
Datenbanken und Informationssysteme

Material for CLIL included

Kurt Hillebrand

September 2014

Inhaltsverzeichnis

| | |
|---|----|
| Inhaltsverzeichnis..... | 1 |
| TEIL I: ALLGEMEINE THEMEN..... | 4 |
| 1 Grundlagen der Datenbanksysteme..... | 5 |
| 1.1 Entwicklung | 5 |
| 1.2 Vergleich: Dateien - Datenbanksystem | 5 |
| 1.3 Eigenschaften von Datenbanksystemen..... | 6 |
| 1.4 Datenbankarchitektur: Drei-Schichten-Konzept | 8 |
| 2 Konzeptionelles Datendesign | 10 |
| 2.1 Grundbegriffe | 10 |
| 2.2 Entity-Relationship-Diagramm..... | 11 |
| 2.3 Komplexitätsgrad von Beziehungs-Typen..... | 12 |
| 2.3.1 Drei Arten von Beziehungs-Typen | 12 |
| 2.3.2 ER-Diagramm mit Arten der Beziehungs-Typen | 14 |
| 2.3.3 (min,max)-Notation | 15 |
| 2.4 Attribute von Beziehungen | 15 |
| 2.5 Beziehungs-Typ oder Entity-Typ | 16 |
| 2.6 Attribut oder Entity-Typ..... | 17 |
| 2.7 Beziehungen zwischen Entities desselben Typs..... | 18 |
| 2.8 Beziehungen zwischen mehr als zwei Entity-Typen..... | 19 |
| 2.9 Weitere Notationsformen von ER-Diagrammen | 21 |
| 3 Vom Konzeptionellen Datenmodell zur Implementierung in Tabellen | 22 |
| 3.1 Identifizierung von Entities | 22 |
| 3.2 Redundanz und Fremdschlüssel..... | 23 |
| 3.3 Implementierung von ER-Diagrammen in Tabellen | 25 |
| 3.4 Zusammenfassung der Vorgangsweise..... | 26 |
| 3.5 Übersicht: Vom ER-Diagramm zu den Tabellen | 27 |
| 4 Weiterführende ER-Modellierung | 29 |
| 4.1 Abhängige Entity-Typen | 29 |
| 4.2 Überlagerte Entity-Typen..... | 29 |
| 4.3 Implementierung von Überlagerten Entity-Typen in Tabellen | 31 |
| 4.4 Arten von Attributen..... | 33 |
| 5 Übungen zum Konzeptionellen Datendesign..... | 34 |
| 5.1 Kursverwaltung..... | 34 |
| 5.2 Produktionsplanungssystem | 34 |
| 5.3 All Airways Association..... | 35 |
| 5.4 Schulinformationssystem | 35 |
| 5.5 Kinokette | 36 |
| 5.6 Rettungsstelle..... | 36 |
| 5.7 Fremdenverkehrsregion | 37 |
| 5.8 Fußballdatenbank..... | 38 |
| 5.9 Tankstellenkette | 38 |
| 5.10 Restaurantbetrieb..... | 39 |
| 5.11 Chess Tournaments (CLIL)..... | 39 |
| 5.12 Library (CLIL) | 40 |
| 5.13 School Organisation (CLIL) | 40 |
| 5.14 Aufträge..... | 41 |
| 5.15 Ableitung ERD in Tabellen | 42 |
| 5.16 Ableitung ERD in Tabellen | 42 |
| 5.17 Einfaches Auftrags-Bestell-System (Handelsbetrieb) | 43 |
| 6 Relationales Datenmodell..... | 44 |
| 6.1 Grundbegriffe | 44 |
| 6.2 Normalformen von Relationen | 45 |
| 6.2.1 Unnormalisierte Relation | 45 |
| 6.2.2 Erste Normalform (1NF) | 46 |
| 6.2.3 Funktionale Abhängigkeit (FA) | 47 |
| 6.2.4 Zweite Normalform (2NF) | 48 |
| 6.2.5 Dritte Normalform (3NF)..... | 50 |
| 6.2.6 Boyce-Codd Normalform (BCNF) | 52 |
| 6.2.7 Weitere Normalformen | 53 |
| 6.2.8 Denormalisieren..... | 53 |
| 6.2.9 Übungen zum Normalisieren | 54 |

| | | |
|--------|--|-----|
| 7 | Relationale Entwurfstheorie..... | 58 |
| 7.1 | Armstrong-Axiome | 58 |
| 7.2 | Hülle einer Menge von Abhängigkeiten..... | 58 |
| 7.3 | Hülle einer Menge von Attributen | 59 |
| 7.4 | Zerlegung von Relationenschemata | 59 |
| 7.4.1 | Verbundtreue Zerlegung | 59 |
| 7.4.2 | Abhängigkeitstreue Zerlegung..... | 60 |
| 7.4.3 | Zusammenfassung | 60 |
| 7.5 | Übungen zur Relationalen Entwurfstheorie..... | 60 |
| 8 | SQL..... | 62 |
| 8.1 | Datenbank LT: Lieferanten - Teile - Lieferungen | 62 |
| 8.2 | Datendefinitionen | 63 |
| 8.3 | Indizes | 64 |
| 8.4 | Datenzugriffsberechtigungen | 64 |
| 8.5 | Datenauswahl - SELECT | 65 |
| 8.5.1 | Projektion..... | 65 |
| 8.5.2 | DISTINCT | 66 |
| 8.5.3 | Restriktion (Selektion)..... | 66 |
| 8.5.4 | ORDER BY | 67 |
| 8.5.5 | Mengenoperation Vereinigung - UNION..... | 68 |
| 8.5.6 | Cross-Join..... | 68 |
| 8.5.7 | Equi-Join | 68 |
| 8.5.8 | Self-Join | 69 |
| 8.5.9 | Sub-Selects in der Search-Condition..... | 70 |
| 8.5.10 | All-Quantor..... | 71 |
| 8.5.11 | Aggregatfunktionen..... | 72 |
| 8.5.12 | Gruppierung..... | 72 |
| 8.5.13 | Theta-Join..... | 73 |
| 8.5.14 | Outer-Join | 74 |
| 8.5.15 | Natural-Join | 74 |
| 8.5.16 | Semi-Join..... | 75 |
| 8.5.17 | Mengenoperationen Durchschnitt und Differenz - INTERSECT, EXCEPT | 75 |
| 8.5.18 | Sub-Selects in der Projektion | 75 |
| 8.5.19 | Sub-Selects in der From-Klausel (Inline View) | 76 |
| 8.5.20 | NULL und dreiwertige Logik | 76 |
| 8.5.21 | Zusammenfassung der Abarbeitung | 76 |
| 8.6 | Datenmanipulationen | 77 |
| 8.6.1 | INSERT | 77 |
| 8.6.2 | UPDATE | 77 |
| 8.6.3 | DELETE | 78 |
| 8.7 | Transaktionssteuerung..... | 78 |
| 8.8 | Virtuelle Tabellen - VIEW | 78 |
| 8.9 | Zeilenweise Verarbeitung - CURSOR | 79 |
| 8.10 | Übungen zu SQL..... | 80 |
| 8.11 | Datenbank LTP: Lieferanten - Teile - Projekte - Lieferungen | 83 |
| 8.12 | Datenbank Schulungsfirma | 85 |
| 9 | Stored Routines und Triggers..... | 90 |
| 9.1 | Stored Routines..... | 90 |
| 9.2 | Triggers | 92 |
| 10 | ESQL (Embedded SQL) | 94 |
| 10.1 | Grundlagen..... | 94 |
| 10.2 | Dynamisches SQL..... | 95 |
| 11 | Systemtabellen und Information Schema Views | 98 |
| 12 | Transaktionen..... | 101 |
| 13 | Recovery..... | 103 |
| 14 | Concurrency | 108 |
| 14.1 | Problemstellung..... | 108 |
| 14.2 | Serialisierbarkeit..... | 109 |
| 14.3 | Lösungsmöglichkeiten..... | 111 |
| 14.4 | Deadlocks | 114 |
| 14.5 | Die Re-Read-Methode | 117 |
| 14.6 | Zusätzlicher Sperrmodus U-Lock..... | 117 |
| 14.7 | Isolation Levels..... | 118 |
| 14.8 | Hierarchische Sperrverfahren | 119 |
| 15 | Vergleich OLTP- und OLAP-Anwendungen | 120 |
| 16 | Performancemessung in Datenbanksystemen..... | 121 |

| | | |
|-----------------------------------|---|-----|
| 17 | XML (CLIL) | 123 |
| | 17.1 Introduction..... | 123 |
| | 17.2 XML Document Structure..... | 123 |
| | 17.3 Document Type Definition (DTD) | 124 |
| | 17.4 Namespaces | 125 |
| | 17.5 XML Schema (XSD)..... | 126 |
| 18 | Client-Server-Architektur und Verteilte Datenbanken..... | 130 |
| | 18.1 Datenbank-Server | 131 |
| | 18.2 Verteilte Datenbanken..... | 132 |
| 19 | Datenkomprimierung | 134 |
| | 19.1 Lauflängencodierung..... | 134 |
| | 19.2 Codieren in variabler Länge | 134 |
| | 19.3 LZW-Verfahren | 135 |
| 20 | Kryptologie..... | 136 |
| | 20.1 Einfache Methoden | 137 |
| | 20.2 Die Datenverschlüsselungsnorm DES (Data Encryption Standard) | 139 |
| | 20.3 Systeme mit öffentlichen Schlüsseln (Public-Key Systems) | 141 |
| 21 | Grundlagen der Datenorganisation..... | 144 |
| | 21.1 Daten und deren Organisationseinheiten..... | 144 |
| | 21.2 Dateien: Organisations- und Zugriffsformen | 146 |
| | 21.3 B-Bäume | 149 |
| 22 | Überblick Graphentheorie..... | 150 |
| 23 | Literatur..... | 152 |
| TEIL II: SPEZIFISCHE SYSTEME..... | | 154 |
| 24 | Microsoft SQLServer | 155 |
| | 24.1 SQLServer Management Studio | 155 |
| | 24.2 Transact-SQL..... | 156 |
| | 24.3 Stored Procedures | 157 |
| | 24.4 Stored Functions | 158 |
| | 24.5 Triggers | 159 |
| | 24.6 Transaktionen | 159 |
| | 24.7 Concurrency..... | 160 |
| 25 | ORACLE..... | 162 |
| | 25.1 Grundlagen..... | 162 |
| | 25.2 Datentypen | 163 |
| | 25.3 SQL | 164 |
| | 25.4 SQL*Plus..... | 165 |
| | 25.5 PL/SQL..... | 167 |
| | 25.6 Stored Subprograms | 168 |
| | 25.7 Packages | 169 |
| | 25.8 Triggers | 170 |
| | 25.9 Collections und Records | 171 |
| | 25.10 Dynamisches SQL..... | 171 |
| | 25.11 Transaktionen | 171 |
| | 25.12 Concurrency..... | 172 |
| | 25.13 Object-Relational Features (CLIL) | 174 |

TEIL I: ALLGEMEINE THEMEN

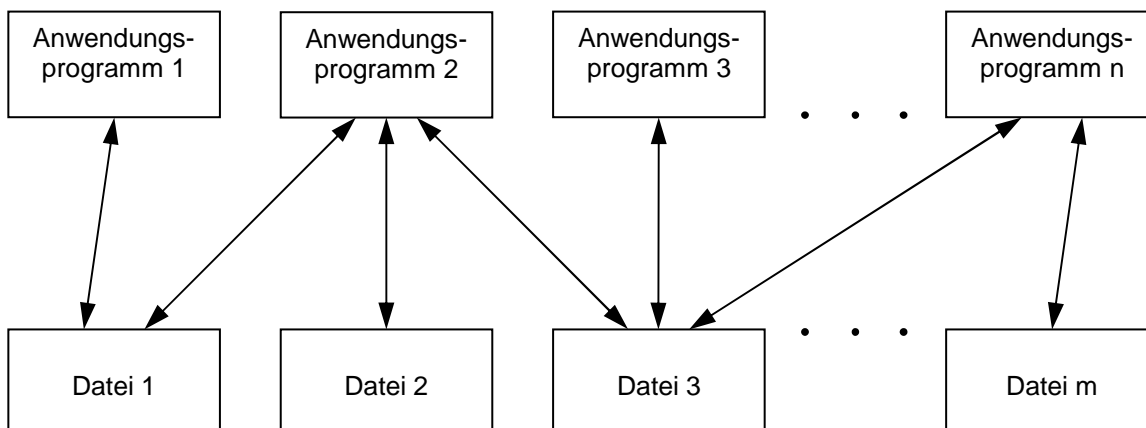
1 Grundlagen der Datenbanksysteme

1.1 Entwicklung

- 60er Jahre: Systeme von Dateien
- 70er Jahre: Hierarchische- und Netzwerk-Datenbanken (Graphenorientierte Datenmodelle)
- 80er Jahre: Relationale Datenbanken (Tabellenorientierte Datenmodelle)
- 90er Jahre: Client-Server Architektur
Datenbankmaschinen
Verteilte Datenbanken
- Weitere Trends: Objektorientierte Datenbanken / Objektrelationale Datenbanken
OLAP / Data Warehousing / Data Mining
XML-Integration
Mobile Datenbanken
Dokumentenmanagementsysteme / Multimedia-Datenbanken
Geo-Datenbanken / Geographische Informationssysteme (GIS)
NoSQL-Datenbanken, NewSQL-Datenbanken
Aktive / Ereignisorientierte Datenbanken
Deduktive Datenbanken / Knowledge Representation / Semantic Networks
- Datenmodell:
Bestimmte Datenstrukturen (z.B. records, files, tables, database-records) und bestimmte Regeln, die bei der Datenmodellierung eingehalten werden müssen.
Für ein Datenmodell müssen folgende Bestandteile definiert werden:
 - Menge von Datenobjekten, aus denen die Datenbank aufgebaut ist.
 - Menge von Integritätsregeln, die die Varianten des Datenbankaufbaus aus den Objekten auf die tatsächlich möglichen einschränken.
 - Menge von Operationen, die auf die Datenobjekte zum Zweck der Informationsänderung und Informationswiedergewinnung angewendet werden können.
- In diesem Sinn wird auch zwischen dem Strukturteil und Operationenteil eines Datenmodells unterschieden

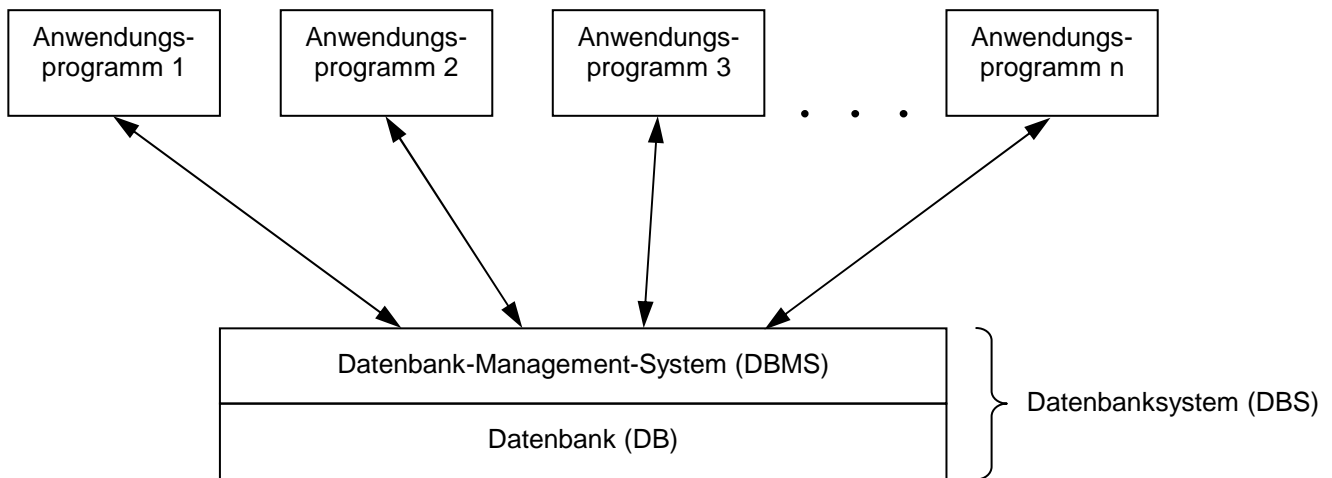
1.2 Vergleich: Dateien - Datenbanksystem

- System von Dateien:



- Anwendungsprogramme greifen direkt, mit elementaren, satzweisen Zugriffsoperationen auf die Datensätze zu.
- Länge und Struktur der Datensätze sind in den Anwendungsprogrammen beschrieben.
- Keine Beziehungen von Datensätzen innerhalb einer oder zwischen verschiedenen Dateien.
- Der Dateiaufbau (Organisation) ist jeweils der Verarbeitung durch das Anwendungsprogramm angepasst.
- Darüberhinaus gibt es öfters auch Dateien, die jeweils nur für ein Programm (sozusagen als 'private' Dateien) vorhanden sind.

- Probleme bei:
 - Änderung, Organisation, Sortierung, Indizierung der Dateien
 - Längenänderungen der Datensätze
 - Änderungen der Datensatzstrukturen
- Datenbanksystem:



- DBMS: Software zur Verwaltung und Benutzung von Datenbanken
- DB: Strukturierter, vom DBMS verwalteter, Datenbestand (Konkrete, physisch existierende Datenbank)
- DBS: Datenbank-Management-System + Datenbank(en)
- Anwendungsprogramme können nur über das DBMS auf die Daten zugreifen
- Komponenten eines Datenbanksystems:
 - Datendefinitionssprache (DDL)
 - Datenkatalog, Data Dictionary
 - Datenmanipulationssprache (DML):
 - Interaktive Sprachen für Ad-Hoc-Abfragen (Query Languages) und -Änderungen
 - Aufrufe des Datenbanksystems (Database Calls) aus einer Programmiersprache (Wirtssprache, Host Language, 3GL oder 4GL)
 - Verwaltungsprogramme (Datensicherung, Datenwiederherstellung, Statistiken über die Datenspeicherung und Performance, etc.)
 - Report Generatoren

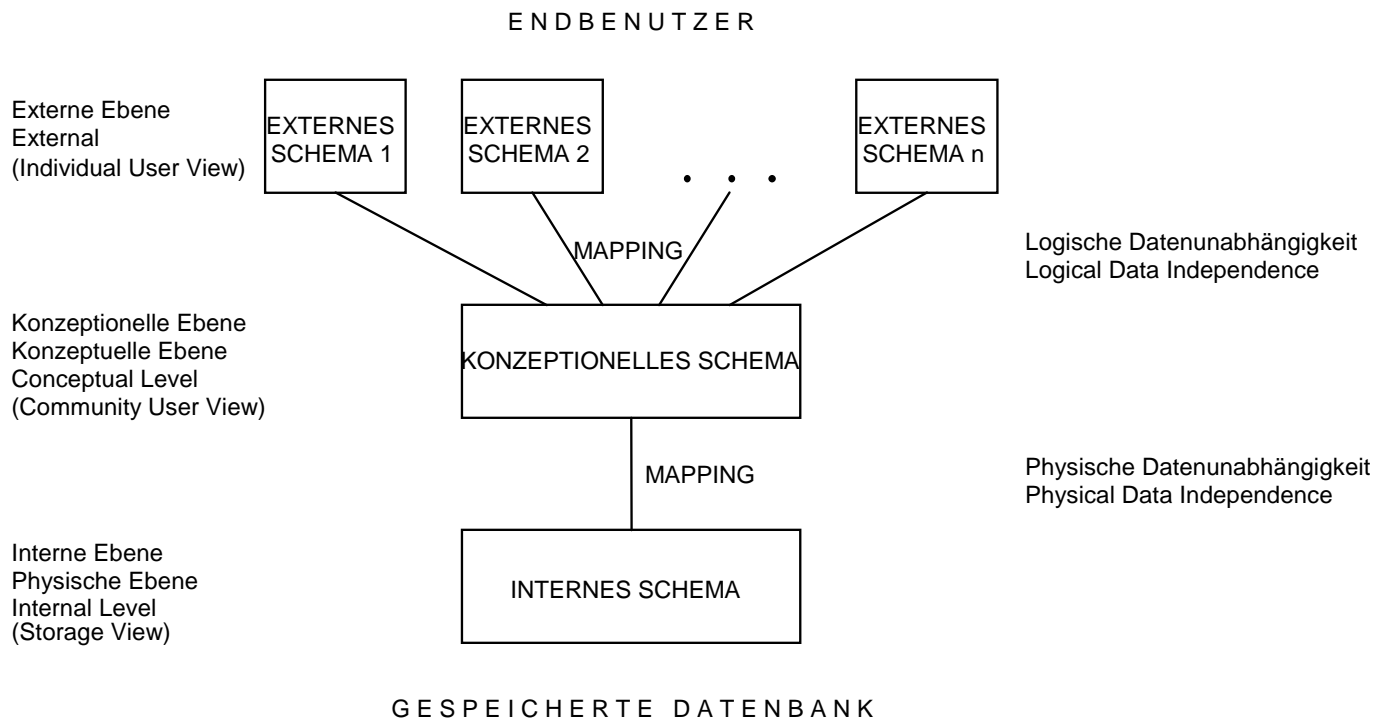
1.3 Eigenschaften von Datenbanksystemen

- Redundanzfreiheit: Ein Sachverhalt ist nur einmal (an einer Stelle) abgespeichert.
Aus bestimmten Gründen (z.B. Performanceüberlegungen) können unter Umständen geringe Redundanzen in Kauf genommen werden, ein solcher Entwurf ist aber fundiert zu begründen und ausführlich zu dokumentieren (besonders die Auswirkungen auf die Verarbeitungslogik).
Folgende Bedingungen sind bei der Verwendung redundanter Daten einzuhalten:
 - Redundante Daten dürfen 'nicht zuviel' zusätzlichen Speicherplatz verbrauchen.
 - Die Benutzung der redundanten Daten muss in einem 'wirtschaftlich vernünftigen' Verhältnis zum Aufwand der Änderung (Wartung) der redundanten Daten stehen.
 - Inkonsistenzen (z.B. wenn nicht alle redundanten Daten geändert werden) müssen vermieden werden können (Transaktionen).
 - Redundanzen sollen bei der Datenbankdefinition angegeben werden können; das Datenbanksystem soll dafür sorgen, dass Änderungen auf alle redundante Daten angewendet werden und damit keine Inkonsistenzen auftreten (propagating updates).

- Integrität, Konsistenz: Korrektheit und Vollständigkeit der Daten.
Dem Datenbanksystem werden bei der Definition der Datenbank die einzuhaltenden Bedingungen bekanntgegeben, damit ist ihre Verletzung nicht mehr möglich.
- Wertebereichsbedingungen von Attributen (Domain Constraints):
 - NULL-Werte ja/nein
 - Aufzählung erlaubter Werte
 - Numerische Werte: Bereich oder quantisierte Werte (z.B. Vielfache von 5)
 - Alphanumerische Werte: Aufbau (z.B. Beginn mit 2 Buchstaben dann 4 Ziffern)
 - Reihungsprinzipien (Ordering): z.B. bei Neuaufnahmen sind jeweils nur größere Werte zugelassen
 - Aggregate Functions: z.B. der Durchschnitt der Gehälter darf nicht größer als 10,000.-- werden
 - Abhängigkeiten zwischen den Werten verschiedener Attribute (Interdomain Constraints):
z.B. der Wert von Obergrenze muss immer größer oder gleich dem Wert von Untergrenze sein
z.B. wenn Monat den Wert 4,6,9,11 hat, dann darf Tag nur einen Wert zwischen 1 und 30 haben
- Integritätsbedingungen in Zusammenhang mit Primär- und Fremdschlüssel (Primary and Foreign Key Constraints):
 - Eindeutige Primärschlüsselwerte
 - Entity Integrität: Keine NULL-Werte in den Attributen des Primärschlüssels
 - Referentielle Integrität ('the glue that holds the database together!'):
 - Zu jedem Fremdschlüsselwert muss es denselben Primärschlüsselwert geben
 - NULL-Werte im Fremdschlüssel ja/nein
 - Ändern eines Primärschlüsselwerts, der als Fremdschlüsselwert existiert:
 - nicht erlaubt (restricted)
 - betroffene Fremdschlüsselwerte werden auf NULL gesetzt (nullifies)
 - betroffene Fremdschlüsselwerte werden mitgeändert (cascades)
 - Löschen eines Satzes mit einem Primärschlüsselwert, der als Fremdschlüsselwert existiert:
 - nicht erlaubt (restricted)
 - betroffene Fremdschlüsselwerte werden auf NULL gesetzt (nullifies)
 - Sätze mit betroffenen Fremdschlüsselwerten werden mitgelöscht (cascades)
 - Sicherstellung einer 1:1-Beziehung (im Fremdschlüssel darf jeder Wert höchstens einmal / muss jeder Wert genau einmal auftreten, abgesehen von NULL-Werten)
 - Ober- und Untergrenzen bei Beziehungen: z.B. maximal 10 Personen sind einem Projekt zugeordnet
- Semantische und Sonstige Integritätsbedingungen (Semantic and Other Integrity Constraints):
 - alle sonstigen Integritätsbedingungen
 - benutzerspezifische Routinen
 - im Extremfall können dahinter ganze Expertensysteme stehen
 - Beispiele:
 - Jeder Lieferant aus 'London' muss das Teil 'T2' liefern
 - Wenn Familienstand 'verheiratet' war, kann nicht mehr der Familienstand 'ledig' erreicht werden
 - Der Gehalt eines Mitarbeiters darf den Gehalt seines Vorgesetzten nicht übersteigen
 - Die Gesamtstundenanzahl, die ein Mitarbeiter in einer Woche an allen Projekten arbeitet, darf 56 nicht übersteigen
- Synchronisation paralleler Zugriffe (Concurrency Control): Mehrfachzugriffe auf die Daten müssen in einem Multitasking- / Multiuser-System so gesteuert werden, dass die Korrektheit der Daten gewährleistet ist.
- Datenunabhängigkeit: Trennung der Daten von den Programmen.
 - Je höher der Grad an Datenunabhängigkeit ist, desto geringer ist der Aufwand für Programmänderungen im Falle einer Änderung bei den Daten; im Idealfall sind keine Programmänderungen bei Änderung des Datenaufbaus notwendig.
 - Beispiele:
 - Änderung der internen Darstellung von numerischen und nicht numerischen Datenfeldern
 - Änderung der Struktur und / oder Länge von Datensätzen
 - Änderung der Organisationsform von Dateien (dann keine Änderungen im Zugriffspfad)
 - Die Datenunabhängigkeit wird auch durch Einführung von zusätzlichen Datentypen (z.B. Datum) erhöht.
- Datensicherheit (Data Security): Maßnahmen gegen Verfälschung, Zerstörung oder Verlust von Daten, 'Schutz der Daten'.
 - Im Besonderen nach Fehlersituationen müssen die Daten wieder korrekt hergestellt werden können (Wiederanlauf, Recovery)
 - Beispiele: Transaktionen, Datensicherung, Backups
- Datenschutz (Data Privacy): Maßnahmen gegen unbefugten Zugriff auf oder missbräuchliche Verwendung von Daten, 'Schutz der beschriebenen Objekte / Personen'.
 - Beispiele: User Management, Passwörter, Zugriffsberechtigungen, Datenverschlüsselung, Kryptographie

1.4 Datenbankarchitektur: Drei-Schichten-Konzept

- ANSI/SPARC Architektur (1975)
SPARC=Standards Planning and Requirements Committee



- Konzeptionelle Ebene:
 - Beschreibung der Realität durch ein Datenmodell, ohne Rücksicht auf hardwaremäßige oder applikations-spezifische Aspekte; 'Image of the Real World'
 - Eine einzige Sichtweise
 - Wird vom Datenbankadministrator (Unternehmensadministrator, Enterprise Administrator) erstellt und betreut
 - Beschreibung mit Data Description Language (DDL), Ergebnis ist das Konzeptionelle Schema
 - Schwierige Analyse- und Planungsarbeit, u.U. beachtliche Investitionen, Stabiler Bezugspunkt, Basis für interne und externe Ebene
- Interne Ebene:
 - Beschreibung der physischen Speicherung der Daten
 - Themen: Organisationsform der Betriebssystem-Dateien in denen die Daten gespeichert sind, Blockgrößen, Cluster, Indizes, Zugriffspfade, Hashing, Pointer, Datenverschlüsselung, etc.
 - Der Datenbankadministrator (Database Administrator) muss bei der Definition dieser Ebene (Internes Schema) auf die bestehende Hardware und das verfügbare Betriebssystem Rücksicht nehmen
- Externe Ebene:
 - Beschreibung der Daten, wie sie für die verschiedenen Anwendungen oder Anwendungsbereiche von Interesse sind
 - Mehrere Sichtweisen
 - Werden vom Datenbankadministrator (Anwendungsadministrator, Application Administrator) zur Verfügung gestellt
 - Externe Ebene wird in Externen Schemata beschrieben
- Verbindung der Ebenen (Mapping):
Transformationsregeln, die definieren, wie ein bestimmtes Objekt einer Ebene aus einem oder mehreren Objekten einer anderen (tieferliegenden) Ebene gebildet wird

