Skript

Infsys

SS 2008

Datenbank allgemein

- Grundlagen, Begriff
- Allgemeine Anforderungen an DBMS
- 3-Ebenen-Architektur
- Datenbankarten
- Nicht-relationale Datenmodelle

Relationale Datenbanken

- Entity Relationship Model (ERM)
- Modellierung von Datenbanken
- Normalisierung
- Relationales Datenmodell
- SQL
- Flotter Flitzer
  Dokumentation einer Datenbank für eine Autovermietung

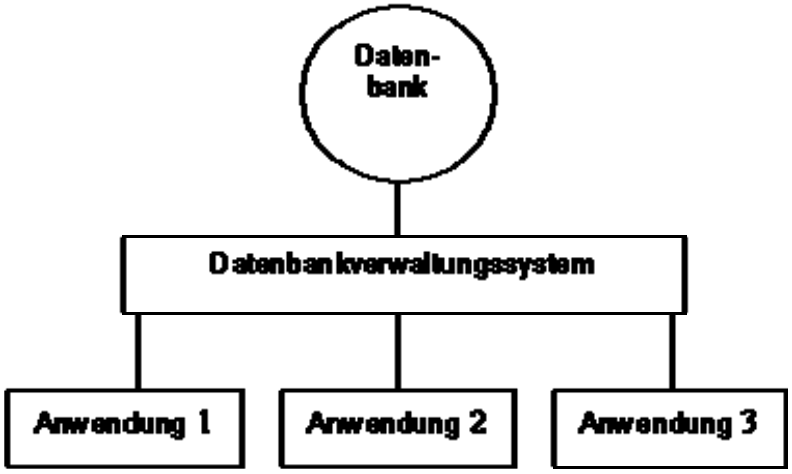
Datenbanken
Grundlagen und Begriff

Eine Datenbank umfasst eine thematisch abgegrenzte Menge von Daten.
(Texte, Zahlen, Tabellen, Bilder ...)

Aus der Organisationsform der Daten ergeben sich prinzipiell zwei verschiedene Datenbankarten, die sich bei Suchanfragen verschieden verhalten:

Information Retrieval Systems (IRS)	**Formatierte Datenbanken**
Die Daten liegen als Fließtexte vor, z. B. Urteile (JURIS)	Daten liegen als Datensätze eines bestimmten Formats vor, z. B. Warenwirtschaftssystem
⇩	⇩
Suchen durch textuelle Suchanfrage (Stichworte)	Suchen durch Objektbeschreibung
⇩	⇩
keine eindeutige Antwort (Relevanzprinzip)	eindeutiges, korrektes Ergebnis

Grundsätzlicher Aufbau eines Datenbanksystems



DBS = DB + DBMS

Ein Datenbanksystem (data base system, DBS) ist ein System zur Beschreibung, Speicherung und Wiedergewinnung von Datenmengen, die von mehreren Anwendungsprogrammen genutzt werden können.

Es setzt sich zusammen aus der Datenbank (oft data base genannt), also der Menge der Daten und einer Software, dem Datenbankverwaltungssystem (data base management system, DBMS), das die Schnittstelle zum Benutzer darstellt.

Die Datenbank enthält sowohl die reinen Nutzdaten als auch Meta-Daten (meta-data), die zur Verwaltung des gesamten Systems nötig sind.

Datenbanken

Allgemeine Forderungen an DBM

Aus den Notwendigkeiten der Datenhaltung ergeben sich allgemeine Anforderungen an ein Datenbank Management System (DBMS):

- **Große Datenbestände** (> 10^7 Datensätze) verwalten ==> erfordert effiziente Suchverfahren
- **Beliebige Verknüpfung** nach inhaltlichen Gesichtspunkten
- **Redundanzfreiheit**
- **Datenkonsistenz**
- **Zentrale Datenhaltung**
- **Gleichzeitige Benutzung** durch viele Anwender (z. B. Flugbuchung)
- **Trennung** von DB-Anwender und DB-Administratoren

Die Realisation solcher Anforderungen hat ein Ergebnis mit völlig neuer Qualität zur Folge.

- Es gibt zu Datenbanken kein materielles Äquivalent (z. B. Sammlung von vielen Karteikästen); d. h. das Modell selbst wird zu einem Objektsystem.
- Der Datenbestand hat eine potentiell unendliche Lebensdauer.

Daraus folgt, dass Techniken erforderlich sind, die vor Verlust und Verfälschung der Daten schützen.

Sicht auf das Datenmodell

Im Gegensatz zu prozeduralen Dateiverwaltungsprogrammen, die im allgemeinen Daten nur einem Anwenderprogramm zugänglich machen und deren Algorithmen beschreiben, wie ein Datensatz zu finden ist, stehen bei einem (relationalen) DBS im Vordergrund

- **Datenobjekte**
  - welche Objekte gibt es?
- **Beschreibung von Daten**
  - welche Eigenschaften haben sie?
  - in welcher Beziehung stehen sie zu einander?
- **Operationen auf Daten**
  - mit einer deskriptiven DB-Sprache (was will ich haben?)

Insgesamt ergeben sich damit hervorragende Möglichkeiten zur Abbildung realer Objekte, aber dennoch sind Datenmodelle - streng betrachtet - kein Ausschnitt der Realwelt, sondern

- die Menge der definierten Strukturmerkmale und
- der darauf möglichen Operationen.

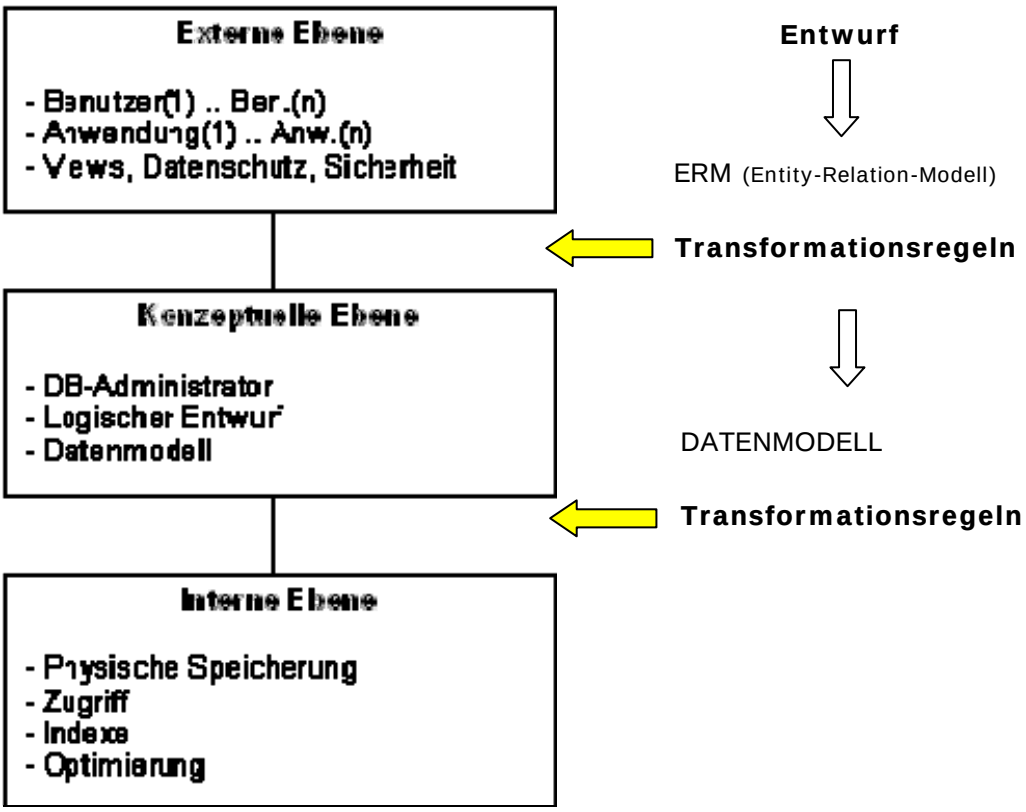
Die Nähe zur objektorientierten Modellierung ist unverkennbar. Die Modellierung mit dem Entity Relationship Model (ERM) kann als ein geistiger Wegbereiter zu OOM angesehen werden.

Datenbanken

3-Ebenen-Architektur

Die allgemein beschriebenen Aufgaben und Anforderungen lassen sich besser verdeutlichen, wenn man ein geeignetes Architekturmodell für Datenbanken hat. Die Grundidee ist die Unterteilung der DBMS in Schichten aus den verschiedenen Benutzersichten und damit verbunden klare Schnittstellen zwischen den Schichten.
Gängig ist das 3-Ebenen-Modell, das seinen Ursprung in der ANSI/SPARC-Architektur hat.

3-Ebenen-Modell



In der Literatur werden außerdem für die Ebenen die Begriffe Schicht und Schema verwendet. Eine gute ausführliche Erläuterung findet man unter Modell der 3-Ebenen-Architektur nach ANSI-Sparc

Datenbanken
Datenbanktypen

Im Laufe der Zeit wurden von Softwarefirmen und wissenschaftl. Instituten eine Vielzahl von Datenbanksystemen entwickelt, denen unterschiedliche Anforderungen und Konzepte zugrunde liegen. In diesem Kurs sollen nach ihrem Betriebskonzept sog. *Stand-alone-Datenbanken*, *File-Share-Datenbanken* und *Client/Sever-Datenbanken* unterschieden werden. Dabei liegt der Schwerpunkt des Kurses auf dem relationalen Strukturierungskonzept das als Konzept zur Datenstrukturierung eine überragende Bedeutung erlangt hat.

Stand-alone-Datenbank

Sie stellt die simpelste Form einer Datenbank dar. In der Regel handelt es sich um eine lokale Datenbank, die sich weder um Mehrfachzugriffsprobleme, wie das, dass zwei Benutzer gleichzeitig versuchen denselben Datensatz zu ändern, noch um ausgefeilte Zugangsberechtigungen für unterschiedliche Programme kümmert. Sie ist lediglich dazu da von einem Benutzer über das immer gleiche Programm angesteuert zu werden. Typische Beispiele dafür sind Adresskarteien, elektronische Telefonbücher, Buchhaltungsprogramme etc. die mit dBase, Access, Filemaker, FoxPro, Paradox oder ähnlichen Programmen erstellt wurden und ihrem Benutzer auf einfache Weise einen mehr oder weniger guten elektronischen Karteikasten zur Verfügung stellen. Mit „echten Datenbanken“ haben diese Implementierungen oft weniger zu tun als ein „Trabi“ mit einem „Rolls Royce“.

File-Share-Datenbank

Im Gegensatz zu einer Stand-alone-Datenbank können auf eine File-Share-Datenbank innerhalb eines Netzwerkes mehreren Benutzer gleichzeitig auf denselben Datenbestand zugreifen. Innerhalb eines Netzwerkes wird der jeweilige Datenbestand „quasi als Datei“ an einer Stelle allen berechtigten Nutzern zur Verfügung gestellt, die von ihren jeweiligen Workstations aus, auf diese Datenbank zugreifen können. Der Zugriff auf die Datenbank erfolgt dabei über ein spezielles „database engine“ genanntes Programm-Modul, das auf der jeweiligen Workstation ausgeführt werden muss. (Die Datenbank liegt wohl auf dem Server, jedoch hat dieser bezüglich der Verwaltung dieser Datenbank keine eigene Intelligenz. Die gesamte Intelligenz liegt separat auf jeder einzelnen Workstation.)

Der Hauptvorteil solcher Datenbanken ist, dass damit die redundante Datenhaltung überwunden wird und Datenänderungen redundanzfrei sofort allen Benutzern zur Verfügung stehen.

Probleme gibt es allerdings dann, wenn viele simultane und vor allem ändernde Zugriffe auf dem Datenbestand erfolgen (z.B. Auftragserfassung) oder häufig umfangreiche Auswertungen erstellt werden müssen. Da bei diesem Konzept jede Anwendung die Datenbank letztlich in seinem lokalen Arbeitsspeicher ggf. Satz für Satz betrachten und auswerten muss, führt dies zu einem umfangreichen Datenverkehr. Bei konkurrierenden Zugriffen lassen sich dabei nur sehr ineffiziente Mechanismen zur Wahrung der operationalen Integrität realisieren, so dass letztlich immer recht große Datenbereiche solange gesperrt werden müssen, bis der lokale Client endlich zu Potte gekommen ist.

Client/Server-Datenbank

Solche Datenbanken sind die „S-Klasse“ und genügen höchsten Ansprüchen. (Das gilt durchaus auch für die Preise.)

Der Kern einer solchen Datenbank ist ein sog. **Datenbankserver** (Softwaresystem), der auf einer dedizierten Maschine innerhalb eines Netzwerkes läuft. Dieser Server kapselt die gesamte Datenbank und bietet im gesamten Netzwerk seine Dienste an. Programme, für welche der Server etwas tun soll, (diese werden **Clients** genannt,) greifen nicht wie bei einer File-Share-Datenbank selbst auf die Daten zu, sondern wenden sich mit Ihren Wünschen lediglich an den Server, der alles für die Clients erledigt.

Die Datenverwaltung ist hier vom Zugriff konsequent getrennt. Die Clients wenden sich lediglich an die standardisierte Schnittstelle des Servers. Wo und wie der Server seine Daten verwaltet ist für die Clients transparent.

Client/Server-Datenbanken bieten viele Sicherheits-, Leistungs- und Flexibilitätsvorteile, erfordern allerdings auch die Betreuung durch einen Datenbankadministrator. Die Produkte aus diesem Bereich haben Namen wie *Informix, Oracle, DB2, MS-SQL-Server, InterBase, MySQL.*

Einige Begriffe und Abkürzungen:

Data Dictionary
Bestandteil einer Datenbank, der alle Informationen über Struktur und Aufbau einer Datenbank enthält. (Wesentliche Informationsquelle für den Datenbankadministrator und InfoBasis für zugreifende Clients.)

Datenbankadministrator (DBA)
Gewährleistet den laufenden Betrieb einer DB. Er überwacht und betreut den täglichen Betrieb einer DB, erteilt Zugriffsberechtigungen, gewährleistet die Datensicherung und veranlaßt und überwacht Systemänderungen.

DBMS
Data **B**ase **M**anagement **S**ystem

ODBC
Open **D**ata**B**ase **C**onnectivity – Standardisierte Softwareschnittstelle von Microsoft, über die Anwendungsprogramme auf unterschiedlichen Datenbanken zugreifen können, (insbesondere bei sog. Client-Server-Applikationen)

Redundanz
ist in einem Datenbestand genau dann vorhanden, wenn ein Teil der Daten ohne Informationsverlust weggelassen werden kann.

SQL
Structured **Q**uery **L**anguage – Standardisierte, systemunabhängige Sprache zur Erstellung, Manipulation und Abfrage von (relationalen) Datenbanken.

Transaktion
Datenbankoperation, die vollständig ausgeführt werden muss, damit die Integrität einer Datenbank nicht gefährdet ist. (Z.B. Registrierung eines Kundenauftrages in einem Warenwirtschaftssystem einschl. Bestandsberichtigung und Rückstandserfassung und Aktualisierung des Kundensaldos.)

Datenbanken

Datenmodelle

Logische Datenmodelle jenseits des Relationenmodells

Referat von
Norman Niemer

1. Hierarchisches Datenmodell

- Gliederung als typisches Beispiel für hierarchisches Datenmodell

- einfachste Form eines Datensystems

- in der "Informatikersprache" auch Bäume (Trees) genannt

- natürliche Hierarchien: Personaldatei (Firma/Abteilung/Mitarbeiter), künstliche Hierarchien: Lieferantenkartei (Artikel/Lieferant/Adresse)

- Arten von Hierarchien:

  - einstufig: genau einem Vaterelement werden ein/mehrere Söhne zugeordnet
  - mehrstufig:
    - Zusammenfügen von mehreren Hierarchien
    - aber: jedes Element darf nur in einer Gruppe Sohnelement sein
    - das Wurzelelement ist selber nirgends Sohnelement (keine Zirkelbezüge)

- Beziehungen:

1:1 - Vater wird Sohn zuordnet

1 : n - Vater werden mehrere Söhne zugeordnet

m : n - nicht darstellbar

- wichtig zum Finden, Ändern, Hinzufügen und Löschen von Daten ist das sequenzielle Auslesen:
  Söhne kommen nach ihrem Vater
  Söhne kommen vor den Brüdern

- programmiertechnisch erfolgt die Zuordnung nicht über Tabellen wie beim Relationenmodell, sondern direkt mit Pointern

- Vorteile: einfaches, effizientes Datenmodell (kommt daher in Directory Servern, Webservern und im Index von z.B. mySQL zur Anwendung)

- Nachteile: unflexibel und bei komplexen Umgebungen schwierig zu modellieren

2. Netzwerkmodell

- baut auf dem hierarchischen Modell auf

- Restriktion, dass Entität nur Mitglied in einer Gruppe sein kann, ist aufgehoben -> mehrere Wurzelelemente möglich

- Modellierung von m:n-Beziehungen möglich

- da es unpraktikabel bzw. nicht realisierbar ist, die m : n-Beziehung direkt zu modellieren, geht man einen Umweg über 2 1:m-Beziehungen (Darstellung: Autor/Beitrag/Buch)

- Zugriff auf die Entitäten erfolgt durch sequenzielles Auslesen der einzelnen Hierarchien

- Vorteile:
  komplexere Umgebungen lassen sich modellieren
  vermaschte Strukturen sind möglich
  m:n-Beziehung darstellbar

- Nachteile:
  Verlust von Einfachheit und Übersichtlichkeit
  sequenzielles Auslesen komplizierter
  ineffizienter und demzufolge langsamer

3. Objektorientiertes Datenmodell

3.1. ODBMS vs. RDBMS

- Problempunkte relationaler Datenbanken:

- Sehr komplexe Objekte und Umgebungen (z.B. CIM, Geoinformationssysteme) lassen sich nur schwer modellieren,
  bzw. müssen auf mehreren Tabellen abgebildet werden, wodurch
  - die Übersichtlichkeit verloren geht
  - Integrität lässt sich schlechter überwachen
  - die Leistung des Datenbanksystems reduziert wird
- Inkompatibilität zwischen Datenbank und Anwendungsentwicklung/Programmiersprache
- Es werden nur Datenattribute, aber keine Operationen abgebildet
- Es gibt keine echte Objektidentität (Identifikation über Schlüssel, die nur das System kennt)

=> Entstehung der objektorientierten Datenbanksysteme

3.2. Anforderungen an ODBMS

- herausgearbeitet von renommierten Forschern -> "The Object-Oriented Database System Manifesto"

- Kernaussage: ein DBMS, erweitert um folgende Konzepte der Objektorientierung

- Unterstützung für komplexe Objekte (Tupels, Listen, Arrays)
- Objektidentität
- Einkapselung/Encapsulation ("Geheimnisprinzip" - private/public)
- Typen und Klassen
- Hierarchien von Typen und Klassen (z.B. "Aggregation")
- Überladen
- Vererbung

- Daneben soll es auch eine vollständige Datenbank-Programmiersprache geben

3.3. ODMG-Standard

- Mitte 1991 taten sich führende Hersteller von ODBMS zusammen um einen Standard für objektorientierte Datenbanksysteme zu entwickeln

- 1993 wurde die ODMG Standard Spezifikation 1.0 veröffentlicht (3.0 ist die neueste Version)

- Bestandteile des Standards:

- **Objektmodell**: Der ODMG-Standard definiert ein gemeinsames Objektmodell für ODBMSe, bestehend aus objektorientierten Modellierungsmitteln und Datenbankelementen.
- **Object Definition Language** (ODL): Die ODL als Objekt-Definitionssprache liefert die Syntax für das Objektmodell. Die ODL kann zur Definition eines Datenbankschemas verwandt werden.
- **Object Query Language** (OQL): Die OQL dient zur Bildung von assoziativen Anfragen auf Datenbankobjekten. Die OQL, die sich an den Anfragemöglichkeiten von SQL orientiert, kann sowohl als eigene ODBMS-Schnittstelle als auch innerhalb der Sprachanbindungen verwendet werden.
- **Sprachanbindungen für C++ und Smalltalk**: Die Sprachanbindungen sind die Umsetzung des Objektmodells auf eine Programmiersprache und dienen als Datenbankschnittstelle für ODBMS-Applikationen. Die Sprachschnittstellen liefern die Syntax und Semantik zur Definition und Manipulation der (Datenbank-) Objekte. Der Standard umfasst derzeit eine C++-Schnittstelle, eine Java-Schnittstelle und eine Smalltalk-Schnittstelle

Quelle: [1], S. 20f

Damit ist auch automatisch Interoperabilität geschaffen: auf Objekte, die mit der Sprache "A" erzeugt wurden, kann auch mit allen anderen Sprache OD-Sprachen zugegriffen werden.

