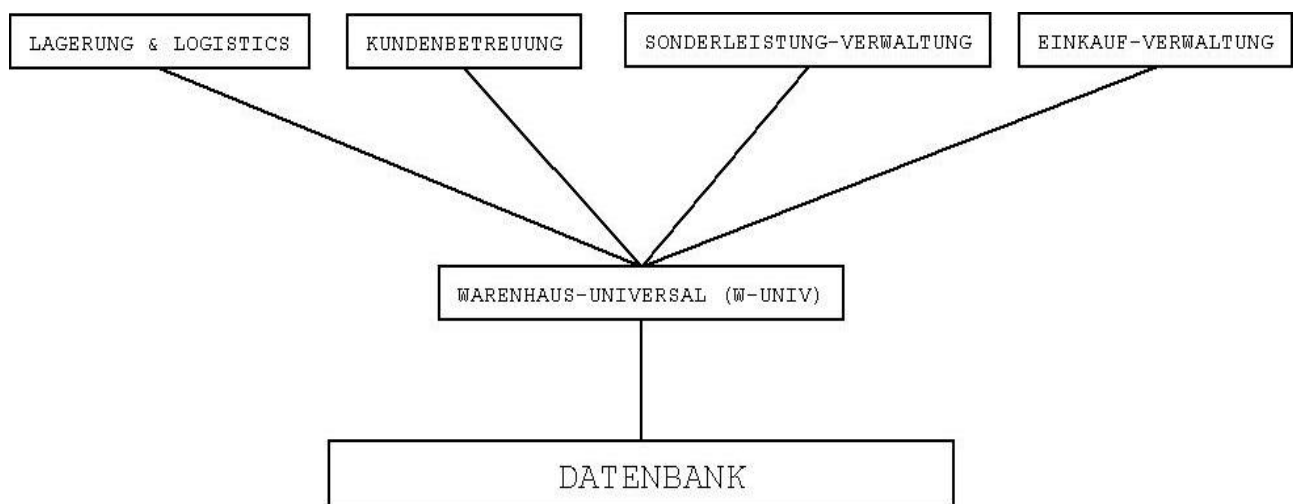
Statt der Anfrage, ohne Sicht würde es so aussehen:

```
select ku.Name, count ( w.Preis )
from kauft ka, KUNDE ku, WARE w
where ku.Kunden# = 543342 and
        ku.Kunden# = ka.Kunden# and
        ka.Waren# = w.Waren#
```

In meinem Datenbanksystem, welche Sichtdefinitionen würden sinnlos sein?
Natürlich braucht nicht jeder Benutzergruppe die ganze Datenmenge von der DB zu sehen.
Für ein höheres Sicherheit führe ich einschränkende Sichten ein, zb.

- für Lagerung,
- für Kundenbetreuung,
- für Sonderleistungen und
- für Einkauf-Anfragen.

Für ein besseres Verständnis hilft die Folgende Zeichnung. Ich definiere verschiedene Sichten auf eine universale Sicht. Diese Universale Sicht enthält alle Attribute von dem Relationalen Schema. Diese vereinfacht die Datenbank, und fasst die Relationen in einer grosse Tabelle zusammen. Diese Sichten können wir ohne änderung des Relationales Schemas definieren und benutzen.



Die universale Sicht: W-UNIV

Die universale Sicht ist ein outer natural join über alle Relationen des Relationalen Schemas. Statt viele, kleine Relationen sieht der Benutzer nur eine, universale Relation.

```
create view W-UNIV (Ort, Abt#, AbtName, KundenName,KundenAdresse, KundenTel#,
                    Kunden#, AngestellterName, Angestellter.Adresse,
                    AngestellterTel#,Angest#, WarenName,Waren#, Preis, Hersteller,
                    Qualitätsklasse, Garantiefrist, Leist#,
                    SonderleistungAnnahmeTermin, SonderleistungFrist,
                    TextilaenderungAenderungswunsch,
                    ReparaturFehlerbeschreibung, Zahlungsart,
                    ZahlungsBetrag, ZahlungsDatum,
                    Zahlungs#, RatenzahlungDauer,
                    RatenzahlungZinssatz, RatenzahlungMindestbetrag,
                    AngestPosition, AngestGehalt, Bonuspunkte, ReklZeitpunkt,
                    ReklBegründung, LiefererName)
as select (KAUFHAUS.Ort, KAUFHAUS.Abt#, KAUFHAUS.AbtName, KUNDE.Name,
           KUNDE.Adresse, KUNDE.Tel#, KUNDE.Kunden#, ANGESTELLTER.Name,
           ANGESTELLTER.Adresse, ANGESTELLTER.Tel#, ANGESTELLTER.Angest#
          WARE.WarenName, WARE.Waren#, Preis, Hersteller, Qualitätsklasse, Garantiefrist
          SONDERLEISTUNG.Leist#, SONDERLEISTUNG.AnnahmeTermin,
          SONDERLEISTUNG.Frist, TEXTILAENDERUNG.Aenderungswunsch,
          REPARATUR.Fehlerbeschreibung, ZAHLUNG.Zahlart, ZAHLUNG.Betrag,
          ZAHLUNG.ZahlungsDatum, ZAHLUNG.Zahlungs#, RATENZAHLUNG.Dauer,
          RATENZAHLUNG.Zinssatz, RATENZAHLUNG.Mindestbetrag,
          Position, Gehalt, Bonuspunkte,Zeitpunkt, Begründung, Zname)
From  ( ( ( ( ( ( ( ( ( ( ( ( ( ( KAUFHAUS      outer natural join KUNDE )
                                              Outer natural join ANGESTELLTER)
                                              Outer natural join WARE)
                                              Outer natural join SONDERLEISTUNG)
                                              Outer natural join REPARATUR)
                                              Outer natural join TEXTILAENDERUNG)
                                              Outer natural join ZAHLUNG)
                                              Outer natural join RATENZAHLUNG)
                                              Outer natural join bietet_an)
                                              Outer natural join arbeitet_in)
                                              Outer natural join verkauft)
                                              Outer natural join kauft)
                                              Outer natural join reklamiert)
                                              Outer natural join beansprucht)
                                              Outer natural join liefert)
```