- Forderung: Datenbanksysteme sollen eine klare Trennung zwischen den einzelnen Ebenen aufweisen (Multiple-Level-DBS)
- Da die Mappings relativ aufwendig (Zeit oder Speicher) sein können, wird bei den meisten Datenbanksystemen die Trennung zwischen den Ebenen nicht sauber vorgenommen
- Datenunabhängigkeit:
Ist die Möglichkeit, das Schema in einer Ebene ändern zu können, ohne Änderungen im Schema der nächsthöheren Ebene vornehmen zu müssen (die Transformationsregeln müssen gegebenenfalls schon geändert werden)
 - Logische Datenunabhängigkeit:
Keine Auswirkungen von Änderungen in der Konzeptionellen Ebene auf die Externe Ebene.
Beispiele: Einfügen oder Löschen von Satzarten, Relationen oder Datenelementen
 - Physische Datenunabhängigkeit:
Keine Auswirkungen von Änderungen in der Internen Ebene auf die Konzeptionelle (oder gar Externe) Ebene.
Beispiele: Reorganisation der internen Dateistruktur (z.B. aus Performancegründen), Aufnehmen oder Löschen von Indizes, Wechsel des Betriebssystems

2 Konzeptionelles Datendesign

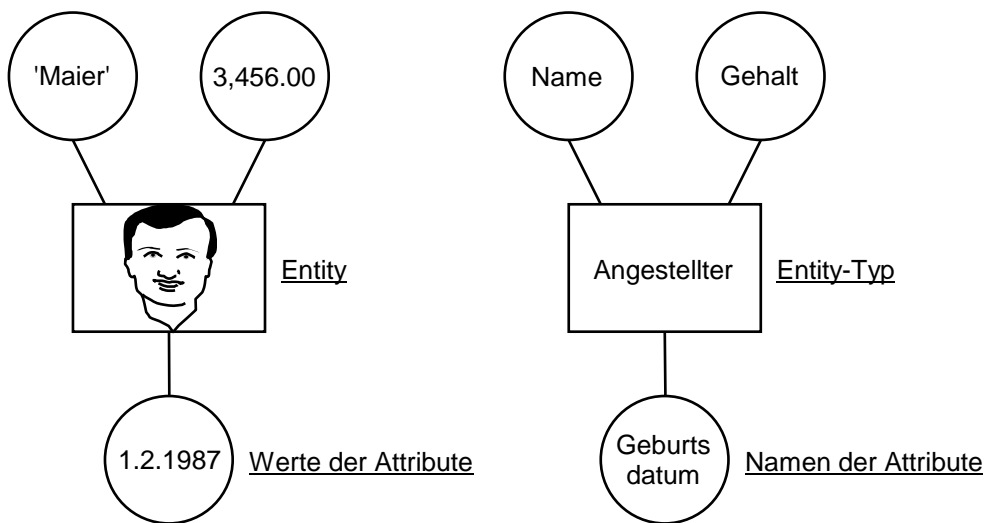
2.1 Grundbegriffe

- Konzeptionelles / Konzeptuelles Datenmodell / Datendesign, Entity-Relationship-Modelling (ERM), Datenstrukturanalyse (DSA), Logische / Semantische Datenmodellierung, Informationsmodell, Information Modelling
- Beschreibt einen Ausschnitt der realen Welt aus einer logischen Gesamtsicht
- Es wird ermittelt, welche Dinge (Entities) für eine Organisation / ein System bedeutend sind, welche Eigenschaften (Attribute) diese Dinge haben und wie sie zueinander in Beziehung (Relations) stehen.
- Modellierung mittels Constraints
- Vorteile:
 - stellt eine Gesamtschau auf die benötigte Information der Gesamtorganisation dar, geht daher über die lokale Sicht von Einzelanwendungen auf die Daten hinaus
 - ist eine gemeinsame sprachliche Basis für die Kommunikation zwischen den Anwendern und den Informatikern
 - ist unabhängig von der DV-technischen Implementierung, also auch vom verwendeten Datenbankmodell ('Struktogramm : Programm = Entity-Relationship-Diagramm : Daten')
 - bildet die Grundlage für die Ableitung der Datenstrukturen in ein konkretes Daten(bank)modell
 - Vergleich:

| Konzeptionelles Datenmodell (Conceptual Data Model - CDM) | Physisches Datenmodell (Physical Data Model - PDM) |
|--|---|
| Darstellung unabhängig von einem bestimmten Datenbankmodell (z.B. Entity-Relationship-Modell) | Darstellung in einem bestimmten Datenbankmodell (z.B. Relationalem Datenbankmodell) |
| sollte der Anwender verstehen | irrelevant für den Anwender |
| <ul style="list-style-type: none"> - Entity-Typ - Attribut - Domäne (Domain) - Beziehungstyp | <ul style="list-style-type: none"> - Tabelle (Datei) - Spalte (Feld) - Datentyp - Fremdschlüssel - Index, Optimierung - Denormalisierung, Redundanzen |

- Entity (Entität): Identifizierbare Einheit der realen Welt ('reale Wesenseinheit'), wie eine Person, ein Ding, ein Objekt, ein Ereignis, ein Begriff etc.
Wesentlich ist, dass die verschiedenen Entities identifizierbar und damit voneinander unterscheidbar sind (Chen: A 'thing' which can be distinctly identified).
Beispiele: ein Angestellter, ein Bauteil, ein Flug, eine Rechnungsverbuchung
Gegenbeispiele: ein Schnee, eine Luft, eine Hoffnung
- Attribut (Eigenschaft): Merkmal eines Entities, das zu seiner Beschreibung wichtig ist.
Jedes Attribut bekommt einen Namen.
Beispiele: Name eines Angestellten, Gehalt eines Angestellten, Geburtsdatum eines Angestellten, Farbe eines Bauteils, Start- und Landezeitpunkt eines Fluges, Betrag einer Buchung
- Werte von Attributen: Für ein konkretes Entity hat jedes Attribut einen bestimmten Wert.
Beispiele: für einen konkreten Angestellten ist der Name gleich 'Maier', ist der Gehalt gleich 3,456.00, ist das Geburtsdatum gleich 1.2.1987; für ein konkretes Bauteil ist die Farbe gleich 'rot'; für einen konkreten Flug ist der Startzeitpunkt gleich 24.12.2000 13:45; für eine konkrete Buchung ist der Betrag gleich 17.30
- Entity-Typ (Entity-Type, Entity-Menge, Entity-Set): Entities mit gleichen Attributen werden zu einem Entity-Typ zusammengefasst.
Jeder Entity-Typ bekommt einen Namen, wobei üblicherweise die Einzahl verwendet wird (z.B. Angestellter, nicht Angestellte)
Beispiele: Angestellte eines Unternehmens, in einem Unternehmen erzeugte Bauteile, Buchungen in einem Unternehmen

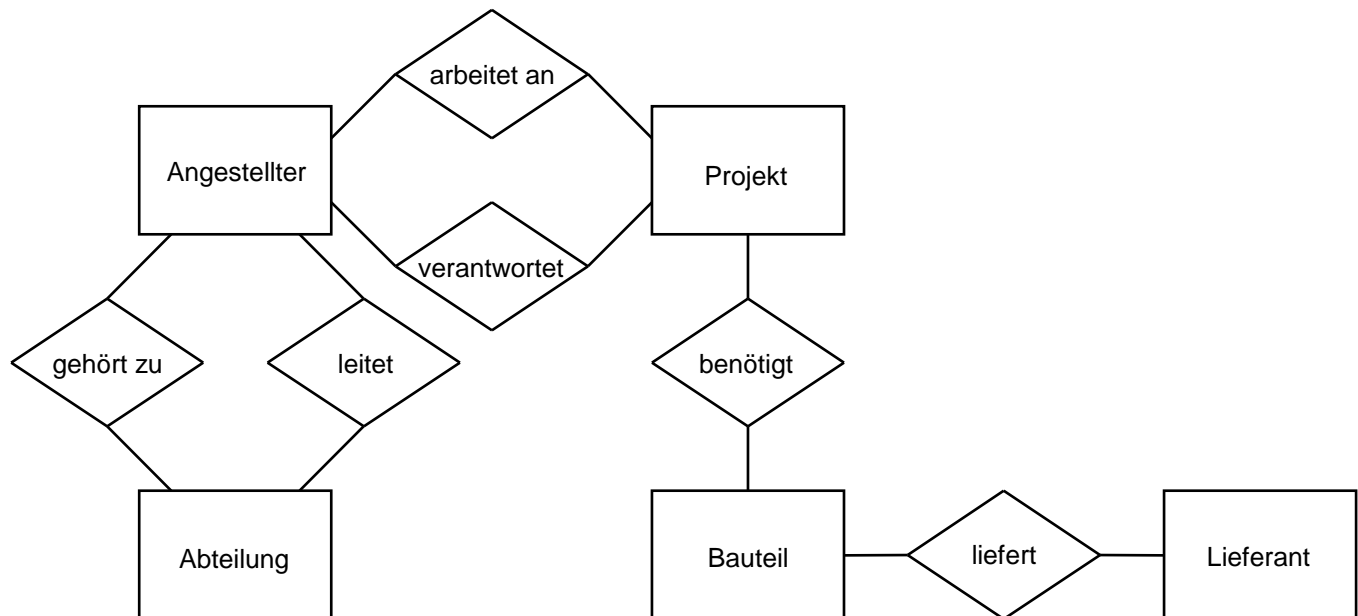
- Graphische Darstellung:
 - Entity-Typ: Rechteck
 - Attribut: Kreis (Lollipop)



- Das Entity selbst kann nicht dargestellt, sondern nur durch seine Attributwerte (eindeutig) beschrieben werden.
- Textliche Darstellung eines Entity-Typs mit seinen Attributen:
Angestellter (Name, Gehalt, Geburtsdatum)
- Eindeutigkeit der Namen: In einem System müssen die Namen der Entity-Typen und der Attribute innerhalb der Entity-Typen eindeutig sein
- Auswertungen (d.h. Informationen, die aus den bestehenden Attributwerten abgeleitet werden können) sind nicht als Entities aufzufassen, z.B. Kundenliste
- Hinweise:
 - Die Begriffe Entity und Entity-Typ sowie Attribut und Attributwert sollen genau unterschieden werden.
 - Oft wird der Begriff Entity statt des Begriffs Entity-Typ verwendet, man spricht vom 'Entity Angestellter'.
 - Es ist zu unterscheiden, ob zwei Entities die gleichen Attribute oder die gleichen Attributwerte haben.
 - Beim Entity-Typ Angestellter ist das Attribut Geburtsdatum einem Attribut Alter vorzuziehen, da letzteres laufend geändert werden müsste.

2.2 Entity-Relationship-Diagramm

- Beziehung (Relation, Relationship): Zwischen zwei (in der Folge auch mehr als zwei) Entities kann eine Beziehung (ein Zusammenhang) einer bestimmten Bedeutung bestehen.
Beispiele: ein Angestellter konstruiert ein Bauteil, ein Angestellter tätigt eine Buchung
- Beziehungs-Typ (Relation-Type, Beziehungs-Menge, Relation-Set): Beziehungen, an denen Entities des gleichen Entity-Typs beteiligt sind und die dieselbe Bedeutung haben, werden zu einem Beziehungs-Typ zusammengefasst.
Jeder Beziehungs-Typ bekommt einen Namen (meist ein Zeitwort).
Beispiele: bestimmte Angestellte konstruieren bestimmte Bauteile, bestimmte Buchungen werden von bestimmten Angestellten getätigt
- Der Begriff Relationship ist ein Überbegriff für Relationship-Instance (Beziehung) und Relationship-Type (Beziehungs-Typ)
- Entity-Relationship-Diagramm, ER-Diagramm (nach P. Chen, 1976):
 - Entity-Typ: Rechteck
 - Beziehungs-Typ: Raute mit Linien zu den, am Beziehungs-Typ beteiligten, Entity-Typen
 - Attribute von Entity-Typen werden in ER-Diagramme, wegen der Übersichtlichkeit, üblicherweise nicht eingezeichnet (sondern in textlicher Darstellung als Legende angegeben)



- Hinweis: Bauteil ist bei diesem Beispiel nicht als physisch konkretes Einzelbauteil, sondern als Bauteil-Art zu verstehen. Es wird nicht jede einzelne M5 Schraube verwaltet, sondern die Schraubenart M5 (ev. mit Anzahl als Attribut)
- Ein Entity-Typ kann an mehreren Beziehungs-Typen beteiligt sein.
Beispiel: Entity-Typ 'Projekt', Beziehungs-Typen 'verantwortet' und 'benötigt'
- Zwischen zwei Entity-Typen können mehrere Beziehungs-Typen (verschiedener Bedeutung) bestehen (Parallelstrukturen).
Beispiel: Entity-Typen 'Angestellter' und 'Abteilung', Beziehungs-Typen 'gehört zu' und 'leitet'
- Benötigte Informationen, die aus bestehenden Beziehungen abgeleitet werden können, dürfen nicht als eigene Beziehung definiert werden (Überbestimmtheit).
Beispiel: Die Information 'Welche Abteilungen wirken bei welchen Projekten mit?' kann aus den Beziehungs-Typen 'gehört zu' und 'arbeitet an' abgeleitet werden, es ist kein eigener Beziehungs-Typ notwendig.
- Die Existenz eines (graphentheoretischen) Weges zwischen zwei Entity-Typen ist jedoch keine Gewähr, dass Beziehungsinformationen zwischen diesen beiden Entity-Typen abgeleitet werden können.
Beispiel: Die Information 'Welche Lieferanten beliefern welche Projekte?' kann aus den Beziehungs-Typen 'benötigt' und 'liefert' nicht abgeleitet werden
- Hinweise:
 - Die Begriffe Beziehung und Beziehungs-Typ sollen genau unterschieden werden.
 - Oft wird der Begriff Beziehung statt des Begriffs Beziehungs-Typ verwendet, man spricht von der 'Beziehung leitet'.
- Implementierungs- und Performanceüberlegungen sind in diesem Stadium der Datenmodellierung nicht anzustellen

2.3 Komplexitätsgrad von Beziehungs-Typen

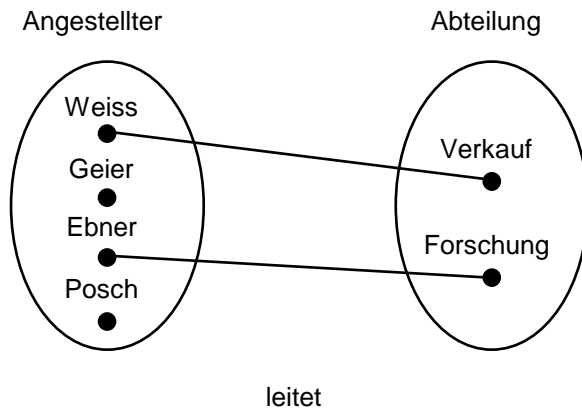
2.3.1 Drei Arten von Beziehungs-Typen

- Die Art des Beziehungs-Typs wird auch Beziehungskardinalität (Cardinality), Konnektivität oder Komplexität eines Beziehungs-Typs genannt.
- 1:1-Beziehungs-Typ (eins zu eins-Beziehungs-Typ): Jedem Entity des einen Entity-Typs ist höchstens ein Entity des anderen Entity-Typs zugeordnet und umgekehrt.
 - Beispiele:


```

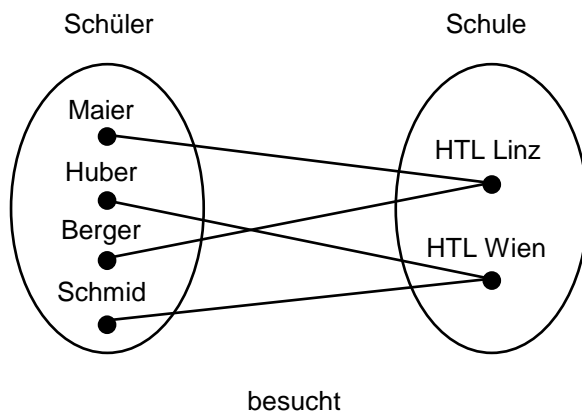
[Angestellter] ---1--- <leitet> ---1--- [Abteilung]
[Mann] ---1--- <ist verheiratet mit> ---1--- [Frau]
[Person] ---1--- <besitzt> ---1--- [Österreichischer Führerschein]
[Mitarbeiter] ---1--- <hat> ---1--- [Firmenauto]
          
```

- Ein Beziehungs-Typ R zwischen den beiden Entity-Typen E1 und E2 kann als Mathematische Relation (Menge von Paaren oder Tupeln)
 $R \subseteq E1 \times E2$
 aufgefasst werden, mit
 $E1 \times E2 = \{ (e1, e2) / e1 \in E1 \ \& \ e2 \in E2 \}$, wobei gilt
 $(e1, e2)$ ist genau dann ein Element von R, wenn e1 und e2 in Beziehung stehen.
- In einem 1:1-Beziehungs-Typ muss die mathematische Relation eine Funktion mit existierender Umkehrfunktion sein.
- Die Entities der beiden Entity-Typen kommen in den Paaren der Relation höchstens einmal vor.
- Beispiel (Mengendiagramm und Mengenschreibweise):



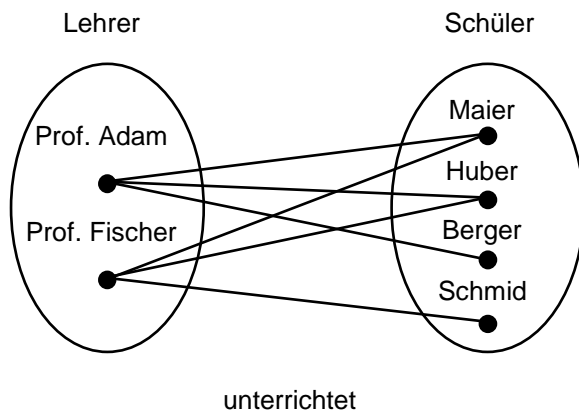
$R = \{ (Weiss, Verkauf), (Ebner, Forschung) \}$

- 1:n-Beziehungs-Typ (eins zu viele-Beziehungs-Typ, hierarchischer Beziehungs-Typ): Jedem Entity des einen Entity-Typs ist höchstens ein Entity des anderen Entity-Typs zugeordnet, nicht aber umgekehrt.
- Beispiele:
 $[Bezirk] \text{ ---1--- } \langle \text{bewohnt von} \rangle \text{ ---n--- } [Niederösterreicher]$
 $[Schüler] \text{ ---n--- } \langle \text{besucht} \rangle \text{ ---1--- } [Schule]$
- In einer 1:n-Beziehung muss die mathematische Relation eine Funktion sein.
- Die Entities des einen Entity-Typs kommen in den Paaren der Relation höchstens einmal vor, die des anderen Entity-Typs kommen mehrmals vor.
- Beispiel (Mengendiagramm und Mengenschreibweise):



$R = \{ (Maier, HTL Linz), (Huber, HTL Wien), (Berger, HTL Linz), (Schmid, HTL Wien) \}$

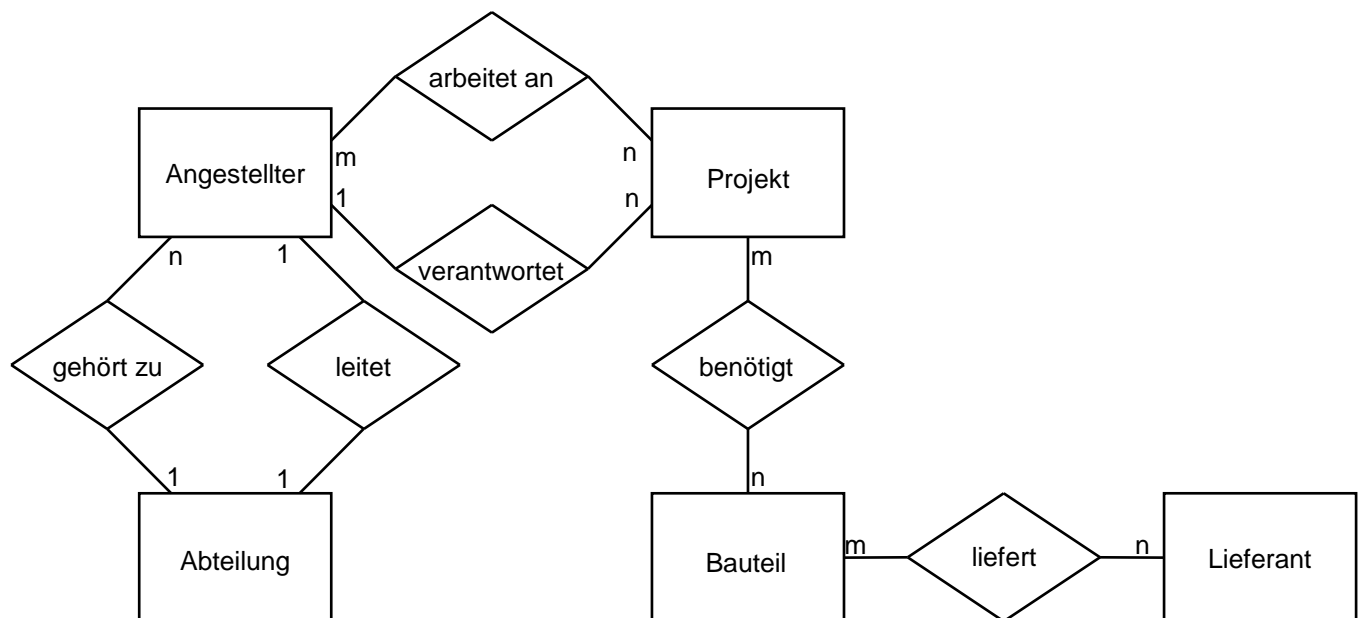
- m:n-Beziehungs-Typ (viele zu viele-Beziehungs-Typ, netzwerkartiger Beziehungs-Typ): Keine Einschränkungen in der Zuordnung zwischen den Entities der beiden Entity-Typen.
- Beispiele:
 [Lehrer] ---m--- <unterrichtet> ---n--- [Schüler]
 [Bauteil] ---m--- <wird geliefert von> ---n--- [Lieferant]
- In einer m:n-Beziehung muss die mathematische Relation keinen Bedingungen genügen.
- Die Entities der beiden Entity-Typen kommen in den Paaren der Relation mehrmals vor.
- Beispiel (Mengendiagramm und Mengenschreibweise):



$R = \{ (Prof. Adam, Maier), (Prof. Adam, Huber), (Prof. Adam, Berger), (Prof. Fischer, Maier), (Prof. Fischer, Huber), (Prof. Fischer, Schmid) \}$

2.3.2 ER-Diagramm mit Arten der Beziehungs-Typen

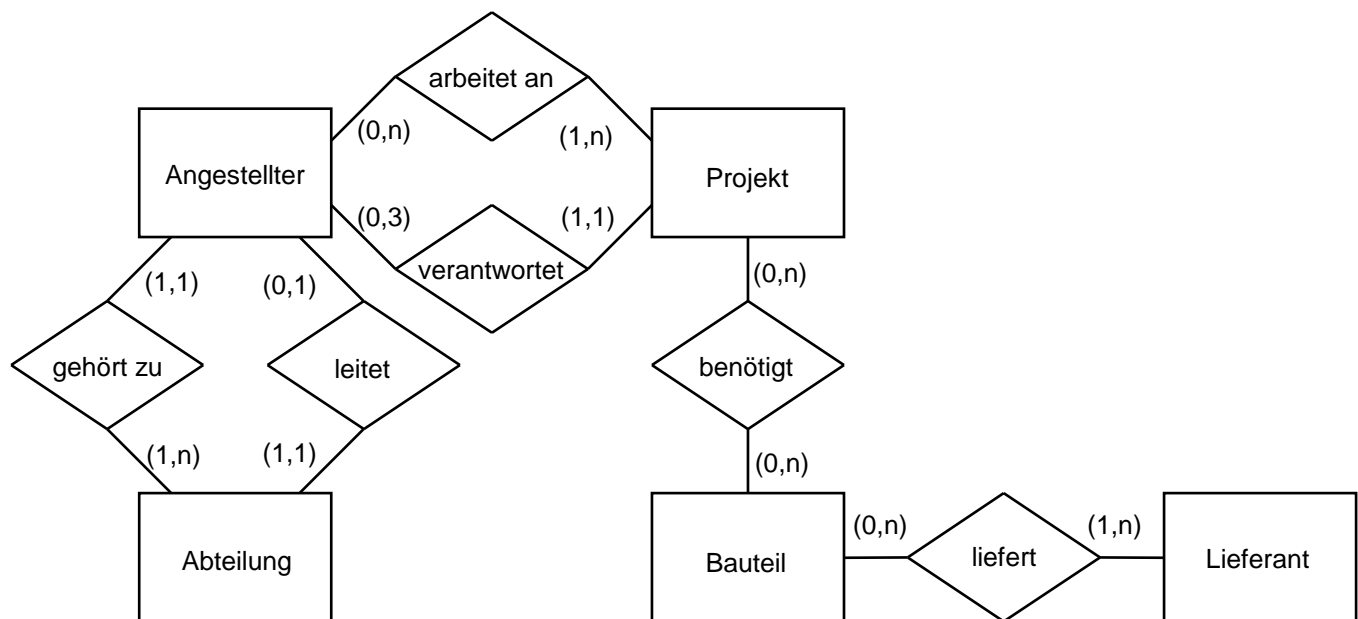
- Beachte: 1:n-Beziehung ist nicht symmetrisch, es ist wesentlich wo 1 und wo n steht!



- Übung: Wie ändern sich die Komplexitätsgrade, wenn Bauteil nicht im Sinne von Bauteil-Art, sondern als Einzelbauteil aufgefasst wird?
- Übung: Wie ändern sich die Komplexitätsgrade, wenn es zu jedem Bauteil nur einen (Stamm-)Lieferant gibt?

2.3.3 (min,max)-Notation

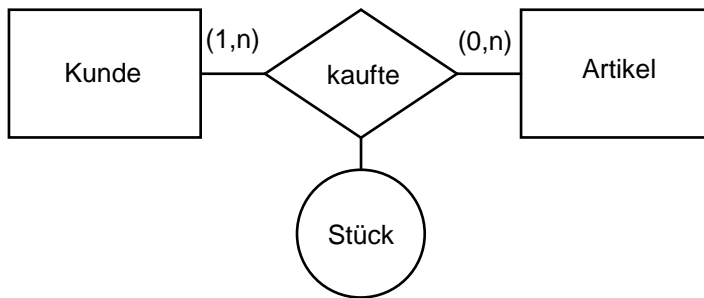
- Charakterisiert den Beziehungs-Typ über die Art hinausgehend noch genauer.
- Jedem Entity-Typ wird für jedem Beziehungs-Typ, an dem er beteiligt ist, ein Zahlenpaar (min,max), mit folgender Bedeutung zugeordnet:
Jedes Entity des Entity-Typs ist mindestens an min und höchstens an max Beziehungen des Beziehungs-Typs beteiligt. Offensichtlich muss gelten: $0 \leq \min \leq \max$ & $\max \geq 1$
- Ein beliebig großes max wird mit n symbolisiert.
- Beziehungs-Typen, die als min=0 enthalten, werden auch Konditionelle Beziehungs-Typen (Kann-Beziehungs-Typen, Partial Relation-Type) genannt.
- Ein Entity-Typ, für den bei einem Beziehungs-Typ min=0 definiert ist, nimmt an diesem Beziehungs-Typ teilweise teil (Partial oder Optional Participation).
- Ein Entity-Typ, für den bei einem Beziehungs-Typ min>0 definiert ist, nimmt an diesem Beziehungs-Typ vollständig teil (Total oder Mandatory Participation).
- Bei einigen Notationsformen sind nur die Werte 0,1 und n für min und max erlaubt.
- Beispiele:
[Angestellter] --- (0,1) --- <leitet> --- (1,1) --- [Abteilung]
[Mann] --- (0,1) --- <ist verheiratet mit> --- (0,1) --- [Frau]
[Person] --- (0,1) --- <besitzt> --- (1,1) --- [Österreichischer Führerschein]
[Mitarbeiter] --- (0,1) --- <hat> --- (1,1) --- [Firmenauto]
[Bezirk] --- (1,n) --- <bewohnt von> --- (1,1) --- [Niederösterreicher]
[Schüler] --- (0,1) --- <besucht> --- (1,n) --- [Schule]
[Lehrer] --- (1,n) --- <unterrichtet> --- (1,n) --- [Schüler]
[Bauteil] --- (0,n) --- <wird geliefert von> --- (1,n) --- [Lieferant]
(es gibt auch eigengefertigte Bauteile)



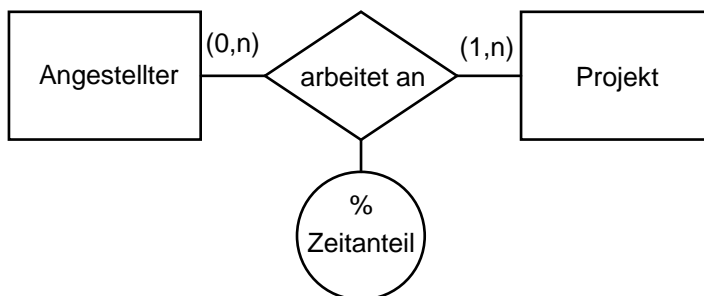
2.4 Attribute von Beziehungen

- Beziehungs-Typen können auch Attribute als Eigenschaften haben. Für eine Beziehung nimmt jedes Attribut einen bestimmten Wert an.
- Attribute sind nur bei m:n-Beziehungs-Typen sinnvoll. Andernfalls kann das Attribut beim Entity-Typ auf der Seite mit max=1 ist dazugegeben werden.
- Attribute von Beziehungs-Typen werden in ER-Diagramme, wegen ihrer Besonderheit, üblicherweise schon eingezeichnet.