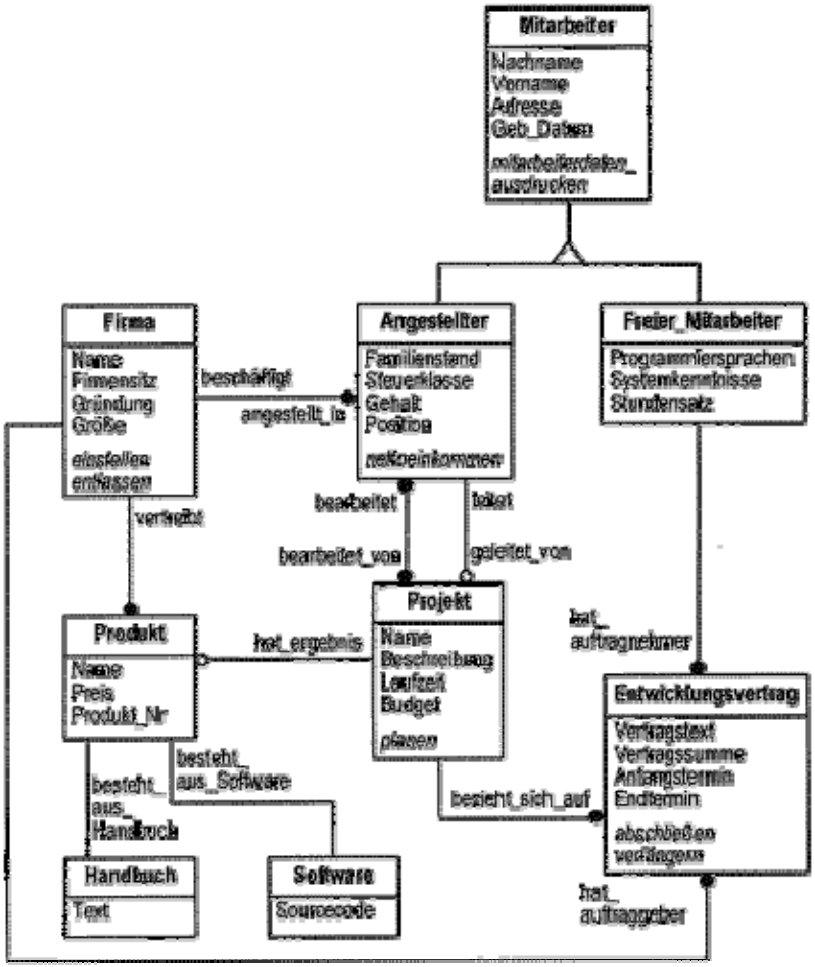
3.4. Das Objektmodell

- C. Beeri entwickelte (ca. 1989) ein theoretisch fundiertes objektorientiertes Datenbankmodell

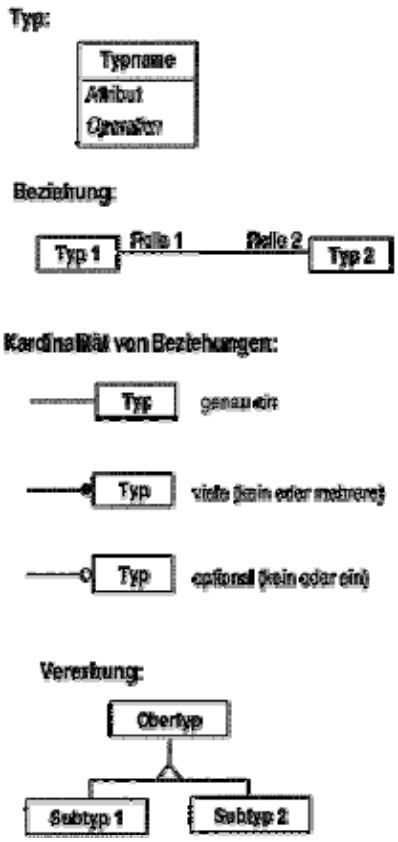
- dies beinhaltete im Großen und Ganzen die o.g. Kriterien des ODBMS-Manifestes

- Objektmodell: Datenmodell nach objektorientierten Konzepten => konkrete Modellierung Datenbankschema

Objektorientiertes Datenbankschema:



Grafische Notation für ein objektorientiertes Datenbankschema:



Quelle: [1], S. 231 f

- im Relationenmodell erfolgt das Speichern der Daten in Tabellen,
  im ODBMS: Speichern der Daten in Objekten
- Attribute, Beziehungen, Schlüssel ebenfalls zu finden
- starke Orientierung an OOP
- Definition des Objektmodells erfolgt mit der ODL - die konkrete Implementation der Objekttypen, Methoden und Abfragen erfolgt in der jeweiligen Programmiersprache

Definition des Objekttyps Firma:

```
interface Firma

( // Typeigenschaften
    extent Firmen
    key (Name, Firmensitz)
)
{ // Instanzeigenschaften

    //Attribute
    attribute String Name;
    attribute Adresse Firmensitz;
    attribute Date Gruendung;
    attribute Short Grosse;

    // Beziehungen
    relationship Set<Produkt> vertreibt;
    relationship Set<Angestellter> beschäftigt inverse angestellt_in;

    // Instanzoperationen
    Boolean einstellen (in Angestellter, in Projekt);
    Boolean entlassen (in Angestellter);
}
```

Quelle: [1] S. 24

3.4.2. Attribute

- haben einen bestimmten Datentyp, der entweder ein Systemdatentyp oder ein Benutzerdatentyp sein kann

- Mengen, Listen und Arrays sind ebenfalls möglich

- können als private oder public deklariert werden

3.4.3. Beziehungen

- die Beziehungen, wie es sie in den Datenmodellen gibt, existieren in objektorientierten Programmiersprachen nicht
  => eigenständiges Konzept

- alle Arten von Beziehungen lassen sich im Objektmodell realisieren

  - 1:1 - Angestellter leitet/geleitet von Projekt
  - 1:n - Firma beschäftigt/angestellt in Angestellten
  - m:n - Angestellter bearbeitet/bearbeitet von Projekt
  - unidirektional - Firma vertreibt Produkt
  - bidirektional - Firma beschäftigt/angestellt in Angestellten
    (muss im verbundenen Objekttyp analog sein - Wahrung der referentiellen Integrität)

- Beziehungen können nur zwei Objekttypen involvieren und keine Attribute besitzen
  => zur Lösung muss ein neuer Objekttyp erstellt werden

3.4.4. Schlüssel

- das Schlüsselkonzept der RDBMS findet sich auch in ODBMS wieder

- dient weniger der Objektidentifizierung, sondern zur Sicherung der Eindeutigkeit bestimmter Attributkombinationen

3.4.5. Methoden

- das objektorientierte Paradigma legt Wert auf strikte Trennung von Speicherung von Daten und Manipulation der Daten

- Konzept auch im ODBMS

- da die Implementation in der Programmiersprache erfolgt ergibt sich ein besonderer Vorteil: Möglichkeit der Nutzung von speziellen Programmieralgorithmen

3.4.6. Vererbung

- identische Verwendung wie in OOP, es ist sogar Polymorphie möglich

- großer Vorteil gegenüber RDBMS

3.4.7. Objektidentität

- wichtige Eigenschaft von ODBMS (wird sowohl im Manifest, als auch im ODMG-Standard gefordert)

- jedem Objekt wird eine ID zugeordnet, die das Objekt eindeutig identifizierbar macht

- Vorteile gegenüber RDBMS

  - die ID ist unabhängig von den Eigenschaften (das Kennzeichen als eindeutiger Primärschlüssel eines Auto-Datentyps ändert sich u.U. doch)
  - die IDs werden automatisch vom DBMS vergeben - die nötige Umgang mit Schlüsseln im RDBMS entfällt

3.5. Datenbankkonzepte

- Persistenz
- Berechnungsvollständigkeit
- Transaktions- und Zugriffskontrolle
- u.a.

4. Glossar

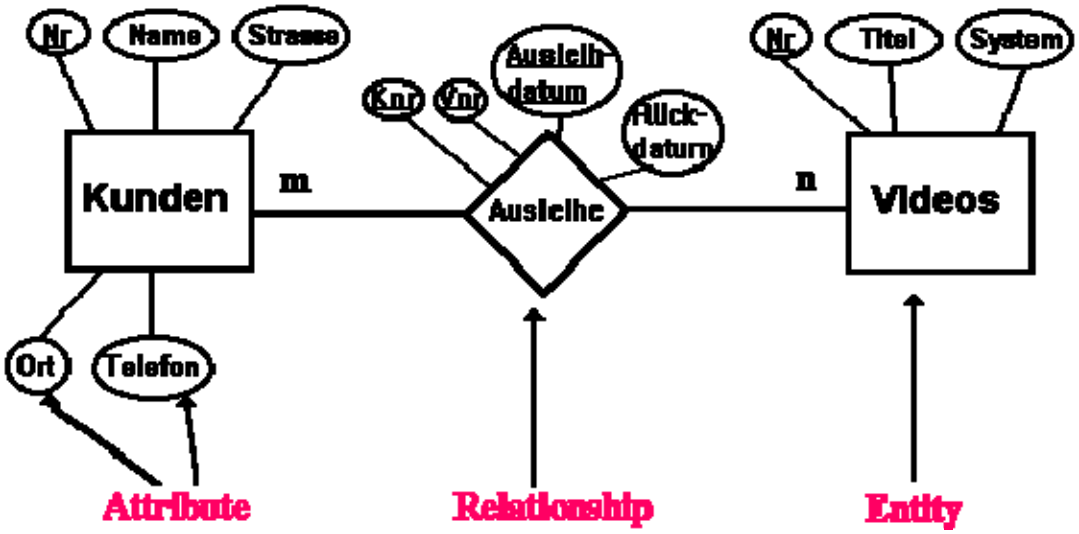
CIM	computer integrated manufacturing
ODBMS	object database management system
ODMG	object database management group - Forum für Standards der ODMBS
OOP	object oriented programming
OOPL	object oriented programming language
RDBMS	relational database management system

Relationale Datenbanken

Entity Relationship Model

- Entity-Relationship-Model
- Entität (Entity)
- Entitätstyp
- Entität und Entitätstyp
- Attribut
- Primärschlüssel
- Fremdschlüssel
- Beziehung (Relationship)
- Konnektivität (Kardinalität)
- 1 : 1 - Beziehung
- 1 : 1 - Beziehung (mehrstellig)
- 1 : n - Beziehung
- m : n - Beziehung

Datenbanken
Entity-Relationship-Model



Alle Primärschlüssel werden unterstrichen.
Die Fremdschlüssel werden bei den Relationship in der Regel nicht aufgeführt.

Entity Relationship Model
Entity

Eine Entität ist ein Ding, welches eindeutig identifizierbar ist. [Chen]

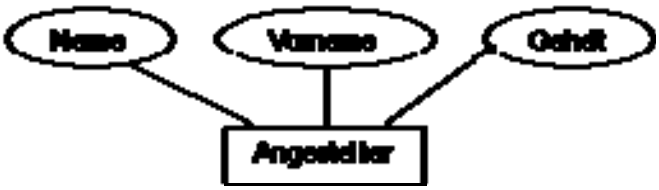
- Entität erscheint wie bei OOP zwitterhaft: als reales Welt-Objekt und/oder als DV-Objekt
- Wie eine Entität modelliert wird, steht per se nicht fest. Es wird durch eine Entwurfsentscheidung festgelegt.
- Jede Entität ist singulär, also ein Exemplar, das durch den Wert seiner Attribute bestimmt wird;
  d. h. es existiert ein eindeutiger Schlüssel, durch den es identifiziert wird.

Entity Relationship Model
Entitätstyp

Ein Entitätstyp definiert eine Klasse gleichartiger Entitäten.

- Im ERM spezifiziert der Entitätstyp eine Menge von Entitäten (Exemplaren, Instanzen) mit gleichen Attributen.
- Zu jedem Entitätstyp kann es beliebig viele Entitäten geben.
- Modelle enthalten nur Entitätstypen
- Entitätstypen repräsentieren die allgemeine Datenbeschreibung
  Entitäten repräsentieren den Datenbestand
- Entitätstypen beschreiben keine Dateien; d. h.
  - damit ist keine Zugriffsorganisation festgelegt,
  - keine (hierarchische) Ordnung der Entitäten definiert.

Entity Relationship Model
Entität und Entitätstyp



Hansen	Horst	4.740
Pirelli	Elli	5.900
Ratlos	Rudi	3.620
....		

Entitätstyp

beschreibt im Modell die Sicht auf die Mitarbeiter des Betriebs.

Entitäten

Die Werte der Attribute enthalten die aktuellen Daten eines jeden Mitarbeiters.

(Zur Veranschaulichung werden manchmal Attributwerte in der Darstellung des Entitätstyps eingetragen!)

Entity Relationship Model
Attribute

Ein Attribut ist die Beschreibung einer bestimmten Eigenschaft der Entitäten einer Entitätsmenge.

- Jedes Attribut einer Entität erhält einen Namen
- Namen sind in einer Entität eindeutig - Identifizierung von Attributen nur über Namen, nicht über Definitionsreihenfolge
- Gleiche Attribut-Namen sind in verschiedenen Entitäten möglich - eindeutig durch: Entitätsname**.**Attributname
- Datentyp und Wertebereich

  Im ERM sind als Attribute auch komplexe Daten möglich, wie z. B. Adresse. Da jedoch das relationale Modell nur "flache" Attribute zulässt, sollten im ERM vorzugsweise auch nur einfache Datentypen genommen werden.

Entity Relationship Model
Identifikationsschlüssel

Der Schlüssel (Primärschlüssel, primary key) dient zur Identifikation einer Entität, der ihn eindeutig von den anderen Entitäten desselben Entitätstyps unterscheidet und der sich während der Lebensdauer nicht ändert.

Jeder Schlüssel besteht aus einem oder mehreren Attributen (Schlüsselattributen). Die restlichen heißen Nicht-Schlüssel-Attribute.

Als Primärschlüssel kann gewählt werden:

- **Natürlicher Schlüssel**,
  ein "normales" Attribut, welches die Entität mitbringt, z.B.

  PKW (Fahrgestellnummer, Modell, Erstzulassung, Datum der Abmeldung),
  da die Nummer einmalig sein muss.

- **Künstlicher Schlüssel**
  Den "natürlichen" Attributen wird bei der Modellierung ein zusätzliches - meist numerisches -Attribut hinzugefügt, das die Einmaligkeit sichert, z. B.

  KUNDE (<u>Kundenummer</u>, Name, Vorname, Plz, Ort, Strasse, Hausnummer)

"Ein Primärschlüssel kann auch aus der Kombination mehrerer Attribute bestehen, wenn diese gemeinsam jeden Satz eindeutig identifizieren. Dies ist dann der so genannte Kombinierte Primärschlüssel oder auch Verbundschlüssel. Hier kann eine Wertekombination aller am Schlüssel beteiligten Attribute nur einmal vorkommen." (http://www.at-mix.de/primaerschluessel.htm), z. B.

PERSON (<u>Name</u>, <u>Vorname</u>, <u>Geburtstag</u>, Strasse, HausNr, TelefonNr)

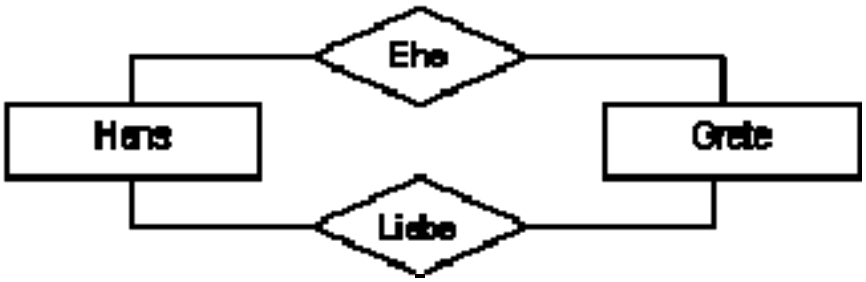
Beziehungen zwischen Tabellen werden dadurch hergestellt, dass der Primärschlüssel einer Relation als Fremdschlüssel in der Relation eingetragen wird zu der eine Beziehung besteht.

Entity Relationship Model

Beziehungen

Eine Beziehung (Relationship) ist die Zuordnung zweier oder mehrerer Entitäten
unter einem bestimmten Aspekt.

- Die Beziehung drückt eine Funktion aus.
- Sie wird üblicherweise von links nach rechts gelesen und kann durch ein Substantiv oder ein Verb beschrieben werden...



- Dieselben Entitäten können mehrere, verschiedene Beziehungen haben.
- **Ob ein "Welt"-Sachverhalt Entität oder Beziehung ist, steht per se nicht fest. Es wird durch eine angemessene Entwurfsentscheidung bei der Modellierung festgelegt.**

Entity Relationship Model

Konnektivität (Kardinalität)

Die Konnektivität (Assoziation E 1, E 2) legt fest, wie viele Entitäten aus E2 einer Entität aus E1 zugeordnet werden können.

- Dabei werden drei Grundtypen unterschieden:
  - **1 : 1**
  - **1 : n**
  - **m : n**
- In dieser Schreibweise gibt die Konnektivität nur an, wie viele Entitäten **maximal** miteinander verbunden sein können.

Andere Notationen sind:
1 : genau eine (1)
c : konditionell, keine oder eine (0,1)
m : multiple (1,*)
mc : (0,1,*)

- Methode der Festlegung und Notation:

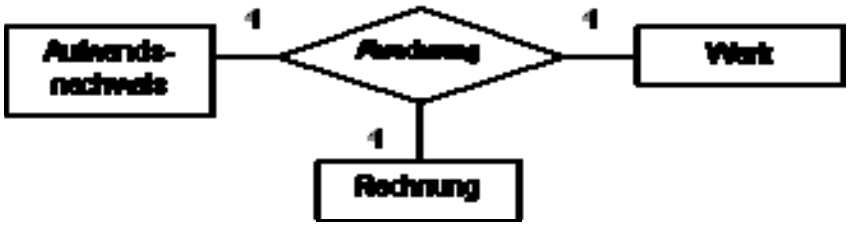
  Die Konnektivität wird jeweils am gegenüber liegenden Entitätstyp notiert,
  z. B. 1 Entität aus E2 kann nur mit genau 1 aus E1 Beziehungen haben.

Die Konnektivität gibt für jeden Entitätstyp an, zu wie viel anderen Entitäten jede einzelne Entität Beziehungen haben kann.

Die 1 : 1 - Beziehung

- 1 Mann steht exklusiv zu 1 einer Frau in Beziehung oder
- 1 Frau steht exklusiv zu 1 Mann in Beziehung

Die 1 : 1 - Beziehung (mehrstellig)



- 1 Aufwandsnachweise gilt für 1 Rechnung in 1 bestimmten Werk
- Zu 1 Rechnung gibt es pro Werk 1 Aufwandsnachweis
- Jedes Werk hat zu 1 Rechnung 1 Aufwandsnachweis.

Die 1 : n - Beziehung

- 1 Abteilung beschäftigt mehrere Mitarbeiter
- 1 Mitarbeiter ist exklusiv in 1 Abteilung beschäftigt

Die m : n - Beziehung

- 1 Projekt kann mehrere Mitarbeiter haben
- 1 Mitarbeiter kann an mehreren Projekten mitarbeiten

Modellierung von Datenbanken

Datenbank-Modellierung

Allgemeines Material

- Arbeitsschritte zum Erstellen von Datenbankanwendungen
- Gliederung der Anforderungsdefinition
- Entity-Relationship-Model
- Abbildungsregeln (einfach)
- Abbildungsregeln (ausführlich)
- Funktionale Abhängigkeit vom Schlüssel
- Integritätsbedingungen
- Normalisierung
- Sichten und Zugriffsrechte (Überblick)
- Zugriffsrechte SQL

Beispieldatenbank "Flotter Flitzer"

- Anforderungsdefinition
- Entity-Relationship-Model
- Anwendung der Abbildungsregeln
- Integritätsbedingungen
- Zugangsberechtigungen
- SQL Script zum Erzeugen des Datenmodells (Informix)
- SQL Script zum Erzeugen des Datenmodells (MySql)
- SQL-Script zur Vergabe der Zugriffsrechte (MySql)
- SQL-Script zur Vergabe der Zugriffsrechte (Informix)

Arbeitsschritte zum Erstellen einer Datenbankanwendung

Im Folgenden werden die wesentlichen Schritte zur Erstellung einer Datenbankanwendung dargestellt. Die Beschreibung gilt für alle Datenbankmaschinen. Die Dokumentation der Verwaltung der Autovermietung Flotter Flitzer kann als Beispiel dienen. Die Tätigkeiten sind nach dem Schichtenmodell getrennt aufgeführt.