Die Lagerung und Logistic-Gruppe braucht unbedingt die folgende Daten zu sehen:

Welche Zulieferer liefert welche Waren für welcher Warenhaus-Filiale. Der Preis und die Qualitätsklasse der Waren können auch noch wichtig sein.

Dafür definiere ich die folgende Sicht:

```
create view LAG (Lieferer, WNummer, WName, WPreis, WKlasse)
as select distinct LiefererName, Waren#, WarenName, Preis, QulaitätsKlasse
from W-UNIV
```

Für Kundenbetreuung, die Zulieferer sind nicht wichtig, desto wichtiger sind aber Probleme mit Waren, also Reklamation. Für Reklamation muss die Abteilung wissen, wann und welche Waren verkauft waren, und auch, ob eine Garantie für die Ware existiert.
Die Bonuspunkte sollte man auch hier anfragen können.

```
create view KUNDENBETR (KundenName, Kunden#, WarenName, Waren#, Preis,
                        Hersteller, Qualitätsklasse, Garantiefrist, BonusSumme,
                        ReklZeitpunkt, ReklBegründung, Abteilung#, AbteilungName
as select distinct KundenName, Kunden#, WarenName, Waren#, Preis, Hersteller,
           Qualitätsklasse, Garatiefrist, count(Bonuspunkte), ReklZeitpunkt,
           ReklBegründung, Abt#, AbtName
from W-UNIV
```

Für die Sonderleistung-Verwaltung, es ist wichtig zu wissen, ob alles richtig läuft und die Kunden mit die Leistungen zufrieden sind. Dafür brauchen sie für jeder Filiale die folgende Sicht:

```
create view SONDERVERW ( Ort, LeistunsNummer, KundenNummer, KundenName,
                         WarenNummer, WarenName, AnnahmeTermin, Frist,
                         Fehlerbeschreibung, Aenderungswunsch )
as select distinct (Ort, Leist#, Kunden#, KundenName, Waren#, WarenName,
            SonderleistungAnnahmeTermin, SonderleistungFrist,
            ReparaturFehlerbeschreibung, TextilaenderungAenderungswunsch )
From W-UNIV
```

Es kann auch oft vorkommen, dass die Verwaltung wissen wollte, wie oft Kunden an verschiedenen Orten Sonderleistungen beanspruchen. Dafür führe ich eine andere Sicht noch ein:

```
create view SONDER-STAT ( Ort, AnzahlLeistungen)
as select Ort, count (Leist#)
from W-UNIV
group by Ort
```

Für Einkauf –Anfragen war mein erstes Beispiel eine Sicht schon, für ein bestimmter Kunde die Einkauf-Daten anzuschauen. Im allgemeinen kann ich die folgende Sicht einführen:

```
create view EINKAUF ( KundenName, Kunden#, Bonuspunkte, Zahlart, Datum, Betrag,
                      Zahlungs#, Dauer, Zinssatz, Mindestbetrag, WarenName, Waren#,
                      Preis, Hersteller, Qualitätsklasse, Garantiefrist)
as select KundenName, Kunden#, Bonuspunkte, Zahlungsart, ZahlungsDatum,
      ZahlungsBetrag, Zahlungs#, RatenzahlungDauer,
      RatenzahlungZinssatz, RatenzahlungMindestbetrag, WarenName, Waren#, Preis,
      Hersteller, Qualitätsklasse, Garantiefrist
from W-UNIV
```

Bemerkung: Die Anwendung von Sichten ist sehr nutzbar in viele Fällen, aber es können verschiedene Probleme bei deren Änderung auftreten. Beim Einfügen (insert) , Änderung (update) oder Löschen (update) von den Relationen können nicht erlaubte Situationen vorkommen, die deshlab in SQL zurückgewiesen werden.
Deshalb muss man bei Planung und Benutzung von Sichten sehr aufmerksam sein.

Literaturverzeichnis

1. Andreas Heuer, Gunter Saake : Datenbanken: Konzepte und Sprachen, mitp 200.
2. Vorlesungsmaterial von Prof. Martin Gogolla, WS 2003/2004 Uni-Bremen
3. Jeffrey D. Ullman – Jennifer Widom: Adatbázisrendszerek (A first course in Database systems) , Prentice Hall 1998.