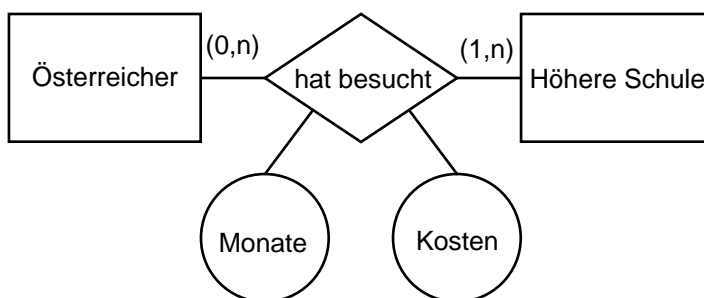
- Beispiel: Kunde hat soviel Stück dieses Artikels gekauft



- Beispiel: Angestellter arbeitet einen bestimmten Zeitanteil bei diesem Projekt mit (Randbedingung: Summen der Zeitanteile eines Angestellten ≤ 100)

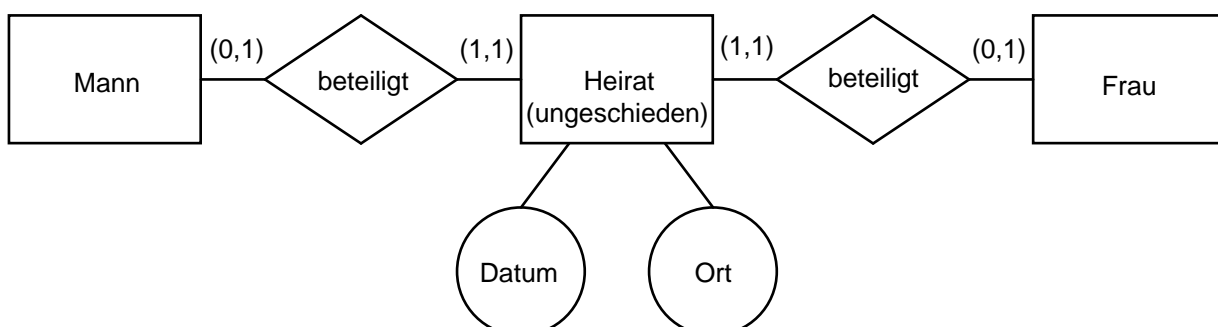


- Beispiel: Österreicher besucht die Höhere Schule eine bestimmte Anzahl von Monaten und verursacht dabei bestimmte Kosten

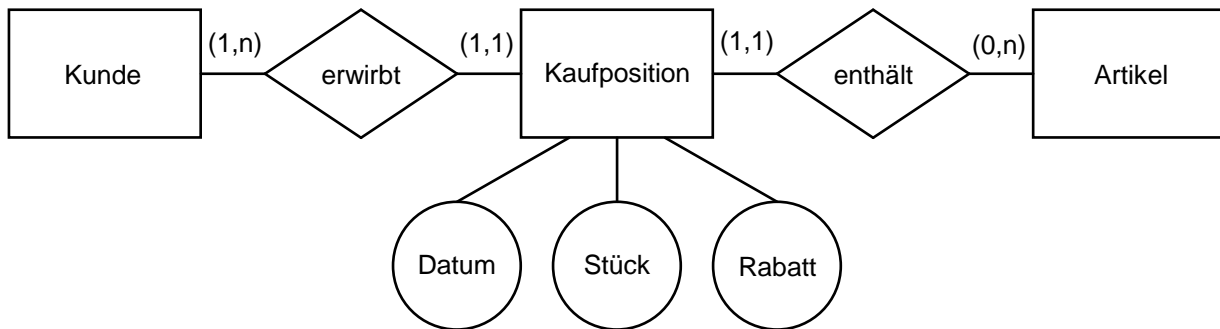


2.5 Beziehungs-Typ oder Entity-Typ

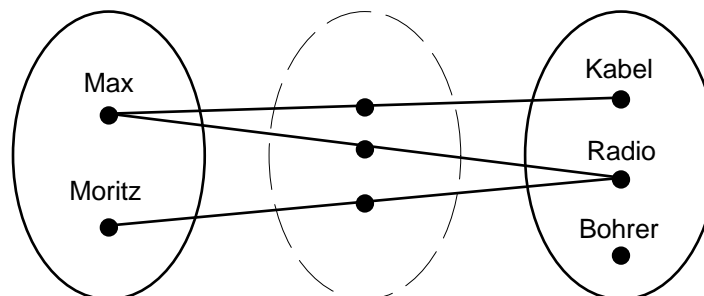
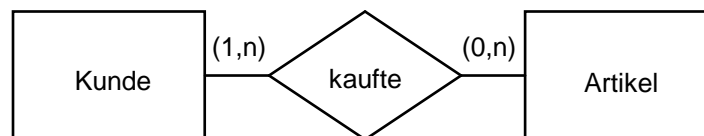
- Ein Sachverhalt, der durch eine Beziehung beschrieben ist, kann meist auch als Entity aufgefasst werden. Dies trifft besonders auf Beziehungen mit Attributen zu.
- Welche Variante zu wählen ist ergibt sich logisch aus der Aufgabenstellung, muss aber teilweise auch willkürlich festgelegt werden.
- Die Beziehung wird meist zum Ereignis (und damit Entity), durch das die Beziehung zustandegekommen ist, umfunktioniert.
- Die verbleibenden Beziehungs-Typen haben keine Attribute mehr.
- Beispiel: Der Mann hat die Frau an einem bestimmten Datum, in einem bestimmten Ort geheiratet



- Beispiel: Der Kunde hat an einem bestimmten Datum soviel Stück des Artikels mit diesem Rabatt gekauft



- Auf diese Art sind auch m:n-Beziehungs-Typen in einen Entity-Typ und zwei 1:n-Beziehungs-Typen auflösbar:

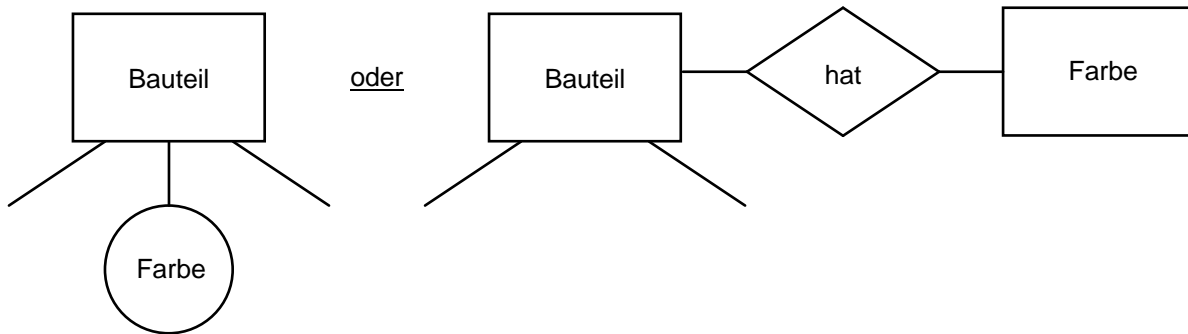


- Für den zusätzlichen Entity-Typ und für die beiden 1:n-Beziehungs-Typen müssen sinnvolle Bezeichnungen gefunden werden.
- Ein so entstandener Entity-Typ wird auch als Assoziativer Entity-Typ (Associative Entity-Type, Intersection Entity-Type) bezeichnet.
- Die (min,max)-Werte an den ursprünglichen Entity-Typen bleiben gleich, die zwei neuen sind immer (1,1)

2.6 Attribut oder Entity-Typ

- Es gibt Aufgabenstellungen, in denen eine Objekteigenschaft als Attribut oder als eigener Entity-Typ definiert werden kann.
- Wenn eine Eigenschaft eines Entity-Typs auch als eigenständige, nicht nur an das Objekt gebundene, Größe zu behandelt ist, so ist diese als eigener Entity-Typ in das Datenmodell aufzunehmen.

- Beispiel:

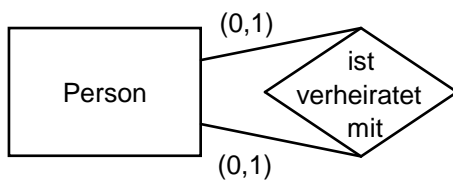


- Wenn eine der folgenden Bedingungen zutrifft, sollte Farbe als eigener Entity-Typ modelliert werden:
 - Bauteile haben mehrere Farben
 - Farben werden noch zusätzlich beschrieben (haben Eigenschaften)
 - die möglichen Werte (des Attributes) sollen auf bestimmte Begriffe eingeschränkt werden (z.B. Schlagworte, Wissensgebiete)

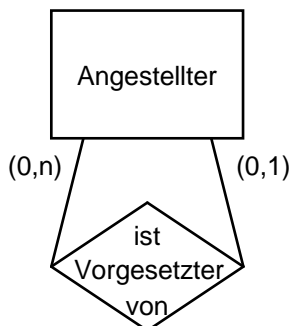
2.7 Beziehungen zwischen Entities desselben Typs

- Beziehungen sind auch zwischen Entities desselben Typs möglich. Man spricht dann auch von Rekursiven oder Reflexiven Beziehungen.

- Beispiel:

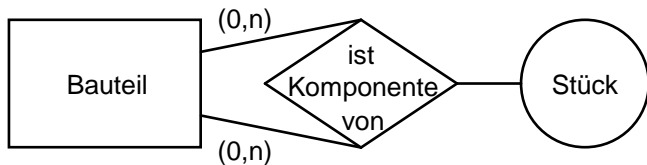


- Beispiel:



- 0 in (0,n) weil nicht jeder Angestellte Vorgesetzter ist ('Die Hackler')
- 0 in (0,1) weil es zumindest einen Angestellten geben wird, der keinen Vorgesetzten hat ('Der Big Boss')

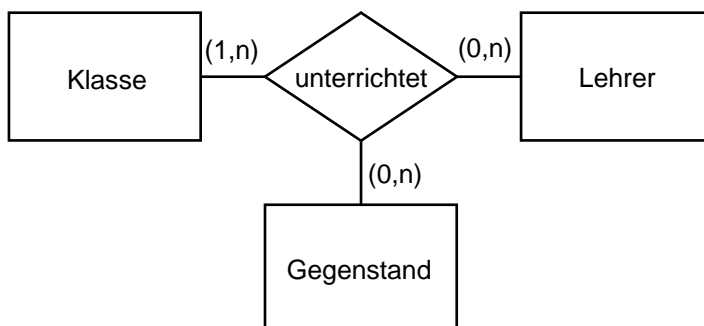
- Beispiel:



- Stücklistenproblematik
 - Konkretes Beispiel dazu:
Bauteile: B1=Fahrrad Type 1, B2= Fahrrad Type 2, B3=Rahmen, B4=Lenker, B5=Rad, B6=Mantel, B7=Schlauch, B8= Felge, B9=Speiche, B10=Nabe,...
ist Komponente von = { (B3,B1,1), (B4,B1,1), (B4,B2,1), (B5,B1,2), (B6,B5,1), (B7,B5,1), (B8,B5,1), (B9,B5,24), (B10,B5,1), ... }
 - 0 in (0,n) oben: Bauteil kommt in keinem anderen Bauteil vor ('Fahrrad Type 1')
 - n in (0,n) oben: Bauteil kommt in mehreren anderen Bauteilen vor ('Lenker')
 - 0 in (0,n) unten: Bauteil besteht aus keinen weiteren Bauteilen mehr ('Speiche')
 - n in (0,n) unten: Bauteil besteht aus mehreren anderen Bauteilen ('Rad')
- Leseart: von links nach rechts oder von oben nach unten

2.8 Beziehungen zwischen mehr als zwei Entity-Typen

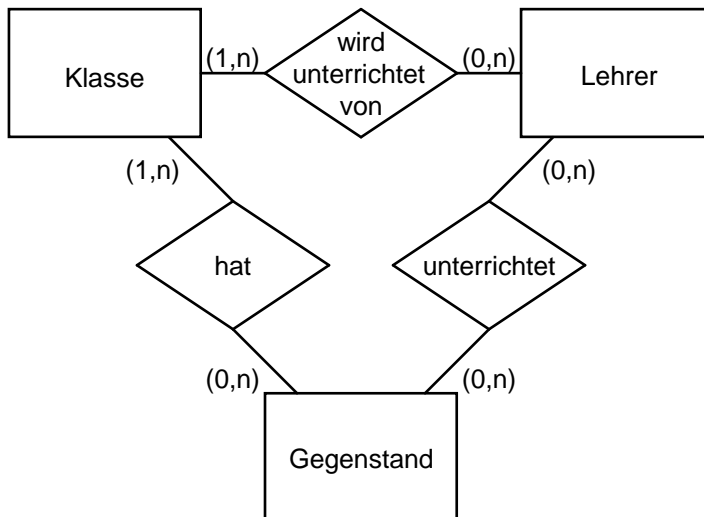
- Die Anzahl der Entities, die an einer Beziehung beteiligt sind, wird Grad oder Wertigkeit der Beziehung (Degree) genannt.
Bis jetzt wurden nur Beziehungs-Typen mit Grad=2 (Zweiwertige oder Binäre Beziehungs-Typen) besprochen, es treten aber auch Beziehungs-Typen mit höheren Graden (Mehrwertige Beziehungs-Typen) häufig auf.
- Wenn der Grad des Beziehungs-Typs gleich g ist so besteht die Relation, die den Beziehungs-Typ darstellt, aus Tupeln mit g Elementen (bei Binären Beziehungs-Typen aus Paaren).
- Beispiel:



unterrichtet = { (3A,Eva,D), (3A,Eva,E), (3A,Max,E), (3B,Eva,D), (3B,Eva,E), (3B,Max,E),... }

- Übung: Geben Sie plausible Erklärungen für die min-Werte in obigem Beispiel an
- Natürlich sind auch in diesem Fall Attribute beim Beziehungs-Typ möglich, Beispiel: Stundenanzahl
- max-Werte müssen alle n sein, sonst ist ein mehrwertiger Beziehungs-Typ nicht notwendig
- Übung: Erstellen Sie ERDs mit Angabe der (min,max)-Werte für die folgenden Situationen
Projekt, Bauteil, Lieferant: Lieferant liefert Bauteil für Projekt
Angestellter, Angestellter, Projekt: Beim Projekt ist der Angestellte dem Angestellten unterstellt

- Es ist zu beachten, dass obiges Beispiel, definiert mit drei Binären Beziehungs-Typen, einen anderen Informationsgehalt hat:

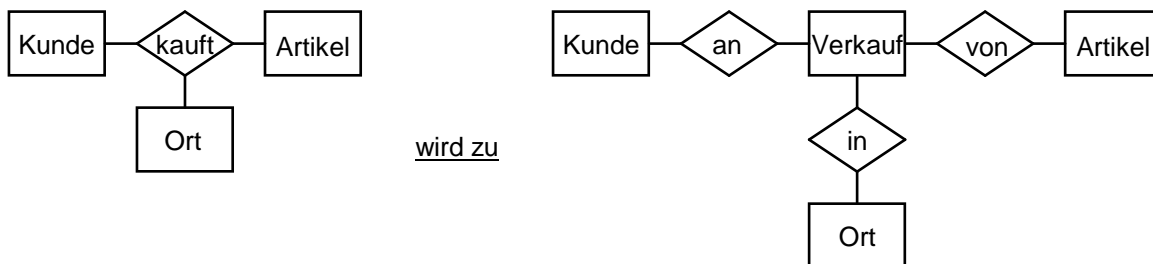


wird unterrichtet von = {(3A,Eva), (3A,Max), (3B,Eva), (3B,Max),... }

hat = {(3A,D), (3A,E), (3B,D), (3B,D),... }

unterrichtet = {(Eva,D), (Eva,E), (Max,E),... }

- Ein Mehrwertiger Beziehungs-Typ kann eliminiert und durch binäre Beziehungs-Typen ersetzt werden:

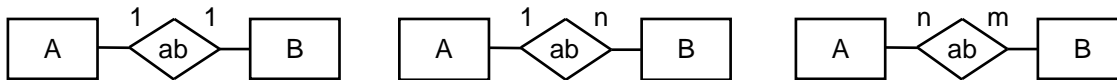


- Die (min,max)-Werte an den ursprünglichen Entity-Typen bleiben gleich, die neuen sind immer (1,1)

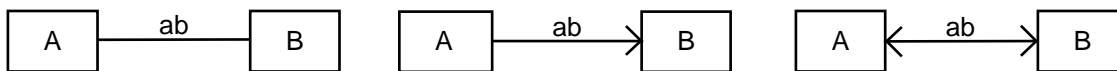
2.9 Weitere Notationsformen von ER-Diagrammen

- Es gibt eine Vielzahl von Notationsformen für ER-Diagramme
- Außer den bisher verwendeten Chen-Diagrammen sind die Diagramme nach C. Bachman weit verbreitet. Bei diesen wird ein Beziehungs-Typ durch eine Linie dargestellt, an der der Name des Beziehungs-Typs vermerkt ist. Damit sind folgende Einschränkungen gegeben:
 - Beziehungs-Typ mit Attributen ist nicht möglich, ein solcher muss als Entity-Typ dargestellt werden.
 - Es sind nur Binäre Beziehungs-Typen möglich, solche mit höherem Grad müssen als Entity-Typen definiert werden.

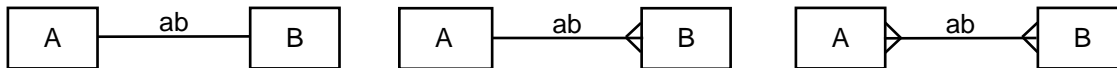
Chen-Diagramm



Bachman-Diagramm mit Arrows

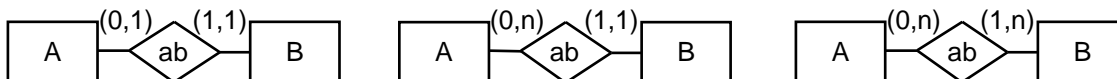


Bachman-Diagramm mit Crowfeet

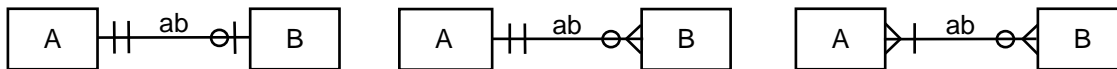


- In einem Diagramm entweder Arrows oder Crowfeet verwenden (nicht mischen)
- Ein rekursiver Beziehungs-Typ mit Crowfeet wird auch 'Pig Ear' genannt
- Die (min,max)-Notation wird auch mit Symbolen (und damit eingeschränkt) verwendet:
 - Die Symbole sind auf der anderen Seite der Linie als die (min,max)-Angaben
 - Das Symbol für max ist immer am Rechteck
 - Symbol für 0: Kreis
 - Symbol für 1: Linie
 - Symbol für n: Pfeilspitze oder umgekehrte Pfeilspitze

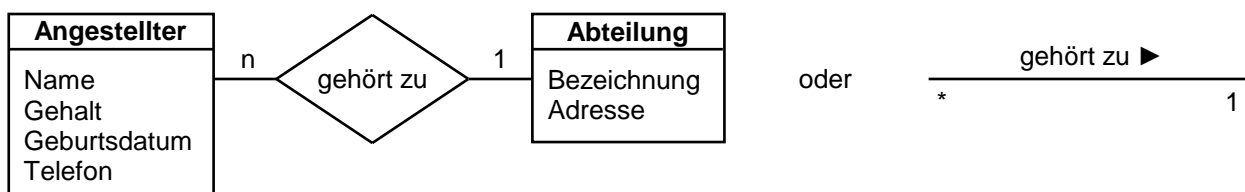
Chen-(min,max)-Notation (same-side cardinality)



Bachman-Symbol-Notation (look-across cardinality)



- In weiteren Varianten werden die Attribute im Rechteck notiert (Name des Entity-Typs, auch fett, wird mit einer Linie von den Namen der Attribute getrennt)
 - Vergleiche: Klassendiagramme (Class Diagrams) in UML (Unified Modelling Language)
 - Vorteile: Übersichtlichkeit, weniger Graphikelemente
 - Nachteile: Rechtecke werden unterschiedlich groß, bei Attributen von Beziehungs-Typen nicht verwendbar
 - Assoziation: Bezeichnung für den Beziehungs-Typ
 - Multiplizität: Bezeichnung für die (min,max)-Werte
- Beispiel:



3 Vom Konzeptionellen Datenmodell zur Implementierung in Tabellen

- Vorbemerkung: In diesem Kapitel können die Begriffe Datei - Satz - Feld analog zu den Begriffen Tabelle - Zeile - Spalte verwendet werden

3.1 Identifizierung von Entities

- Definitionsgemäß müssen Entities von einander unterscheidbar und damit identifizierbar sein.
- Superschlüssel (Superkey, Oberschlüssel, Identifizierende Attributkombination): Menge von Attributen, bei denen sich je zwei Entities des Entity-Typs in mindestens einem Attributwert unterscheiden.
- Schlüssel(kandidat) (Key, Candidate Key): Superschlüssel, der minimal ist (d.h. es kann kein Attribut weggelassen werden, ohne dass die identifizierende Eigenschaft verloren geht). Entweder hat ein Entity-Typ auf Grund der bisherigen Erhebungen mindestens einen Schlüssel oder ein eigener Schlüssel muss nun eingeführt werden (das muss immer möglich sein, da Entities definitionsgemäß voneinander unterscheidbar sein müssen)
- NULL-Wert: Speziell definierter Wert, der bei jedem Attribut auftreten kann und bedeutet, dass bei diesem Entity keine Information über den Wert des Attributs vorhanden ist (Nicht zu verwechseln mit den Werten 0, " oder 'Null' eines Attributs).
- Entity-Integrität: Bedingung, dass es für jeden Entity-Typ (mindestens) einen Schlüssel geben muss, in dessen Attributwerten kein NULL-Wert auftritt.
- Einfacher Schlüssel: Schlüssel besteht aus einem Attribut
- Zusammengesetzter Schlüssel (Concatenated Key): Schlüssel besteht aus mehreren Attributen
- Teilschlüssel: Ein oder mehrere Attribute eines Zusammengesetzten Schlüssels
- Primärschlüssel (Primary Key): Ein Schlüssel(kandidat) wird als Primärschlüssel definiert. Die anderen Schlüssel(kandidaten) werden als Alternate Keys bezeichnet.
- Regeln für die Wahl des Primärschlüssels:
 - Der Primärschlüsselwert muss beim Hinzufügen eines Entities vollständig bekannt sein (Entity-Integrität)
 - Da der Primärschlüssel das Entity während der ganzen Lebenszeit identifiziert, darf sein Wert nicht geändert werden müssen (dadurch würde eigentlich ein neues Entity entstehen).
 - Der Primärschlüssel soll möglichst wenige Attribute umfassen, die Darstellung der Attributwerte soll möglichst kurz sein.
- Die Attribute, die den Primärschlüssel darstellen, werden durch Unterstreichen kenntlich gemacht (bei Bedarf auch im Entity-Relationship-Diagramm):
Angestellter (Personalnummer, Name, Gehalt, Geburtsdatum)
- Künstlicher Schlüssel (Surrogatschlüssel): Zur einfacheren Identifizierbarkeit von Entities wird oft auch ein zusätzliches, künstliches Attribut (meist fortlaufende Nummer) als Primärschlüssel eingeführt.
Beispiele: Personalnummer, Teilenummer
 - Der Künstliche Schlüssel muss beim Anlegen des Entities vom System vergeben werden können (in Ausnahmefällen auch Wahl durch den Benutzer); ein neuer Schlüssel soll daher einfach ermittelt werden können (z.B. fortlaufend nummerieren)
 - Beim Zugriff auf ein Entity braucht der Benutzer diesen (meist nicht sprechenden) Schlüssel nicht zu wissen, sondern muss über Attributwerte, die ihm bekannt sind, ein Entity auswählen können.
 - Vollständigkeitskontrolle ist bei einer fortlaufenden Nummerierung leicht möglich.
 - Alle Regeln für die Wahl des Primärschlüssels sind eingehalten.
 - Viele Datenbanksysteme sehen einen eigenen Datentyp für die automatische, fortlaufende (+1) Nummerierung vor (z.B. ZÄHLER, SERIAL, IDENTITY).
- Beispiel:
Änderung einer Abteilungsbezeichnung ist einfach möglich, wenn statt einer Abteilungsbezeichnung eine abstrakte Abteilungsnummer als Primärschlüssel verwendet wird.
- Beispiel: Chemische Elemente haben mehrere Schlüssel(kandidaten), nämlich Name (z.B. Eisen), Chemisches Zeichen (z.B. Fe), Ordnungszahl (z.B. 26), Atomgewicht (z.B. 55.847), etc.
Übung: Was würde man als Primärschlüssel wählen?

- Beispiel: Attribute des Entity-Typs Angestellter

| Personal# | Vorname | Nachname | Geburtsdatum | |
|-----------|---------|----------|--------------|---------------------------------------|
| x | x | x | | Superschlüssel |
| x | | | x | Superschlüssel |
| | x | x | x | Schlüssel (eventuell nicht eindeutig) |
| x | | | | Schlüssel / Primärschlüssel |

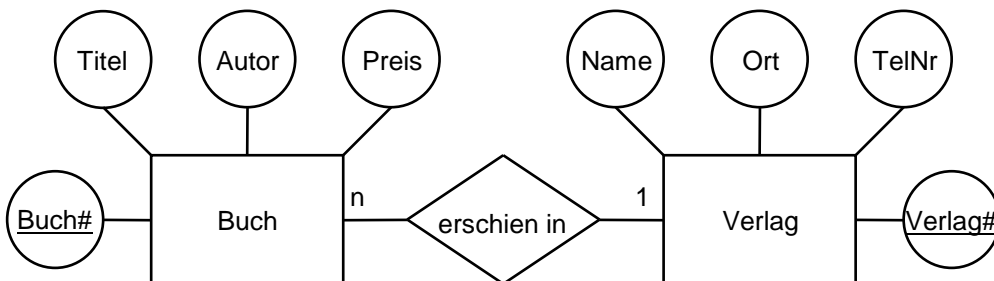
- Beispiel: Aus der Personalverwaltung

| Möglicher Primärschlüssel | Vorteil | Nachteil |
|----------------------------------|--|---|
| Personalnummer | kurz | (nicht sprechend) |
| Geburtsdatum+laufende Nummer | | (nicht sprechend) |
| Familiennamen+laufende Nummer | (sprechend) | kann sich ändern (z.B. Heirat) |
| Sozialversicherungsnummer | muss ohnehin geführt werden | bei Eintritt eventuell (noch) nicht bekannt |
| Eintrittsdatum+laufende Nummer | Eintrittsdatum muss ohnehin geführt werden | kann sich bei Wiedereintritt ändern |
| Abteilungsnummer+laufende Nummer | | kann sich ändern |

- Beispiel: Kennzeichen als Primärschlüssel für ein Auto ist nicht gut geeignet, da ein Auto im Laufe der Zeit verschiedene Kennzeichen haben kann. Außerdem bringt die Verwendung von Wechselkennzeichen Probleme mit sich.
- Beispiel: Wie beurteilen Sie die Wahl von Telefonnummer als Primärschlüssel für eine Firma?