Modell

Nr.	Tätigkeit	Dokument
1	Anforderungsdefinition schreiben	Anforderungsdefinition
2	Datenanalyse und Datenmodellierung mit Hilfe des Entity-Relationship-Modells (ERM)	Entity-Relationsship-Model
3	Überführung des ERM in Tabellen anhand der Abbildungsregeln	Verzeichnis der Tabellen
4	Überprüfung der Tabellen mit Hilfe der Normalisierung	
5	Festlegen der Integritätsbedingungen	Aufstellung der Integritätsbedingungen
6	Erzeugen der Datenbank und der entsprechenden Tabellen	Kommentiertes SQL-Script
7	Festlegen der Sichten und der jeweiligen Zugangsberechtigungen	Verzeichnis der Zugangsberechtigungen
8	Erstellen der Sichten und Zugangsberechtigungen	Kommentiertes SQL-Script
9	Erzeugen von Indices für jeden Schlüssel bzw. zur Beschleunigung des Ablaufs (nur bei mehr als 200 Datensätzen)	Kommentiertes SQL-Script

Mit diesen Schritten ist die Datenbasis der Anwendung mit den Elementaroperationen (select, insert, modify und delete) fertig, allerdings ohne jeglichen Komfort. Die Ein- und Ausgabe von Daten kann nur auf der Promptebene mit SQL erfolgen.
Weiterhin fehlt die Sicht der Funktionen. Um die Datenbank über das Internet bedienen zu können und um die Intergritätsbedingungen einzubauen sind folgende Schritte möglich:

View

Nr.	**Tätigkeit**	**Dokument**
1	Anforderungsdefinition für die Website in der die Datenbank eingebunden werden soll erstellen oder anpassen.	Anforderungsdefinition
2	Festlegen der Navigationsstruktur für die Website	Zeichnung der Navigationsstruktur
3	Erstellen eines Prototyps mit Platzhaltern für die Tabellen	
4	Erstellen von Webseiten zur Datenpflege in HTML.	Beispiel: HTML-Seiten zur Eingabe
5	Skripte erzeugen (z.B. in PHP, Pearl usw.), um die Verbindung der HTML-Seite mit der Datenbank herzustellen.	Beispiel: Skript zum Ausgeben einer Kundenliste
6	Testen der Anwendung	Testprotokoll
7	Datenbankanwendung öffentlich zugänglich machen	Logfiles

Gliederung der Anforderungsdefinition

Aufgabenstellung

Beschreibung von Zweck und Zusammenhang des künftigen Produkts

Anforderungen an die Benutzerinnen und Benutzer

Erforderliche Vorkenntnisse, die zum Umgang mit dem Produkt vorausgesetzt werden

Anforderungen an die Arbeitsweise

Eingliederung des zu erarbeitenden Softwareprodukts in bestehende Betriebs- und Arbeitsabläufe

Anforderungen an den Leistungsumfang

Kurzdarstellung der durch das Produkt zur Verfügung gestellten Leistungen, aufgeschlüsselt nach Benutzungsgruppen

Anforderungen an mögliche Ausbaustufen

Aufstellung denkbarer, zukünftiger Erweiterungen

Anforderungen an das Fehlerverhalten

Aussagen zum Verhalten des Produkts bei Fehlbedienung, insbesondere der möglichen Konsequenzen für das Umfeld (Datenverlust, Systemstillstand etc.)

Anforderungen an die Qualität

Angaben zur Qualität des Produkts, z.B. zum Aufwand für die Portierbarkeit und zur Implementation von Erweiterungen

Anforderungen an den Datenschutz

Beschreibung der zu treffenden Maßnahmen zur Sicherstellung des Datenschutzes sowohl rechtlich als auch aus der Sicht des Auftraggebers bzw. der Auftraggeberin.

Anforderungen an die Ergonomie

Darstellung der Anforderungen an die Benutzerschnittstelle

Anforderungen an die Dokumentation

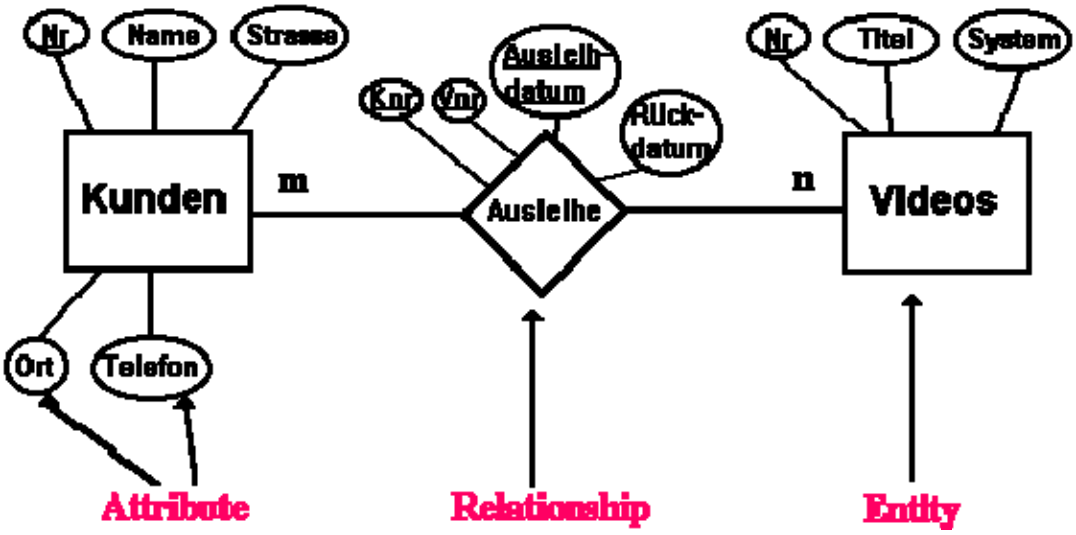
Aussagen zum Umfang der zum Produkt gehörenden Dokumentation

Anforderungen an die Basismaschine

Voraussetzungen, die von einer Maschine (einschließlich Software) zur Verwendung des Produkts erfüllt sein müssen

Datenbanken

Entity-Relationship-Model



Alle Primärschlüssel werden unterstrichen.
Die Fremdschlüssel werden bei den Relationship in der Regel nicht aufgeführt.

Transformation von ERM in das relationale Modell

Abbildungsregeln (einfach)

Die Transformation des ER-Modells in Relationen ist ein mehr technischer Prozess, der nach festen Regeln abläuft und der heutzutage auch bereits von Werkzeugen übernommen werden kann.

Transformationsregeln (Abbildungsregeln)

1. Regel:

Jeder Entitätstyp wird als Tabelle dargestellt.

2. Regel

Jede n : m - Beziehung wird durch eine eigene Tabelle dargestellt.

Die Beziehung wird dadurch hergestellt, dass die Primärschlüssel von E1 und E2 als Fremdschlüsselattribute in der Beziehungsrelation aufgenommen werden.

Beispiel: Einem Projekt gehören mehrere Mitarbeiter an, die auch in mehreren verschiedenen Projekten mitarbeiten.

PERSONAL (P-Nr, Name)
PROJEKT (Pro-Nr, Bezeichung, Termin)
ZUGEHÖRIGKEIT (P-Nr, Pro-Nr)

3. Regel

Jede 1:n und 1:1 - Beziehung, die eigene Attribute hat, wird durch eine eigene Tabelle dargestellt.

4 . Regel

Hat eine 1:n - Beziehung keine eigenen Attribute dann gilt:
Ist eine Entität aus E2 *zwingendes* Mitglied (d. h. genau 1) einer 1 : n - Beziehung mit einer Entität aus E1, dann erhält E2 den primären Schlüssel von E1 als Attribut.

Sofern eine Entität aus E2 *freies* Mitglied (d. h. 0,1 - konditionell) der 1 : n - Beziehung mit einer Entität aus E1 ist, wird diese Beziehung gewöhnlich in einer eigenen Relation dargestellt.
Diese Bedingungen gelten für 1:1 und 1: n - Beziehungen.

Beispiele für Regel 4

Zwingendes Mitglied am Beispiel Abteilungszuordnung:

Eine Mitarbeiterin gehört zwingend genau einer Abteilung an und eine Abteilung hat mehrere Mitarbeiterinnen.

Da hier eine Entität (Mitarbeiterin) von E2 zwingendes Mitglied einer Entität aus E1 (Abteilung ist), wird die Beziehung in den Tabellen dadurch hergestellt, in dem der Primärschlüssel von E1 (A-Nr) in E2 als Fremdschlüsselattribut aufgenommen wird.

ABTEILUNG (A-Nr, A-Name**)**
PERSONAL (P-Nr, Name, A-Nr)

Freigestelltes Mitglied am Beispiel Betriebssportgemeinschaft:

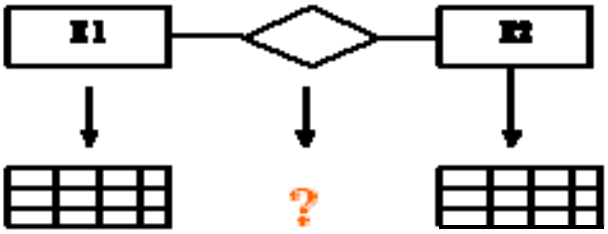
Eine Mitarbeiterin kann muss aber nicht Mitglied genau einer Betriebssportgemeinschaft sein.

Da hier eine Entität aus E2 (Mitarbeiterin) freies Mitglied der Entität E1 (BSG) ist, wird die Beziehung in einer eigenen Tabelle dargestellt und die Primärschlüssel von E1 und E2 werden als Fremdschlüssel eingetragen.

BSG (BSG-Nr, BSG-Name**)**
PERSONAL (P-Nr, Name)
MITGLIED (Mitgliedsnummer, BSG-Nr, P-Nr)

Transformation von ERM in das relationale Modell (ausführlich)

Die Transformation des ER-Modells in Relationen ist ein mehr technischer Prozess, der nach festen Regeln abläuft und der heutzutage auch bereits von Werkzeugen übernommen werden kann. Bei den Entitätstypen ist es klar - sie werden in Relationen (Tabellen) überführt. Bei den Beziehungsmengen (Relationship) sind einige Besonderheiten in Abhängigkeit von der Konnektivität zu beachten.



Transformationsregeln

1. Regel:

Jeder Entitätstyp wird als Tabelle dargestellt.

2. Regel

Jede n : m - Beziehung wird durch eine eigene Tabelle dargestellt.

Die Beziehung wird dadurch hergestellt, dass die Primärschlüssel von E1 und E2 als Fremdschlüsselattribute in der Beziehungsrelation aufgenommen werden.

Beispiel: Einem Projekt gehören mehrere Mitarbeiter an, die auch in mehreren verschiedenen Projekten mitarbeiten.

PERSONAL (**P-Nr**, Name)
PROJEKT (**Pro-Nr**, Bezeichung, Termin)
ZUGEHÖRIGKEIT (**P-Nr**, **Pro-Nr)**

3. Regel

Jede Beziehung, die eigene Attribute hat, wird durch eine eigene Tabelle dargestellt.

4 . Regel

Hat eine 1:n - Beziehung keine eigenen Attribute dann gilt:
Ist eine Entität aus E2 *zwingendes* Mitglied (d. h. genau 1) einer 1 : n - Beziehung mit einer Entität aus E1, dann erhält E2 den primären Schlüssel von E1 als Attribut.

Sofern eine Entität aus E2 *freies* Mitglied (d. h. 0,1 - konditionell) der 1 : n - Beziehung mit einer Entität aus E1 ist, wird diese Beziehung gewöhnlich in einer eigenen Relation dargestellt.
Diese Bedingungen gelten für 1:1 und 1: n - Beziehungen.

Wann nimmt man 2 Tabellen - wann drei?

Beispiel zwei Tabellen:

Abteilungszuordnung:

Mehrere Mitarbeiter gehören 1 Abteilung an.

Die Beziehung wird in den Tabellen dadurch hergestellt, in dem der Primärschlüssel von E1 (A-Nr) in E2 als Fremdschlüsselattribut aufgenommen wird.

ABTEILUNG (**A-Nr**, A-Name**)**
PERSONAL (P-Nr, Name, **A-Nr**)

Beispiel drei Tabellen:
Szenen einer Ehe

Scheinbar liegt hier eine 1 : 1 - Beziehung nach abendländischem Recht vor.

Bei einer **zwingenden** Beziehung muss in 'Eheland' Heiratspflicht bestehen und es müssten gleichgroße Entitätsmengen vorhanden sein. Wenn ein Partner - Adam - stirbt (gelöscht wird), tritt eine Löschanomalie auf, die nur dadurch geheilt werden kann, dass Eva ebenfalls gelöscht wird ('indische Ehe').
Die gleiche Problematik liegt bei 1 : n - Beziehungen vor ('muslimische Ehe').

Falsche Lösung mit Heiratspflicht und "indischen Verhältnissen".

MÄNNER

Name	Ehefrau
Adam	Eva
Ike	Tina
Udo	?
Rudi	?

FRAUEN

Name	Ehemann
Berta	?
Eva	Adam
Tina	Ike

Da bei uns nicht jeder verheiratet sein muss, also eine freie Beziehung besteht, und die Männer schon mal früher sterben als Frauen, gibt es in der Tabelle nach dem obigen Modell verbotene leere Felder (Nullstellen). Die Lösung ergibt sich durch Einführen der Beziehungstabelle EHE mit den Fremdschlüsseln MÄNNER.Name und FRAUEN.Name.

Richtige Lösung:

MÄNNER

Name
Adam
Ike
Udo
Rudi

EHE

Ehefrau	Ehemann
Eva	Adam
Tina	Ike

FRAUEN

Name
Berta
Eva
Tina

Funktionale Abhängigkeit von Schlüsseln

Funktionale Abhängigkeit vom Schlüssel heißt:

Nicht-Schlüsselattribute sind aus dem Schlüssel bestimmbar

1. Funktionale Abhängigkeit bei einem Schlüsselattribut:
   KUNDE (**K-Nr**, Firma, Plz, Ort, Strasse, HausNr)
   Kunden-Firma ist aus K-Nr bestimmbar.
2. Funktionale Abhängigkeit von zusammengesetzten Schlüsseln:
   (Bei diesem Beispiel sei unterstellt, dass genau eine PLZ einem Ort zugeordnet ist.)

   VERTRETER (**Name**, Ort, PLZ, Vorname, Umsatz)

Meier	A-Stadt	12123	Anton	50.000
Meier	K-Stadt	34517	Anton	80.000

3. Der Umsatz eines Vertreters in einer bestimmten Stadt ist vom gesamten Schlüssel abhängig

Meier	A-Stadt	50.000
Meier	K-Stadt	80.000

4. Die Postleitzahl ist nur von einem Teilschlüssel abhängig.

A-Stadt	12123
K-Stadt	34517

1.

***Voll funktional* abhängig sind die Attribute einer Relation genau dann, wenn sie von jedem Teil der Attributskombination abhängen, aber nicht bereits von Teilen der Attributskombination**

Integritätsbedinungen

Integritätsregeln können in drei Kategorien einordnet werden:

1. Regeln für die Wertebereichsintegrität (Beispiel)

"Sie umfassen den Erhalt der Korrektheit der Attributwerte innerhalb der Relationen"

Für jedes Attribut ist der zulässige Wertebereich festzulegen. Dieses Konzept entspricht der Typprüfung bei Programmiersprachen. Neben dem genauen Datentyp sollten, wenn möglich, auch Ober- und Untergrenzen für die Werte angegeben werden.

Zu dieser Integritätsprüfung gehört auch die Forderung, dass der Wertebereich für ein beliebiges Schlüsselelement nicht NULL beinhalten darf.

Methoden zur Einhaltung sind z. B. die Plausibilitätsprüfung und das Prüfziffernverfahren.

2. Regeln für die intra-relationale Integrität (Beispiel)

"Sie umfassen die Korrektheit der Beziehungen zwischen den Attributen in einer Relation und der Erhaltung der Eindeutigkeit des Schlüssels"

Im wesentlichen geht es hier um die Eindeutigkeit der Schlüssel, Indexe und z. B. fortlaufende Nummerierung. Viele relationale Datenbanksysteme erzwingen diese Eindeutigkeit nicht.

3. Regeln für die referentielle Integrität (Beispiel)

"Sie umfassen die Erhaltung der Korrektheit und Konsistenz der Beziehungen zwischen den Relationen"

Die Erhaltung der referentiellen Integrität gilt vor allem der Überprüfung von Fremdschlüsseln. So sollten z.B. in einer Bücherei keine Benutzer gelöscht werden können, die noch ein Buch ausgeliehen haben. Außerdem sollten nur Bücher an tatsächlich vorhandene Benutzer ausgeliehen werden können. Verstöße gegen die referentielle

Normalisierung

- Normalisierung - Ziele u. Verfahren
- Normalisierung – Beispiel
  - 1 Normalform (mit. Übung)
  - 2 Normalform
  - 3 Normalform
  - Lösung
- weitere Übungsaufgabe zur Normaliserung
- Redundanz
- Mutationsanomalien
- Übung Mutationsanomalien
- Übung Mutationsanomalien - Lösung

Die Einhaltung der Integritätsbedingungen bei einem RDBS erzwingt bestimmte Anforderungen an die Form der Tabellen. Der Weg dahin wird Normalisierung genannt. Am Ende der Normalisierung stehen Relationen, die einer vorgegebenen Normalform entsprechen. Je höher die gewählte Normalform (NF), desto größer die Anforderung an die innere Struktur der Relation. Kurz gesagt bezweckt die Normalisierung zwei Ziele:

Ziele: Eliminieren von Redundanzen und Vermeiden von Anomalien.

Es gibt zwei unterschiedliche Ansätze: Relationensynthese und Dekomposition, die zu unterschiedlichen Ergebnissen führen können.

Relationensynthese

Relationensynthese ist ein Entwurfsverfahren für ein relationales Datenbankschema. Ausgangspunkt dafür sind die Attribute und ihre funktionalen Abhängigkeiten, welche man in geeigneten Relationen zusammenfasst.

Dekomposition

Nicht immer gelingt es bei einem Entwurf die Relationen so zu gestalten, dass Informationen, die nicht zusammen gehören, auch nicht in einer Tabelle stehen. Solche Relationen führen in der Praxis zu Fehlern bei der Verarbeitung, die Anomalien genannt werden.

Die Grundidee der Dekomposition besteht darin, ausgehend von der 1. Normalform(1NF), Relationen, die nicht der geforderten Normalform entsprechen, in zwei oder mehr Relationen so zu zerlegen, dass die neuen Relationen der Normalform entsprechen.
Bedingung: es dürfen keine Informationen verloren gehen und die alte Relation muss aus den neuen wieder herstellbar sein (Verbundtreue oder Abhängigkeitstreue). Außerdem muss die Minimalität in Bezug auf die Anzahl der Relationen gewährleistet sein.

Das übliche Verfahren ist die Dekomposition. Aus Sicherheitsgründen sollte man auch einen scheinbar guten Entwurf damit überprüfen.

Übung

Die nachfolgenden drei Arbeitsblätter*) zeigen den Normalisierungsprozess an einem einfachen Beispiel.

- **1 NF** (1. Normalform)
- **2 NF** (2. Normalform)
- **3 NF** (3. Normalform)
- **Lösung**

Für den die erste Übung ist es günstig, die Tabellen auszufüllen, um den Fortschritt der Normalisierung zu veranschaulichen, später reicht die Kurznotation.

1. Normalform

"Eine Tabelle liegt in der ersten Normalform vor, wenn jeder Attributwert eine atomare, nicht weiter zerlegbare Dateneinheit ist."

Beispiel: Fahrradgeschäft

Knr	Name	Str.	Ort	Rahmennr	Marke	Versich.	Ort der Versich.	Reparatur-datum	Diagnose
100	Meyer	Hofweg 6	Berlin	123	Diamant	Allianz	Köln	12.12.94	Platten
101	Müller	Solweg 5	Berlin	690	Kettler	Allianz	Köln	14.12.94	Schleicher
100	Meyer	Hofweg 6	Berlin	432	Winora	Signal	Mainz		

2. Normalform

"Eine Tabelle liegt in der zweiten Normalform (2NF) vor, wenn sie in der 1NF ist und jedes Nichtschlüsselattribut voll funktional abhängig vom Primärschlüssel ist."

Die Attribute Name, Str., Ort, Marke, Versicherung, Ort der Versicherung, Reparaturdatum, Diagnose sind nicht voll funktional abhängig vom Primärschlüssel (Kombinationsschlüssel Knr und Rahmennr). Die Tabelle muss so aufgeteilt werden, dass diese Nichtabhängigkeiten beseitigt werden.

Fahrräder:

Rahmen-nr	Knr	Marke	Versich.	Ort der Versich.
123	100	Diamant	Allianz	Köln
690	101	Kettler	Allianz	Köln
432	100	Winora	Signal	Mainz

Kunden:

Knr	Name	Str.	Ort
100	Meyer	Hofweg 6	Berlin
101	Müller	Solweg 5	Berlin

Reparaturen

Rahmennr	Reparaturdatum	Diagnose
123	12.12.94	Platten
690	14.12.94	Schleicher

3. Normalform

"Eine Tabelle liegt in der dritten Normalform (3NF) vor, wenn sie sich in der 2NF befindet und jedes Nichtschlüsselattribut nicht transitiv abhängig vom Primärschlüssel ist."

Es gibt eine transitive Abhängigkeit: Rahmennummer -> Versicherung -> Ort der Versicherung. Aus diesem Grund muss eine neue Tabelle Versicherung angelegt werden.

Die Tabellen Kunden und Reparaturen bleiben unverändert.

Versicherungen

Versicherung	Ort der Versicherung
Allianz	Köln
Signal	Mainz

Fahrräder

Rahmennr	Knr	Marke	Versicherung
123	100	Diamant	Allianz
690	101	Kettler	Allianz
432	100	Winora	Signal

Normalisierung - 1NF

Ziel: Eliminieren von Redundanzen und Vermeiden von Anomalien

Weg: Attribute den Relationen so zuordnen, dass innerhalb der Relation keine Redundanz auftritt.

"Naive" Tabelle, nicht in der 1 NF (erste Normalform). Mitarbeiter können in mehreren Projekten beteiligt sein. Es wird jeweils notiert, wie viele Stunden sie in einem Projekt geleistet haben.

PERSONAL-PROJEKT

P#	P-Name	Abt#	Abt-Name	Prj#	Prj-Name	Prj-Std
101	Müller	1	Motoren	11, 12	A, B	60, 40
102	Meier	2	Karosserie	13	C	100
103	Krause	2	Karosserie	11, 12, 13	A, B, C	20, 50, 30
104	Schmidt	1	Motoren	11, 13	A, C	80, 20

Eine Tabelle liegt in der ersten Normalform vor, wenn jeder Attributwert eine atomare, nicht weiter zerlegbare Dateneinheit ist.
oder
Eine Tabelle ist nicht in der 1 NF, wenn Attribute mehrfach oder komplex in einer Spalte auftreten; d. h. 1 NF ist eine Strukturierungsvorschrift.

Rezept: Auslagern der nicht atomaren Attribute in verschiedene Zeilen oder mehrere Spalten oder eigene Tabellen.

PERSONAL-PROJEKT

P#	P-Name	Abt#	Abt-Name	Pj#	Pj-Name	Pj-Std

Normalisierung - 2NF

Zwischenergebnis: 1 NF

PERSONAL-PROJEKT

P#	P-Name	Abt#	Abt-Name	Prj#	Prj-Name	Prj-Std
101	Müller	1	Motoren	11	A	60
101	Müller	1	Motoren	12	B	40
102	Meier	2	Karosserie	13	C	100
103	Krause	2	Karosserie	11	A	20
103	Krause	2	Karosserie	12	B	50
103	Krause	2	Karosserie	13	C	30
104	Schmidt	1	Motoren	11	A	80
104	Schmidt	1	Motoren	13	C	20

Eine Tabelle liegt in der zweiten Normalform (2NF) vor, wenn sie in der 1NF ist und jedes Nichtschlüsselattribut voll funktional abhängig vom Primärschlüssel ist.
oder
Eine Tabelle ist nicht in der 2 NF, wenn Attribute von *einem Teil* des Schlüssels eindeutig identifiziert werden. Voraussetzung ist außerdem 1 NF

Rezept: *Auslagern von Teilschlüsseln und zugehörigen Informationen in eigene Tabellen nach Sachgebieten; bzw. separate Entitätstypen mit eigenem Schlüssel finden. Beim Auslagern durch entsprechende Beziehungen darauf achten, dass Informationen nicht verloren gehen.*

Normalisierung - 3NF

Zwischenergebnis: 2NF

PERSONAL

P#	P-Name	Abt#	Abt-Name
101	Müller	1	Motoren
102	Meier	2	Karosserie
103	Krause	2	Karosserie
104	Schmidt	1	Motoren

PERSONAL-PROJEKT

P#	Prj#	Prj-Std
101	11	60
101	12	40
102	13	100
103	11	20
103	12	50
103	13	30
104	11	80
104	13	20

PROJEKT

Prj#	Prj-Name
11	A
12	B
13	C

Eine Tabelle liegt in der dritten Normalform (3NF) vor, wenn sie sich in der 2NF befindet und jedes Nichtschlüsselattribut nicht transitiv abhängig vom Primärschlüssel ist
oder
Eine Tabelle ist nicht in der 3 NF, wenn Attribute von anderen Nicht-Schlüsselattributen identifiziert werden. Voraussetzung ist außerdem 2 NF.

Rezept: Auslagern der "transitiv abhängigen" Attribute in eigene Tabellen.

Normalisierung - Lösung 3. Normalform

1. Normalform (1NF)
PERSONAL-PROJEKT (P#, P-Name, Abt#, Abt-Name, Prj#, Prj-Name, Prj-Std)

2. Normalform (2NF)
PERSONAL (P#, P-Name, Abt#, Abt-Name)
PROJEKT (Prj#, Prj-Name)
PERSONAL-PROJEKT (P#, Prj#, Prj-Std)

3. Normalform (3NF)
PERSONAL (P#, P-Name, Abt#)
PROJEKT (Prj#, Prj-Name)
PERSONAL-PROJEKT (P#, Prj#, Prj-Std)
ABTEILUNG (Abt#, Abt-Name)

Übungsaufgabe zur Normalisierung

Aufgabenstellung:

Der Systemanalytiker Hannes von Massenheim wurde beauftragt, für eine Autovermietung eine ERM zu erstellen und dieses Modell in Tabellen zu übertragen.

Er liefert folgende 3 Tabellen ab (Schlüsselfelder sind unterstrichen):

Kunden

Knr	**Name**	**Straße.**	**PLZ**	**Ort**	**Geburtsdatum**
100	Meyer	Hofweg 6	14169	Berlin	13.04.56
101	Müller	Solweg 5	10999	Berlin	12.09.46

PKW

Kennzeichen	**Erst-zulassung**	**Klima-anlage**	**Modell-name**	**Lei-stung**	**Länge**	**Herstel-lername**	**Strasse**	**PLZ**	**Ort**
B - W 123	10.05.95	ja	Opel Manta	78	390	Opel	Torstr. 5	54321	Rüsselsheim
B - K 345	10.09.95	nein	Opel Manta	78	390	Opel	Torstr. 5	54321	Rüsselsheim

Ausleihe

Kennzeichen	**Knr**	**Ausleihtag**	**Anfangkm**	**Rückgabetag**	**Endekm**
B - W 123	1011	18.09.95	23457	20.09.95	25456
B - K 345	1017	18.09.95	1256	19.09.95	2341

Marietta Mandelheim, Auszubildende zur Datenverarbeitungskauffrau, sieht diese Tabellen und lästert: "Das sieht doch jede, dass diese 3 Tabellen nicht der 3. Normalform entsprechen."

Aufgabe:

Prüfen Sie, ob Marietta recht hat und stellen Sie ggf. alle Tabellen in 3. Normalform dar!

Redundanz

Um Inkonsistenzen zu vermeiden, sollte eine Datenbank keine Daten mehrfach enthalten. Dies ist z. B. dann der Fall, wenn Daten über Studentinnen und Studenten gleichzeitig in der Fachbereichsbibliothek und im Immatrikulationsbüro gespeichert werden.

Redundanz: mehrmaliges Speichern der gleichen Daten, d. h. es können Daten ohne Informationsverlust weggelassen werden.

Wenn Daten mehrfach gespeichert werden,

1. führt dies zu einem redundanten Arbeitsaufwand, z. B. muss eine Änderung der Anschrift im obigen Beispiel zweimal vorgenommen werden.
2. wird Speicherplatz verschwendet.
3. können Mutationsanomalien auftreten, wenn nicht alle Daten in allen beteiligten Dateien geändert werden, z. B. eine Änderung der Anschrift nur in der Fachbereichsbibliothek und nicht im Immatrikulationsbüro vorgenommen wird.

Eine Normalisierung überprüft ein Datenmodell auf Redundanzen und stellt somit sicher, dass keine Daten mehrfach vorhanden sind.

Relationale Datenbanken

Mutationsanomalien

Wenn einen Datenbank nicht normalisiert wurde und dadurch Redundanzen aufweist können Mutationsanomalien vorkommen. Diese Anomalien entstehen auch, wenn die Integritätsbedingungen einer Datenbank, insbesondere die referentielle Integrität zwischen Relationen nicht eingehalten werden.

Weist eine eine Datenbank Mutationsanomalien auf, dann sind die enthaltenen Daten inkonsistent, z. B. kann es dann möglich sein, dass von einer Person mehrere Anschriften in der Datenbank vorhanden sind.

Am folgenden Beispiel sollen die einzelnen Anomalien erklärt werden:

Ausleihe (Achtung, diese Tabelle ist nicht normalisiert, also falsch und dient nur als Demonstrationsobjekt)

Ausleih-nr	**Buchnr**	**Name**	**Vorname**	**Straße**	**PLZ**	**Ort**	**Ausleih-datum**	**Rückgabe-datum**
1	100	Meyer	Hans	Hofweg 8	12345	Berlin	2005-11-03	2006-01-03
2	245	Meyer	Hans	Hofweg 8	12345	Berlin	2005-11-03	2006-01-03
3	444	Müller	Hilde	Hagweg 9	10101	Berlin	2006-01-03	

Buch

Buchnr	**Titel**	**Autor**	**Verlag**
100	Grundlagen von Datenbanksystemen	Elmasri	Addison-Wesley
245	Datenbankmodelle	Vossen	Oldenbourg
444	Relationale Datenbanken	Meier	Springer

Einfüge-Anomalie

In eine Relation mit Fremdschlüssel können neue Tupel nur eingetragen werden, wenn der konkrete Datensatz in der referenzierten Relation vorhanden ist. Ein neues Tupel kann also in der Relation Ausleihe erst eingetragen werden, wenn das auszuleihende Buch in der Relation Buch vorhanden ist.

Lösch-Anomalie

Ein Tupel darf aus einer Relation erst gelöscht werden, wenn keine Fremdschlüsseleinträge für diesen Tupel in anderen Relationen vorliegen. Geschieht dies trotzdem gehen alle Informationen verloren. Wird das Buch mit der Buchnummer 444 in der Relation Buch gelöscht, sind alle Daten zu diesem Buch nicht mehr nachvollziehbar, obwohl das Buch noch ausgeliehen ist.

Änderungs-Anomalie

Wenn, wie in der Relation Ausleihe, ein Datensatz mehrfach vorhanden ist und sich ändert, kann es vorkommen, dass nicht alle betroffenen Datensätze geändert werden. Zieht Hans Meyer um, dann müssen in der Relation Ausleihe die beiden vorhandenen Adressen geändert werden, sonst entsteht eine Änderungs-Anomalie. Durch die Auslagerung der Kundendaten in eine eigene Relation Kunde, können Änderungsanomalien in diesem Fall verhindert werden.

Mutationsanomalien

Mutationsanomalien sind schädliche Folgen für die Konsistenz einer Datenbank, die sich bei Änderungen von Daten ergeben können, wenn sich die Tabellen nicht in einer normalisierten Form befinden und Redundanzen vorhanden sind.

Beispiel

Für diese Tabelle soll gelten:

- Jedes Fach hat genau einen Fachleiter und einen Fachhelfer
- Ein Fach kann von mehreren Lehrern unterrichtet werden

FACH-LEHRER

Lehrerkürzel	**Lehrername**	**Fachname**	**FachleiterKürzel**	**FachhelferName**
Hei	Heitech	Info	Hei	Lotech
Mid	Middletech	Info	Hei	Lotech
Zwi	Zwitech	Info	Hei	Lotech
Jag	Jagger	Musik	Jag	Brahms
Sch	Schiller	Deutsch	Schil	Ratlos
Goe	Goethe	Deutsch	Schil	Ratlos

1. Was kann man bei Betrachtung der Datenintegrität zu der Tabelle sagen?
2. Wie wirkt es sich auf die Datenintegrität aus, wenn ...

- **Einfügeanomalie**
  Ein neues Fach Kunst soll eingeführt werden, jedoch ist noch kein Lehrer dafür verfügbar.
- **Löschanomalie**
  Lehrer Jagger (Jag) wird pensioniert.
- **Änderungsanomalie**
  Das Fach Info wird der Fachhelfer Lotech durch den Fachhelfer Fiffikus ersetzt.

Mutationsanomalien – Lösung

FACH-LEHRER

Lehrerkürzel	Lehrername	Fachname	FachleiterKürzel	FachhelferName
Hei	Heitech	Info	Hei	Lotech
Mid	Middletech	Info	Hei	Lotech
Zwi	Zwitech	Info	Hei	Lotech
Jag	Jagger	Musik	Jag	Brahms
Sch	Schiller	Deutsch	Schil	Ratlos
Goe	Goethe	Deutsch	Schil	Ratlos

Was kann man zu der Tabelle sagen?
Die Tabelle enthält viele Redundanzen.

Wie wirkt es sich auf die Datenintegrität aus, wenn ... ?

- **Einfügeanomalie**
  Ein neues Fach Kunst soll eingeführt werden, jedoch ist noch kein Lehrer dafür verfügbar.

  Das Fach kann nicht eingeführt werden - man muss warten bis ein Lehrer einsetzbar ist, weil das Attribut Lehrerkürzel zum Schlüssel gehört und immer einen Wert (NOT NULL) haben muss.
- **Löschanomalie**
  Lehrer Jagger (Jag) wird pensioniert.

  Wenn Jagger die Schule verlässt, dann muss das ganze Fach Musik gelöscht werden. Außerdem geht die Information verloren, dass Brahms Fachhelfer ist.
- **Änderungsanomalie**
  Das Fach Info wird der Fachhelfer Lotech durch den Fachhelfer Heiend ersetzt.

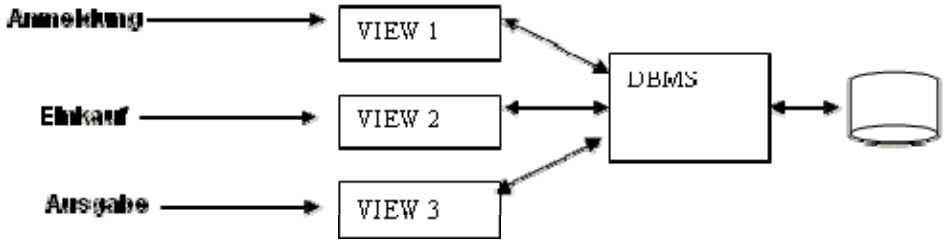
  Wenn Fiffikus statt Lotech Fachhelfer wird, dann müssen Änderungen in drei Datensätzen gemacht werden.

Sichten und Zugriffsrechte

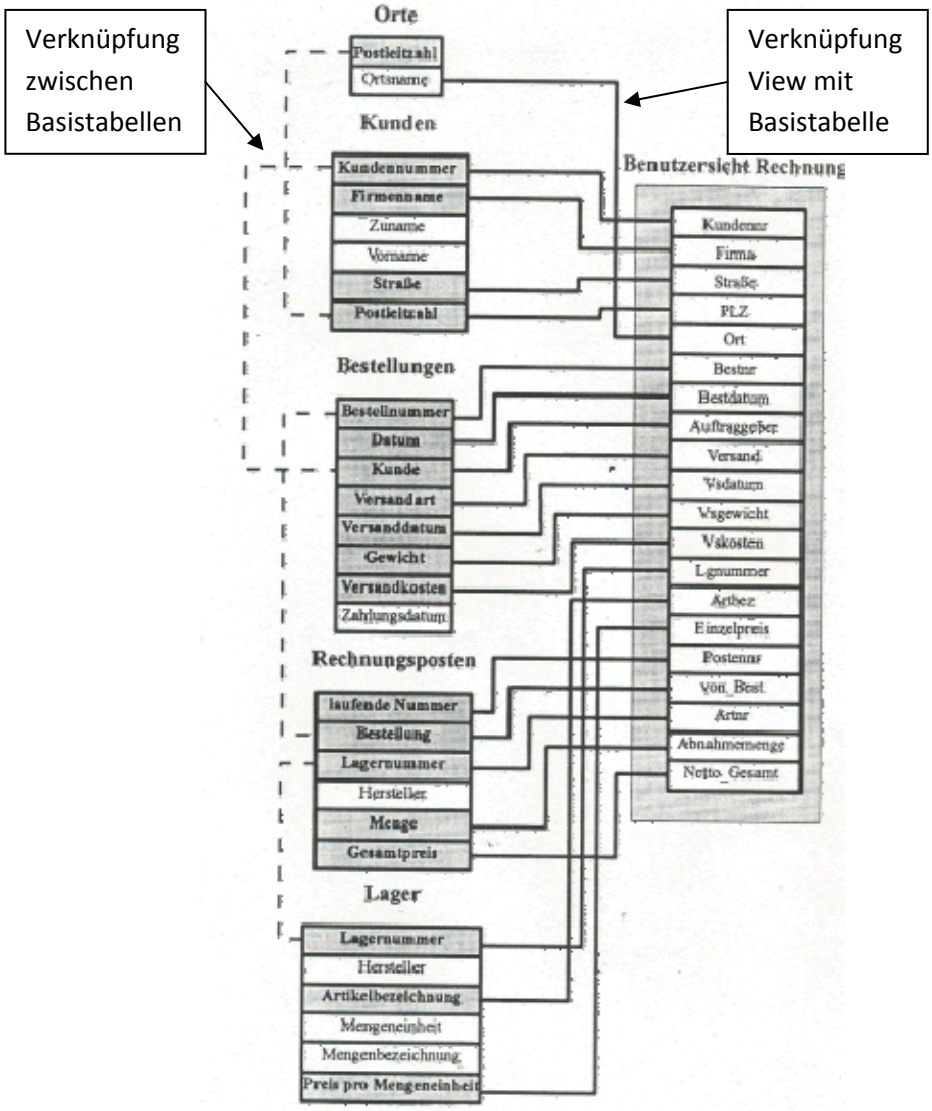
Nicht jeder Mitarbeiter muss und soll Zugriff auf alle Daten haben.

Sichten

Es können Benutzersichten auf Teilmengen (einer oder mehrerer Tabellen und/oder Teilen daraus) des Datenbestandes eingerichtet werden. Sie werden mit der Anweisung **Create view** ... vergeben und können dem Benutzer zugänglich gemacht werden.



Beispiel für eine Benutzersucht (view) 'Rechnung'.



Quelle: Husch, B: Praktische SQL-Anwendung, 1994.

Zugriffsrechte

können mit **grant** select on kunden to mueller
insert ..
delete ..
update .. usw.

jedem einzelnen Benutzer gegeben werden.

Daneben sind die üblichen Zugangskontrollen(ID, password) erforderlich.