3.2 Redundanz und Fremdschlüssel

- Redundanz: Mehrmaliges Abspeichern desselben Sachverhalts.
- Ein Ziel des Datenentwurfs soll natürlich sein, Redundanz möglichst zu vermeiden ('One fact at one place')
- Beispiel 1):
Folgendes ER-Diagramm ist gegeben:



Buch (Buch#, Titel, Autor, Preis)

Verlag (Verlag#, Name, Ort, TelNr)

In einer Tabelle wird für jedes Buch eine Zeile abgespeichert. Die Information über den zugehörigen Verlag wird ebenfalls in dieser Zeile festgehalten.

Buch+Verlag

| <u>Buch#</u> | Titel | Autor | Preis | <u>Verlag#</u> | Name | Ort | TelNr |
|--------------|----------------------|-----------|-------|----------------|-----------|------|---------------|
| 17 | 999 Zaubertränke | Miraculix | 150 | 5 | Styria | Graz | +43 316 70630 |
| 23 | Römer für Anfänger | Asterix | 130 | 3 | Oldenburg | Wien | +43 1 727258 |
| 48 | Gallische Superdiät | Obelix | 120 | 5 | Styria | Graz | +43 316 70630 |
| 70 | Selbstverteidigung | Asterix | 240 | 7 | Manz | Wien | +43 1 533171 |
| 151 | Handbuch des Druiden | Miraculix | 350 | 5 | Styria | Graz | +43 316 70630 |
| 258 | Tarnen und Täuschen | Asterix | 130 | 5 | Styria | Graz | +43 316 70630 |

Redundanz: Die Tatsache, dass ein Verlag mit einer bestimmten Nummer den entsprechenden Namen hat, am entsprechenden Ort ist und die entsprechende Telefonnummer hat, wird mehrmals (nämlich bei jedem Buch, das der Verlag herausgebracht hat) abgespeichert.

- Beispiel 2):
In einem Bestellsystem werden pro Bestellung ein Kopf (Bestellnummer, Lieferant, Gesamtwert der Bestellung,...) und die Bestellpositionen (Artikel, Menge, Preis,...) abgespeichert.
Redundanz: Die Tatsache, dass die Bestellung einen bestimmten Wert hat, ist zweimal festgehalten (nämlich als Gesamtwert der Bestellung abgespeichert und als Summe der Positionswerte errechenbar).
- Hauptprobleme bei redundanten Daten:
 - Speicherplatzverbrauch
 - Änderungsanomalie (Update Anomaly / Dependency): Tatsache, dass bei Führung redundanter Daten die Änderung einer Tatsache an mehreren Stellen vorgenommen werden muss.
Beispiel 1): Änderung der Telefonnummer des Styria-Verlages
 - Zeitverbrauch: Auf Grund der Änderungsanomalie.
 - Inkonsistenz (Inconsistency) der Daten: Die Daten sind in einem, in sich logisch falschen, Zustand; zu einem Sachverhalt liegen widersprüchliche Informationen vor.
Dateninkonsistenz kann auf Grund der Änderungsanomalie auftreten.
Beispiel 1): Änderung des Ortes beim Styria-Verlag. Aus verschiedenen Gründen (Programmierfehler bis Hardwarefehler) wird der Ort nicht in allen betroffenen Sätzen geändert.
 - Einfüge- und Löschanomalien (Insertion, Deletion Anomaly / Dependency): Vor dem Einfügen des ersten und nach dem Löschen der letzten Zeile, die die (potentiell) redundante Information enthält, ist diese Information nicht vorhanden.
Beispiel 1): Bevor in einem Verlag das erste Buch erscheint, existieren die Daten über den Verlag noch nirgends; nachdem ein Buch aus dem Sortiment genommen wird (z.B. Buch# 70), sind die Daten über den Verlag verloren (Umgehung mit fiktiven Buchnummern wäre in diesem Fall möglich, diese sind aber bei der Verarbeitung der Buchinformationen immer zu berücksichtigen).
 - Zugang zu den redundanten Informationen ist umständlich und zeitaufwendig.
Beispiel 1): Liste aller Verlage.
- Lösung zu Beispiel 1):
In einer Tabelle 'Verlag' wird für jeden Verlag eine Zeile abgespeichert. In einer Tabelle 'Buch' wird für jedes Buch eine Zeile abgespeichert. Um festhalten zu können, in welchem Verlag das Buch erschien, wird in dieser Tabelle noch eine Spalte definiert in dem jeweils der Primärschlüsselwert des Verlages abgespeichert wird.

Buch

| Buch# | Titel | Autor | Preis | Verlag# |
|-------|----------------------|-----------|-------|---------|
| 17 | 999 Zaubertränke | Miraculix | 150 | 5 |
| 23 | Römer für Anfänger | Asterix | 130 | 3 |
| 48 | Gallische Superdiät | Obelix | 120 | 5 |
| 70 | Selbstverteidigung | Asterix | 240 | 7 |
| 151 | Handbuch des Druiden | Miraculix | 350 | 5 |
| 258 | Tarnen und Täuschen | Asterix | 130 | 5 |

Verlag

| Verlag# | Name | Ort | TelNr |
|---------|-----------|------|---------------|
| 3 | Oldenburg | Wien | +43 1 727258 |
| 5 | Styria | Graz | +43 316 70630 |
| 7 | Manz | Wien | +43 1 533171 |

- Fremdschlüssel (Foreign Key, External Key): Spalte (oder Spalten) einer Tabelle, die in einer anderen Tabelle den Primärschlüssel bildet (oder bilden).
Beispiel 1): Die Spalte 'Verlag#' in der Tabelle 'Buch' ist ein Fremdschlüssel (weil diese Spalte in der Tabelle 'Verlag' Primärschlüssel ist).
- Detailtabelle (Childtable): Tabelle, die den Fremdschlüssel enthält (bei 1:n-Bezeichnung auf der n-Seite)
- Mastertabelle (Parenttable): Tabelle, die nicht den Fremdschlüssel enthält (bei 1:n-Bezeichnung auf der 1-Seite)
- Referentielle Integrität (Referential Integrity): Bedingung, dass jeder Fremdschlüsselwert als Primärschlüsselwert existieren muss oder der NULL-Wert ist.
- Maßnahmen, um die Referentielle Integrität sicherzustellen:
 - Bei jeder Aufnahme oder Änderung eines Fremdschlüsselwerts muss überprüft werden, ob dieser Wert als Primärschlüsselwert existiert.
Beispiel 1): Wenn der Wert der Spalte 'Verlag#' in der Tabelle 'Buch' aufgenommen oder geändert wird, muss überprüft werden, ob dieser Wert in der Spalte 'Verlag#' der Tabelle 'Verlag' existiert.
 - Beim Löschen einer Zeile aus der Tabelle, wo der Fremdschlüsselwert als Primärschlüsselwert existiert, muss überprüft werden, ob der Primärschlüsselwert nirgends als Fremdschlüsselwert auftritt.
Beispiel 1): Beim Löschen einer Zeile aus der Tabelle 'Verlag' muss überprüft werden, ob der Wert der Spalte 'Verlag#' in keiner Zeile der Tabelle 'Buch' als Wert der Spalte 'Verlag#' auftritt.

- Bemerkung zu Beispiel 2):
Führen des Gesamtwertes der Bestellung im Bestellkopf kann sinnvoll sein, wenn
 - die Gesamtwerte der Bestellungen oft gelesen, jedoch vergleichsweise selten geändert werden und
 - sichergestellt ist, dass es zu keinen Inkonsistenzen kommt (Transaktionen, Angabe der Integritätsbedingung bei der Definition der Datenbank, Triggers).
 Der zusätzliche Speicherplatzbedarf für den Gesamtwert ist in diesem Fall vertretbar.

3.3 Implementierung von ER-Diagrammen in Tabellen

- Überführung des Konzeptionellen Datenmodells in das Relationale Datenmodell
- Entity-Typ: Für jeden Entity-Typ E wird eine Tabelle T angelegt
 - Für jedes Attribut wird eine entsprechende Spalte definiert
 - Jedes Entity wird durch eine Zeile repräsentiert
 - Die Attributwerte werden in den entsprechenden Spalten der Zeilen (Zellen) als Inhalt abgelegt
- 1:1-Beziehungs-Typ:
 - T1 und T2 seien die beiden Tabellen, die die Entity-Typen darstellen, die am Beziehungs-Typ beteiligt sind
 - Die Tabelle T1 wird um Spalten erweitert, in denen der Primärschlüssel der Tabelle T2 als Fremdschlüssel geführt wird
 - Wenn das Entity, das durch die Zeile repräsentiert wird, an keiner Beziehung des Beziehungs-Typs beteiligt ist, enthält der Fremdschlüssel den NULL-Wert
 - Wenn ein Entity-Typ vollständig am Beziehungs-Typ teilnimmt, soll die entsprechende Tabelle als T1 gewählt werden
 - Da es sich um einen 1:1-Beziehungs-Typ handelt, muss sichergestellt werden, dass im Fremdschlüssel jeder Primärschlüsselwert höchstens einmal auftritt (abgesehen von NULL-Werten)
 - Hinweis: Wenn beide Entity-Typen vollständig am Beziehungs-Typ teilnehmen und sie nicht an anderen Beziehungs-Typen beteiligt sind, könnten beide Entity-Typen samt Beziehungs-Typ auch in einer gemeinsamen Tabelle dargestellt werden (die beiden Entities, die in Beziehung stehen, kommen jeweils in eine Zeile). Dieser Fall tritt jedoch sehr selten auf.
 - Beispiele:

| | |
|--|--|
| Abteilung (<u>Abteilungs#</u> , ..., Personal# des Leiters) | Personal# des Leiters muss eindeutig sein |
| Mann (<u>Mann#</u> , ..., Frau# der Gattin) | Frau# der Gattin muss (abgesehen von NULL-Werten) eindeutig sein |
| Person (<u>Personen#</u> , ..., Personen# des Gatten) | Personen# des Gatten muss (abgesehen von NULL-Werten) eindeutig sein |
- 1:n-Beziehungs-Typ:
 - T1 sei die Tabelle, die den Entity-Typ auf der n-Seite, T2 die Tabelle, die den Entity-Typ auf der 1-Seite des Beziehungs-Typs, darstellt
 - Die Tabelle T1 wird um Spalten erweitert, in denen der Primärschlüssel der Tabelle T2 als Fremdschlüssel geführt wird
 - Wenn das Entity, das durch die Zeile repräsentiert wird, an keiner Beziehung des Beziehungs-Typs beteiligt ist, enthält der Fremdschlüssel den NULL-Wert
 - Beispiele:

| |
|--|
| Angestellter (<u>Personal#</u> , ..., Abteilungs# zu der er gehört) |
| Projekt (<u>Projekt#</u> , ..., Personal# des Leiters) |
| Angestellter (<u>Personal#</u> , ..., Personal# des Vorgesetzten) |
- m:n-Beziehungs-Typ:
 - T1 und T2 seien die beiden Tabellen, die die Entity-Typen darstellen, die am Beziehungs-Typ beteiligt sind
 - Es wird eine eigene Tabelle T erstellt, in der die Primärschlüssel der Tabellen T1 und T2 als Fremdschlüssel geführt werden - Assoziative Tabelle (Junction Table)
 - Der Primärschlüssel der Tabelle T ist ein zusammengesetzter Schlüssel, der aus den Spalten beider Fremdschlüssel besteht
 - Für jedes Attribut des Beziehungs-Typs wird die Tabelle T um eine Spalte erweitert
In diesem Fall muss (bei mehreren Beziehungen zwischen demselben Entity-Paar) der Primärschlüssel um ein oder mehrere Attribute (des Beziehungs-Typs) erweitert werden (so dass Eindeutigkeit erreicht wird) oder überhaupt ein künstlicher Schlüssel eingeführt werden (eher bei eigenem Entity-Typ)
 - Hinweis: Es könnten auch 1:1- und 1:n-Beziehungs-Typen auf diese Weise, in einer eigenen Tabelle, abgebildet werden. Dies ist im Besonderen dann sinnvoll, wenn sehr wenige Beziehungen zwischen den Entities existieren und daher die Spalten mit den Fremdschlüsseln sehr oft den NULL-Wert enthalten oder wenn NULL-Werte prinzipiell nicht verwedet werden sollen.
Es ist dann unbedingt sicherzustellen, dass diese Tabelle tatsächlich einen 1:1- oder 1:n-Beziehungstyp darstellt (Eindeutigkeit des / der entsprechenden Fremdschlüssels)

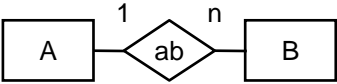
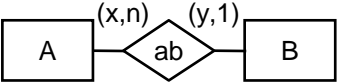
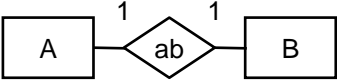
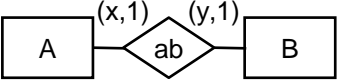
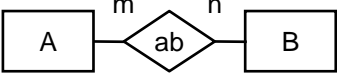
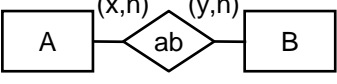
- Beispiele:
 - arbeitet an (Personal#, Projekt#, %-Zeitanteil)
 - benötigt (Projekt#, Bauteil#)
 - liefert (Bauteil#, Lieferanten#)
 - kauft (Kunden#, Artikel#, Stück)
 - hat besucht (Personen#, Schul#, Monate, Kosten)
 - ist Komponente von (Bauteil# der Komponente, Bauteil# des Zusammenbaus, Stück)
 - kauft (Kunden#, Artikel#, Datum, Stück, Rabatt) Annahme: An einem Tag bei gleichem Kunden und Artikel nur ein Rabatt
- Mehrwertiger Beziehungs-Typ:
 - Analog m:n-Beziehungs-Typ wird eine eigene Tabelle T erstellt, in der alle Primärschlüssel der Tabellen, die die am Beziehungs-Typ beteiligten Entity-Typen darstellen, als Fremdschlüssel geführt werden
 - Der Primärschlüssel der Tabelle T ist ein zusammengesetzter Schlüssel, der aus den Spalten aller Fremdschlüssel besteht
 - Für jedes Attribut des Beziehungs-Typs wird die Tabelle T um eine Spalte erweitert
In diesem Fall muss (bei mehreren Beziehungen zwischen demselben Entity-Tupel) der Primärschlüssel um ein oder mehrere Attribute (des Beziehungs-Typs) erweitert werden (so dass Eindeutigkeit erreicht wird) oder überhaupt ein künstlicher Schlüssel eingeführt werden (eher bei eigenem Entity-Typ)
 - Beispiele:
 - unterrichtet (Klassen#, Lehrer#, Gegenstands#, Stundenanzahl)
 - liefert (Bauteil#, Lieferanten#, Projekt#, Datum, Menge) Annahme: keine Teillieferungen an einem Tag
- Vollständiges Beispiel (siehe 2.3.3):

| | |
|---|---|
| Angestellter (<u>Personal#</u> , sonstige Attribute von Angestellter, <u>Abteilungs#</u>) Abteilung (<u>Abteilungs#</u> , sonstige Attribute von Abteilung, <u>Personal#</u>) Projekt (<u>Projekt#</u> , sonstige Attribute von Projekt, <u>Personal#</u>) Bauteil (<u>Bauteil#</u> , sonstige Attribute von Bauteil) Lieferant (<u>Lieferanten#</u> , sonstige Attribute von Lieferant) arbeitet an (<u>Personal#</u> , <u>Projekt#</u>) benötigt (<u>Projekt#</u> , <u>Bauteil#</u>) liefert (<u>Bauteil#</u> , <u>Lieferanten#</u>) Bedingungen: <ul style="list-style-type: none"> - Wert von <u>Abteilungs#</u> in Angestellter darf nirgends NULL sein (min=1) - Wert von <u>Personal#</u> in Abteilung darf nirgends NULL sein (min=1) und derselbe Wert darf nur einmal auftreten (1:1-Beziehungs-Typ) - Wert von <u>Personal#</u> in Projekt darf nirgends NULL sein (min=1) und derselbe Wert darf höchstens 3 mal auftreten (max=3) - usw. | Beziehungs-Typ: gehört zu Beziehungs-Typ: leitet Beziehungs-Typ: verantwortet |
|---|---|

3.4 Zusammenfassung der Vorgangsweise

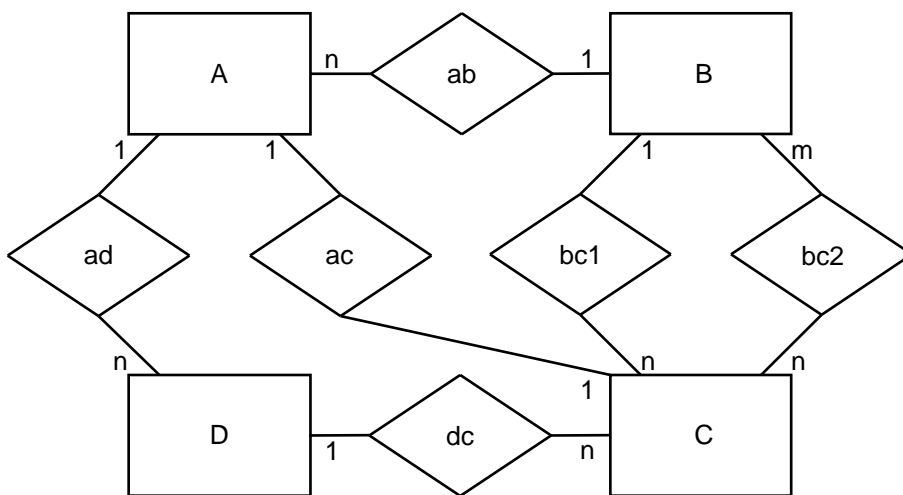
- Bestimme die Entity-Typen
- Bestimme die Beziehungs-Typen
- Bestimme die Komplexitätsgrade der Beziehungs-Typen (Art oder (min,max))
- Bestimme die Attribute von Entity-Typen und Beziehungs-Typen samt ihrer Wertebereiche (Domains)
- Bestimme die Primärschlüsseln
 - geeignete Kombinationen aus bestehenden Attributen
 - Künstliche Schlüssel
- Bestimme eventuelle zusätzliche, anwendungsspezifische Integritätsbedingungen (z.B. ein Angestellter ist für höchstens 3 Projekte verantwortlich)
- Wenn das System im Relationalen Datenmodell implementiert wird, so leite die benötigten Tabellen mit ihren Spalten (Zeilen Aufbau) ab
- Wenn das System mit Dateien implementiert wird, so leite die benötigten Dateien mit ihren Feldern (Satz Aufbau) ab

3.5 Übersicht: Vom ER-Diagramm zu den Tabellen

| Beziehungskardinalität | (min,max)-Notation |
|---|--|
|  <p>A (<u>A#</u>, ...) B (<u>B#</u>, ..., A# > A)</p> |  <p>Tabellenstruktur siehe links</p> <p>x=0 -</p> <p>x=1 Jeder PK-Wert von A.A# muss als FK-Wert in B.A# mindestens einmal vorkommen, daher: insert A → (insert B oder update B.A#) delete B oder update B.A# → FK-Wert noch einmal vorhanden? (Zeile mit PK-Wert löschen???)</p> <p>y=0 FK-Wert darf NULL sein</p> <p>y=1 FK-Wert darf nicht NULL sein</p> |
|  <p>A (<u>A#</u>, ...) B (<u>B#</u>, ..., A# > A)</p> <p>oder</p> <p>B (<u>B#</u>, ...) A (<u>A#</u>, ..., B# > B)</p> <p>FK-Wert muss (abgesehen von NULL-Werten) eindeutig sein = 'UNIQUE except NULL'</p> |  <p>x=0 & y=0 Tabellenstruktur siehe links Wo kommen weniger NULL-Werte zustande?</p> <p>x=0 & y=1 A (<u>A#</u>, ...) B (<u>B#</u>, ..., A# > A) A# NOT NULL und UNIQUE</p> <p>x=1 & y=0 B (<u>B#</u>, ...) A (<u>A#</u>, ..., B# > B) B# NOT NULL und UNIQUE</p> <p>x=1 & y=1 Sehr selten, Designfehler?, Nur ein ET?</p> |
|  <p>A (<u>A#</u>, ...) B (<u>B#</u>, ...) AB (<u>A#</u> > A, <u>B#</u> > B)</p> <p>Beziehungsattribute zusätzlich in AB, dann PK von AB beachten</p> <p>Höherwertige BT analog</p> |  <p>Tabellenstruktur siehe links</p> <p>x/y=0 -</p> <p>x/y=1 Jeder PK-Wert von A / B muss als FK-Wert in AB mindestens einmal vorkommen, daher: insert A / B → insert AB delete AB → FK-Wert noch einmal vorhanden? (Zeile mit PK-Wert löschen???)</p> |

- Beispiel

ERD:



Tabellen:

B (B#, ...)A (A#, ..., B# > B)D (D#, ..., A# > A)C (C#, ..., D# > D, B# > B, A# > A / UNIQUE except NULL)BC(B# > B, C# > C)

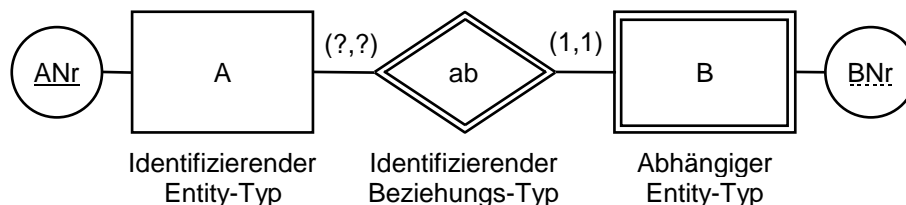
Die Reihenfolge der Tabellen-Definitionen ist zu beachten.

Für den 1:1-BT wurde A# als FK in C verwendet, andernfalls gibt es Probleme bei der Definitions-Reihenfolge.

4 Weiterführende ER-Modellierung

4.1 Abhängige Entity-Typen

- Kern-Entity-Typ (Kernel-Entity-Type, Regular Entity-Type, Fundamental Entity-Type, Dominant Entity-Type):
 - Jedes Entity existiert unabhängig von der Existenz anderer Entities
- Abhängiger Entity-Typ (Dependent Entity-Type, Weak Entity-Type, Subordinate Entity-Type):
 - Kein Entity kann ohne einem übergeordneten Entity (Owner-Entity) eines bestimmten Entity-Typs (Identifizierender Entity-Typ, Identifying Entity-Type) existieren. Wird auch Existenzabhängigkeit (Existence Dependency) genannt.
 - Zwischen dem übergeordneten und dem abhängigen Entity-Typ besteht ein Beziehungs-Typ bei dem der (min,max)-Wert beim abhängigen Entity-Typ immer (1,1) ist (muss vollständig am Beziehungs-Typ teilnehmen). Dieser Beziehungs-Typ wird Identifizierender Beziehungs-Typ (Identifying Relation-Type) genannt.
Hinweis: Die Tatsache, dass der (min,max)-Wert bei einem Entity-Typ (1,1) ist, bedeutet nicht das Vorliegen eines abhängigen Entity-Typs.
 - Die Beziehung vom abhängigen Entity zum identifizierenden Entity ändert sich während der Existenz des abhängigen Entities nicht, non-transferable Relationships (d.h. der Fremdschlüsselwert ändert sich nicht).
 - Der Primärschlüssel des Identifizierenden Entity-Typs ist im Primärschlüssel des Abhängigen Entity-Typs als Fremdschlüssel enthalten (ist Teilschlüssel im Abhängigen Entity-Typ).
 - Üblicherweise werden mit dem Identifizierenden Entity alle von ihm Abhängigen Entities mitgelöscht (siehe SQL: ON DELETE CASCADE).
- Graphische Darstellung im ERD:

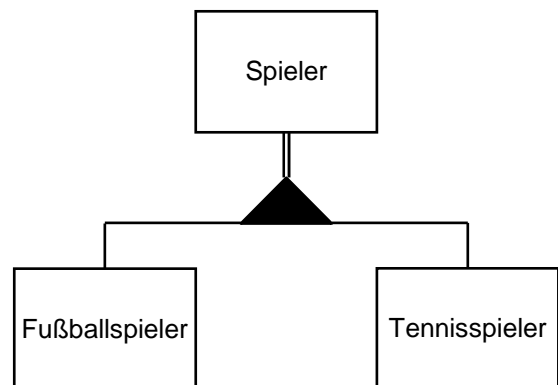
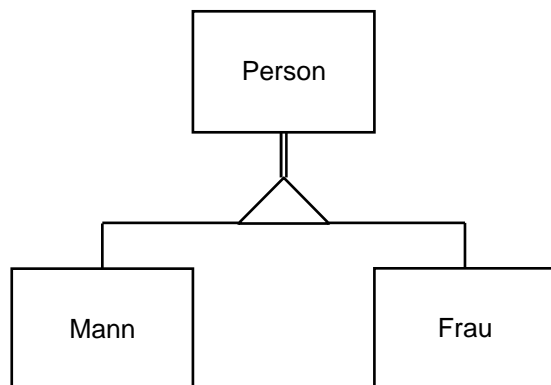


- Tabellen:
 - A (ANr, ...)
 - B (ANr > A, BNr, ...)
- Strichliert unterstrichener Primärschlüssel symbolisiert, dass er alleine nicht identifizierend ist
- Ein Abhängiger Entity-Typ kann auch von mehreren Entity-Typen abhängen, es gibt dann mehrere Identifizierende Beziehungs-Typen alle entsprechenden Fremdschlüssel werden Teil des Primärschlüssels.
- Beispiele:
 - Auftrag – Auftragsposition, Gebäude – Raum, Angestellter – Angehöriger des Angestellten, Geschäft – Geschäftskündigung

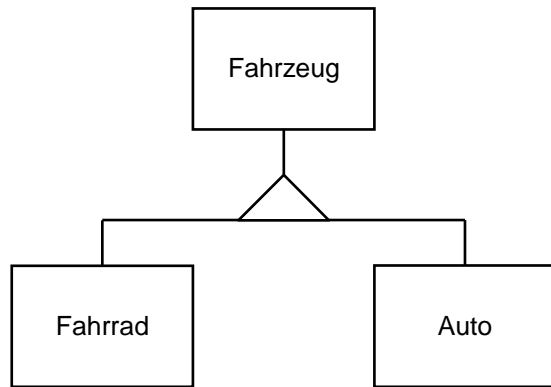
4.2 Überlagerte Entity-Typen

- Sub- und Super-Entity-Typen, Unter- und Ober-Entity-Typen (abgekürzt auch Sub- und Super-Typen)
 - Es gibt einen Super-Entity-Typ und dazu einen oder mehrere Sub-Entity-Typen
 - Jedes Entity eines Sub-Entity-Typs muss auch als Entity im Super-Entity-Typ enthalten sein (die Sub-Entity-Typen sind echte Teilmengen des Super-Entity-Typs)
 - Der Super-Entity-Typ enthält alle Attribute, die den Entities gemeinsam sind.
 - Die Sub-Entity-Typen enthalten die individuellen Attribute (haben in der Regel denselben Primärschlüssel wie der Super-Entity-Typ).
 - Zwischen dem Super-Entity-Typ und den einzelnen Sub-Entity-Typen besteht jeweils ein 1:1-Beziehungs-Typ, der als 'ist ein' ('is-a') bezeichnet wird.
- Übung: Welche Werte treten bei der (min,max)-Notation auf?