Zugriffsrechte SQL

Vorbemerkungen

Alle im weiteren erwähnten Zugriffsrechte beziehen sich auf den SQL1-Standard. Die Beispiele gelten für MySQL und entsprechen teilweise nicht diesem Standard.

Zugriffsebenen

Zugriffsrechte für Datenbanken können auf vier Ebenen vergeben werden:

- Globale Ebene: Die Ebene des Datenbank Management Systems (DBMS), d.h. wer überhaupt Zugang zu allen Datenbanken hat.
- Datenbank-Ebene: Der Zugang zu einzelnen Datenbanken, d.h. wer eine Datenbank benutzen darf
- Tabellen-Ebene: Der differenzierte Zugriff auf einzelne Tabellen, d.h. welche Operationen auf einer Tabelle (Relation) ausgeführt werden dürfen
- Spalten-Ebene: Der differenzierte Zugriff auf einzelne Spalten, d.h. welche Operationen auf einer Spalter (Attribut) ausgeführt werden dürfen.

Die dargestellten Rechte werden in der angegebenen Reihenfolge ausgewertet. Das bedeutet: Rechte die auf der globalen Ebene vergeben werden, können auf unteren Ebene nicht wieder eingeschränkt werden. Auf der globalen Ebene sollte deshalb nur die Administration des DBMS über alle Rechte verfügen und allen anderen Benutzerinnen und Benutzer sollten nur das Zugangsrecht zum DBMS haben. Alle weiteren Rechte sollten nur jeweils für Datenbanken, Tabellen und Spalten vergeben werden.

Zugang zum Datenbank Management System (Globale Ebene)

Globale Berechtigungen betreffen alle Datenbanken auf einem gegebenen Server**.** Diese Rechte sollten sehr sparsam oder gar nicht vergeben werden. Es reicht, wenn alle Benutzerinnen und Benutzer auf dieser Ebene Zugang zur Datenbank haben.

Der Befehl bei MySQL dafür lautet für den Benutzer ausleihe mit dem Passwort auff: `GRANT USAGE on *.* to ausleihe@localhost IDENTIFIED BY 'auff';` Die USAGE-Berechtigung erlaubt bei MySQL, einen Benutzer ohne Berechtigungen anzulegen.

Für das gesamte Datenbank Management System werden Administratorenrechte vergeben. Diese Rechte sind nicht eingeschränkt und erlauben es alle Datenbanken und Rechte zu verändern und zu löschen. Wichtigste Aufgabe der globalen Datenbankadministration ist es leere Datenbank zu erzeugen und dafür die Rechte der Datenbankadministration zu vergeben.

Zugang zur Datenbank

Dieses Zugriffsrecht legt fest, wer die Datenbank aufrufen und damit nutzen darf. Entweder kann dieses Recht allen Benutzerinnen und Benutzern oder einzelnen Personen zugeteilt werden. Greift jemand unberechtigterweise auf die Datenbank zu, so wird er mit einer entsprechenden Fehlermeldung abgewiesen.

Um allen Benutzerinnen und Benutzern bei MySQL einen Zugriff auf die Datenbank "FF" zu gewähren lautet der Befehl: `GRANT USAGE on FF.* to ''@localhost;`

Für jede Datenbank werden besondere Rechte für die Administration vergeben. Das Recht der Datenbankadministration umfasst alle Zugriffe auf die Datenbank. Insbesondere können mit diesem Recht:

- alle Zugriffsrechte vergeben und entzogen
- Tabellen, Views und Indices erzeugt, verändert und gelöscht
- die Datenbank gelöscht

werden. Wer die Datenbank erzeugt, wird automatisch Datenbankadministrator dieser Datenbank, d.h. er besitzt DBA-Rechte. Dieses Recht kann vom Administrator weiteren Personen zuerkennen.

DBA-Rechte sollten auf wenige Personen beschränkt bleiben.

Zugriff auf Tabellen

Für jede Tabelle wird entschieden, wer auf diese Tabelle mit welchen Rechten zugreifen darf. Folgende Zugriffsrechte können vergeben werden:

- Ansehen (SELECT)
- Hinzufügen (INSERT)
- Ändern (UPDATE)
- Löschen (DELETE)
- alle obigen Zugriffsrechte (ALL PRIVILEGES)

Weiter können differenziert Rechte zur Tabellenveränderung erteilt werden. Dies ist in den entsprechenden Handbüchern nachzulesen, und wird hier nicht näher erläutert.

Standardmäßig werden zunächst alle Tabellenrechte für die Allgemeinheit freigegeben. Die Einschränkung auf bestimmte Personen ist gesondert zu programmieren. Die allgemeinen Rechte sind dabei ausdrücklich rückgängig zu machen.

Folgende Tabelle kann als Musterdokumentation für die Festlegung von Tabellenrechten dienen:

Tabellenname

Benutzernamen	**SELECT**	**INSERT**	**UPDATE**	**DELETE**
Name	ja/nein	ja/nein	ja/nein	ja/nein
Name	ja/nein	ja/nein	ja/nein	ja/nein
Name	ja/nein	ja/nein	ja/nein	ja/nein

Soll ein Zugriff auf bestimmte Tabellen auch für unangemeldete Benutzer möglich sein, so ist es ratsam dafür einen besonderen Benutzer (z.B. Dummy) mit entsprechenden Rechten einzurichten.

Beispiel für die Regelung der Zugriffe auf die Tabelle Werkstatt in MySql:

```
GRANT SELECT ON FF.werkstatt TO fuhrpark@localhost,
wartung@localhost;
GRANT INSERT ON FF.werkstatt TO wartung@localhost;
GRANT UPDATE ON FF.werkstatt TO wartung@localhost;
GRANT DELETE ON FF.werkstatt TO wartung@localhost;
```

Der Benutzer "wartung" darf die Tabelle ansehen und kann Werte einfügen, ändern und löschen. Der Benutzer "fuhrpark" hat nur Leserechte.

Zugriff auf Spalten

Für jede Spalte wird entschieden, wer auf diese Tabelle mit welchen Rechten zugreifen darf. Folgende Zugriffsrechte können vergeben werden:

- Ansehen (SELECT)
- Hinzufügen (INSERT)
- Ändern (UPDATE)
- Löschen (DELETE)
- alle obigen Zugriffsrechte (ALL PRIVILEGES

Views

Durch geeignet definierte Views wird erreicht, dass je nach Bedarf ein bestimmter Ausschnitt der Datenbank zur Verfügung gestellt wird. Dieser Ausschnitt kann aus einer Kombination von Feldern aus verschiedenen Tabellen bestehen. So können zum Beispiel aus Sicherheitsgründen kritische Daten (wie Kontostand) vor nicht-privilegierten Benutzern verborgen werden, während privilegierte Benutzer auf alle Daten zugreifen dürfen. Views werden wie Tabellen behandelt, d.h. es können alle Tabellenrechte vergeben werden.

Bei der Kombination von Feldern verschiedener Tabellen müssen die Zugriffsrechte aller beteiligten Tabellen für den jeweiligen Benutzer der Views definiert werden.

Relationales Modell

Das relationale Datenmodell (nach Codd)

Die Tabelle ist die einzige Datenstruktur des relationalen Modells.

Unabhängig von den Aspekten der jeweiligen Implementierung und physikalischen Realisierung auf einem bestimmten Rechner, bieten relationale Datenbanken dem Anwender eine einfache logische Schnittstelle zu den Datenbeständen.

Diese werden dem Benutzer in Form von Tabellen zur Verfügung gestellt. Entgegen einem hierarchischen Datenmodell müssen keine detaillierten Informationen über die Beziehungen (Referenzen, links) zwischen den einzelnen Objekten verwaltet werden, sondern das System bietet die Daten lediglich als Tupels an, zwischen denen der Benutzer später weitgehend beliebige Beziehungen - auch während der Anwendung - herstellen kann.

KUNDE

K-Nr	Vorname	Name
12	Hans	Meier
47	Otto	Waalkes
....	...	...
1020	Rudi	Ratlos

VIDEO

V-Nr	Titel	Genre
101	Casablanca	Liebe
348	Alpenglühen	Heimat
...	...	...

AUSLEIHE

K-Nr	Vorname	Name	V-Nr	Titel	Genre
47	Otto	Waalkes	101	Casablanca	Liebe
47	Otto	Waalkes	348	Alpenglühen	Heimat
...					

Beziehungsrelation (naive Darstellung)

Jede dieser Tabellen hat folgende Eigenschaften:

- Jede Tabelle ist autonom und wird durch einen eindeutigen Schlüssel identifiziert.
- Auf jeden elementaren Wert kann garantiert zugegriffen werden durch eine Kombination von Tabellenname, Primärschlüssel und Spaltenname
- NULL-Werte (unbelegte Attributfelder unabhängig von Datentyp) dürfen keine möglichen Primärschlüssel sein und müssen bei Bedarf auch in anderen Spalten ausgeschlossen werden können.
- Jede Spalte hat einen Bezeichner, der innerhalb der Tabelle eindeutig ist.
- Die Anordnung der Spalten ist für alle Tupel (Zeilen) in einer Tabelle einheitlich, aber ohne jede Bedeutung.
- Spalten sind elementar. Dies heißt, dass Tabellen "flache" Datenstrukturen sein müssen, in denen ein Datenfeld keine weitere Datenstruktur enthalten darf.
- All Einträge innerhalb einer Spalte sind von demselben Typ (meist NUMERIC(n), AMOUNT(n), STRING(n), DATE(n); BLOB)
- Die Reihenfolge der Tupel ist in der Tabelle beliebig.

Ein Datenbanksystem kann dann relational genannt werden, wenn es die allgemeinen Anforderungen an ein DBS erfüllt und

1. die Daten dem Benutzer ausschließlich als Relationen (Tabellen) zugänglich macht,
2. die Primärschlüsselbedingung und die referenzielle Integrität (Fremdschlüsselbedingungen) automatisch überwacht,
3. die Operationen der Relationenalgebra und die Änderungsoperationen auf Tabellen ausführen kann, ohne dass physische Zugriffe angegeben werden müssen.

Operationen auf Tabellen – Projektion

Projektion : Spalten auswählen

KUNDE

K-Nr	Vorname	Name
12	Hans	Meier
47	Otto	Waalkes
..	...	...
...	...	...
...	...	...
512	Elli	Pirelli

⟹

proj (Name)

Name
Meier
Waalkes
...
...
...
Pirelli

Liste die Namen aller Kunden auf!

Operationen auf Tabellen – Selektion

Selektion : Zeilen (Tupels) auswählen

KUNDE

K-Nr	Vorname	Name
12	Hans	Meier
47	Otto	Waalkes
..	...	...
308	Erich	Meier
...	...	...
512	Elli	Pirelli

⟹

sel(Name = 'Meier')

K-Nr	Vorname	Name
12	Hans	Meier
308	Erich	Meier

Liste alle Kunden mit dem Namen '**Meier**' auf!

Operationen auf Tabellen - Proj. + Sel.

Projektion und Selektion kombinieren

KUNDE

K-Nr	Vorname	Name
12	Hans	Meier
47	Otto	Waalkes
..	...	...
308	Erich	Meier
...	...	...
512	Elli	Pirelli

⟹ **proj(Nr,Name); sel(Name= 'Meier')**

K-Nr	Name
12	Meier
308	Meier

Liste die Nummer und Namen aller Kunden auf, die '**Meier**' heissen!

Operationen auf Tabellen – Join

Natural Join : Tabellen verbinden über die gemeinsamen Attribute

KUNDE

K-Nr	Vorname	Name
12	Hans	Meier
47	Otto	Waalkes
..	...	...
308	Erich	Meier
...	...	...
512	Elli	Pirelli

⟹ **join(KUNDE.K-Nr = AUSLEIHE.K-Nr)**

Vorname	Name
Otto	Waalkes

⟹

AUSLEIHE

K-Nr	V-Nr	....
47	101	

Liste die Vorname und Namen aller Kunden auf, die ein Video geliehen haben!

Weitere Operationen auf Tabellen sind:

- Einfügen von Zeilen in Tabellen
- Löschen von Zeilen
- Ändern von Spaltenwerten

Dazu steht eine standardisierte Datenbanksprache zur Verfügung : **SQL**

Kartesisches Produkt

Das kartesische Produkt ist die Menge aller möglichen Wertekombinationen

KUNDE

K-Nr	Name
1	Abel
2	Bebel
47	Waalkes

AUSLEIHE

K-Nr	V-Nr	....
1	300	
47	400	

kartesisches Produkt

1	Abel	1	300
1	Abel	47	400
2	Bebel	1	300
2	Bebel	47	400
47	Waalkes	1	300
47	Waalkes	47	400

Daraus lässt sich eine Definition ableiten:

Eine Relation ist eine Teilmenge des kartesischen Produkts, wobei alle Tupel unter sich verschieden sind

... und noch einmal: join (KUNDE.K-Nr = AUSLEIHE.K-Nr)

Suche Kunden-Nr, Name aller Kunden, die ein Video ausgeliehen haben!

1	Abel	1	300
1	Abel	47	400
2	Bebel	1	300
2	Bebel	47	400
47	Waalkes	1	300
47	Waalkes	47	400

Selektion

1	Abel	1	300
47	Waalkes	47	400

und nun noch die Projektion

1	Abel
47	Waalkes

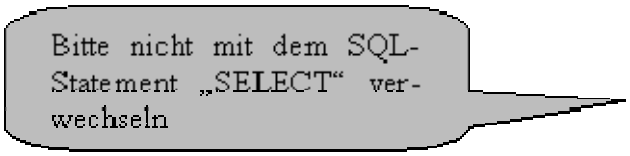
Grundoperationen auf Tabellen (Relationen)

Neben dem EINFÜGEN, LÖSCHEN und VERÄNDERN Zeilen (Tupel), sind in RDBMS auch Operationen definiert, die sich auf ganze Tabellen (Relationen) beziehen.

Im einzelnen sind dies:

- Tabelle erzeugen
- Tabelle löschen
- Tabelle mit Zugriffsrechten belegen
- Für eine bestimmte Tabelle einen Index zur Beschleunigung der Zugriffe erzeugen.

Zur Auswertung der in den jeweiligen Tabellen enthaltenen Informationen stehen folgende **Auswahloperationen** zur Verfügung:



- **Projektion**
- **Selektion**
- **Join (Verbund)**
- **View**

Beachte!

Das Ergebnis einer Auswahloperation ist Relation (Tabelle) mit mindestens einem Attribut (einer Spalte), die entweder leer ist oder die gesuchten Werte enthält.

Projektion (Auswahl von bestimmten Spalten)

Artikel				
ArtNr	**Bezeichnung**	**Bestand**	**Preis**	**StammLief**
100	Bohrer	100	4,85	4711
101	Pinsel	210	7,60	3299
103	Hammer	17	8,30	4711
105	Zange	25	24,50	4711
110	Kelle	9	16,80	3000
111	Gips	200	9,60	*NULL*
112	Zement	180	10,40	*NULL*

Erstellen Sie eine Artikelliste mit Nr., Bezeichnung und Preisen aller unserer Artikel

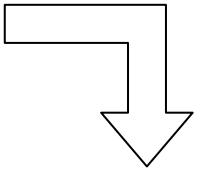
```
SELECT ArtNr,
Bezeichnung, Preis
FROM Artikel
```

ArtNr	Bezeichnung	Preis
100	Bohrer	4,85
101	Pinsel	7,60
103	Hammer	8,30
105	Zange	24,50
110	Kelle	16,80
111	Gips	9,60
112	Zement	8,40

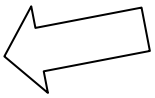
Selektion (Auswahl von bestimmten Zeilen)

Artikel				
ArtNr	**Bezeichnung**	**Bestand**	**Preis**	**StammLief**
100	Bohrer	100	4,85	4711
101	Pinsel	210	7,60	3299
103	Hammer	17	8,30	4711
105	Zange	25	24,50	4711
110	Kelle	9	16,80	3000
111	Gips	200	9,60	*NULL*
112	Zement	180	10,40	*NULL*

Erstellen Sie eine Artikelliste aller Artikel die 10 DM oder mehr kosten.



```
select *
from Artikel
where Preis >= 10.00
```

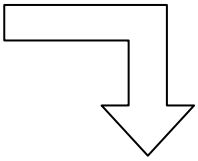


Artikel				
ArtNr	Bezeichnung	Bestand	Preis	StammLief
105	Zange	25	24,50	4711
110	Kelle	9	16,80	3000
112	Zement	180	10,40	*2000*

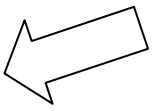
In der Regel werden Selektion und Projektion innerhalb derselben Abfrage verwendet.

Artikel				
ArtNr	**Bezeichnung**	**Bestand**	**Preis**	**StammLief**
100	Bohrer	100	4,85	4711
101	Pinsel	210	7,60	3299
103	Hammer	17	8,30	4711
105	Zange	25	24,50	4711
110	Kelle	9	16,80	3000
111	Gips	200	9,60	*NULL*
112	Zement	180	10,40	*NULL*

Erstellen Sie eine Artikelliste aller Artikel die 10 DM oder mehr kosten.
Das Ergebnis soll nur die jeweilige ArtNr, Bezeichnung und den Preis enthalten



```
select ArtNr,
Bezeichnung, Preis
from Artikel
where Preis >=10.00
```



ArtNr	Bezeichnung	Preis
105	Zange	24,50
110	Kelle	16,80
112	Zement	10,40

JOIN (VERBUND) (Zusammenziehen der Informationen aus verschiedenen Tabellen)

Solange keine der verbundenen Tabellen NULL-Werte enthält sind Verbunde relativ unkompliziert zu handhaben. (Zur Problematik "Verbundspalten mit NULL-Werten" beachte die nachfolgende Seite.)
Der einfachste Verbund ist ein sog. "**CROSS JOIN**" bei dem das Ergebnis aus dem Kreuzprodukt der beteiligten Tabellen besteht. Z.B.:

```
SELECT *
FROM Artikel, Lieferanten
```

```
SELECT *
FROM Artikel CROSS JOIN Lieferanten
```

Das Ergebnis einer solchen Abfrage ist aber in aller Regel nicht besonders erleuchtend, so dass es in aller Regel erforderlich ist, das Ergebnis einer Verbundoperation auf eine bestimmte Teilmenge einzugrenzen.
Dies geschieht z.B. in der nachfolgenden Aufgabe, bei dem auf zwei verschiedenen Lösungswegen ein sog. "natürlichen Verbund" erzeugt wird.

Aufgabe: Erstellen Sie eine Artikelliste mit ArtNr, Artikelbezeichnung und Name und Ort des jeweiligen Lieferanten

Artikel

ArtNr	Bezeichnung	Bestand	Preis	StammLief
100	Bohrer	100	4,85	4711
101	Pinsel	210	7,60	3299
103	Hammer	17	8,30	4711
105	Zange	25	24,50	4711
110	Kelle	9	16,80	3000
111	Gips	200	9,60	2000
112	Zement	180	10,40	2000

Lieferanten

LiefNr	Name	Ort	...
2000	Lederer AG	Berlin	…
2300	Teufel & Co	Potsdam	…
3000	Auf & Ab KG	Bernau	…
3299	Trims OHG	Berlin	…
4700	Bert & Kage	Berlin	…
4711	Kugler GmbH	Kremmen	…
4800	Aal & Söhne	Potsdam	…

```
SELECT ArtNr, Bezeichnung, Name,
Ort
FROM Artikel, Lieferanten
WHERE Artikel.StammLief =
Lieferanten.LiefNr
```

```
SELECT ArtNr, Bezeichnung, Name,
Ort
FROM Artikel INNER JOIN
Lieferanten
  ON Artikel.StammLief =
Lieferanten.LiefNr
```

ArtNr	Bezeichnung	Name	Ort
100	Bohrer	Kugler GmbH	Kremmen
101	Pinsel	Trims OHG	Berlin
103	Hammer	Kugler GmbH	Kremmen
105	Zange	Kugler GmbH	Kremmen
110	Kelle	Auf & Ab KG	Bernau
111	Gips	Lederer AG	Berlin
112	Zement	Lederer AG	Berlin

JOINS unter Tabellen, die NULL-Werte enthalten

Sobald aber eine der beteiligten Tabellen NULL-Werte enthält, wird die Sache etwas interessanter.

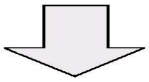
Nun erhält man je nachdem, ob man sich für einen **INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, oder FULL OUTER JOIN** entscheidet ein jeweils anderes Ergebnis. Ändert man das zugrunde liegende Beispiel derart, dass in der Tabelle Artikel die Spalte Stammlief NULL-Werte enthält, würden **beide Varianten** der Verbundabfrage auf der letzten Seite ein irreführendes Ergebnis liefern.

Artikel				
ArtNr	Bezeichnung	Bestand	Preis	StammLief
100	Bohrer	100	4,85	4711
101	Pinsel	210	7,60	3299
103	Hammer	17	8,30	4711
105	Zange	25	24,50	4711
110	Kelle	9	16,80	3000
111	Gips	200	9,60	NULL
112	Zement	180	10,40	NULL

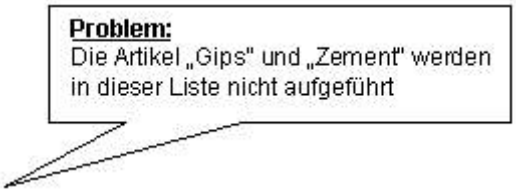
Lieferanten			
LiefNr	Name	Ort	...
2000	Lederer AG	Berlin	…
2300	Teufel & Co	Potsdam	…
3000	Auf & Ab KG	Bernau	…
3299	Trims OHG	Berlin	…
4700	Bert & Kage	Berlin	…
4711	Kugler GmbH	Kremmen	…
4800	Aal & Söhne	Potsdam	…

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Artikel, Lieferanten
WHERE Artikel.StammLief =
Lieferanten.LiefNr
```

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Artikel INNER JOIN Lieferanten
ON Artikel.StammLief =
Lieferanten.LiefNr
```



ArtNr	Bezeichnung	**Name**	Ort
100	Bohrer	**Kugler GmbH**	Kremmen
101	Pinsel	**Trims OHG**	Berlin
103	Hammer	**Kugler GmbH**	Kremmen
105	Zange	**Kugler GmbH**	Kremmen
110	Kelle	**Auf & Ab KG**	Bernau



Aus dieser Ergebnisliste ist nicht zu entnehmen, dass für die Artikel "Gips" und "Zement" kein Lieferant enthalten ist. Wird die Aufgabe wie folgt modifiziert, so führt nur die nachfolgende Lösung zum Erfolg.

Aufgabe: Erstellen Sie eine Artikelliste in der alle Artikel enthalten sind, und zu jedem Artikel der Stammlieferant (soweit vorhanden) angegeben wird

Hier führt ein sog. "LEFT OUTER JOIN" zum Erfolg, der das nachfolgende Ergebnis liefert.

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Artikel, Lieferanten
WHERE Artikel.StammLief =
Lieferanten.LiefNr
    OR Artikel.StammLief = NULL
```

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Artikel LEFT OUTER JOIN
Lieferanten
ON Artikel.StammLief =
Lieferanten.LiefNr
```

ArtNr	Bezeichnung	Name	Ort
100	Bohrer	Kugler GmbH	Kremmen
101	Pinsel	Trims OHG	Berlin
103	Hammer	Kugler GmbH	Kremmen
105	Zange	Kugler GmbH	Kremmen
110	Kelle	Auf & Ab KG	Bernau
111	Gips	NULL	NULL
112	Zement	NULL	NULL

LEFT OUTER JOIN (linksseitige Außenverknüpfung) Enthält sämtliche Zeilen der im FROM-Statment genannten Tabelle, auch solche, die im Vergleichskriterium NULL-Werte enthalten und verknüpft diese mit der Tabelle auf der Lieferanten. Kurz: Ein "LEFT OUTER JOIN" enthält alle Zeilen, die ein "INNER JOIN" liefern würde und zusätzlich alle die Zeilen der linken Tabelle, die beim Vergleichskriterium einen NULL-Werte enthalten.

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Artikel LEFT OUTER JOIN Lieferanten
       ON Artikel.StammLief = Lieferanten.LiefNr
```

RIGHT OUTER JOIN (rechtsseitige Außenverknüpfung) Enthält sämtliche Zeilen der im JOIN-Statement genannten Tabelle, auch solche, die im Vergleichskriterium NULL-Werte enthalten und verknüpft diese mit der Tabelle auf der linken Seite (hier ebenfalls Lieferanten). Kurz: Ein "RIGHT OUTER JOIN" enthält alle Zeilen, die ein "INNER JOIN" liefern würde und zusätzlich alle die Zeilen der rechten Tabelle, die beim Vergleichskriterium einen NULL-Werte enthalten.

(Eine rechtsseitige Außenverknüpfung liefert dieselben Ergebnisse wie eine linksseitige Außenverknüpfung, wenn die Spalten entsprechend vertauscht werden – she. Folgendes Beispiel.)

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Lieferanten RIGHT OUTER JOIN Artikel
    ON Artikel.StammLief = Lieferanten.LiefNr
```

FULL OUTER JOIN (vollständige Außenverknüpfung)

Enthält sämtliche Reihen, die in der Bedingung übereinstimmen sowie übereinstimmenden Reihen aus sowohl der linksseitigen als auch der rechtsseitigen Tabelle.

Kurz: Ein "FULL OUTER JOIN" enthält alle Zeilen, die ein "INNER JOIN" liefern würde und zusätzlich alle Zeilen sowohl der linken wie auch rechten Tabelle, die beim Vergleichskriterium einen NULL-Werte enthalten.

```
SELECT ArtNr, Bezeichnung, Name, Ort
FROM Artikel FULL OUTER JOIN Lieferanten
      ON Artikel.StammLief = Lieferanten.LiefNr
```

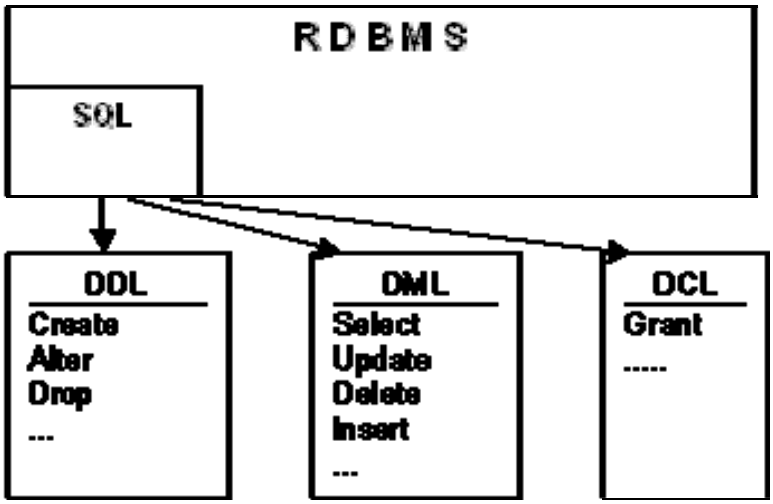
Kurzbeschreibung SQL

SQL – Sprachumfang

In seiner Regel 5 fordert Codd, dass in relationalen Systemen eine Sprache vorhanden sein muss, die umfassend alle wichtigen Anforderungen erfüllt:

- Datendefinition
- Datenmanipulation (als Programm oder interaktiv)
- Definition von Datensichten (Views)
- Integritätsbedingungen kontrollieren
- Autorisierung (Zugang regeln)
- Transaktionskontrolle (start, ende, rollback, ..)

Neben der Möglichkeit von SQL als Query Language werden diese Anforderungen als Data Definition Language (DDL), Data Manipulation Language (DML) und Data Control Language (DCL) erfüllt.



SQL-Funktionen nach erweitertem ANSI-Standard.

Häufig wird auf die explizite Darstellung der DCL verzichtet und die Funktionen werden der DDL/DML zugeordnet.

Kurzbeschreibung SQL I

Definition

SQL (**S**TRUCTURED **Q**UERY **L**ANGUAGE) ist eine international genormte Sprache zur Kommunikation mit relationalen Datenbanksystemen (RDBMS). Man kann heute sagen, dass alle ernstzunehmenden Datenbanksysteme über eine SQL-Schnittstelle verfügen.

Neben den Anweisungen zur Abfrage/Auswahl von Informationen aus einer Datenbank, sind in SQL auch Anweisungen zur Definition und Anlage einer Datenbank, dem Hinzufügen, Entfernen und Verändern einzelner Tabellen, der Vergabe von Zugriffsrechten sowie der Manipulation (einfügen, verändern, löschen) der gespeicherten Daten enthalten.

Die Abfrage einer Datenbank ist über SQL sehr einfach zu bewerkstelligen, da man dazu in der Regel nur drei bis sechs sehr leistungsfähige Anweisungen kennen muss, die nachfolgende kurz vorgestellt werden.

Bei der Nutzung der "Informationsquelle Datenbank" entsteht schon nach kurzer Zeit das Bedürfnis, dass man die Informationsflut durch zusätzliche Auswahl- und Verdichtungskriterien konzentrieren oder bestimmte Informationen arithmetisch oder logisch miteinander kombinieren will. Dazu stellt SQL einen imperativen Programmiersprachen vergleichbaren Satz von praktischen Operationen zur Verfügung, welche die Abfrage erleichtern.

Bei realen Datenbankanwendungen werden die Abfragemöglichkeiten der Endbenutzer i.d.R. über vorgegebene Masken eingeschränkt (und erleichtert). Die für die jeweilige Abfrage erforderlichen SQL-Statements wurden hier von den Programmentwicklern in die Anwendungsprogramme integriert und bleiben so dem Benutzer verborgen. Beim der Programmausführung kommunizieren die jeweiligen Anwendungsprogramme (Clients) über SQL dem entsprechenden Datenbanksystem..

Die sechs Komponenten einer SQL-Abfrage

SELECT [ALL | DISTINCT] *Spaltenausdruck* [, ...]	Zeige mir die Werte aus folgenden Spalten … (Ausgabedaten) Default ist ALL
FROM *Tabellenbezeichner* [, ...]	… aus folgenden Tabelle(n)
[**WHERE** *Bedingung*]	… schränke die Auswahl wie folgt ein …
[**GROUP BY** *Spalte* [, ...]]	Gruppenbildung Verdichte die Ausgabe von allen Zeilen mit gleichem Attributwert zu einer einzigen Ausgabezeile
[**HAVING** Having-Bedingung	Zusätzliche Auswahlbedingung auf Gruppenebene
[**ORDER BY** [[ASC | DESC] Spalte [, ...]] …	sortiere Ausgabe anhand folgender Spalten Default ASC

Datenbanken und Tabellen erzeugen

1. Datenbanken erzeugen (CREATE DATABASE)

Zunächst muss mit dem Befehl **CREATE DATABASE dbname** eine Datenbank erzeugt werden. Dafür müssen Sie die Berechtigung zum Anlegen von Datenbank haben. Wenn Sie Zugang zum Datenbanksystem haben, dürfen Sie Testdatenbanken anlegen. Einzige Voraussetzung: Der Name der Datenbank muss mit den Buchstaben test beginnen. Diese Testdatenbank ist ungeschützt. Es hat sich bewährt, für die Erzeugung von Datenbanken und Tabellen ein Skript zu erstellen und mit dem Befehl: **mysql -u XXXX -p XXXXX < skriptname.sql** (Für XXXX sind Benutzer und Passwort einzusetzen) auszuführen. Weitere Möglichkeiten SQL-Befehle auszuführen finden Sie hier.

2. Tabellen erzeugen (CREATE TABLE)

Zunächst müssen Sie sicherstellen, dass Sie die richtige Datenbank bearbeiten. Mit dem Befehl use dbname legen Sie die Defaultdatenbank fest. Für einfache Tabellen reicht folgende Syntax:

CREATE TABLE [IF NOT EXISTS] tabellenname (spaltenname1, datentyp, option, spaltenname2, datentyp, option, PRIMARY KEY (spaltenname));	CREATE TABLE IF NOT EXISTS kunden (kunr integer DEFAULT '10000' NOT NULL AUTO_INCREMENT, kuname CHAR(30) NOT NULL, PRIMARY KEY (kunr));

Wenn die Tabelle schon existiert kommt es zu einer Fehlermeldung. Dies kann mit dem Zusatz **IF NOT EXISTS** verhindert werden. Gängige Datentypen sind: **INTEGER** für ganze Zahlen, **FLOAT** für Fließkommazahlen, **CHAR(n) und VARCHAR(n)** für Zeichenketten und **DATE** für das Datum.

Mit der Option NOT NULL wird festgelegt, dass ein Eintrag in dieser Spalte vorhanden sein muss. Dies ist eine zwingende Option für den Primärschlüssel. AUTO_INCREMENT fügt in die Spalte eine Zahl ein, die um Eins größer ist als der vorige Wert dieser Spalte.

Beispiel-Skript für die Erzeugung der Datenbank Flotter Flitzer.

Mit dem Befehl DESCRIBE table können Sie überprüfen, ob alles geklappt hat.

Kurzbeschreibung SQL II

Doppelte Werte aus der Ausgabe ausblenden

Schlüsselwort: DISTINCT

Beispiel: Von wieviel verschiedenen Lieferanten SELECT **DISTINCT** Stammlief

wird unser Unternehmen beliefert? from Artikel

Einsatz von Operatoren / Notation von Konstanten

Beim Einsatz von arithmetischen und logischen Funktionen ist es wichtig, zwischen numerischen Ausdrücken und Texten (Strings) zu unterschieden.

- Arithmetische Operatoren und Aggregatfunktionen können i.d.R. nur auf numerische Werte angewendet werden.
- Soll ein Ergebnis mit Nachkommastellen ausgegeben werden, wie z.B. (Menge / 3) müssen die jeweiligen Werte in der Formel mit Dezimalpunkt eingegeben werden.
- Sogenannte Matchcodes (siehe Seite 4) können nur auf Texte angewendet werden.
- **String-Literale und Datums-Literale müssen in einfachen Anführungszeichen notiert werden.**
  Währungsliteralen muss ein $-Zeichen vorangestellt werden.
  z.B.
  INSERT INTO Artikel (ArtBezeichnung, Preis)
  VALUES ('Persil Subber', **$**12.80)
  WHERE ArtBezeichnung = 'Persil'

Vergleichsbedingungen (log. Operatoren)

Operator	Bedeutung	Beispiel
>, <	größer, kleiner	
>=	größer gleich	ArtNr >= 100
<=	kleiner gleich	
=	gleich	Name = 'Meier'
<>	ungleich	Saldo <> 0.00
AND	Konjunktion	ArtNr > 100 AND ArtNr < 500
OR	Disjunktion	Name = 'A%' OR Name = 'B%'
NOT	Negation	NOT (Ort = 'Berlin')
BETWEEN	Bereichsangabe	WHERE KuNr BETWEEN 100 AND 500
LIKE	einfache Mustervergleiche über Match-codes	*(Siehe nachfolgende Beispiele zu "Patternmatching".)*
IN	Bezug auf eine fest	SELECT Name, Ort FROM Kunden

	vorgegebene Menge Bezug auf das Ergebnis einer Unterabfrage	WHERE Ort in ('Berlin', 'Bonn', 'Hamburg', 'Wien') SELECT KuNr, Name, Ort FROM Kunden WHERE KuNr IN (SELECT KuNr FROM Abverkauf WHERE ReBetrag > $10000) (Siehe dazu auch Seite 6 "Unterabfragen")

Relationale Datenbanken
Kurzbeschreibung SQL III

Pattern Matching (Jokerzeichen)

Sogenannte 'Matchcodes' können nur für Texte verwendet werden!

Matchcodes können zu beliebigen sinnvollen Mustern kombiniert werden!

Matchcode	steht für	Beispiel	Ergebnis
%	beliebige Folge von Zeichen wozu auch NULL gehört.	WHERE Name LIKE 'B**%**' LIKE '%er' LIKE 'B%mann' LIKE '%'	Alle Namen, die … mit "B" beginnen … mit "er" enden … die mit "B" beginnen **und** mit "mann" enden. Alle Namen die ein "ü" enthalten.
_	ein beliebiges einzelnes Zeichen	WHERE Name LIKE '_al' LIKE 'S__' LIKE 'Ma_er'	Alle dreistelligen Namen, die mit "al" enden. Alle dreistelligen Namen, die mit "S" beginnen. Alle, 5-stelligen Namen die mit "Ma" beginnen und mit "er" enden.
[A-F]	ein beliebiges einzelnes Zeichen innerhalb des angegebenen Bereichs bzw. der angegebenen Menge	WHERE Name LIKE '[A-H]**%**' LIKE '[A-H]**%**ing' LIKE '_[KR]%' `LIKE '[A-FKRT]%'`	Alle Namen, die … "A" bis "H" beginnen … dto. und mit "ing" enden. … die an zweiter Stelle ein "K" oder "R" enthalten und beliebig enden. … die mit "A" bis "F", "K", "R" oder

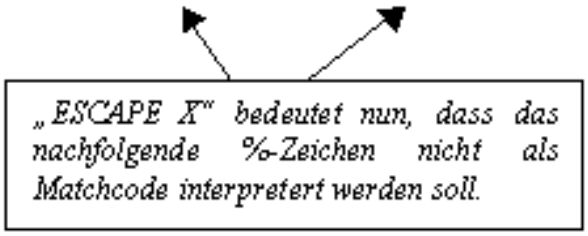
			"T" beginnen.
[^A-F]	ein beliebiges einzelnes Zeichen außerhalb des angegebenen Bereiches	WHERE Name LIKE '[^A-H]**%**' LIKE '_[^C]%'	Alle Namen, die … nicht "A" bis "H" beginnen (Äquivalent zu WHERE NAME NOT LIKE '[A-H]**%**' Alle, die an zweiter Stelle kein "C" haben.

Anmerkung:

Wollen Sie z.B. in einer Tabelle Finanzen alle Einträge der Spalte "Zinsart" suchen, welche ein %-Zeichen enthalten, stehen Sie vor einem Problem. Sie können das Zeichen % nicht verwenden, da es als Matchcode interpretiert wird.

Dieses kann über die sog. ESCAPE-Option wie folgt gelöst werden:

```
SELECT * FROM Finanzen
WHERE Zinsart LIKE '%X%-Satz%' ESCAPE 'X'
```



Oder

Alle Namen suchen, welche ein "Underline" enthalten.

```
SELECT * FROM Kunden
WHERE Name LIKE '%#_%' ESCAPE '#'
```

SQL-Befehle ausführen

Es gibt sehr viele unterschiedliche Möglichkeiten SQL-Befehle auszuführen. Für alle Möglichkeiten müssen sich sicherstellen, dass Sie die erforderlichen Zugriffsrechte besitzten. Im folgenden werden die wichtigsten aufgeführt:

1. Kommandointerpreter

Hierfür müssen Sie auf dem Datenbankserver mit einer Console angemeldet sein. Mit dem Befehl mysql -u username -p passwort bekommen Sie Zugang zum Kommandointerpreter. Mit use datenbankname wählen Sie die gewünschte Datenbank. Jetzt können Sie die SQL-Befehle direkt eintippen und mit RETURN ausführen. Beachten Sie, dass jeder Befehl mit einem Semikolon abgeschlossen werden muss.

2. Skripte erstellen und ausführen

Auch für diese Möglichkeit müssen Sie mit einer Console angemeldet sein. Zunächst erstellen Sie ein Skript mit den gewünschten SQL-Befehlen. Stellen Sie sicher, dass das Skript aufrufbar ist. Sie können das Skript auf einem anderen Rechner erzeugen und dann mit FTP in Ihren Benutzerbereich auf dem Datenbankserver übertragen. Mit dem Befehl: **mysql -u username -p passwort < skript.sql** können Sie nun den Befehl ausführen. Bei diesem Verfahren sind die SQL-Befehle immer vorhanden und lassen sich leicht verändern.

3. phpMyAdmin benutzen

Dieses Programm besteht aus einer Reihe von PHP-Skripts mit deren Hilfe eine fast vollständige Administration einer Datenbank möglich ist und somit auch leicht SQL-Befehle erzeugt werden können. Rufen Sie zunächst das Programm auf und melden Sie sich bei DBMS an. Falls Sie eine existierende Datenbank bearbeiten wollen, wählen Sie diese links aus der Liste aus. Jetzt können Sie durch die Anwahl von SQL in der oberen Leiste einen Editor öffnen mit dem Sie SQL-Befehle eingeben und mit OK ausführen können.