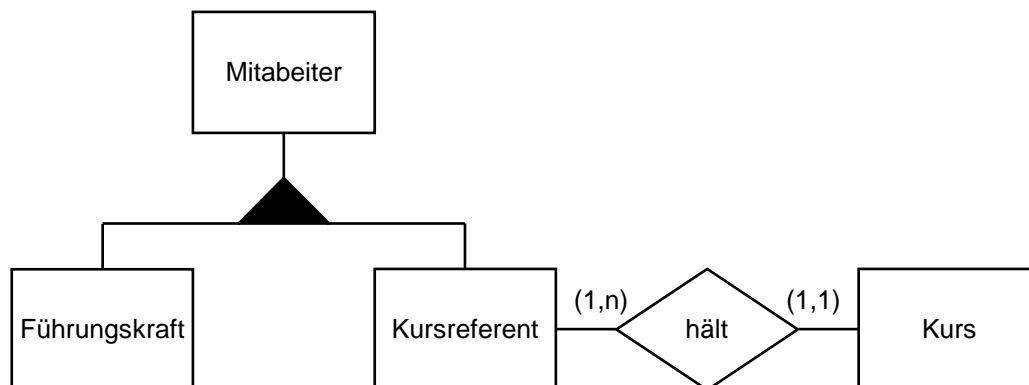
- Beispiele:
 Person - Mann, Frau
 Mitarbeiter - Vollbeschäftigter Mitarbeiter, Teilbeschäftigter Mitarbeiter
 Person - Lehrer, Student
 Geschäft - Kreditgeschäft, Dokumentengeschäft
 Turbine - Axialturbine, Francis turbine, Pelton turbine
 Führerscheinbesitzer - PKW-Fahrer, LKW-Fahrer, Motorradfahrer
- Die Aufteilung in Sub-Entity-Typen oder die Sub-Entity-Typen selbst werden als Spezialisierung(en) (Specialization) des Super-Entity-Typs bezeichnet.
 Beispiel: Mann und Frau sind Spezialisierungen von Person
- Die Zusammenfassung zum Super-Entity-Typ oder der Super-Entity-Typ selbst wird als Generalisierung (Generalization, Verallgemeinerung) der Sub-Entity-Typen bezeichnet.
 Beispiel: Person ist die Generalisierung von Mann und Frau
- Sub-Entity-Typen können selbst wieder Sub-Entity-Typen haben, sodass eine ganze Typ-Hierarchie entsteht (vergleiche Objektorientierte Konzepte wie Vererbung, etc.)
 Beispiel: Mitarbeiter - Sekretär, Ingenieur, Techniker;
 Ingenieur - Kfz-Ingenieur, Elektronik-Ingenieur, Flugzeugbau-Ingenieur;
 Beispiel: Luftfahrzeug - Flugzeug, Hubschrauber, Luftkissenfahrzeug, anderes Luftfahrzeug
 Flugzeug - Segelflieger, Motorflieger, Düsenflugzeug;
- Wenn eine Spezialisierung in mehr als einen Sub-Entity-Typ vorliegt, so können zwei Bedingungen angegeben werden, die die Art der Spezialisierung näher beschreiben:
 - Disjointness Constraint: Gibt es Entities des Super-Entity-Typs, die in mindestens zwei Sub-Entity-Typen enthalten sind?
 nein: disjunkt, disjoint
 ja: nicht disjunkt, overlapping
 - Completeness Constraint (Vollständigkeitsbedingung): Ist jedes Entity des Super-Entity-Typs in mindestens einem Sub-Entity-Typen enthalten?
 ja: total
 nein: partiell, partial
- Sei E der Super-Entity-Typ und seien E1, E2, ..., En die Sub-Entity-Typen für eine bestimmte Spezialisierung:
 - Die Spezialisierung ist disjunkt wenn
 $E_i \cap E_j = \{\}$ für $i \neq j$ und $i, j \in \{1, 2, \dots, n\}$
 - Die Spezialisierung ist total wenn
 $E_1 \cup E_2 \cup \dots \cup E_n = E$
- Für den 'ist ein'-Beziehungs-Typ werden spezielle graphische Darstellungsarten verwendet (Dreieck), mit denen auch die oben angeführten Bedingungen ausgedrückt werden können:
 - total, disjunkt:
 - total, nicht disjunkt (Sportverein mit Sektionen Fußball und Tennis):



- partiell, disjunkt:



- partiell, nicht disjunkt:



Mitarbeiter (Personal#, Name, Adresse, ...)

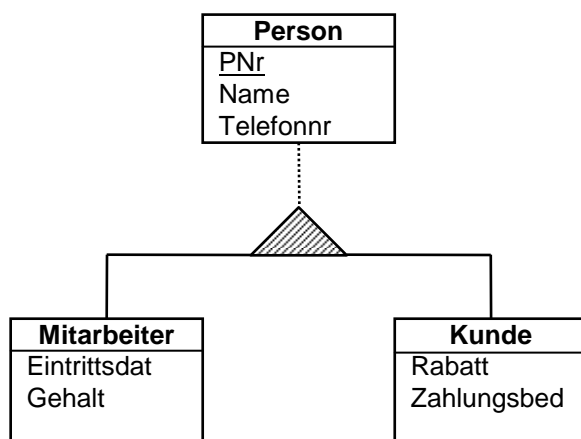
Führungskraft (Personal# > Mitarbeiter, Überstundenpauschale, Anz Sekretärinnen, Geleitete Abteilung, ...)

Kursreferent (Personal# > Mitarbeiter, Stundenkosten, Gehaltene Kursstunden, ...)

Kurs (Kurs#, Titel, ..., Personal# > Kursreferent)

4.3 Implementierung von Überlagerten Entity-Typen in Tabellen

- Beispiel



- Übung: Interpretieren Sie die verschiedenen Möglichkeiten der Disjointness- und Completeness-Constraints

1) Eine Tabelle pro Entity-Typ

Die Primärschlüssel der Sub-Entity-Typen sind dieselben wie der Primärschlüssel des Super-Entity-Typs, gleichzeitig sind sie auch Fremdschlüssel zum Super-Entity-Typ.

a) Die Tabellen für die Sub-Entity-Typen übernehmen den Primärschlüssel vom Super-Entity-Typ (Vertical Mapping)

Person (PNr, Name, Telefonnr)

Mitarbeiter (PNr > Person, Eintrittsdat, Gehalt)

Kunde (PNr > Person, Rabatt, Zahlungsbed)

- bevorzugte Lösung
- Vorteile: strukturell sauber, keine Redundanzen
- Nachteile: Vollständige Information (alle Attribute) über einen Sub-Typ auf zwei Tabellen verteilt (view mit join), Information über Sub-Typ-Zuordnung in eigenen Tabellen, viele Tabellen

b) Die Tabellen für die Sub-Entity-Typen übernehmen alle Attribute vom Super-Entity-Typ (Horizontal Mapping)

Person (PNr, Name, Telefonnr)

Mitarbeiter (PNr > Person, Name, Telefonnr, Eintrittsdat, Gehalt)

Kunde (PNr > Person, Name, Telefonnr, Rabatt, Zahlungsbed)

- Vorteile: Vollständige Information (alle Attribute) über einen Sub-Typ in einer Tabelle
- Nachteile: Redundanzen (sogar mehrfach, wenn overlapping), Information über Sub-Typ-Zuordnung in eigenen Tabellen, viele Tabellen

2) Eine Tabelle pro Sub-Entity-Typ

Mitarbeiter (PNr, Name, Telefonnr, Eintrittsdat, Gehalt)

Kunde (PNr, Name, Telefonnr, Rabatt, Zahlungsbed)

- Vorteile: Vollständige Information (alle Attribute) über einen Sub-Typ in einer Tabelle
- Nachteile: Speicherung der Attribute des Super-Typs bei Completeness Constraint partial, Redundanzen (sogar mehrfach, wenn overlapping), Information über Sub-Typ-Zuordnung in eigenen Tabellen, Zugriff auf alle Entities des Super-Typs (union)
- unübliche Lösung (nur überlegenswert, wenn total und disjoint)

3) Eine Tabelle für alle Entity-Typen (Filtered Mapping)

Person (PNr, Name, Telefonnr, Eintrittsdat, Gehalt, Rabatt, Zahlungsbed)

Zuordnung zu den Sub-Entity-Typen über Abfrage der NULL-Werte aufwendig, daher

a) zusätzlich ein Attribut: ist von welchem Sub-Typ (wenn disjoint)

Person (PNr, Name, Telefonnr, PArt, Eintrittsdat, Gehalt, Rabatt, Zahlungsbed)

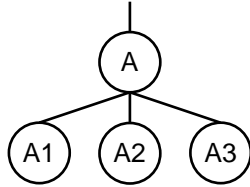
b) zusätzlich pro Sub-Typ ein Attribut (Datentyp Logical): ist von diesem Sub-Typ (wenn overlapping)

Person (PNr, Name, Telefonnr, is-Mitarbeiter, Eintrittsdat, Gehalt, is-Kunde, Rabatt, Zahlungsbed)

- Vorteile: Vollständige Information (alle Attribute) über einen Sub-Typ in einer Tabelle, Information über Sub-Typ-Zuordnung direkt beim Super-Typ, nur eine Tabelle
- Nachteile: viele NULL-Werte in den Attributen der Sub-Typen (speziell, wenn partial und disjoint)
- überlegenswert, wenn Sub-Typen wenige zusätzliche Attribute enthalten (und diese überwiegend nicht NULL sind)
- Mit anderer Anzahl von Attributen in den Entity-Typen oder mehr als zwei Sub-Entity-Typen analoge Vorgangsweise
- Übung: Untersuchen Sie die Beispiele aus 4.2 auf sinnvolle Implementierungsmöglichkeiten
- Übung: Wie verändern sich die Überlegungen, wenn es nur einen Sub-Entity-Typ gibt (welche Disjointness- und Completeness-Constraints sind dann überhaupt sinnvoll)?
- Übung: Erweitern Sie die Überlegungen auf Hierarchien von Super-Sub-Entity-Typen

4.4 Arten von Attributen

- Einfaches Attribut (Simple Attribute, Atomic Attribute): Ein Attributwert ist nicht weiter unterteilt.
- Zusammengesetztes Attribut (Structured Attribute, Composite Attribute): Der Attributwert besteht aus mehreren Teilen.
Beispiel: Adresse ist unterteilt in Straße, Hausnummer, Stadt, Land. Name ist unterteilt in Vorname, Nachname.
Darstellung: Adresse(Straße,Hausnummer,Stadt,Land), Name(Vorname,Nachname)
Graphisch:



- Einwertiges Attribut (Singlevalued Attribute): Bei einem Entity hat das Attribut einen Wert.
- Mehrwertiges Attribut (Multivalued Attribute): Bei einem Entity treten mehrere Werte in diesem Attribut auf.
Beispiel: Farbe eines Autos, Titel einer Person, Vornamen einer Person, Telefonnummern eines Angestellten, Qualifikation eines Angestellten
Darstellung: In geschwungenen Klammern
Graphisch:



- Basis-Attribut (Base Attribute): Der Attributwert eines Entities kann von keinem (keinen) anderen Wert eines Attributs (Werten von Attributen) abgeleitet werden.
- Abgeleitetes Attribut (Derived Attribute): Der Attributwert eines Entities kann von einem Wert (mehreren Werten) eines Attributs (mehrerer Attribute) abgeleitet werden.
Führt zu Redundanzen in der Datenverwaltung und ist im allgemeinen zu vermeiden.
Beispiele: Alter - Geburtsdatum, Auftragswert - Summe Positionswerte des Auftrags
Graphisch:



5 Übungen zum Konzeptionellen Datendesign

5.1 Kursverwaltung

Die Firma TEACH-WARE veranstaltet Kurse in verschiedensten Gebieten (Betriebswirtschaft, EDV, Management, etc.). Die Planung, Abwicklung und Abrechnung der Kurse soll mit einem EDV-System unterstützt werden. Es wurde ein erstes Gespräch mit dem Geschäftsführer der Firma, Herrn Burger, geführt:

Zunächst wird geplant, welche Kurse überhaupt angeboten werden sollen. Jeder Kurs wird mit einer prägnanten Identifikation versehen. Natürlich sind auch Titel, Kurzbeschreibung, Dauer, maximale Teilnehmeranzahl und Preis jedes Kurses interessant. Für eine Abrechnung über Punktekonto ist die in Abzug zu bringende Punkteanzahl festzuhalten.

Es kann auch vorkommen, dass Kurse aufeinander aufbauen, das heißt, dass ein Kurs erst besucht werden soll, wenn zuvor andere Kurse absolviert wurden.

Dann wird das Kursprogramm erstellt, in dem festgelegt wird, welcher Kurs zu welchen Terminen abgehalten wird. Natürlich kann ein Kurs auch mehrmals abgehalten werden. Hier muss auch festgelegt werden, wer den Kurs hält. Ein Kurs kann von mehreren Vortragenden oder Trainern gehalten werden, einer davon ist aber immer der Hauptverantwortliche, der alle Fragen und Probleme, die diesen Kurs betreffen, koordinieren muss. Für Dispositionszwecke ist es interessant zu wissen, welcher Vortragender für welchen Kurs qualifiziert ist.

Es müssen auch die entsprechenden Kursräumlichkeiten gefunden werden. Wir haben selber zwar einige Räume im Haus, mieten zusätzlich aber auch Räume in anderen Gebäuden an.

Das Kursprogramm wird dann verschickt und wir bekommen die Anmeldungen der Teilnehmer für die verschiedenen Kurse herein. Oft müssen jetzt noch andere Räume für die Kurse gesucht werden, weil in dem geplanten Raum zu wenig Plätze zur Verfügung stehen.

Eine Woche vor Beginn eines Kurses werden Kurseinladungen an die Teilnehmer verschickt. Das hat zwei Gründe: Erstens ist darauf vermerkt, wann der Kurs genau beginnt, wann er endet und wo er genau stattfindet, also die Adresse und Raumnummer. Zweitens müssen die Kurseinladungen zum Kurs mitgebracht werden; dadurch kann überprüft werden, ob nur Teilnehmer den Kurs besuchen, die sich auch angemeldet haben.

Am letzten Kurstag bekommen die Teilnehmer eine Bestätigung über den Kursbesuch.

In der Folge ist angedacht, dass die Teilnehmer auch Prüfungen über besuchte Kurse ablegen können. Wann die Prüfungen stattfinden, wer sich anmeldet, wer antritt und welchen Erfolg er dabei hat ist von Interesse.

Nach Abschluss eines Kurses werden Rechnungen an die Teilnehmer verschickt. Die Zahlungen, die in der Folge von den Teilnehmern eingehen, werden registriert. Wenn noch Beträge offen sind, müssen diese eingemahnt werden.

5.2 Produktionsplanungssystem

Die Firma TOOLS & SON produziert Werkzeuge verschiedenster Art. Beim Fertigungsverfahren wird ein zugekaufter Rohling (Guss- oder Schmiedestück, einer bestimmten Qualität und Abmessung) in entsprechenden Bearbeitungsschritten (Drehen, Bohren, Fräsen, Schleifen, Polieren, Prüfen, etc.) zum Endprodukt gefertigt. Für den Materialeinsatz sind nur die Rohlinge von Bedeutung, in den einzelnen Arbeitsschritten fließt kein Material mehr in die Fertigung ein.

Es handelt sich um eine individuelle Auftragsfertigung, das heißt für jedes Endprodukt wird in der Arbeitsvorbereitung ein maßgeschneiderter Arbeitsplan für das konkrete Endprodukt erstellt.

Ein Arbeitsplan besteht aus einem Kopfteil und mehreren Positionen. Im Kopfteil wird festgehalten, um welches Fertigprodukt es sich handelt und wieviele Stück dieses Teiles zu fertigen sind. Es muss auch ersichtlich sein, für welchen Kundenauftrag (samt Kundennamen) die Teile gehören. Die Arbeitsgänge (Arbeitsplanpositionen) stellen eine chronologisch sequentielle Auflistung der einzelnen Produktionsschritte dar. Dabei ist von der Arbeitsvorbereitung anzugeben, auf welchem Arbeitsplatz (Betriebsmittel) zu fertigen ist und wie lange die Bearbeitung eines Stückes dauert.

Pro Betriebsmittel sind neben den identifizierenden Daten (Nummer, Benennung, Standort, Kostenstelle) auch Kapazitätsdaten (Nutzungszeit pro Zeiteinheit) und Kostendaten (Kostensatz pro Zeiteinheit) anzugeben.

Eine Verknüpfung Endprodukt zu benötigtem Einsatzmaterial ist (nach erfolgter Materialrechnung) ebenfalls herzustellen, wobei auch die Kosten des Einsatzmaterials festgehalten werden sollen.

Aus Basis der Kostendaten (Material und Bearbeitung) soll eine Vorkalkulation des Fertigprodukts vorgenommen werden können.

Auf Grund der Zeitdaten ist eine Auslastungsplanung der Betriebsmittel zu ermöglichen und eine Terminierung für die Fertigstellung der Produkte vorzusehen.

Im Fertigungsprozess soll auch ein Rückmeldewesen integriert werden, wobei die tatsächlich benötigten Zeiten (von, bis), sowie die tatsächlich gefertigten Stück (Ausschussmenge mit Ausschussursachenschlüssel) erfasst werden. Damit ist eine Nachkalkulation der Produkte (Kostenträgerrechnung) zu realisieren.

5.3 All Airways Association

Die All Airways Association (AAA) ist eine Vereinigung, in der alle Fluggesellschaften zusammengeschlossen sind. Diese Vereinigung plant, ein umfassendes Informationssystem für die Flugabwicklung zu installieren. Eine erste Erhebung ergibt folgenden Situationsbericht:

Wenn ein Passagier (Passagiernummer, Name, Mr/Mrs, Titel, etc.) einen Flug (oder mehrere) buchen will, gibt er zunächst den gewünschten Abflug- und Zielflughafen an, außerdem das gewünschte Flugdatum und eventuell auch einen Zeitrahmen, wann er wegfliegen oder ankommen will.

Es gibt verschiedene Fluggesellschaften (Name, Firmensitz, etc.), die Flüge veranstalten. Fluggesellschaften werden mit einem dreistelligen Code identifiziert (z.B. PA für PanAm, FUA für Futura Air). Jede Fluggesellschaft betreibt Flugzeuge (Flugzeugnummer, Internationale Registrierungsnummer, Name, Datum der Inbetriebstellung, etc.) verschiedenster Flugzeugtypen (Typidentifikation, Hersteller, Reichweite, etc.).

Die Flughäfen (Name, Stadt, Land, Kapazität in Flugzeugen, etc.) werden ebenfalls mit einem dreistelligen Code verschlüsselt (z.B. VIE für Wien-Schwechat, JFK für New York-John F. Kennedy, IBZ für Ibiza). Die Entfernungen zwischen den Flughäfen muss festgehalten werden, um die Reichweite des Flugzeugtyps bei der Erstellung des Flugplanes berücksichtigen zu können.

Jeder Flug hat einen Abflug- und einen Ankunftsflughafen, die Flüge werden innerhalb einer Fluggesellschaft mit einer dreistelligen Zahl fortlaufend nummeriert (z.B. PA039 zwischen VIE und JFK, FUA916 zwischen IBZ und VIE). Jeder Flug hat eine fixe geplante Abflug- und Ankunftszeit, außerdem ist festgelegt, an welchen Tagen dieser Flug stattfindet (bei Linienflügen z.B. täglich, täglich außer Sa und So, wöchentlich Mo, bei Charterflügen die einzelnen Tage). Vorab wird für jeden Flug festgelegt, wieviele Plätze welcher Klasse zur Verfügung stehen, die Anzahl der verbleibenden freien Plätze eines Fluges muss ebenfalls laufend zu ermitteln sein.

Die vom Passagier gebuchten Flüge werden auf einem Ticket (Ticketnummer, Ausstellungsdatum, Preis, Währung, Verkaufsbüro, etc.) zusammengefasst.

Bevor er den Flug antritt, bekommt der Passagier am Flughafen eine Einsteigekarte (Boarding Card) ausgehändigt, auf der außer Flugnummer, Datum, Abflughafen, Zielflughafen und Name des Passagiers, auch der zugeteilte Sitz (Reihe als Zahl, Sitz als Buchstabe, z.B. 18D) aufscheint. Die Sitzeinteilung hängt von der Flugzeugtype des Flugzeuges ab, mit dem der Flug durchgeführt wird. Für jeden Sitz sind Klasse (First Class / Economy) und Lokation (Fenster, Gang, Mitte) festzuhalten.

Bei jedem Flug muss auch die tatsächliche Start- und Landezeit festgehalten werden können, um Auswertungen über die Pünktlichkeit der einzelnen Flüge erstellen zu können.

5.4 Schulinformationssystem

Ein Schulinformationssystem für eine HTL soll entwickelt werden:

Die abteilungsweise Gliederung einer HTL ist wiederzugegeben. Jeder Lehrer ist einer Abteilung als Stammapteilung zugeordnet, kann aber auch in Klassen anderer Abteilungen unterrichten. Jede Abteilung wird von einem Lehrer als Abteilungsvorstand geleitet.

Für jede Ausbildungsform der Abteilungen ist im Lehrplan festgehalten, welche Gegenstände in welchen Jahrgängen, in welchem Ausmaß (Theorie- und Übungsstunden) unterrichtet werden müssen.

Die Klassen eines Schuljahres werden von den Lehrern in den einzelnen Gegenständen in einem bestimmten Stundenausmaß (Theorie- und Übungsstunden) unterrichtet. Jeder Schüler wird mit einer Semester- und einer Jahresnote pro Klasse und Gegenstand beurteilt. In dem System sollen diese Informationen für mehrere Schuljahre festgehalten werden können.

Die Klassenvorstände der verschiedenen Klassen sollen eruierbar sein, ebenso von wem welche Funktionen (Klassensprecher, Kassier, etc.) ausgeübt werden oder wurden.

Die Entlohnung der Lehrer erfolgt nicht nach gehaltenen Stunden, sondern nach gehaltenen Werteinheiten: Jeder Gegenstand ist einer bestimmten Lehrverpflichtungsgruppe (I bis VI) zugeordnet. Für jede Lehrverpflichtungsgruppe ist ein Faktor (1,167 bis 0,75) festgelegt, der zur Umrechnung von Stunden in Werteinheiten herangezogen wird.

Für jeden Schüler ist ein Erziehungsberechtigter verantwortlich (sofern der Schüler nicht eigenberechtigt ist), wenn Geschwister die Schule besuchen, soll dies ebenfalls ermittelt werden können.

5.5 Kinokette

Die Firma STAR-MOVIES betreibt eine Kinokette mit mehreren Kinos (Name, Adresse,...). In jedem Kino können mehrere Säle untergebracht sein, in denen die Filme gezeigt werden. Der Sitzplan jedes Saales soll festgehalten werden; für jeden Sitz muss eine Reihe und ein Platz angegeben sein. Eine Loge soll wie eine Reihe verwaltet werden.

Die Erstellung eines Spielplanes muss möglich sein. Es können pro Saal natürlich mehrere Filme an einem Tag gezeigt werden.

Um die freien Sitze einer Vorstellung feststellen zu können, ist jeder Kartenkauf zu vermerken. Auf jeder Eintrittskarte soll aufscheinen: Kino, Saal, Filmtitel, Datum, Beginnzeit, laufende Nummer innerhalb der Vorstellung, Reihe, Platz, Preis.

Für die Preisgestaltung ist vorzusehen: Jede Reihe eines Saales hat einen Standardpreis, für bestimmte Vorstellungen können die Reihenpreise aber auch individuell festgelegt werden.

Für Auskunftszwecke sollen die Schauspieler mit ihren persönlichen Daten (Nachname, Vorname, Nationalität, Geburtsdatum, Todesdatum, Bemerkung,...) erfasst werden und die Aussage möglich sein, welche Schauspieler in welchen Filmen mitgespielt haben.

Die analogen Aussagen sollen auch für Regisseure möglich sein, wobei angenommen werden kann, dass es für einen Film nur einen Regisseur gibt. Es ist allerdings möglich, dass bei einem Film der Regisseur auch mitspielt. Die sonstigen Daten eines Filmes umfassen: Titel, Art (Krimi, Western, Jugendfilm,...), Herstellungsjahr, Land, Sprache, Dauer, Verleih, etc.

5.6 Rettungsstelle

Die Rettungsstelle des Roten Kreuzes in Kleinhinterstetten beschließt ein EDV-System zu entwickeln, mit dem sämtliche Einsätze genau erfasst und entsprechende Auswertungen (Abrechnungen, Statistiken, etc.) erstellt werden können.

Es gibt geplante Einsätze (Kontrollfahrten, Therapiefahrten, etc.) und ungeplante Einsätze (Unfälle, akute Krankheiten, etc.). Jeder Einsatz beginnt und endet zu einem bestimmten Zeitpunkt. Bei geplanten Einsätzen sind auch Planbeginn und Planende wesentlich. Die Beschreibung des Einsatzes, Einsatzort, sowie Art und Zeitpunkt der Einsatzanforderung (z.B. Unfallmeldung), bei geplanten Einsätzen auch Rücksprachemöglichkeiten, sind ebenfalls festzuhalten.

Die freiwilligen Mitarbeiter der Rettungsstelle werden beschrieben durch: Name, Wohnadresse, Geburtsdatum, Dienstgrad, Ausbildung.

Der Fuhrpark umfasst verschiedene Fahrzeuge: Marke, Modell, Kennzeichen, Ausstattung.

Jeder Einsatz wird von einem Mitarbeiter geleitet. Im Rahmen eines Einsatzes werden eine oder mehrere Fahrten durchgeführt. Für jede Fahrt sind ein Fahrzeug und eventuell mehrere Mitarbeiter, die verschiedene Aufgaben übernehmen (Fahrer, Beifahrer, Sanitäter, etc), notwendig. Jede Fahrt wird beschrieben durch einen Ausgangsort, einen Zielort, eine Entfernung und eine Zeitangabe (von, bis). Bei einer Fahrt können mehrere Patienten transportiert werden. Dabei soll eine Beschreibung der wahrscheinlichen Verletzungen möglich sein.

Von den Patienten ist zu speichern: Name, Wohnadresse, Geburtsdatum, männlich/weiblich, Sozialversicherungsnummer, Krankenversicherungsanstalt. Bei Mitversicherten soll der eigentliche Versicherungsnehmer ermittelt werden können.

Um häufig angefahrte Orte (z.B. Krankenhäuser, Rehabilitationszentren, etc.) nicht jedesmal wieder eingeben zu müssen, sind diese adressmäßig festzuhalten und gegebenenfalls einer Fahrt als Ausgangs- oder Zielort zuzuordnen. Für diese Fälle sind auch die Entfernungen und Fahrzeiten (soll) zwischen den Orten zu erfassen.

Zusatzaufgaben:

- Pro Fahrt (Einsatz) sollen Beginn- und Endkilometerstand des verwendeten Fahrzeuges (der verwendeten Fahrzeuge) festgehalten werden.
- Da auch Mitarbeiter der Rettungsstelle Patienten sein können (und als solche besondere Konditionen genießen), sollen Person - Mitarbeiter - Patient geeignet als überlagerte Entity-Typen modelliert werden.

5.7 Fremdenverkehrsregion

Alle Unterkünfte der Fremdenverkehrsregion SULMTAL sollen erfasst und verwaltet werden. Im System sollen Funktionen wie Information und Auskunft über Quartiere, Reservierung und Buchung von Zimmern, Erfassung sonstiger Leistungen und Abrechnung der Aufenthalte integriert realisiert werden.

Im Sulmtal befinden sich mehrere Orte (Waglsberg, Radlbach, Kraxndorf, usw.), deren Charakteristika erfasst werden: Name, Postleitzahl, Seehöhe, Adresse und Telefonnummer des Gemeindeamts, etc.

Die Quartiere der Orte sind umfangreich beschrieben: Name, Besitzer / Pächter, Adresse, Telefonnummer, Faxnummer, Art (Hotel, Gasthaus, Frühstückspension, Privatzimmer, Appartementhaus, etc.), Kategorie / Sterne, Gesamtbettenanzahl, Lage (Planquadrat), etc. Mögliche Ausstattungen von Quartieren sollen katalogisiert werden (z.B. Garage, Sauna, Tischtennis). Welche Ausstattung bei welchem Quartier vorhanden ist, muss zu Vergleichszwecken ersichtlich sein (in Prospekten wird diese Information üblicherweise als Piktogramme bei den Quartieren angegeben). Die Benutzung einer derartigen Ausstattung kann auch kostenpflichtig sein (Mengeneinheit, Preis).

Um dem Gast Informationen über die Infrastruktur der Orte zu geben, sollen sämtliche entsprechenden Einrichtungen ebenfalls verzeichnet sein: Bezeichnung, Art (Tennisplatz, Hallenbad, Sportschule, Kaufhaus, Restaurant, Diskothek, usw.), Adresse, Telefonnummer, Lage (Planquadrat). Eventuell sind auch die genauen Entfernungen der Einrichtungen von den Quartieren festzuhalten.

Ein Gast (Herr/Frau, Name, Adresse, Land, Telefonnummer, Geburtsdatum, Staatsangehörigkeit, Beruf) gibt bei der Reservierung oder Buchung eines Quartiers an: Zeitraum (Datum-von, Datum-bis), Anzahl der Personen, Anzahl der Zimmer in bestimmten Varianten (EZ-Einzelzimmer, DZ-Doppelzimmer, 3Z-Dreibettzimmer, DU-Dusche, WC, BK-Balkon, Telefon, etc.), Verpflegung (Ü-Übernachtung, ÜF-Übernachtung mit Frühstück, HP-Halbpension, VP-Vollpension), Bemerkungen.