Wenn Sie eine Testdatenbank einrichten wollen, so wählen Sie die Datenbank test aus der Liste aus, klicken dann SQL an und geben Sie im Editor den Befehl "create database test_meinname;" an. Drücken Sie auf "OK" und schon ist die Datenbank eingerichtet und Sie können Sie in der Datenbankliste anwählen und bearbeiten.

Kurzbeschreibung SQL IV

Arithmetische Operatoren

Operator	Bedeutung	Beispiel
+	Addition	EK – Skonto + Bezugskosten
–	Subtraktion	
*****	Multiplikation	(VK – EK) * 100 / VK
/	Division	
%	Modulo *(nur für ganze Zahlen)*	(Zeilen % 60) AS 'Seiten'

Achtung:

Die Anzahl der ausgegebenen Nachkommastellen ist von den beteiligten Typen und den jeweiligen Literalen abhängig!

Gegeben seien folgende Felder

Menge : Integer (mit dem aktuellen Wert 10)

Preis : Money (mit dem aktuellen Wert 12.50)

Gewicht : Float (mit dem aktuellen Wert 2.0)

Notation	liefert dieses Ergebnis
Menge / 2	5
Menge / 2.0	5.0000000
Menge / $2.0	5.00
Preis / Menge	1.25
Preis / 5	2.50
Preis / 5.0	2.5000000
Preis / $5.0	2.50
Gewicht / 3	0.666667
Gewicht / 2	1.0

Kurzbeschreibung SQL V

Aggregatfunktionen

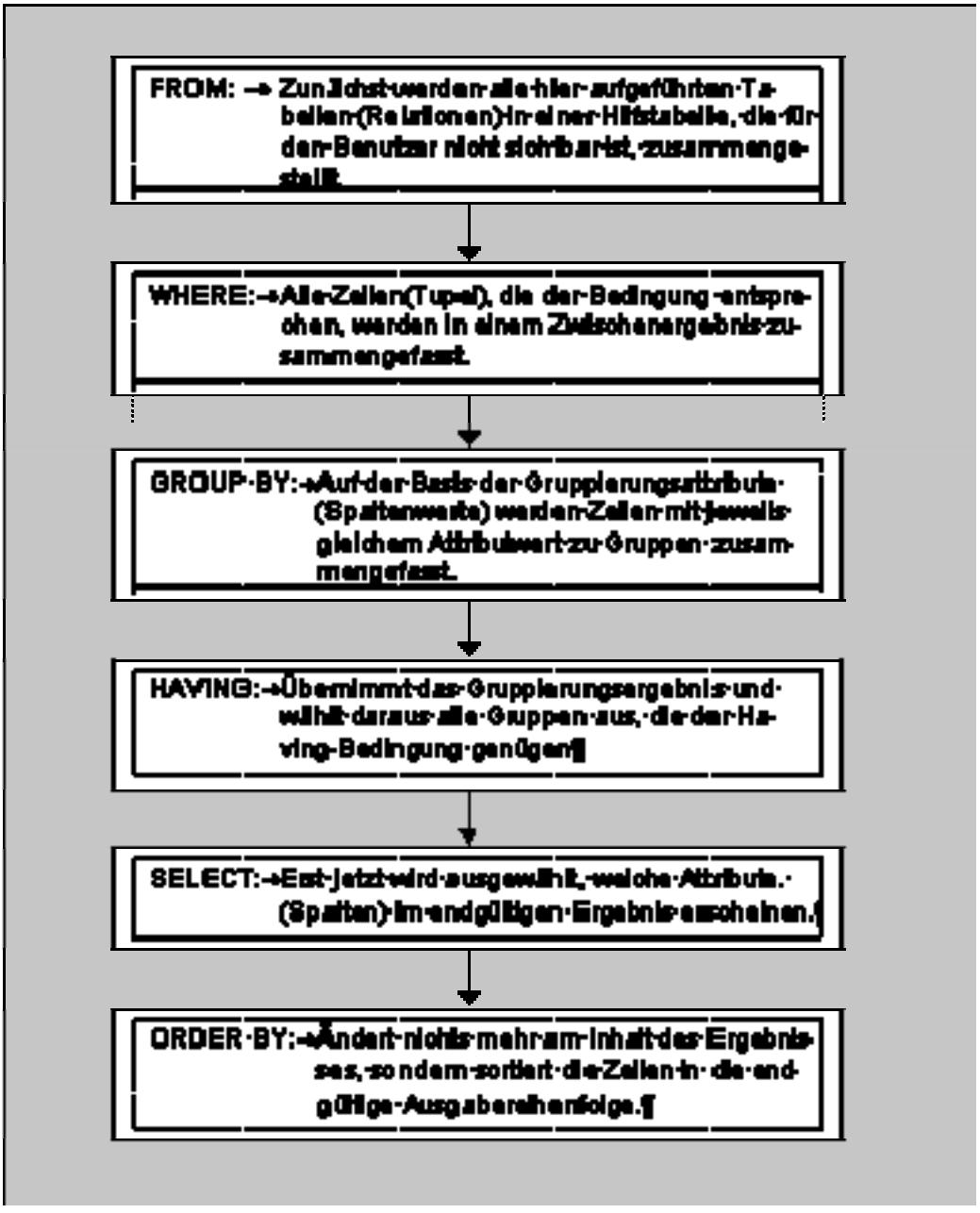
Aggregatfunktionen dienen zur Zusammenfassung oder Verdichtung von Ergebnissen und arbeiten jeweils auf allen Zeilen der entsprechenden Tabelle bzw. der entsprechenden Untergruppe. Sie können nur in 'select'- und 'having'-Anweisungen benutzt werden.

Eine HAVING-Anweisung kann i.d.R. nur im Verbindung mit einer Aggregatfunktion benutzt werden.

Operator	Bedeutung	Beispiel
count(*)	Liefert die Anzahl der Tupel des jeweil. Zwischenergebnisses	SELECT **count(*)** AS 'Artikelzahl' FROM Bestand
sum()	Liefert die Summe der Werte einer Spalte	SELECT **sum**(Sollsaldo) FROM Kunden
min() **max()**	Liefert den kleinsten / größten Wert einer Spalte.	SELECT ArtNr, ArtBez, **min**(Bestand) FROM Lager
avg()	Liefert das arithmetische Mittel (Durchschnittswert) über alle Werte in der jeweiligen Spalte.	SELECT **avg**(Umsatz) AS 'DuUmsatz' FROM Kunden

Kurzbeschreibung SQL VI

Prinzipielle Abarbeitungsreihenfolge der einzelnen Komponenten



Kurzbeschreibung SQL VII

Unterabfragen "INNER SELECT"

Hinter Bedingungsoperatoren in WHERE- oder HAVING-Ausdrücken dürfen Unterabfragen verwendet werden. Einschränkungen:

Die innere SELECT-Anweisung darf nur einen Tabellenausdruck enthalten
Die ORDER BY – Klausel darf innerhalb einer Unterabfrage nicht verwendet werden.

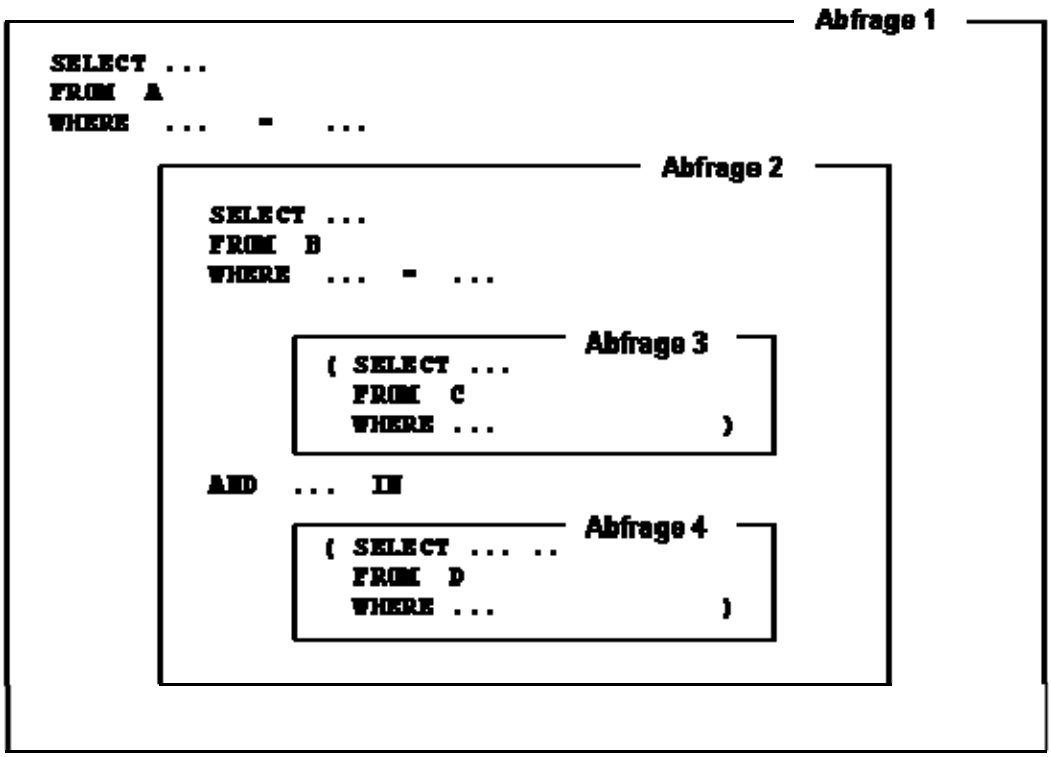
Beispiel:

Es sollen alle Kunden mit Kundennummer, Familienname und Ausweisnummer aus einer fiktiven Video-Verleih-Datenbank herausgesucht werden, die Horrorvideos ausgeliehen haben und die in der Tabelle Gewaltverbrecher mit einer Vorstrafe registriert sind.

```
SELECT KundenNr, Familienname, Ausleihe.AusweisNr
FROM Kunden
WHERE KundenNr =

        (SELECT Ausleihe.KundenNr
        FROM Ausleihe
        WHERE Ausleihe.VideoNr =

                (SELECT VideoNr
                FROM Katalog
                WHERE Genre = 'Horror' ) )

AND Ausleihe.AusweisNr IN

        (SELECT Personenkennzeichen
        FROM Gewaltverbrecher
        WHERE Vorstrafen > 0 )
```

Gültigkeitsbereich von Spalten (Attributen) bei Unterabfragen

Ein wichtiger Gesichtspunkt bei Unterabfragen ist die Frage, welche Datenobjekte (Datenfelder) in den jeweiligen Unterabfragen gültig (bekannt) sind und verwendet werden dürfen.



Regel

Attribute (Spalten, Datenfelder) aus den jeweils äußeren Abfrageblöcken dürfen im inneren Abfrageblock an jeder Stelle verwendet werden - aber nicht umgekehrt - .

So können im Abfrageblock 1 'Abfrage 1' nur Attribute aus der Tabelle A verwendet werden. Im Abfrageblock 2 können Attribute aus den Tabellen A und B verwendet werden und im Abfrageblock 3 Attribute aus A, B und C.

Werden in den Tabellen Attribute mit gleichen Attributbezeichnern verwendet, z.B. KundenNr, müssen im jeweiligen Abfrageblock qualifizierte Bezeichner (A.KundenNr, B.KundeNr, C.KundenNr) benutzt werden.

Man spricht bei der Frage des Gültigkeitsbereiches auch von der Reichweite eines Attributes in einer Tabelle. So haben in obigem Beispiel die Attribute der Tabelle A die Reichweite Abfrage 1, Abfrage 2, Abfrage 3 und Abfrage 4. Die Attribute der Tabelle B haben lediglich die Reichweite Abfrage 1 und Abfrage 3.

Beispieldatenbank Flotter Flitzer

Modellierung

- Anforderungsdefinition
- Entity-Relationship-Model
- Anwendung der Abbildungsregeln
- Integritätsbedingungen
- Zugangsberechtigungen

Implementierung

- SQL Script zum Erzeugen des Datenmodells (MySql)
- SQL Script zum Erzeugen des Datenmodells (Informix)
- SQL-Script zur Vergabe der Zugriffsrechte (MySql)
- SQL-Script zur Vergabe der Zugriffsrechte (Informix)
- Skripte für eine Benutzungsschnittstelle der Tabelle Kunden (PHP und HTML)

Online Versionen

- Flotter Flitzer ohne Zugangskontrolle (Nur intern) (Autor: Maximillian Möller)
- Flotter Flitzer ohne Zugangskontrolle (Autor: Maximillian Möller)

Anforderungsdefinition

Anforderungsdefinition für Autovermietung Flotter Flitzer

Aufgabenstellung

Die Autovermietung "Flotter Flitzer" ist auf die Vermietung luxuriöser Sportwagen spezialisiert. Dieser Wagentyp bildet das Hauptangebot, daneben werden aber auch Mittelklassewagen verliehen.

Nunmehr soll die Verwaltung auf EDV umgestellt werden. Vorgesehen ist eine relationale Datenbank, mit der zunächst die Vermietung und die Verwaltung des Fuhrparks (die Angaben über den Fahrzeugtyp und die Dokumentation von Reparaturen) abgewickelt werden können.

Anforderungen an die Benutzerinnen und Benutzer

Bei der Abwicklung der *Ausleihe* sind zwei Arbeitsbereiche zu unterscheiden:

- **die Ausleihe**: Hier werden die Personalien der Kundinnen und Kunden, die ausgegebenen PKW und der Kilometerstand erfasst.
- **die Rückgabe**: Die Erfassung der zurückgegebene PKW und der Kilometerstand.

Für die *Verwaltung* des Fahrzeugbestandes sind ebenfalls zwei Bereiche einzurichten:

- **die PKW-Verwaltung**: Hier werden die PKW aufgenommen bzw. abgemeldet.
- **die PKW-Wartung**: Die Dokumentation der Reparaturen und die Verwaltung von Informationen über die Werkstätten.

Allen Bereichen müssen sämtliche Angaben zur Verfügung stehen, die für die Erfüllung ihrer jeweiligen Arbeitsaufgaben notwendig sind. Die Daten- und Systempflege verbleibt bei der Systemverwaltung.

Die Erlaubnis, Löschungen vorzunehmen, ist wie folgt geregelt:

- Kundinnen, Kunden und PKW dürfen nur von der Systemverwaltung gelöscht werden.
- Hersteller dürfen von der PKW- und Systemverwaltung gelöscht werden.
- Werkstätten dürfen von der Systemverwaltung und der Wartung gelöscht werden.
- Daten zur Ausleihe und zur Reparatur dürfen von niemanden gelöscht werden.

Anforderungen an die Arbeitsweise

Bei den Angestellten der Autovermietung kann auf keine EDV-Erfahrung zurückgegriffen werden. Die Benutzung muss daher möglichst einfach und fehlertolerant sein. Online-Hilfen sind zu den wesentlichen Punkten einzubauen. Die Eingaben sind so zu gestalten, dass auch bei einem hektischen Massenbetrieb eine schnelle, fehlerarme Erfassung möglich ist.

Anforderungen an den Leistungsumfang

Die Ausleihe soll mit Hilfe des Programms die Kundinnen und Kunden registrieren und ihnen einen PKW zuteilen, dabei ist auch der aktuelle Tachostand einzugeben. Alle Ausleihvorgänge werden automatisch durchnummeriert. Bei der Rückgabe des Autos werden das Rückgabedatum und der aktuelle Tachostand erfasst.

Als Daten der Kundschaft werden verpflichtend Name, Anschrift, Datum des Führerscheinerwerbs und Geburtsdatum gespeichert. Für die PKW werden verpflichtend Kennzeichen, Hersteller mit Namen, Anschrift und Kontaktperson, Modell mit Namen, Leistung, Hubraum, Klimaanlage, Schiebedach, Länge und Breite und das Datum der Erstzulassung erfasst. Es werden nur Neuwagen direkt beim Hersteller beschafft. Werden PKW aus dem Fuhrpark entfernt, so wird nur die Abmeldung in die Datenbank eingetragen. Der PKW wird erst am Ende des Geschäftsjahres vollständig gelöscht.

Von den Werkstätten werden Name, Anschrift und Kontaktperson gespeichert. Als Daten der Reparatur sind Kennzeichen des entsprechenden PKW, Name der Werkstatt, Reparaturdatum sowie Art und Dauer der Reparatur vorzusehen.

Um Manipulationen vorzubeugen, sind alle Vorgänge zu protokollieren. Die Protokolldatei darf nur von der Systemverwaltung eingesehen und von niemanden gelöscht werden.

Als Geschäftsjahr gilt das Kalenderjahr. Jeweils am Ende des Geschäftsjahres werden alle fortlaufenden Daten, in diesem Fall alle abgeschlossenen Ausleih- und Reparaturvorgänge, aus der Datenbank herausgenommen und für Steuerzwecke auf einem Magnetband gespeichert.

Anforderungen an mögliche Ausbaustufen

In der ersten Ausbaustufe werden die finanziellen Transaktionen nicht erfasst. Als weitere Ausbaustufe soll die Abwicklung des gesamten Zahlungsverkehrs einbezogen werden.

Auch die Verwaltung des Personals soll in einer späteren Ausbaustufe realisiert werden.

Anforderungen an das Fehlerverhalten

Um einen reibungslosen Betrieb zu gewährleisten, sollen so weit wie möglich Fehleingaben abgewiesen werden. Auch darf eine Fehlbedienung nicht zum Absturz des Systems und erst recht nicht zum Datenverlust führen. Im Fehlerfall sind aussagekräftige, kontextabhängige Online-Hilfen zu geben.

Anforderungen an die Qualität

Es ist anzustreben, ein möglichst fehlerarmes und wartungsfreundliches Programmsystem zu erstellen. Das Programm soll mit geringem Aufwand auf andere Hard- und Softwareplattformen übertragen werden können. Deshalb ist es notwendig, so weit dies möglich ist, herstellerunabhängige Standards zu benutzen.

Anforderungen an den Datenschutz

Die Kundinnen und Kunden werden mit Abschluss des Vertrages darauf hingewiesen, dass ihre Daten zur Abwicklung der Vermietung elektronisch gespeichert werden. Eine Weitergabe der Daten an Dritte wird ausgeschlossen.

Anforderungen an die Ergonomie

Das Programm soll möglichst optimal ergonomischen Gesichtspunkten genügen. Dies gilt insbesondere für die schnelle Eingabe von Daten an der Ausleihe. Alle Masken sind einheitlich zu gestalten und die Bildschirmformulare sollten mit ggf. vorhandenen Papiervorlagen im Aufbau identisch sein. Alle Masken sollen mindestens Angaben zum Standort im Programm, zum Verlassen der Maske und zum Abbrechen enthalten.

Anforderungen an die Dokumentation

Die Struktur des Programms, inklusive der verwendeten Datenstruktur, der Masken und der kommentierten Programmlistings werden in einem Wartungshandbuch dokumentiert. Das System soll so benutzungsfreundlich gestaltet werden, so dass kein Handbuch nötig ist.

Anforderungen an die Basismaschine

Das System soll auf gängigen aktuellen Rechnern unabhängig vom Betriebssystem implementiert werden können. Wünschenswert wäre eine Wartungsmöglich über das Internet mit gängigen Browsern.