Beispiel 1: 01.07.2005-05.07.2005 4 Personen 2DZ,DU,Wc / HP

Beispiel 2: 03.07.2005-10.07.2005 4 Personen 2DZ,DU,Wc,BK / 1ÜF,3VP
2 Personen 2EZ,DU,Wc / 1HP,1VP

Das Vorhandensein dieser Leistungen oder angebotenen Kombinationsmöglichkeiten (preislich bewertet) können eine Auswirkung auf die Quartierwahl haben. Die vorhandenen freien Kapazitäten der Quartiere sind vor einer Buchungsbestätigung zu berücksichtigen.

Bei oder kurz vor Antritt des Aufenthalts werden die passenden Zimmer per Zimmernummer zugeordnet. Dabei kann noch das Stockwerk oder die Lage eines Zimmers als Kriterium für die Zuteilung herangezogen werden. Bei der Zuteilung kann auch eine Rolle spielen, dass gewisse Zimmer über eine Verbindungstür verfügen.

Der Zimmerpreis richtet sich tageweise nach Quartier, Zimmerausstattung und Saison (von-Datum, bis-Datum), wobei auch zu berücksichtigen ist, ob nur Ü, ÜF, HP oder VP in Anspruch genommen wird.

Bei der Abrechnung werden auch die jeweils benutzten kostenpflichtigen Quartiereinrichtungen (siehe oben) angeführt (Menge). Ortsspezifisch wird eine nach Erwachsene und Kinder differenzierte, tageweise Ortstaxe bei der Abrechnung eingehoben. Diese Einnahmen müssen an die jeweilige Gemeinde abgeführt werden.

Varianten und Erweiterungen:

- Unterscheidung Reservierung - Buchung:
Reservierung ist eine unverbindliche Option auf einen Aufenthalt für eine bestimmte Dauer oder bis zu einem bestimmten Datum, dann buchen oder verwerfen.
Buchung ist die definitive Bestellung eines Aufenthaltes, in diesem Fall eventuell Stornierung mit entsprechender Stornogegebühr.
- Zu den Gästen werden deren verschiedene Interessen (Tennis, Schifahren, Surfen, etc.) erfasst, um einen geeigneten Urlaubsort oder ein passendes Quartier (Ausstattung, Lage) vorschlagen zu können.
- Anzahlungen (beim Buchen, bei Antritt des Aufenthalts,...) werden erfasst und bei der Abrechnung berücksichtigt.
- Erfassung der Daten für das Gästebuch / den Gästepass (Name und Geburtsdatum der Gattin / des Gatten, Name und Geburtsdatum der Kinder, Anzahl der Teilnehmer bei Reisegruppen, geplantes und tatsächliches Abreisedatum)
- Diverse Rabatte und Wochenpreise sollen angeboten werden können. Die Preisgestaltung soll personen- oder zimmerbezogen erfolgen können.
- Für Kinder können, wenn sie in Zusatzbetten im Zimmer der Eltern / Angehörigen übernachten, bestimmte Ermäßigungen gewährt werden.
- Die von den Telefonapparaten der Zimmer geführten externen Telefongespräche werden erfasst, dokumentiert und bei der Abrechnung berücksichtigt.
- Sonstige Nebenkosten (Getränke, zusätzliche Speisen,...) werden mit Quittung erfasst und am Ende eines Aufenthalts in Summe abgerechnet; pro Leistung soll auf einer Aufstellung ersichtlich sein: Datum, Bezeichnung, Menge, Einzelpreis, Wert, Gesamtwert

5.8 Fußballdatenbank

Für eine Sportzeitschrift wird eine Fußballdatenbank entworfen:

- In dieser Datenbank werden verschiedene Fußballmannschaften verwaltet. Jede Mannschaft hat einen eindeutigen Namen, ist in einem bestimmten Jahr gegründet worden und ist an einer Adresse beheimatet.
- Zu jeder Fußballmannschaft gehören Fußballspieler. Für jeden Spieler soll die SVNr gespeichert werden, welche ihn identifiziert. Jeder Spieler hat einen Namen, eine Wohnadresse und ein Geburtsdatum, sowie eine Position an der er spielt.
- Fußballmannschaften beteiligen sich an Spielen. Diese werden durch die Adresse des Stadions, Tag und Uhrzeit eindeutig festgelegt. Für sie werden die beiden beteiligten Mannschaften und der Schiedsrichter gespeichert. Das Ergebnis soll ermittelt werden können.
- Falls das Spiel zu einem Turnier gehört, so ist diese Tatsache ebenfalls zu speichern.
- Falls ein Spieler in einem Spiel Tore geschossen hat, soll die Anzahl der Tore gespeichert werden.
- Für jeden Schiedsrichter werden dieselben Daten gespeichert wie für die Spieler, außer die Position. Zusätzlich wird noch das Datum der Schiedsrichterprüfung und die Berechtigungsklasse verwaltet.
- Für jedes Turnier werden eine von der FIFA vergebene eindeutige Nummer, der Name, Beginn- und Enddatum, sowie die beteiligten Mannschaften in der Datenbank gespeichert.

5.9 Tankstellenkette

Eine Tankstellenkette möchte in allen größeren Orten der Region Tankstellen einrichten und die relevanten Daten in einer Datenbank speichern.

In Kleinkennsternich (Bezirk Hinterberg, 2 000 Einwohner) gibt es noch keine Tankstelle, in Benzhausen (Bezirk Mobilland, 75 000 Einwohner) bereits drei Tankstellen. Eine dieser Tankstellen befindet sich in der Rennstraße 77 (PLZ 98765) und hat eine Fläche von 3 700 m². Sie hat 13 Mitarbeiter, über die Personalnummer, Name und Adresse gespeichert werden. Ein Mitarbeiter kann mehreren Tankstellen zugeordnet sein. Mitarbeiter können andere Mitarbeiter anleiten, wobei ein Mitarbeiter mehrere Chefs haben kann.

Jede Tankstelle der Kette kann selbst wählen, von welchem Großhändler sie den Kraftstoff bezieht. Der Großhändler 'Peter Petrolius AG' ist für die Kette von großem Interesse, auch wenn er noch mit keiner Tankstelle zusammenarbeitet. Zu jedem Großhändler wird über den eindeutigen Firmennamen hinaus noch die Anschrift des Hauptsitzes gespeichert.

Die oben genannte Tankstelle hat 8 Kraftstofftanks, wobei der Tank mit der Nummer 3 ein Fassungsvermögen von 70 000 l und einen Füllstand von 35 % hat. Er enthält den Kraftstoff mit der Bezeichnung 'Superbenzin bleifrei'. Der Kraftstoff 'Superbenzin bleifrei' hat eine Oktanzahl von 95 und kostet heute (an einem bestimmten Tag) an dieser Tankstelle 1,129 Euro. Die Tankstelle verfügt über 12 Zapfsäulen. Eine Zapfsäule ist jeweils mit mehreren Kraftstofftanks verbunden, ein Tank kann mehrere Zapfsäulen speisen. Der Tank 8 ist vorübergehend stillgelegt: Er enthält keinen Kraftstoff und versorgt keine Zapfsäule.

Es werden Angaben über die Tankvorgänge gespeichert. Zu jedem Tankvorgang muss ersichtlich sein, an welcher Zapfsäule welcher Kraftstoff in welcher Menge getankt wurde und wie hoch der Tankpreis war. Gegebenenfalls kann dem Tankvorgang auch das betankte Fahrzeug zugeordnet werden – nämlich dann, wenn das Bezahlen 'vergessen' wurde. So wurde beispielsweise in der betrachteten Tankstelle am 11.11.2002 um 11:22 Uhr an der Zapfsäule 11 ein blauer VW Golf mit dem polizeilichen Kennzeichen 'GA UNER7' betankt, ohne dass bezahlt wurde. Das ist besonders ärgerlich, weil dieses Fahrzeug schon zum dritten Mal einem solchen Tankvorgang zugeordnet wurde.

5.10 Restaurantbetrieb

Für einen Restaurantbetrieb soll zur Verwaltung eine Datenbank entwickelt werden. Zeichnen Sie aufgrund der vorliegenden Informationen ein ER-Diagramm. Verwenden Sie (min,max)-Notation.

Es gibt mehrere Filialen. Jede Filiale ist eindeutig identifiziert durch den Namen (NAME) und den Bezirk (BEZIRK) in dem sie sich befindet. Außerdem ist eine Adresse (ADRESSE) bekannt. Manche Filialen haben einen Gastgarten. Bei diesen Filialen werden zusätzlich die Öffnungszeiten (VON), (BIS) des Gartens gespeichert.

Mitarbeiter haben eine eindeutige Sozialversicherungsnummer (SVNR) und einen Namen (NAME). Grundsätzlich wird zwischen Koch und Kellner unterschieden. Bei jedem Kellner ist bekannt, wie hoch sein monatliches Fixum ist (FIXUM) und ob er umsatzbeteiligt ist oder nicht (UMSATZ), bei jedem Koch, wie hoch sein Stundenlohn ist (LOHN) und wie viele Stunden pro Woche er (im Regelfall) arbeitet (STUNDEN).

Jeder Tisch wird von einem, maximal jedoch von zwei Kellnern betreut. Jeder Tisch ist eindeutig identifiziert durch die Filiale und eine Tischnummer (TNR). Außerdem ist für jeden Tisch bekannt, ob es sich um einen Rauchtisch (RAUCHER) handelt oder nicht. Die Tatsache, dass bestimmte Tische benachbart sind, soll festgehalten werden.

Jedes Gericht ist eindeutig durch den Namen (NAME) gekennzeichnet, außerdem wird der Preis (PREIS) und die Art (ART) gespeichert, also ob es sich beispielsweise um eine Hauptspeise, Zuspese oder Vorspeise handelt. Es ist bekannt, welche Köche welche Speisen zubereiten können. Außerdem wird in der Datenbank gespeichert, welche Menge (MENGE) von welcher Zutat für die Zubereitung einer bestimmten Speise gebraucht wird. Zutaten sind eindeutig identifiziert durch ihre Bezeichnung (BEZ). Zusätzlich wird der Lagerstand (STAND) von jeder Zutat gespeichert.

Daten über Lauf-Kundschaften werden nicht gespeichert. Für Stammkunden hingegen werden eine eindeutige Kundennummer (KNR), der Name (NAME) und die Anzahl der Treuepunkte (PUNKTE) gespeichert.

Jede Bestellung ist eindeutig identifiziert durch eine BestellungsID (BID). Außerdem ist bekannt, wann die Bestellung erfolgte (WANN), welche Menge (ANZAHL) von welchen Gerichten die Bestellung umfasst und an welchem Tisch die Bestellung aufgegeben wurde. Wenn die Bestellung von einem Stammkunden aufgegeben wurde, wird auch dieser bei der Bestellung gespeichert.

5.11 Chess Tournaments (CLIL)

Design and implement a database for chess clubs, which also organize chess tournaments.

These are the facts:

Each chess club is identified by an ID, has a name, an address and a year of foundation. Each member has got a unique number, a first name, a last name, an address, a phone number and an email address. Every member belongs to one club; the date, when the member joined the club, has to be stored. Specific members are playing members (players), in this case they have a nationality, an actual ranking, and the date, when they achieved this ranking. The valid rankings shall be documented: e.g. CL for club-level, IM for international master, GM for grandmaster.

A tournament is identified by a code, has a name, a location, a starting date, an ending date and is hosted by one club. A chess club can host many tournaments. Various members of chess clubs can be responsible for the organization of a tournament. They don't necessarily have to belong to the same club that hosts the tournament.

Players can participate in zero or many tournaments. In every tournament there are a certain number of matches between two participating players, one playing white and one black. A match is identified by a consecutive number within the tournament and has a starting and an ending point of time. The number of moves and the result of each match have to be recorded: W=won by white, B=won by black, D=draw. The final result (place) of every player in every tournament is also to be registered.

The following tasks need to be done:

- 1) Design an entity-relationship model for the given situation, use (min,max)-notation and draw all attributes
- 2) Transfer the ERD into a relational database schema (mark all primary and foreign keys)
- 3) Create an appropriate SQL-script for creating the tables (with all constraints)
At the begin of the script write the statements to drop all tables
- 4) Fill the tables with test data in an SQL-script
- 5) Do the following SQL queries (and manipulations, if applicable) in a script:
 - a) List of all clubs, which did not host a tournament so far
 - b) List of all clubs, which have only players of Austrian nationality
 - c) How many players have participated in the different tournaments?
(also newly announced tournaments where nobody is registered at the time)
 - d) How many matches has every single player won?
(also players who have not won so far)

Remark: Use only English identifiers (from the names in the ERD to the names of the table elements)

5.12 Library (CLIL)

A library keeps books and magazines.

Every book is published by a publishing company, for every book several copies may exist. They can be borrowed by customers.

Magazines are published in multiple issues, they only exist once, borrowing is not possible.

Both - articles, published in the issues of magazines, and books - have to be administrated in a way, that convenient information retrieval is possible: In addition to the assignment of a subject area, there shall be keywords associated with each book and article. The relevance of all associated keywords and also synonym keywords have to be recorded. Articles and books can always have multiple authors. Certain books can be the translation of another book, articles can reference each other.

Each borrowing is done by an employee. An employee is also responsible for returning the book's copy.

Customers can reserve books, normally each reservation leads to a borrowing.

Books and magazines are stored in shelves. All issues of one magazine are stored in one shelf. For the storage of books each shelf is assigned to a subject area.

5.13 School Organisation (CLIL)

Each school is defined by its individual number, a description and an address. Each school has several classes with their unique identification code within the school. For each class a type must be defined and a classroom assigned. Moreover, each class has a teacher as its form master.

Teachers teach at individual schools; they earn a salary and can be sorted by their exam dates. Each teacher is assigned to a specific school as his/her main school. For students, their birth dates and (parents') telephone numbers are to be registered. Each student is assigned to a class. Both teachers and students can be described by a unique personal code, a name and an address.

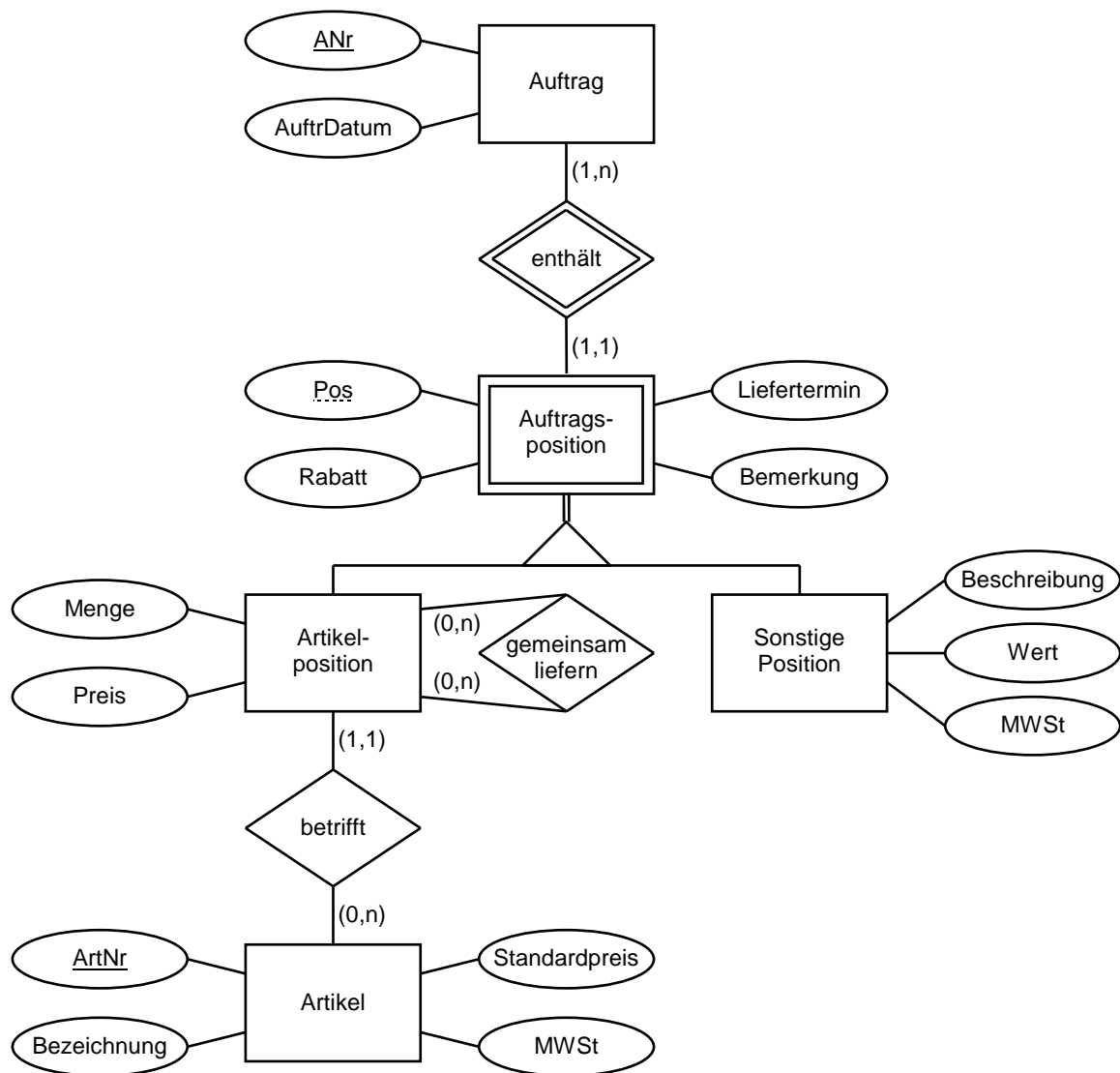
The subjects taught are represented by a short form as well as a full description. It must be registered which teachers teach which classes in which subjects which number of hours. Students are graded in each subject by a winter term and a summer term mark.

Additional remarks:

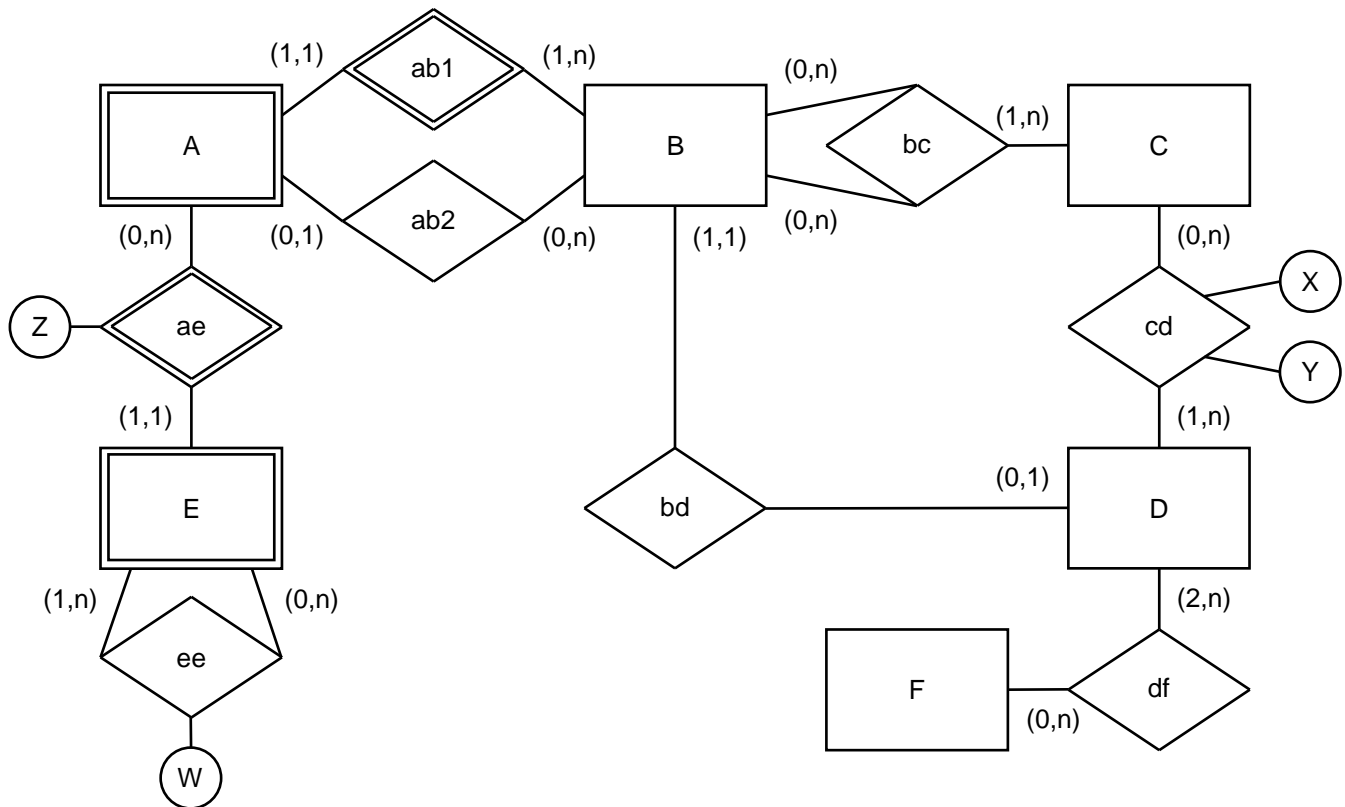
All information shall be stored for the single current school year only. There is no archive function required for more than one school year. Teachers can also teach in several schools (i. e. others than their main school).

A class may be taught in a single subject by more than one teacher.

5.14 Aufträge

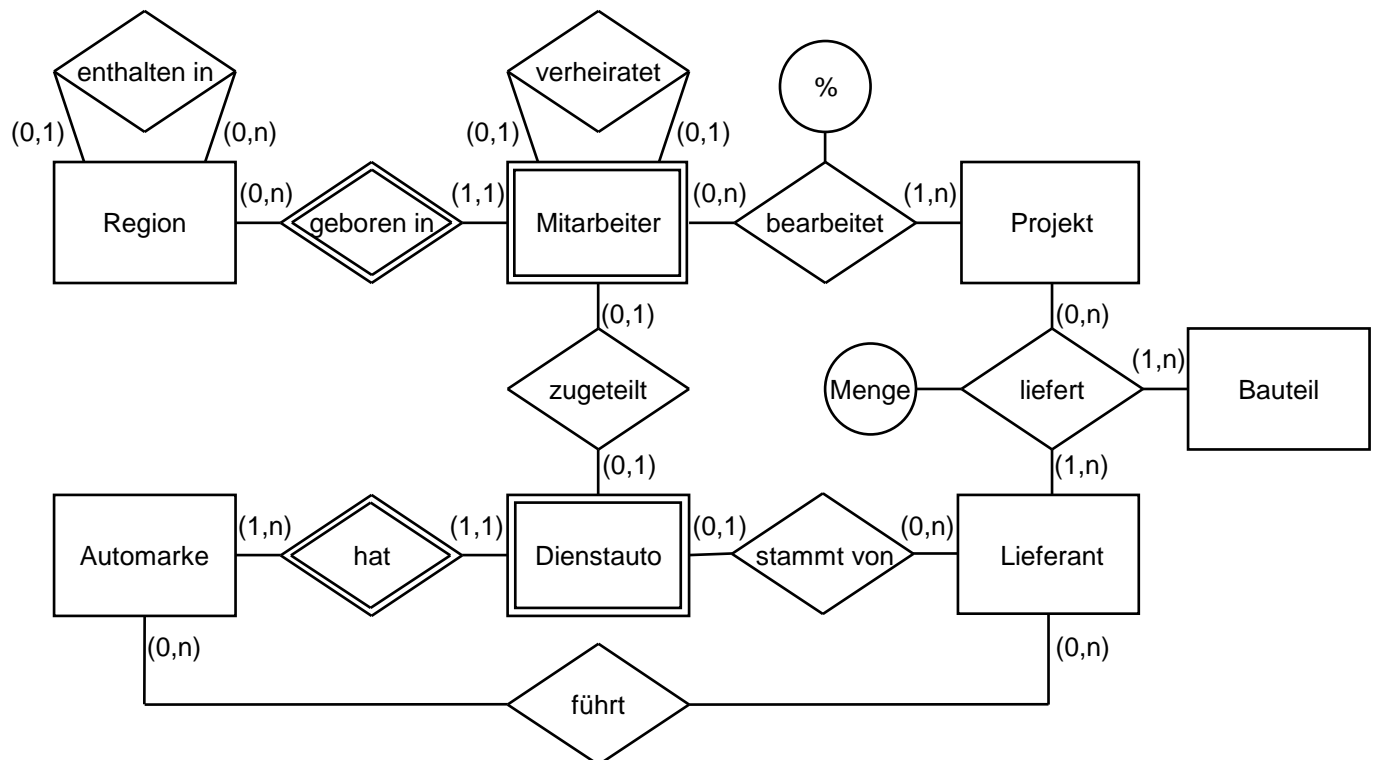


5.15 Ableitung ERD in Tabellen



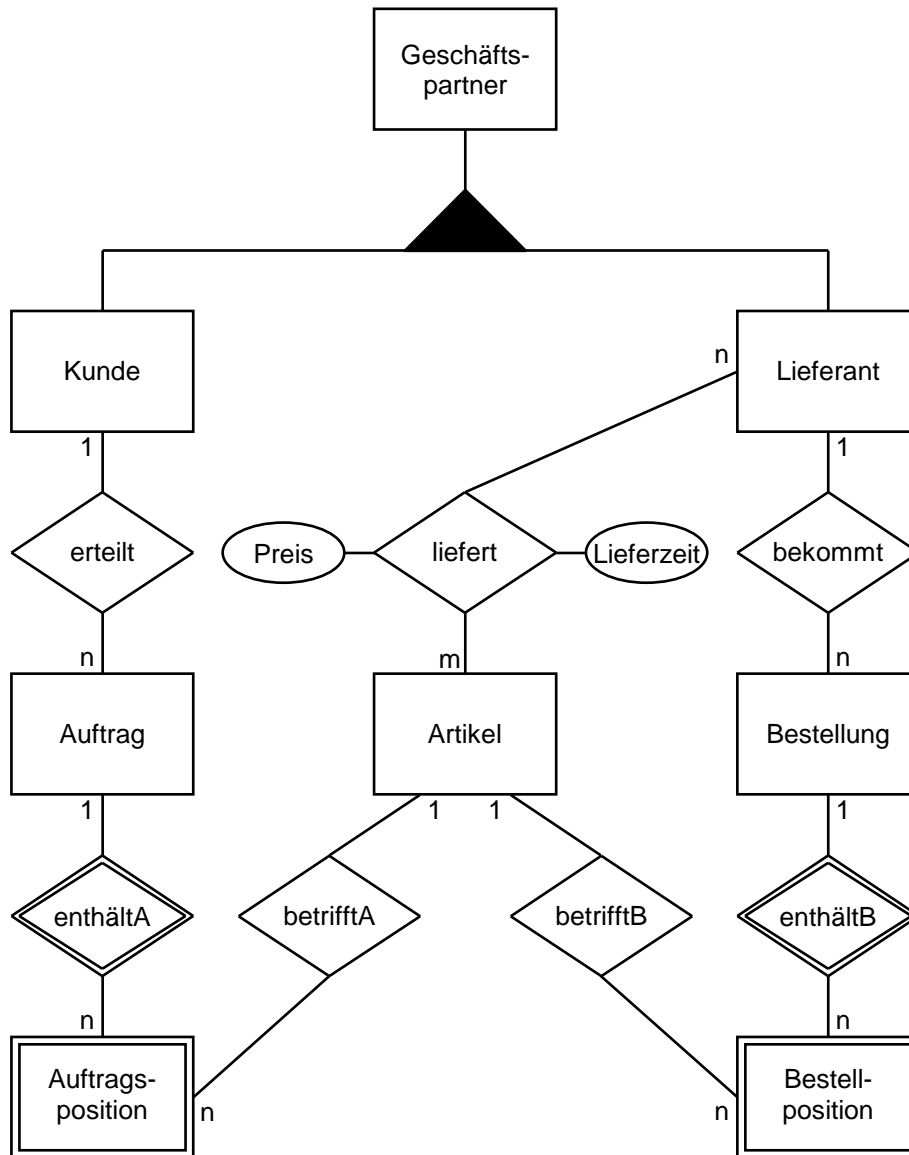
- Das ERD enthält einen Fehler, beschreiben Sie ihn
- Der Primärschlüssel von A sei A#, von B sei B#, etc.
- Leiten Sie das ERD in Tabellen ab (Attributschreibweise), achten auf eine gültige Definitionsreihenfolge
Variante 1: ohne abhängige Entity-Typen, Variante 2: mit abhängigen Entity-Typen (so wie dargestellt)

5.16 Ableitung ERD in Tabellen



- Der Primärschlüssel von Region sei RegNr, von Mitarbeiter sei MitNr, etc.
- Leiten Sie das ERD in Tabellen ab (Attributschreibweise), achten auf eine gültige Definitionsreihenfolge
Variante 1: ohne abhängige Entity-Typen, Variante 2: mit abhängigen Entity-Typen (so wie dargestellt)

5.17 Einfaches Auftrags-Bestell-System (Handelsbetrieb)



Attribute der Entity-Typen:

Geschäftspartner: GNr, GName, Adresse

Kunde: Jahresumsatz, Kreditlimit, Zahlungsbedingungen

Auftrag: ANr, ADatum, AStatus

Auftragsposition: APosNr, Menge, Preis, VersandDatumSoll, VersandDatumIst

Artikel: TNr, TBezeichnung, Mengeneinheit, VerkaufsPreis, Mwst, DurchschnittlicherEinkaufspreis, AktuelleBestandsmenge, MindestBestandsmenge

Lieferant: Bankleitzahl, Kontonummer, Lieferbedingungen

Bestellung: BNr, BDatum, BStatus

Bestellposition: BPosNr, Menge, Preis, LieferDatumSoll, LieferDatumIst

Einschränkungen:

- Keine eigene Lieferadresse, keine Teilversendungen, keine Teillieferungen
- Keine Lagerführung mit Lagerorten
- Keine verschiedenen Lager-, Auftrags- und Bestell-Mengeneinheiten
- Keine Lieferscheine und Rechnungen (damit auch keine Zahlungs-/ Lieferbedingungen, Zahlungseingänge, Mahnungen, etc.)
- Keine Rabatte und Rabattstaffeln im Ein- und Verkauf
- Artikelführung ist lieferantenanonym, d.h. welcher Kunde die Artikel welches Lieferanten bekommt, ist nicht nachvollziehbar
- Änderung z.B. Kundendaten
 - aktuelle Kundendaten jeweils in den Auftrag kopieren (= Stand der Kundendaten bei Auftragserteilung)
 - Kundendaten historisieren; Kundendaten des Auftrags auf Grund des Auftragsdatums immer rekonstruieren

6 Relationales Datenmodell

6.1 Grundbegriffe

- Entwickelt von E.F. Codd, 1970 (Physische Datenunabhängigkeit, Abfragesprache)
- Alle Daten werden in Tabellenform repräsentiert, eine solche Tabelle ist eine übersichtliche Darstellungsform für eine mathematische Relation.
- Tabelle (zweidimensional):
 - a) Jede Tabelle besitzt einen Namen (eindeutig innerhalb der Datenbank)
 - b) Die Spalten haben einen Spaltennamen (eindeutig innerhalb der Tabelle), die Anordnung (Reihenfolge) der Spalten ist beliebig
 - c) Die Anordnung (Reihenfolge) der Zeilen ist beliebig, es gibt keine zwei Zeilen mit gleichem Inhalt
 - d) In den Zellen (Kreuzungspunkten von Spalten und Zeilen) stehen die einzelnen Datenwerte
 - e) Spalten, die dem Primärschlüssel angehören, sind unterstrichen (da keine zwei Zeilen denselben Inhalt haben dürfen, muss es immer einen Primärschlüssel geben, dieser besteht gegebenenfalls aus allen Spalten)

KUNDE ← a)

| <u>KUNDEN#</u> | KUNDENNAME | ORT |
|----------------|------------|--------|
| 007 | Bond | Wien |
| 128 | Fleming | London |
| 013 | Bond | London |

Diagramm zur Tabelle KUNDE:

- Die Spaltenüberschriften sind mit Pfeilen von 'b)' markiert.
- Die Zeilenüberschriften sind mit Pfeilen von 'c)' markiert.
- Die Zelleninhalte sind mit Pfeilen von 'd)' markiert.
- Die Spalte KUNDEN# ist mit einem Pfeil von 'e)' markiert.

- Kardinalität (Cardinality) der Tabelle: Anzahl der Zeilen
- Grad (Degree) der Tabelle: Anzahl der Spalten
- Relationenschema (Attributschreibweise) ohne Angabe der konkreten Attributwerte:

KUNDE (KUNDEN#, KUNDENNAME, ORT)

- Eine Relationale Datenbank besteht aus beliebig vielen Tabellen oben beschriebener Art.
- Relation:
 - Entity-Typ hat Attribute: A1, A2, ..., An
 - Menge aller möglichen Werte eines Attributs in einem Entity-Typ heißt Domäne (Domain, Wertebereich, Value Set) des Attributs: dom(A1), dom(A2), ..., dom(An)
 - Anzahl der verschiedenen möglichen Werte einer Domäne heißt Kardinalität (Cardinality) der Domäne: |dom(A1)|, |dom(A2)|, ..., |dom(An)|
 - Beispiel:
KUNDE hat Attribute: KUNDEN#, KUNDENNAME, ORT
dom(KUNDEN#) = {i / i ∈ N & i < 1000}
dom(KUNDENNAME) = Menge aller Firmennamen
dom(ORT) = Menge aller Stadtnamen
 - Relation: $R \subseteq \text{dom}(A1) \times \text{dom}(A2) \times \dots \times \text{dom}(An)$
 - Tupel: $r \in R$
 - Beispiel
Relationenschema: KUNDE (KUNDEN#, KUNDENNAME, ORT)
Relationeninhalte (Ausprägung, Instanz): {(007,'Bond','Wien'), (128,'Fleming','London'), (013,'Bond','London')}
- Da die Tabelle nur eine übersichtliche Darstellung der Relation (also einer Menge) ist, muss auf Grund der Mengeneigenschaft gelten:
 - Die Reihenfolge der Zeilen (Tupel) ist nicht relevant (Reihenfolge der Informationen nicht relevant)
 - Die Unterschiedlichkeit der Zeilen (Tupel) ist gewährleistet (Unterschiedlichkeit der Informationen gewährleistet, Primärschlüssel sicher vorhanden)
 - Mengenoperationen (Vereinigung, Durchschnitt, Differenz) sind problemlos möglich

- Folgende Tabellen (R1, R2, R3, R4) stellen dieselbe Relation dar

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|---|---|---|--|---|---|---|---|---|---|--|---|---|---|---|---|---|--|---|---|---|---|---|---|
| R1 | R2 | R3 | R4 | | | | | | | | | | | | | | | | | | | | | | | | |
| <table><tr><td>A</td><td>B</td></tr><tr><td>1</td><td>3</td></tr><tr><td>2</td><td>1</td></tr></table> | A | B | 1 | 3 | 2 | 1 | <table><tr><td>A</td><td>B</td></tr><tr><td>2</td><td>1</td></tr><tr><td>1</td><td>3</td></tr></table> | A | B | 2 | 1 | 1 | 3 | <table><tr><td>B</td><td>A</td></tr><tr><td>3</td><td>1</td></tr><tr><td>1</td><td>2</td></tr></table> | B | A | 3 | 1 | 1 | 2 | <table><tr><td>B</td><td>A</td></tr><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>1</td></tr></table> | B | A | 1 | 2 | 3 | 1 |
| A | B | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | B | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| B | A | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| B | A | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | |

$R1(A, B) = \{(1, 3), (2, 1)\}$

$R2(A, B) = \{(2, 1), (1, 3)\}$

$R3(B, A) = \{(3, 1), (1, 2)\}$

$R4(B, A) = \{(1, 2), (3, 1)\}$

- Wichtige (mathematische) Eigenschaft des Relationenmodells ist die Abgeschlossenheit (Closure): Das Ergebnis jeder Operation (mit einer oder mehreren Relationen) ist wieder eine Relation.

- Vergleich:

| Relation | Tabelle | Table | Datei |
|------------------------|------------------------------|------------------------|--------------------------------|
| Tupel | Zeile | Row | Datensatz (Satz) |
| Attribut | Spalte | Column | Datenfeld (Feld) |
| Domäne eines Attributs | Zulässige Werte einer Spalte | Domain of an Attribute | Zulässiger Inhalt eines Feldes |

- Unterschiede: Datei - Tabelle (Relation)

| Datei | Tabelle (Relation) |
|--|---|
| - Zugriffsreihenfolge auf die Sätze relevant | - Reihenfolge der Zeilen nicht relevant |
| - Zwei Sätze mit gleichem Inhalt möglich | - Zwei Zeilen mit gleichem Inhalt nicht möglich |
| - Feldeinteilung und Feldnamen nicht Bestandteil der Datei | - Spaltennamen Bestandteil der Tabelle |
| - In Datei kein (eindeutiger) Primärschlüssel notwendig | - In Tabelle (eindeutiger) Primärschlüssel notwendig |
| - Satzweise Operationen (record processing) | - Tabellenweise Operationen (set processing) |
| - Prozedurale Bearbeitung ('wie'): Öffne Datei; Lies ersten Satz; Solange nicht Dateiende: Überprüfe Satz gemäß Kriterien, Wenn Prüfung positiv: Gib Satz aus; Lies nächsten Satz; Schließe Datei; | - Deskriptive Bearbeitung ('was'): Gib alle Zeilen aus, die den Kriterien entsprechen; |
| - Verknüpfung von Dateien beim Zugriff nicht möglich | - Verknüpfung von Tabellen beim Zugriff möglich |

6.2 Normalformen von Relationen

- Modellierung mittels Datenabhängigkeiten
- Zerlegung von Relationen, um ein redundanzfreies, konsistentes und realitätskonformes Modell der Wirklichkeit zu bekommen (Dekomposition, Decomposition)

6.2.1 Unnormalisierte Relation

- Für eine Firma soll folgender Sachverhalt in einer Relation festgehalten werden:
 - Ein Mitarbeiter hat jederzeit einen Namen, einen Wohnort und ist in einer Abteilung tätig
 - Er arbeitet an mehreren Produkten eine vorbestimmte Zeit (Prozentsatz)
 - Jede Abteilung und jedes Produkt hat einen Namen
 - In einer Abteilung sind mehrere Personen tätig
 - An einem Produkt können mehrere Personen arbeiten

- Tabellendarstellung (4 Zeilen, 6 Spalten)

PERSON

| PERS# | NAME | WOHNORT | ABT# | ABT-NAME | ARBEIT | | |
|-------|---------|---------|------|-----------|--------|----------|------|
| | | | | | PRO# | PRO-NAME | ZEIT |
| 101 | Schmidt | Wien | 1 | Verkauf | 11 | Brot | 60 |
| | | | | | 12 | Käse | 40 |
| 102 | Huber | Linz | 2 | Forschung | 13 | Wurst | 100 |
| 103 | Maier | Wien | 2 | Forschung | 11 | Brot | 20 |
| | | | | | 12 | Käse | 50 |
| | | | | | 13 | Wurst | 30 |
| 104 | Müller | Krems | 1 | Verkauf | 11 | Brot | 80 |
| | | | | | 13 | Wurst | 20 |

- Es treten an Kreuzungspunkten von Zeilen und Spalten mehrere Werte (eine weitere Relation) auf.
- Rein mathematisch betrachtet kann in einer Relation eine Komponente eines Tupels wieder eine Relation sein (relation-valued domain, nested relation, Relationenwertige Domäne).
- Relationenschema

PERSON (PERS#, NAME, WOHNORT, ABT#, ABT-NAME, ARBEIT (PRO#, PRO-NAME, ZEIT))

- Relationeninhalt

PERSON = { (101,'Schmidt','Wien',1,'Verkauf',{(11,'Brot',60),(12,'Käse',40)}),
 (102,'Huber','Linz',2,'Forschung',{(13,'Wurst',100)}),
 (103,'Maier','Wien',2,'Forschung',{(11,'Brot',20),(12,'Käse',50),(13,'Wurst',30)}),
 (104,'Müller','Krems',1,'Verkauf',{(11,'Brot',80),(13,'Wurst',20)}) }

- Im Relationenmodell heißen derartige Relationen (Element eines Tupels ist wieder eine Relation) unnormalisiert, deren Verwendung ist nicht erlaubt.
- Hinweis: Eine Implementierung in einer Datei wäre unter Verwendung von variabel langen Sätzen (Wiederholungsgruppen) sehr wohl möglich, auch in XML sind derartige Strukturen vorgesehen, sowie zusammengesetzte Datentypen in verschiedenen Programmiersprachen.
- Hinweis: Objektorientierte Datenbanksysteme unterstützen Strukturen mit Wiederholungsgruppen (Nested Tables, NF²-Systeme = Non First Normalform-Systeme)

6.2.2 Erste Normalform (1NF)

- Eine Relation befindet sich in Erster Normalform (ist eine Flache Relation), wenn sie keine Attribute aufweist, deren Werte sich aus mehreren Elementen zusammensetzen (wieder Relationen sind).
 Andere Umschreibungen:
 Jedes Attribut stellt einen atomaren Wert dar (nicht zerlegbare Attribute).
 An allen Kreuzungspunkten von Zeilen und Spalten steht maximal ein Wert.
- Vorgangsweise:
 Mehrfache Wertaussprägungen von Attributen in einer Zeile (einem Tupel) werden in einfache Wertaussprägungen mehrerer Zeilen (Tupel) transformiert.
 Einfache Wertaussprägungen müssen in die neu erstellten Zeilen kopiert werden.
 Dabei muss auch der Primärschlüssel in der Regel um Attribute erweitert oder völlig neu definiert werden.
- Relationenschema

PERSON-1NF (PERS#, NAME, WOHNORT, ABT#, ABT-NAME, PRO#, PRO-NAME, ZEIT)

- Relationeninhalt

```
PERSON-1NF = {
(101, 'Schmidt', 'Wien', 1, 'Verkauf', 11, 'Brot', 60),
(101, 'Schmidt', 'Wien', 1, 'Verkauf', 12, 'Käse', 40),
(102, 'Huber', 'Linz', 2, 'Forschung', 13, 'Wurst', 100),
(103, 'Maier', 'Wien', 2, 'Forschung', 11, 'Brot', 20),
(103, 'Maier', 'Wien', 2, 'Forschung', 12, 'Käse', 50),
(103, 'Maier', 'Wien', 2, 'Forschung', 13, 'Wurst', 30),
(104, 'Müller', 'Krems', 1, 'Verkauf', 11, 'Brot', 80),
(104, 'Müller', 'Krems', 1, 'Verkauf', 13, 'Wurst', 20)
}
```

- Tabellendarstellung (8 Zeilen, 8 Spalten)

PERSON-1NF

| PERS# | NAME | WOHNORT | ABT# | ABT-NAME | PRO# | PRO-NAME | ZEIT |
|-------|---------|---------|------|-----------|------|----------|------|
| 101 | Schmidt | Wien | 1 | Verkauf | 11 | Brot | 60 |
| 101 | Schmidt | Wien | 1 | Verkauf | 12 | Käse | 40 |
| 102 | Huber | Linz | 2 | Forschung | 13 | Wurst | 100 |
| 103 | Maier | Wien | 2 | Forschung | 11 | Brot | 20 |
| 103 | Maier | Wien | 2 | Forschung | 12 | Käse | 50 |
| 103 | Maier | Wien | 2 | Forschung | 13 | Wurst | 30 |
| 104 | Müller | Krems | 1 | Verkauf | 11 | Brot | 80 |
| 104 | Müller | Krems | 1 | Verkauf | 13 | Wurst | 20 |

- Nachteile:
 - Hohe Datenredundanz ist gegeben, mit den bekannten Anomalien und Gefahren von Inkonsistenzen.
 - Beispiel: Einfügen des Tupels (105,'Lechner','Wien',1,'Produktion',12,'Schokolade',70)

6.2.3 Funktionale Abhängigkeit (FA)

- Ein Attribut Y ist von einem Attribut X funktional abhängig (functional dependent, FD) wenn gilt
 - Wenn in zwei verschiedenen Zeilen die Werte von X gleich sind, dann müssen in diesen Zeilen auch die Werte von Y gleich sein
 - oder
 - Es darf nie vorkommen, dass in zwei verschiedenen Zeilen die Werte von Y unterschiedlich sind, die Werte von X jedoch gleich
- In der Folge wird abhängig / Abhängigkeit gleichbedeutend mit funktional anhängig / funktionale Abhängigkeit verwendet
- Schwierigkeit: Diese Abhängigkeit muss im Relationenschema gelten, d.h. für alle möglichen denkbaren Relationen / Inhalte
- X wird determinierendes Attribut oder (funktionale) Determinante genannt
- Y wird (funktional) abhängiges Attribut genannt
- Schreibweise:
 - $X \longrightarrow Y$ für Y ist von X abhängig
 - $X \not\longrightarrow Y$ für Y ist von X nicht abhängig
- Beispiele:
 - $PERS\# \longrightarrow NAMEN$
 - $WOHNORT \not\longrightarrow ABT\#$
- X und Y können auch Attributmengen (Attributkombinationen, zusammengesetzte Attribute) sein
Schreibweisen:
 - $\{A,B\} \longrightarrow \{C,D\}$ für $\{C,D\}$ ist von $\{A,B\}$ abhängig
 - $A,B \longrightarrow C,D$ für C und D sind von der Attributkombination aus A und B abhängig
 - $AB \longrightarrow CD$ Notation so nur möglich, wenn auch ohne Beistriche eindeutig

- Beispiele:
 $PERS\#, PRO\# \rightarrow ZEIT$
 $PERS\# \rightarrow NAME, WOHNORT$
 $PRO\#, PRO-NAME \nrightarrow ZEIT$
 $ABT\# \nrightarrow NAME, PRO\#$
 $NAME \nrightarrow WOHNORT$
 $PERS\# \rightarrow TELEFONNR ?$
 $TELEFONNR \rightarrow PERS\# ?$
- Mathematisch bedeutet funktionale Abhängigkeit, dass die Relation (X, Y) immer eine Funktion ist, d.h. jedem Element aus X immer genau ein Element aus Y zugeordnet ist
- Andere Umschreibungen für FA:
 - Wenn von jedem Attributwert von X direkt auf den Attributwert von Y geschlossen werden kann
 - Zu einem Wert von X ist höchstens ein Wert von Y möglich
 - Werte aus X identifizieren die Werte aus Y ; wenn ein Wert aus X gegeben ist, so liegt der zugehörige Wert aus Y fest
 - $SELECT DISTINCT Y FROM R WHERE X=C$
 ist für jeden möglichen Wert / jede mögliche Wertekombination von C skalar (einzeilig)
- Volle funktionale Abhängigkeit (full functional dependency) liegt vor, wenn die Determinante nicht reduziert werden kann, d.h. das abhängige Attribut von keiner Teilmenge der Determinante abhängig ist
 Schreibweise: $X \twoheadrightarrow Y$
- Triviale funktionale Abhängigkeit (trivial functional dependency) liegt vor, wenn bei $X \rightarrow Y$ gilt: $Y \subseteq X$.
 Diese sind praktisch nicht von Bedeutung, normalerweise versteht man unter Abhängigkeit immer eine nicht triviale (nontrivial) Abhängigkeit.
- Beispiel mit konkretem Tabellen- / Relationeninhalt

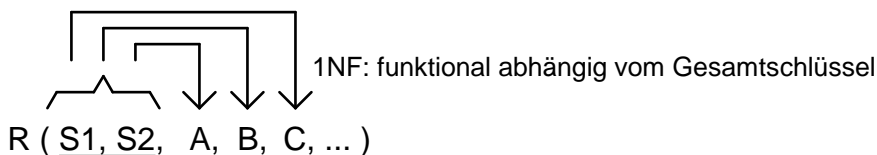
R

| A | B | C |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 1 | 2 |
| 3 | 3 | 1 |
| 4 | 2 | 1 |

| | | | |
|-------------------|--------------------|-----------------------|---------------------------------------|
| $A \rightarrow B$ | $B \nrightarrow A$ | $A, B \rightarrow C$ | $A, B \rightarrow A$ (triviale FA) |
| $A \rightarrow C$ | $C \nrightarrow A$ | $A, C \rightarrow B$ | $A, B \rightarrow A, B$ (triviale FA) |
| $B \rightarrow C$ | $C \nrightarrow B$ | $B, C \nrightarrow A$ | not $(A, B \twoheadrightarrow C)$ |

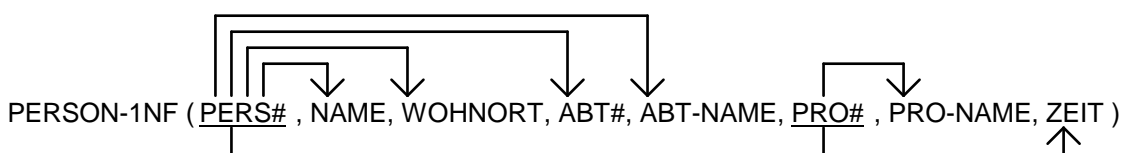
Achtung: Damit FA tatsächlich vorliegt, darf die Bedingung nicht nur in einer konkreten Wertaussprägung gelten, sondern muss zu allen Zeiten in jedem (gültigen) Datenzustand eingehalten sein

- 1NF kann somit auch definiert werden als:
 Jedes, nicht dem Schlüssel angehörige Attribut, ist vom Schlüssel abhängig
- Schematische Darstellung der 1NF:

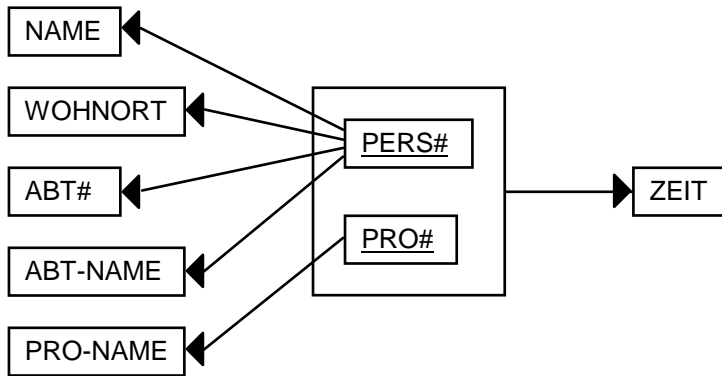


6.2.4 Zweite Normalform (2NF)

- Eine Relation befindet sich in der Zweiten Normalform, wenn jedes nicht dem Schlüssel angehörende Attribut vom Gesamtschlüssel abhängig ist (1NF-Kriterium), nicht aber von einem Teilschlüssel.
- Um eine Relation in 2NF bringen zu können, müssen zunächst alle Abhängigkeiten von den Schlüsselattributen ermittelt werden:



- Für eine übersichtliche Darstellung werden auch Abhängigkeitsdiagramme (Dependency Diagrams) verwendet:



- Vorgangsweise:

Die Relation wird aufgespalten. Jene Attribute, die von Teilschlüsseln abhängig sind, werden zusammen mit diesen Teilschlüsseln in getrennten Relationen zusammengefasst. In den abgespaltenen Relationen werden die determinierenden Attribute der ursprünglichen Relation zum Schlüssel. Die determinierenden Attribute bleiben jedoch auch in der ursprünglichen Relation (als Verbindungsglieder zu den abgespaltenen Relationen) erhalten.

- Relationenschemata

PERSON-2NF (PERS#, NAME, WOHNORT, ABT#, ABT-NAME)

PRODUKT (PRO#, PRO-NAME)

ARBEIT (PERS#, PRO#, ZEIT)

- Relationeninhalte (wegen Mengeneigenschaft keine Tupeln gleichen Inhalts)

PERSON-2NF =

```
{ (101, 'Schmidt', 'Wien', 1, 'Verkauf'),
  (102, 'Huber', 'Linz', 2, 'Forschung'),
  (103, 'Maier', 'Wien', 2, 'Forschung'),
  (104, 'Müller', 'Krems', 1, 'Verkauf') }
```

PRODUKT =

```
{ (11, 'Brot'),
  (12, 'Käse'),
  (13, 'Wurst') }
```

ARBEIT =

```
{ (101, 11, 60),
  (101, 12, 40),
  (102, 13, 100),
  (103, 11, 20),
  (103, 12, 50),
  (103, 13, 30),
  (104, 11, 80),
  (104, 13, 20) }
```

- Tabellendarstellung

PERSON-2NF

| <u>PERS#</u> | NAME | WOHNORT | ABT# | ABT-NAME |
|--------------|---------|---------|------|-----------|
| 101 | Schmidt | Wien | 1 | Verkauf |
| 102 | Huber | Linz | 2 | Forschung |
| 103 | Maier | Wien | 2 | Forschung |
| 104 | Müller | Krems | 1 | Verkauf |

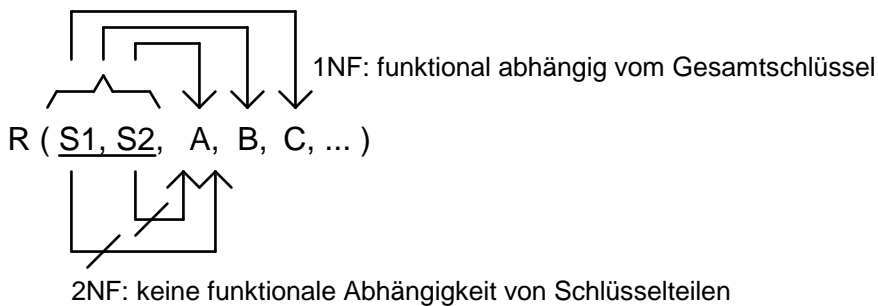
ARBEIT

| PERS# | PRO# | ZEIT |
|-------|------|------|
| 101 | 11 | 60 |
| 101 | 12 | 40 |
| 102 | 13 | 100 |
| 103 | 11 | 20 |
| 103 | 12 | 50 |
| 103 | 13 | 30 |
| 104 | 11 | 80 |
| 104 | 13 | 20 |

PRODUKT

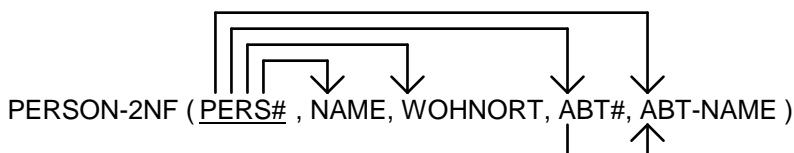
| PRO# | PRO-NAME |
|------|----------|
| 11 | Brot |
| 12 | Käse |
| 13 | Wurst |

- Bei einfachen Schlüsseln kann keine Verletzung der 2NF vorliegen (es gibt keine Schlüsselteile)
- Vorteile: Redundanzen sind teilweise beseitigt
- Nachteile:
 - Weiterhin ist Datenredundanz gegeben, mit den bekannten Anomalien und Gefahren von Inkonsistenzen.
 - Beispiel: Einfügen des Tupels in PERSON-2NF (105,'Lechner','Wien',1,'Produktion')
- Schematische Darstellung der 2NF:

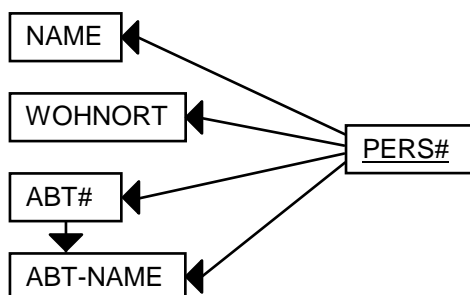


6.2.5 Dritte Normalform (3NF)

- Eine Relation befindet sich in Dritter Normalform, wenn sie in 2NF ist und nicht dem Schlüssel angehörige Attribute voneinander nicht abhängig sind.
- Um eine Relation in 3NF bringen zu können, müssen zunächst alle Abhängigkeiten von Nicht-Schlüsselattributen ermittelt werden:



- Abhängigkeitsdiagramm für Relation PERSON-2NF:



- Vorgangsweise:

Die Relation wird aufgespalten. Funktional abhängige Attribute werden zusammen mit den Determinierenden Attributen in getrennten Relationen zusammengefasst. In den abgespaltenen Relationen werden die determinierenden Attribute der ursprünglichen Relation zum Schlüssel. Die determinierenden Attribute bleiben jedoch auch in der ursprünglichen Relation (als Verbindungsglieder zu den abgespaltenen Relationen) erhalten.