Entity-Relationship-Model

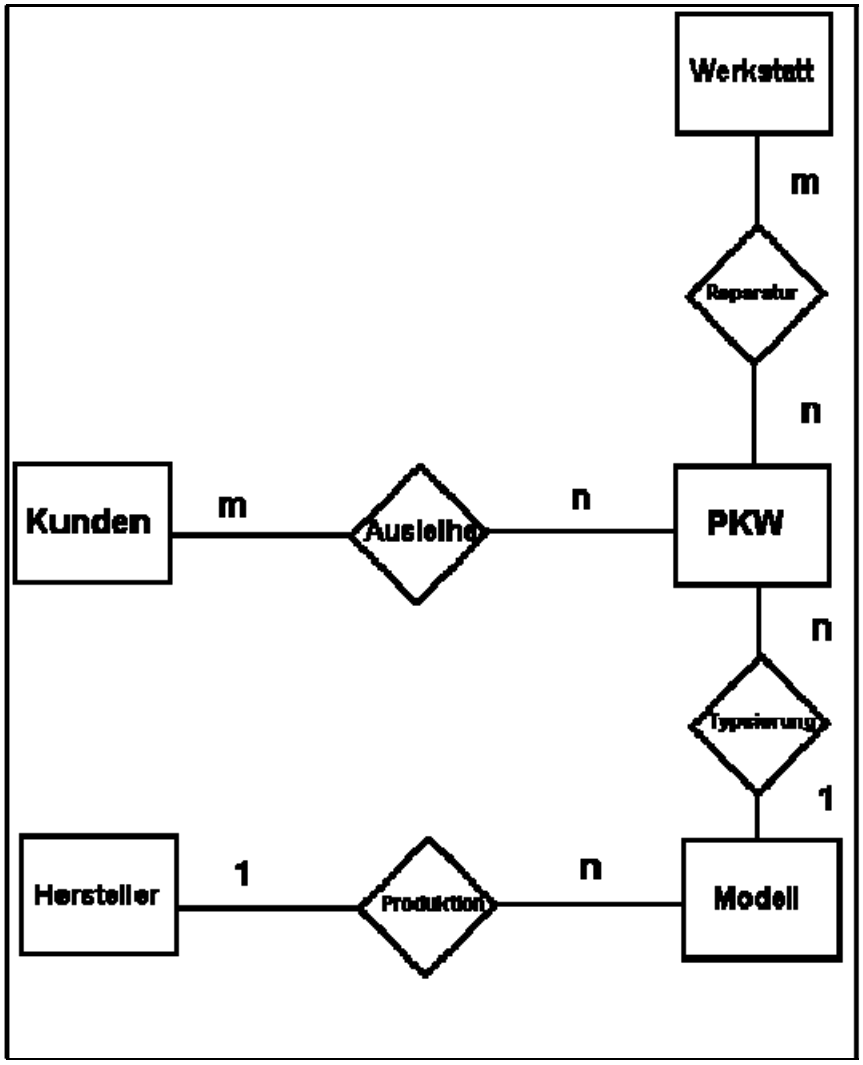


Abbildung des ERM in Tabellen

Anhand der Abbildungsregeln wird das ERM der Autovermietung Flotter Flitzer in Tabellen überführt.

Nach der 1. Regel werden alle Entitätstypen zu einer Tabelle:

Kunden (Nummer, Name, Vorname, Straße, PLZ, Ort, Geburtsdatum, Telefon, Datum des Führerscheinerwerbs)

Pkw (Kennzeichen, Modell, Erstzulassung, Datum der Abmeldung, Schiebedach, Klimaanlage)

Werkstatt (Name, Straße, PLZ, Ort, Telefon, Kontaktperson)

Modell (Name, Hersteller, Leistung, Länge, Breite, Hubraum)

Hersteller (Name, Straße, PLZ, Ort, Telefon, Kontaktperson)

Nach der 2. Regel wird jede m:n-Beziehung eine Tabelle:

Ausleihe (fortlaufende Nummer, Kundennummer, Kennzeichen, Ausleihdatum, Anfangskilometer, Rückgabedatum, Endkilometer)

Reparatur (Werkstatt, Kennzeichen, Reparaturdatum, Dauer, Art)

Die 3. Regel findet keine Anwendung, da es keine 1:n oder 1:1-Beziehung mit Attributen gibt.

Regel 4.

Nach der 4. Regel wird eine 1:n oder 1:1-Beziehung die freigestellt ist zu einer eigenen Tabelle. Da beide 1:n-Beziehungen zwingend sind, findet auch diese Regel keine Anwendung. Die Beziehungen sind Typisierung und Produktion sind zwingend.

Da die Entität PKW (E2) zwingendes Mitglied der Entität Modell (E1) ist, wird die Beziehung in den Tabellen dadurch hergestellt, in dem der Primärschlüssel von E1 (Modell.Name) in PKW (E2) als Fremdschlüsselattribut aufgenommen wird. Nach dem gleichen Muster wird bei der Beziehung Produktion verfahren.

Pkw (Kennzeichen, Modell, Erstzulassung, Datum der Abmeldung, Schiebedach, Klimaanlage, Modell.Name)

Modell (Name, Hersteller, Leistung, Länge, Breite, Hubraum, Hersteller.Name)

Integritätsbedingungen

Wertebereichsintegrität

kunden

Feldname	Beschreibung	Datentyp	Wertebereich	Null
kunr	Kundennummer	SERIAL	Fortlaufende Nummer mit 1000 beginnend	nein
kuname	Name	CHAR	30 Stellen	nein
kuvorname	Vorname	CHAR	20 Stellen	nein
kustrasse	Straße	CHAR	30 Stellen	nein
kuplz	Postleitzahl	CHAR	8 Stellen, 3 Buchstaben für das Land, Rest nur Ziffern	nein
kuort	Ort	CHAR	30 Stellen	nein
kutelefon	Telefon mit Vorwahl	CHAR	30 Stellen	ja
kugebdat	Geburtsdatum	DATE	mindestens vor 18 Jahren	nein
kudatfschein	Datum des Führerscheinerwerbs	DATE	mindestens 18 Jahre nach dem Geburtsdatum	nein

pkw

Feldname	Beschreibung	Datentyp	Wertebereich	Null
pkennzeichen	Polizeiliches Kennzeichen	CHAR	11 Stellen	nein
moname	Name des Modells	CHAR	35 Stellen	nein
perstzulassung	Datum der Erstzulassung	DATE	nicht älter als 10 Jahre, nicht nach der Abmeldung	ja
peabmeldung	Datum der Abmeldung	DATE	älter als Erstzulassung	ja
pschiebedach	Schiebedach vorhanden	CHAR	1 Stelle 'j' oder 'n'	ja
pklima	Klimaanlage vorhanden	CHAR	1 Stelle 'j' oder 'n'	ja

modelle

Feldname	Beschreibung	Datentyp	Wertebereich	Null
moname	Name des Modells	CHAR	35 Stellen	nein
hename	Name des Herstellers	CHAR	35 Stellen	nein
moleistung	Leistung in KW	INTEGER	positiv 60 - 500 KW	ja

mohubraum	Hubraum in ccm	INTEGER	positiv 500 - 5000 ccm	ja
molaenge	Länge des PKW in cm	INTEGER	positiv 300 - 1200 cm	ja
mobreite	Breite des PKW in cm	INTEGER	positiv 200 - 400 cm	ja

hersteller

Feldname	Beschreibung	Datentyp	Wertebereich	Null
hename	Name des Herstellers	CHAR	35 Stellen	nein
hestrasse	Straße	CHAR	30 Stellen	ja
heplz	Postleitzahl	CHAR	5 Stellen	ja
heort	Ort	CHAR	30 Stellen	ja
hetelefon	Telefon mit Vorwahl	CHAR	20 Stellen	ja
hekontakt	Kontaktperson beim Hersteller	CHAR	20 Stellen	ja

werkstatt

Feldname	Beschreibung	Datentyp	Wertebereich	Null
wename	Name der Werkstatt	CHAR	35 Stellen	nein
westrasse	Straße	CHAR	30 Stellen	ja
weplz	Postleitzahl	CHAR	5 Stellen	ja
weort	Ort	CHAR	30 Stellen	ja
wetelefon	Telefon mit Vorwahl	CHAR	20 Stellen	ja
wekontakt	Kontaktperson bei der Werkstatt	CHAR	20 Stellen	ja

ausleihe

Feldname	Beschreibung	Datentyp	Wertebereich	Null
annr	Ausleihnummer	SERIAL	Fortlaufende Nummer mit 1 beginnend	nein
pkennzeichen	Polizeiliches Kennzeichen	CHAR	11 Stellen	nein
kunr	Kundennummer	INTEGER	> 999	nein
auleiausdat	Ausleihdatum	DATE	Datum des Ausleihtags, nicht vor dem aktuellen Datum	nein
auanfangkm	Kilometerstand beim Ausleihen	INTEGER	positiv < 100000	nein

auleirueckdat	Rückgabedatum	DATE	später als das Ausleih-datum, nicht vor dem aktuellen Datum	ja
auendkm	Kilometerstand bei der Rückgabe	INTEGER	positiv < 100000	ja

reparatur

Feldname	**Beschreibung**	**Datentyp**	**Wertebereich**	**Null**
pkennzeichen	Polizeiliches Kennzeichen	CHAR	11 Stellen	nein
wename	Name der Werkstatt	CHAR	35Stellen	nein
redatum	Datum der Reparatur	DATE		ja
reart	Art der Reparatur	CHAR	35 Stellen	ja
redauer	Dauer der Reparatur in Tagen	INTEGER	positiv < 100	ja

Intra-relationale Integrität

Im Folgenden werden nur die Maßnahmen zur Sicherung eines eindeutigen Schlüssels dokumentiert:

Tabellenname	**Maßnahmen**
kunden	Fortlaufende Kundennummern mit 1000 beginnend durch das Datenbanksystem
hersteller	durch einen eindeutigen Index für den Herstellernamen
modelle	durch einen eindeutigen Index für den Modellnamen
pkw	durch einen eindeutigen Index für das Kennzeichen
werkstatt	durch einen eindeutigen Index für den Werkstattnamen
ausleihe	Fortlaufende Nummerierung aller Ausleihvorgänge mit 1 beginnend durch das Datenbanksystem
reparatur	durch einen eindeutigen Index für die Fremdschlüssel Kennzeichen und Werkstattname

Die Integritätsbedingungen, die sich auf die Beziehung einzelner Felder innerhalb einer Relation beziehen, wurden aus Gründen der besseren Darstellungsmöglichkeiten bei der Wertebereichsintegrität dokumentiert.

Referentielle Integrität

Die hierfür notwendigen Maßnahmen betreffen im wesentlichen die Relationen Ausleihe und Reparatur.

Bei der **Ausleihe** muss sichergestellt werden, dass nur Ausleihen vorgenommen werden, bei denen alle Daten über Kunden und PKW vorliegen. Das System muss jede Ausleihe ablehnen, bei denen kein Verweis auf die Relation Kunden und PKW möglich ist. Da alle Ausleihdaten ein Jahr aktuell im System gehalten werden, dürfen Kunden und PKW nur einmal im Jahr, am Ende des Geschäftsjahres gelöscht werden.

Ähnliches gilt für die Relation **Reparatur**. Auch hier kann nur eine Reparatur eingegeben werden, die einen eindeutigen Verweis auf die Relation PKW und Werkstatt zulässt. Löschungen von Werkstätten sind nur am Ende des Geschäftsjahres erlaubt.

Weniger rigide sollten die Beziehungen zwischen PKW, Modelle und Hersteller gehandhabt werden. So kann es in der Praxis vorkommen, dass bereits ein PKW ausgeliehen wird, von dem noch nicht alle genauen Daten vorliegen. Deshalb soll es möglich sein, in der Relation PKW Eintragungen vorzunehmen, obwohl die entsprechenden Verweise in den Relationen Hersteller und Modelle fehlen.

Zugriffsrechte

Datenbankadministration

Das Recht der Datenbankadministration erhält nur der Ersteller bzw. die Erstellerin, in diesem Fall **"systff".**

Zugang zum Datenbanksystem und zur Datenbank Flotter Flitzer

Der Allgemeinheit wird das Zugriffsrecht entzogen. Vergeben wird es an die folgenden Bereiche:

- ausleihe
- rueck
- kuverw
- fuhrpark
- wartung und
- systff

Zugriff auf Tabellen

kunden

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	ja	ja	ja	nein
rueck	ja	nein	nein	nein
kuverw	ja	ja	ja	nein
fuhrpark	nein	nein	nein	nein
wartung	nein	nein	nein	nein

fuhrpark

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	ja	nein	nein	nein
rueck	ja	nein	nein	nein
kuverw	ja	nein	nein	nein
fuhrpark	ja	ja	ja	nein
wartung	ja	nein	nein	nein

werkstatt

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	nein	nein	nein	nein
rueck	nein	nein	nein	nein
kuverw	nein	nein	nein	nein
fuhrpark	ja	nein	nein	nein
wartung	ja	ja	ja	ja

hersteller

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	ja	nein	nein	nein
rueck	ja	nein	nein	nein
kuverw	nein	nein	nein	nein
fuhrpark	ja	ja	ja	ja
wartung	ja	nein	nein	nein

modelle

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	ja	nein	nein	nein
rueck	ja	nein	nein	nein
kuverw	nein	nein	nein	nein
fuhrpark	ja	ja	ja	ja
wartung	ja	nein	nein	nein

ausleihe

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	ja	ja	nein	nein
rueck	ja	ja	ja	nein
kuverw	ja	nein	nein	nein
fuhrpark	nein	nein	nein	nein
wartung	nein	nein	nein	nein

reparatur

Name	SELECT	INSERT	UPDATE	DELETE
ausleihe	nein	nein	nein	nein
rueck	nein	nein	nein	nein
kuverw	nein	nein	nein	nein
fuhrpark	ja	nein	nein	nein
wartung	ja	ja	nein	nein

Zugriff auf Spalten

Es werden keine gesonderten Rechte für einzelne Attribute vergeben.

Views

Es werden keine Views erzeugt.

Autovermietung Flotter Flitzer

SQL-Script zum Erzeugen der Datenbank und der Tabellen (MYSQL)

```
# Skriptname create_ff.sql
#
# Funktion Erstellt die Tabellen fuer die Verwaltung der
# Autovermietung Flotter Flitzer
#
# Autor: Johann Penon (BICS)
#
# DBMS: MySql 4.X
#
# erstellt am 30. Oktober 2005
#
# Aufruf:  mysql -u XXXX -p XXXXX < create_ff.sql (Für XXXX sind Benutzer
und Passwort einzusetzen).
#
# Die Fremdschlüsselbefehle (FOREIGN KEY) wurden von der Informix-Version
übernommen,
# haben aber keine Wirkung bei MySql.
#
# drop database FF; (Kommentarzeichen wegnehmen, wenn die Datenbank bereits
existiert)
CREATE DATABASE FF;
use FF;

CREATE TABLE IF NOT EXISTS kunden
(
        kunr int (5) DEFAULT '10000' NOT NULL AUTO_INCREMENT,
        kuname CHAR(30) NOT NULL,
        kuvorname CHAR(20) NOT NULL,
        kustrasse CHAR(30) NOT NULL,
        kuplz CHAR(08) NOT NULL,
        kuort CHAR(30) NOT NULL,
        kutelefon CHAR(20),
        kugebdat DATE NOT NULL,
        kudatfschein DATE NOT NULL,
        PRIMARY KEY (kunr)
);
```

```
CREATE TABLE IF NOT EXISTS hersteller
(
        hename CHAR(35) NOT NULL,
        hestrasse CHAR(30),
        heplz CHAR(05),
        heort CHAR(30),
        hetelefon CHAR(20),
        hekontakt CHAR(20),
        PRIMARY KEY (hename)
);

CREATE TABLE IF NOT EXISTS modelle
(
        moname CHAR(35) NOT NULL,
        hename CHAR(35) NOT NULL,
        moleistung SMALLINT,
        mohubraum SMALLINT,
        molaenge SMALLINT,
        mobreite SMALLINT,
        PRIMARY KEY (moname),
        FOREIGN KEY(hename) REFERENCES hersteller(hename)
);

CREATE TABLE IF NOT EXISTS pkw
(
        pkennzeichen CHAR(11) NOT NULL,
        moname CHAR(35) NOT NULL,
        perstzulassung DATE,
        peabmeldung DATE,
        pschiebedach CHAR(1),
        pklima CHAR(1),
        PRIMARY KEY (pkennzeichen),
        FOREIGN KEY(moname) REFERENCES modelle(moname)
);

CREATE TABLE IF NOT EXISTS werkstatt
(
        wename CHAR(35) NOT NULL,
        westrasse CHAR(30),
        weplz CHAR(05),
        weort CHAR(30),
        wetelefon CHAR(20),
        wekontakt CHAR(20),
        PRIMARY KEY (wename)
);
```

```
CREATE TABLE IF NOT EXISTS ausleihe
(
        aunr int (6) DEFAULT '100000' AUTO_INCREMENT,
        pkennzeichen CHAR(11) NOT NULL,
        kunr INTEGER NOT NULL,
        auleiausdat DATE NOT NULL,
        auanfangkm INTEGER NOT NULL,
        auleirueckdat DATE,
        auendkm INTEGER,
        PRIMARY KEY (aunr),
        FOREIGN KEY(pkennzeichen) REFERENCES pkw(pkennzeichen),
        FOREIGN KEY(kunr) REFERENCES kunden(kunr)
);


CREATE TABLE IF NOT EXISTS reparatur
(
        pkennzeichen CHAR(11) NOT NULL,
        wename CHAR(35) NOT NULL,
        redatum DATE NOT NULL,
        reart CHAR(30),
        redauer SMALLINT,
        PRIMARY KEY (pkennzeichen, wename, redatum),
        FOREIGN KEY(pkennzeichen) REFERENCES pkw(pkennzeichen),
        FOREIGN KEY(wename) REFERENCES werkstatt(wename)
)
```

Autovermietung Flotter Flitzer

Script zum Erzeugen der Datenbank und der Tabellen (Informix)

```
{ Skriptname    create.sql                                          }
{ Fundort       scripts/create.sql                                  }
{                                                                   }
{ Funktion      Erstellt die Tabellen fuer die Verwaltung der       }
{               Autovermietung Flotter Flitzer                      }
{                                                                   }
{ Autoren       Johann Penon (BICS), Hermann Peltzer (BICS)         }
{                                                                   }
{ DBMS          Informix 4.1                                        }
{                                                                   }
{ erstellt am   18. September 1995                                  }

CREATE DATABASE ff WITH LOG IN "/schule/anwen/db/ff/log1.log";
```

```
CREATE TABLE kunden
(
        kunr                    SERIAL(1000),
        kuname                  CHAR(30) NOT NULL,
        kuvorname               CHAR(20) NOT NULL,
        kustrasse               CHAR(30) NOT NULL,
        kuplz                   CHAR(08) NOT NULL,
        kuort                   CHAR(30) NOT NULL,
        kutelefon               CHAR(20),
        kugebdat                DATE     NOT NULL,
        kudatfschein            DATE     NOT NULL,
        PRIMARY KEY (kunr)
);

CREATE TABLE hersteller
(
        hename                  CHAR(35) NOT NULL,
        hestrasse               CHAR(30),
        heplz                   CHAR(05),
        heort                   CHAR(30),
        hetelefon               CHAR(20),
        hekontakt               CHAR(20),
        PRIMARY KEY (hename)
);

CREATE TABLE modelle
(
        moname                  CHAR(35) NOT NULL,
        hename                  CHAR(35) NOT NULL,
        moleistung              SMALLINT,
        mohubraum               SMALLINT,
        molaenge                SMALLINT,
        mobreite                SMALLINT,
        PRIMARY KEY (moname),
        FOREIGN KEY(hename) REFERENCES hersteller(hename)
);

CREATE TABLE pkw
(
        pkennzeichen            CHAR(11) NOT NULL,
        moname                  CHAR(35) NOT NULL,
        perstzulassung          DATE,
        peabmeldung             DATE,
        pschiebedach            CHAR(1),
        pklima                  CHAR(1),
        PRIMARY KEY (pkennzeichen),
        FOREIGN KEY(moname) REFERENCES modelle(moname)
);
```

```
CREATE TABLE werkstatt
(
        wename                  CHAR(35) NOT NULL,
        westrasse               CHAR(30),
        weplz                   CHAR(05),
        weort                   CHAR(30),
        wetelefon               CHAR(20),
        wekontakt               CHAR(20),
        PRIMARY KEY (wename)
);

CREATE TABLE ausleihe
(
        aunr                    SERIAL(1),
        pkennzeichen            CHAR(11) NOT NULL,
        kunr                    INTEGER  NOT NULL,
        auleiausdat             DATE     NOT NULL,
        auanfangkm              INTEGER  NOT NULL,
        auleirueckdat           DATE,
        auendkm                 INTEGER,
        PRIMARY KEY (aunr),
        FOREIGN KEY(pkennzeichen) REFERENCES pkw(pkennzeichen),
        FOREIGN KEY(kunr)         REFERENCES kunden(kunr)
);

CREATE TABLE reparatur
(       pkennzeichen            CHAR(11) NOT NULL,
        wename                  CHAR(35) NOT NULL,
        redatum                 DATE     NOT NULL,
        reart                   CHAR(30),
        redauer                 SMALLINT,
        PRIMARY KEY (pkennzeichen, wename),
        FOREIGN KEY(pkennzeichen) REFERENCES pkw(pkennzeichen),
        FOREIGN KEY(wename)       REFERENCES werkstatt(wename)
);
```