- Relationenschemata

ABTEILUNG (ABT#, ABT-NAME)
 PERSON-3NF (PERS#, NAME, WOHNORT, ABT#)
 PRODUKT (PRO#, PRO-NAME)
 ARBEIT (PERS#, PRO#, ZEIT)

- Relationeninhalte (wegen Mengeneigenschaft keine Tupeln gleichen Inhalts)

ABTEILUNG =
 { (1, 'Verkauf'),
 (2, 'Forschung') }

PERSON-3NF =
 { (101, 'Schmidt', 'Wien' 1),
 (102, 'Huber', 'Linz', 2),
 (103, 'Maier', 'Wien', 2),
 (104, 'Müller', 'Krems', 1) }

PRODUKT =
 { (11, 'Brot'),
 (12, 'Käse'),
 (13, 'Wurst') }

ARBEIT =
 { (101, 11, 60),
 (101, 12, 40),
 (102, 13, 100),
 (103, 11, 20),
 (103, 12, 50),
 (103, 13, 30),
 (104, 11, 80),
 (104, 13, 20) }

- Tabellendarstellung

ABTEILUNG

| ABT# | ABT-NAME |
|------|-----------|
| 1 | Verkauf |
| 2 | Forschung |

PERSON-3NF

| <u>PERS#</u> | NAME | WOHNORT | ABT# |
|--------------|---------|---------|------|
| 101 | Schmidt | Wien | 1 |
| 102 | Huber | Linz | 2 |
| 103 | Maier | Wien | 2 |
| 104 | Müller | Krems | 1 |

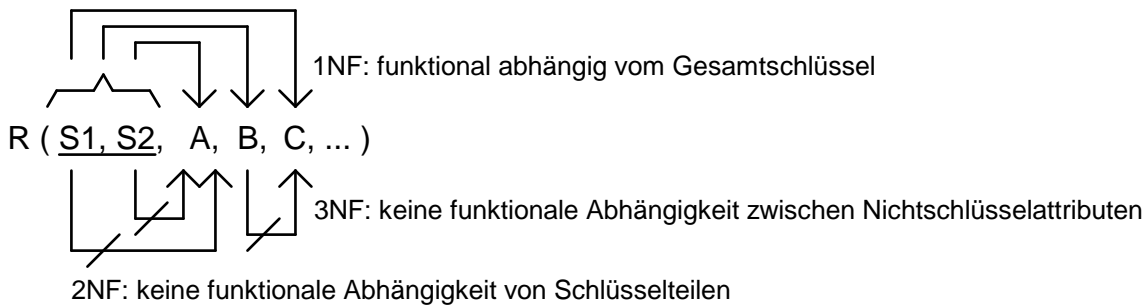
ARBEIT

| <u>PERS#</u> | <u>PRO#</u> | ZEIT |
|--------------|-------------|------|
| 101 | 11 | 60 |
| 101 | 12 | 40 |
| 102 | 13 | 100 |
| 103 | 11 | 20 |
| 103 | 12 | 50 |
| 103 | 13 | 30 |
| 104 | 11 | 80 |
| 104 | 13 | 20 |

PRODUKT

| <u>PRO#</u> | PRO-NAME |
|-------------|----------|
| 11 | Brot |
| 12 | Käse |
| 13 | Wurst |

- Vorteile: Alle Redundanzen sind beseitigt
- Schematische Darstellung der 3NF:



- Andere Definition der 3NF:
 - Ein Attribut C ist von einem Attribut A dann transitiv abhängig, wenn es ein Attribut B gibt, sodass gilt:
 $A \rightarrow C$ und $A \rightarrow B$ und $B \rightarrow C$
 (dabei darf B kein Schlüsselkandidat sein und $B \not\rightarrow A$)
 - Eine Relation befindet sich in der 3NF, wenn sie in 2NF ist und kein Attribut vom Schlüssel transitiv abhängig ist.
- 'Data-Designer's Oath':
 Each attribute is placed in a relation
 where it is depending on the key,
 the whole key and
 nothing but the key –
 so help me Codd!
- Meistens kann auf Basis der 3NF ein redundanzfreies, konsistentes und realitätsgetreues Datenmodell erstellt werden
- Wenn mehrere zusammengesetzte Schlüsselkandidaten in einer Relation sind, die sich mit dem Primärschlüssel überlappen, dann können zusätzliche Normalformen notwendig sein

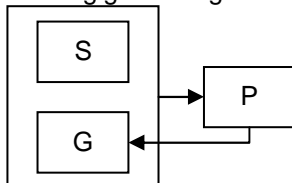
6.2.6 Boyce-Codd Normalform (BCNF)

- Eine Relation befindet sich in BCNF, wenn jede Determinante ein Schlüsselkandidat ist (stärker als 3NF). Die Determinante darf nicht reduzierbar (volle Abhängigkeit), die Abhängigkeit nicht trivial ($B \subseteq A$) sein.
- Beispiel: Schüler, Gegenstände, Professoren
 - In jedem Gegenstand wird ein Schüler von höchstens einem Professor unterrichtet
 - Jeder Professor unterrichtet höchstens einen Gegenstand

Tabelle
SGP

| <u>S</u> | <u>G</u> | P |
|----------|------------|-------------|
| Schmid | Mathematik | Prof. Weiss |
| Schmid | Physik | Prof. Grün |
| Jonas | Mathematik | Prof. Weiss |
| Jonas | Physik | Prof. Braun |

- Abhängigkeitsdiagramm



- In 3NF, da keine Teilschlüsselabhängigkeiten und nur ein Nicht-Schlüsselattribut
- Lösch-Anomalie: Wenn gelöscht wird, dass Jonas Physik studiert, ist auch die Information, dass Prof. Braun Physik unterrichtet nicht mehr vorhanden
- Nicht in BCNF: P ist Determinante, aber kein Schlüsselkandidat

- Weiterer Schlüsselkandidat der Relation SGP: S,P
Es gibt zwei zusammengesetzte Schlüsselkandidaten, die sich in einem Attribut überlappen - daher kann es zur Verletzung der BCNF kommen. Im konkreten Fall ist die Relation tatsächlich nicht in BCNF.
- Lösung des Beispiels durch Aufspaltung in zwei Relationen:

| S | P |
|--------|-------------|
| Schmid | Prof. Weiss |
| Schmid | Prof. Grün |
| Jonas | Prof. Weiss |
| Jonas | Prof. Braun |

| P | G |
|-------------|------------|
| Prof. Weiss | Mathematik |
| Prof. Grün | Physik |
| Prof. Braun | Physik |
- Speicher-Anomalie beseitigt (die Zerlegung ist verbundtreu, da SP natural join PG wieder SGP ergibt), jedoch können die beiden Relationen nicht unabhängig voneinander upgedated werden (die Zerlegung ist nicht abhängigkeittreu, da $S, G \rightarrow P$ verloren ging):
z.B. Einfügen von (Schmid, Prof. Braun) in SP muss nach Kontrolle in PG verhindert werden (Schmid wird bereits von Prof. Grün in Physik unterrichtet)
- Konflikt zwischen:
 - a) Zerlegung in BCNF
 - b) Zerlegung in 'unabhängige' Relationen
- Übung: Schüler, Gegenstand, Rang (kein ex aequo)
Geben Sie alle Schlüsselkandidaten an. Ist die Relation in BCNF?

6.2.7 Weitere Normalformen

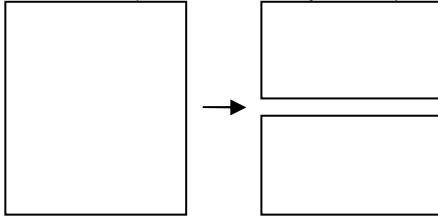
- Vierte Normalform (4NF)
Analyse von Mehrwertigen Abhängigkeiten (Multivalued Dependencies, MVD).
In einer Relation darf nicht mehr als eine MVD auftreten.
- Fünfte Normalform (5NF)
Analyse von Verbundabhängigkeiten (Join Dependencies, JD)
- Domain-Key Normalform (DKNF)
Frei von allen Änderungsanomalien

6.2.8 Denormalisieren

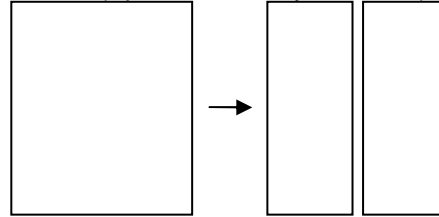
- 'Best denormalization is no denormalization'
- Tendenziell liefert eine hohe Normalform viele Tabellen, das Zusammenführen der Information (für Auswertungszwecke) kann aufwendig sein (Performanceprobleme)
- Denormalisierung: Man geht nicht bis zur höchsten Normalform und erhält weniger Tabellen
- Logischer DB-Entwurf sollte auf alle Fälle in höchster Normalform sein
- Performanceprobleme auf der internen / physischen Ebene zu lösen versuchen (Indexing, Clustering, Materialized Views, Pre-Joined Tables, etc.)
- Denormalisierung aus Performancegründen nur durchführen, wenn alles andere nichts hilft
- Besonders sorgfältige Dokumentation ist unbedingt notwendig
- 'Normalize until it hurts, Denormalize until it works'
- Beispiel:
KUNDEN (KNr, Stammdaten, ...) KONTEN (BLZ, KONTO#, KNr > KUNDEN, ...)
Kunde nicht mehr geschäftsfähig, Sperrmerkmal logisch bei Stammdaten des Kunden. Bei jedem Konto muss geprüft werden, ob es nicht einem gesperrten Kunden gehört (Prüfung in 99% der Fälle negativ).
Sperrmerkmal wird (auch) bei den Konten des Kunden abgespeichert. Welche Normalform liegt vor?
- Beispiel: Abgeleitete / Berechnete / Aggregierte Daten werden redundant mitgeführt,
z.B. der Auftragswert (aus den Positionsdaten berechenbar) wird im Auftragskopf ebenfalls gespeichert.

- Beispiel: Wiederholungsgruppen (Repeating Groups),
z.B. die Umsätze der jeweils letzten 12 Monate / 10 Jahre werden beim Kunden mitabgespeichert.

- Beispiel: Zerlegung von Tabellen,
Horizontal (Zeilenweise, by Rows)



Vertikal (Spaltenweise, by Columns)

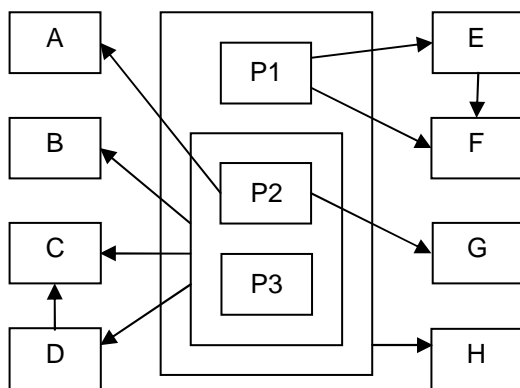


- Durch den Einsatz von Trigger können Inkonsistenzen, die durch Änderungen bewirkt werden, verhindert werden
- Weitere Einsatzgebiete: Data Warehouse (siehe OLAP-Anwendungen), Systemtabellen

6.2.9 Übungen zum Normalisieren

Übung 1):

Gegeben ist eine Relation in 1NF R (P1, P2, P3, A, B, C, D, E, F, G, H).
Folgendes Abhängigkeitsdiagramm wurde ermittelt:



Geben Sie die Relationenschemata für die 2NF und die 3NF an.

Übung 2): Freigegegenstände

In einer Schule soll (für ein Schuljahr) verwaltet werden welche Schüler sich für welche Freigegegenstände angemeldet haben und wie sie darin beurteilt wurden.

Folgende Attribute wurden ermittelt (S steht für Schüler, FG für Freigegegenstand):

SName, SGebDat, Jahrgang, KatalogNr, Jahrgangsvorstand, Stammklassenraum, Trakt, Stockwerk, FGKurzzeichen, FGBezeichnung, FGWochenstunden, Semesternote, Jahresnote

Geben Sie alle (nicht trivialen, vollen) Abhängigkeiten an.

Wie lautet der Primärschlüssel?

Erstellen Sie die 2NF und die 3NF.

Wie würde ein ERD ausschauen, das zum selben Ergebnis führt?

Übung 3): Vorlesungsverwaltung

Folgende Relation in Tabellenschreibweise ist gegeben:

| MATR-NR | NAME | LV-NR | LV-TITEL | PRÜFUNGSERGEBNISSE | | |
|---------|--------|--------|---------------|--------------------|------|------|
| | | | | DATUM | NOTE | SAAL |
| 6625337 | Maier | 116468 | Programmieren | 20000601 | 5 | 2.13 |
| | | | | 20060415 | 5 | 1.47 |
| | | | | 20070219 | 1 | 2.13 |
| 6625337 | Maier | 116138 | Compilerbau | 20021220 | 1 | 1.47 |
| 7025123 | Müller | 116138 | Compilerbau | 20021220 | 4 | 1.47 |
| 6830478 | Hofer | 116468 | Programmieren | 20000601 | 5 | 2.13 |
| | | | | 20031015 | 1 | E.05 |

MATR-NR = Matrikelnummer, LV-NR = Lehrveranstaltungsnummer, LV-TITEL = Lehrveranstaltungstitel, DATUM = Prüfungsdatum, SAAL = Prüfungssaal

- An einem Tag findet in einer Lehrveranstaltung maximal eine Prüfung (ein Saal pro Prüfung) statt, ein Student kann aber an einem Tag durchaus zu mehrere Prüfungen antreten
- Was ist der Primärschlüssel?
- Geben Sie das Relationenschema in der UNF an
- Wieviele Zeilen und Spalten hat die UNF?
- Bringen Sie die Relation in 1NF und unterstreichen Sie die Attribute des Primärschlüssels (Tabellenschreibweise)
- Wieviele Zeilen und Spalten hat die 1NF?
- Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm
- Bringen Sie die Relation in 2NF und unterstreichen Sie die Attribute des Primärschlüssels (Tabellenschreibweise)
- Wie unterscheidet sich in diesem Beispiel die 3NF von der 2NF?

Übung 4): Spital

Für ein Spital soll folgender Sachverhalt in einer Relation festgehalten werden (eine Zeile pro Patient):

- Ein Patient hat eine Nummer (PNR), einen Namen (PNAME), einen Wohnort (PORT) und liegt in einem Zimmer
- Jeder Patient wird von mehreren Ärzten behandelt; jeder Arzt diagnostiziert höchstens eine Krankheit (KRHEIT) bei einem behandelten Patienten
- Jeder Arzt hat eine Nummer (ANR), einen Namen (ANAME) und ein Fachgebiet (AFACH)
- Ein Arzt behandelt mehrere Patienten
- Jedes Zimmer hat eine eindeutige Nummer (ZNR) und eine Bezeichnung (ZBEZ) des Traktes, wo es liegt
- In einem Zimmer liegen mehrere Patienten

| PNr | PName | POrt | ZNr | ZBez | Behandlung | | | |
|------|--------|---------|-----|------------|------------|------------|------------|---------------|
| | | | | | ANr | AName | AFach | Krheit |
| 1234 | Maier | Bregenz | 5 | Westtrakt2 | 7 | Dr.Mabuse | Hautarzt | Röteln |
| | | | | | 10 | Dr.No | Internist | Magengeschwür |
| 1255 | Huber | Graz | 13 | Osttrakt1 | 10 | Dr.No | Internist | Gelbsucht |
| 1313 | Berger | Wien | 5 | Westtrakt2 | 13 | Dr Kurt O. | Kinderarzt | Röteln |
| | | | | | 3 | Dr.Frank | Internist | Bluthochdruck |
| 1350 | Huber | Wien | 13 | Osttrakt1 | 10 | Dr.No | Internist | Herzinfarkt |
| | | | | | 3 | Dr.Frank | Internist | Herzinfarkt |
| | | | | | 7 | Dr.Mabuse | Hautarzt | Gürtelrose |

- Geben Sie das Relationenschema in der UNF an
- Wieviele Zeilen und Spalten hat die UNF?
- Bringen Sie die Relation in 1NF und unterstreichen Sie die Attribute des Primärschlüssels (Attributschreibweise)
- Wieviele Zeilen und Spalten hat die 1NF?
- Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm
- Bringen Sie die Relation in 2NF und unterstreichen Sie die Attribute der Primärschlüssel (Attributschreibweise)
- Wie unterscheidet sich in diesem Beispiel die 3NF von der 2NF?

Übung 5): Baufirma

Für eine Baufirma soll folgender Sachverhalt in einer Relation festgehalten werden (eine Zeile pro Bauprojekt):

- Ein Bauprojekt hat eine Nummer (PNR), einen Namen (PNAME) und findet an einem Ort (PORT) statt
 - Für ein Projekt werden Bauteile von Lieferanten an bestimmten Tagen (LDAT) in bestimmten Mengen (LMENG) geliefert
 - Es kommt nicht vor, dass für ein Bauprojekt das gleiche Bauteil vom selben Lieferanten öfters (an mehreren Tagen) geliefert wird
 - Ein Bauteil hat eine eindeutige Nummer (TNR) und irgendeinen Typ (TTYP)
 - Ein Lieferant hat eine Nummer (LNR), einen Namen (LNAME) und einen Standort (LORT)
 - Das gleiche Bauteil kann in mehreren Bauprojekten benötigt und von verschiedenen Lieferanten geliefert werden
 - Ein Lieferant kann verschiedene Bauteile liefern; ein bestimmter Lieferant verlangt für ein bestimmtes Bauteil einen bestimmten Preis (PREIS)
 - In einem Bauprojekt können mehrere Bauteile benötigt werden
 - Die verschiedenen Lieferanten gewähren für verschiedene Projekte außerdem noch einen Preisnachlass in Prozent (RABATT)
- Skizzieren Sie die unnormalisierte Relation (Tabellenschreibweise mit Beispielwerten)
 - Bringen Sie die Relation in 1NF und unterstreichen Sie die Attribute des Primärschlüssels (Attributschreibweise)
 - Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm
 - Bringen Sie die Relation in 2NF und unterstreichen Sie die Attribute der Primärschlüssel (Attributschreibweise)
 - Wie unterscheidet sich in diesem Beispiel die 3NF von der 2NF?

Übung 6): Gebäudereinigung

Eine Reinigungsfirma hat in einer Stadt mehrere Gebäude zu reinigen. Jedes Gebäude hat einen Code und eine Adresse.

In jedem Gebäude sind mehrere Konferenzräume zu reinigen. Für die Reinigung eines Raumes sind ein oder mehrere Pfleger zuständig. Ein Raumpfleger säubert mehrere Räume.

Jeder Raum ist mit einer bestimmten Anzahl von Sitzgelegenheiten ausgestattet, die pro Raum vom gleichen Typ sind. Es sind mehrere Räume mit dem gleichen Sitztyp ausgestattet. Für jeden Sitztyp wird ein Pflegehinweis gegeben.

Der vorliegende Sachverhalt kann in einer Tabelle zusammengefasst werden:

| Gebäude Code | Adresse | Raum Nr. | Sitz Anzahl | Sitz Typ | Pflegehinweis | Pfleger Nr. | Pflegername |
|--------------|--------------|----------|-------------|----------|------------------|-------------|-------------|
| A | Amtsstraße 1 | 1 | 52 | A | Feucht abwischen | P01 | Maier |
| | | 1 | 52 | A | Feucht abwischen | P03 | Bauer |
| | | 2 | 20 | B | Saugen | P02 | Huber |
| | | 3 | 11 | B | Saugen | P03 | Bauer |
| | | 4 | 30 | C | Echtholzpflge | P02 | Huber |
| B | Stadtplatz 5 | 1 | 38 | B | Saugen | P02 | Huber |
| | | 2 | 25 | C | Echtholzpflge | P02 | Huber |
| | | 3 | 46 | A | Feucht abwischen | P01 | Maier |
| | | 3 | 46 | A | Feucht abwischen | P03 | Bauer |

Bringen Sie die obige Tabelle unter Erstellung entsprechender Abhängigkeitsdiagramme in 1., 2. und 3. Normalform.

Übung 7): Krankenkasse

Eine Krankenkasse verwaltet zur Abrechnung der Behandlungshonorare die notwendigen Daten in einer Tabelle (eine Zeile pro Arzt):

- Ein Arzt hat eine Nummer (ANR), einen Namen (ANAME) und ein Fachgebiet (AFACH)
- Pro Arzt sind alle Behandlungen, die er bei den Patienten durchgeführt hat, eingetragen.
- Eine Behandlung betrifft einen Arzt und einen Patienten; ein Arzt kann einen Patienten auch öfters behandeln.
- Für jeden Patienten ist die Nummer (PNR), der Name (PNAME) und das Geschlecht (PGESCH) vermerkt.
- Der Beginnzeitpunkt (Datum, Uhrzeit) jeder Behandlung (BBEGIN) ist samt Behandlungscode (BCODE), Behandlungstext (BTEXT) und Behandlungskosten (BKOST) angegeben.
- Der Behandlungscode legt sowohl Behandlungstext, als auch Behandlungskosten fest.

Geben Sie alle Schlüsselkandidaten der ersten Normalform an.

Bringen Sie die obige Tabelle unter Erstellung eines entsprechenden Abhängigkeitsdiagramms in zweite und in dritte Normalform.

Übung 8): Classic-CD-World

Die Organisation 'Classic-CD-World' richtet ein Informationssystem über CDs, Labels, Musikstücke, Komponisten, Interpreten und Aufnahmen bezüglich klassischer Musik ein. In einer Relation (ein Tupel entspricht einer CD) soll folgender Sachverhalt gespeichert werden:

- Eine CD ist durch eine Nummer (CDNR) identifiziert, hat einen Titel (TITEL) und einen Preis (PREIS); jede CD wird von einem Label herausgebracht: Labelnummer (LABNR), Bezeichnung (BEZ), www-Adresse (WWW).
- Im Rahmen einer Aufnahme spielt ein Interpret ein Musikstück an einem Tag (DAT) ein. Die Aufnahme findet an einem bestimmten Ort (ORT) statt und hat (wenn sie fertiggestellt ist) eine bestimmte Dauer (DAUER). Ein Interpret nimmt dasselbe Stück höchstens einmal an einem Tag auf.
- Pro CD sollen die enthaltenen Aufnahmen abgespeichert sein. Eine CD enthält nie dieselbe Aufnahme mehrmals.
- Die Daten eines Musikstücks sind: Stücknummer (STNR), Stückbezeichnung (STBEZ), Komponistennummer (KOMNR), Komponistenname (KOMNAME), Geburtsdatum des Komponisten (KOMGEB). Ein Musikstück stammt von einem Komponisten.
- Über die Interpreten wird vermerkt: Interpretennummer (INTNR), Interpretenname (INTNAME), Geburtsdatum des Interpreten (INTGEB).
- Ein Interpret bekommt pro verkaufter CD ein bestimmtes Honorar (HONO)

- a) Was ist der Primärschlüssel in der unnormalisierte Relation?
- b) Bringen Sie die Relation in 1NF (Attributschreibweise) und unterstreichen Sie die Attribute des Primärschlüssels.
- c) Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm.
- d) Bringen Sie die Relation in 2NF (Attributschreibweise) und unterstreichen Sie die Attribute der Primärschlüssel.
- e) Bringen Sie die Relationen in 3NF (Attributschreibweise) und unterstreichen Sie die Attribute der Primärschlüssel.
- f) In welcher Weise könnte man Komponisten und Interpreten sinnvoll zusammenfassen?
- g) Welche Änderungen ergeben sich, wenn die Reihenfolge (NR), in der die Aufnahmen auf einer CD sind, ebenfalls festgehalten werden soll?

7 Relationale Entwurfstheorie

Im Folgenden bezeichnen

| | |
|---------------|--|
| R, S | Relationenschemata |
| A, B, C, ... | Attribute |
| W, X, Y, Z, K | Attributmengen |
| F, G | Mengen von funktionalen Abhängigkeiten |

7.1 Armstrong-Axiome

Interference Rules

- Problemstellung:
Im Relationenschema R ist eine Menge von funktionalen Abhängigkeiten F gegeben.
Welche (zusätzlichen) funktionalen Abhängigkeiten können alle daraus abgeleitet werden?
Kann eine bestimmte (zusätzliche) funktionale Abhängigkeit daraus abgeleitet werden?
- Unter Anwendung der Regeln von Armstrong (1974) kann obige Problemstellung gelöst werden
- Armstrong's Axioms:
 - Reflexivität (Reflexivity)
wenn $Y \subseteq X$, dann $X \rightarrow Y$ (auch triviale Abhängigkeit genannt)
 - Erweiterung, Verstärkung (Augmentation)
wenn $X \rightarrow Y$, dann $XZ \rightarrow YZ$ (für alle Attributmengen Z)
Hinweis: XZ ist die Kurzform für $X \cup Z$, YZ für $Y \cup Z$
 - Transitivität (Transitivity)
wenn $X \rightarrow Y$ und $Y \rightarrow Z$, dann $X \rightarrow Z$
- Die Axiome sind
korrekt (sound), d.h. sie leiten von F nur wirklich gültige Abhängigkeiten ab und
vollständig (complete), d.h. sie erzeugen bei wiederholter Anwendung alle von F implizierten Abhängigkeiten
- Beispiel:
R (A, B, C, D, E, H), $F = \{AB \rightarrow C, BC \rightarrow D, D \rightarrow EH, BE \rightarrow C\}$
gilt auch $AB \rightarrow EH$?

| | | |
|-----|----------------------|---------------------------------------|
| (1) | $AB \rightarrow C$ | gegeben |
| (2) | $ABB \rightarrow CB$ | Armstrong - Erweiterung |
| (3) | $AB \rightarrow CB$ | Mengeneigenschaft |
| (4) | $BC \rightarrow D$ | gegeben |
| (5) | $AB \rightarrow D$ | Armstrong – Transitivität (3) und (4) |
| (6) | $D \rightarrow EH$ | gegeben |
| (7) | $AB \rightarrow EH$ | Armstrong – Transitivität (5) und (6) |
- Aus den ursprünglichen drei Armstrong-Axiomen können weitere abgeleitet werden:
 - Vereinigung (Additivity)
wenn $X \rightarrow Y$ und $X \rightarrow Z$, dann $X \rightarrow YZ$
 - Zerlegung (Projektivität)
wenn $X \rightarrow YZ$, dann $X \rightarrow Y$ und $X \rightarrow Z$
 - Pseudotransitivität (Pseudotransitivity)
wenn $X \rightarrow Y$ und $YW \rightarrow Z$, dann $XW \rightarrow Z$

7.2 Hülle einer Menge von Abhängigkeiten

Closure of a Set of Dependencies

- Die (transitive) Hülle einer Menge von Abhängigkeiten F (notiert als F^+) ist die Menge aller funktionalen Abhängigkeiten, die aus F (durch Anwendung der Armstrong Axiome) ableitbar sind
- Fragestellung: Wann sind zwei Mengen (G, F) von Abhängigkeiten gleichwertig (äquivalent)?
Antwort: Genau dann, wenn $G^+ = F^+$
- Die komplette Hülle einer Menge von von Abhängigkeiten umfasst sehr viele Elemente, sie wird daher praktisch nie explizit ermittelt

7.3 Hülle einer Menge von Attributen

Closure of a Set of Attributes

- Welche Attribute werden (direkt oder indirekt) durch die Attributmenge X bestimmt?
- Unter der Hülle einer Menge von Attributen X (notiert als X^+) bezüglich F versteht man die Menge aller Attribute A , sodass $X \rightarrow A$ aus F^+ ist, also $X^+ = \{ A \mid (X \rightarrow A) \in F^+ \}$
- Algorithmus CLOSURE

```

result := X
repeat
    temp := result
    for each  $Y \rightarrow Z$  in  $F$  do
        if  $Y \subseteq \text{result}$  then
            result := result  $\cup$   $Z$ 
until temp = result

```
- Fragestellung: Gegeben F , kann $X \rightarrow Y$ daraus abgeleitet werden, d.h. $(X \rightarrow Y) \in F^+$?
 Antwort: Genau dann, wenn $Y \subseteq X^+$
- Fragestellung: Ist die Attributmenge K Superschlüssel (Superkey) der Relation R ?
 Antwort: Genau dann, wenn $K^+ = \text{Menge aller Attribute von } R$
- Fragestellung: Ist die Attributmenge K Schlüsselkandidat (Candidate Key) der Relation R ?
 Antwort: Genau dann, wenn $K^+ = \text{Menge aller Attribute von } R$ und K minimal (irreduzibel) ist
- Primes Attribut (Prime Attribut) kommt in einem Schlüsselkandidaten vor
- Nicht primes Attribut (Non-prime Attribut) kommt nicht in einem Schlüsselkandidaten vor
- Hinweis zur Ermittlung eines Schlüsselkandidaten: Er enthält sicher alle Attribute, die nicht auf der rechten Seite einer Funktionellen Abhängigkeit vorkommen, aber gegebenenfalls auch noch weitere.

7.4 Zerlegung von Relationenschemata

Decomposition of Relation Schemas

7.4.1 Verbundtreue Zerlegung

Verlustfreie Zerlegung, Lossless Decomposition

- Eine Zerlegung ist verbundtreu, wenn der Natural Join zwischen den zerlegten Relationen wieder die ursprüngliche Relation liefert
- Beispiel 1): verbundtreue Zerlegung

| R | | R1 | R2 |
|-------|------------|-----|-----|
| A B C | zerlegt in | A B | B C |
| 1 2 3 | | 1 2 | 2 3 |
| 4 2 3 | | 4 2 | |

R1 Natural Join R2 liefert R

- Beispiel 2): nicht verbundtreue Zerlegung, 'lossy' Decomposition

| R | | R1 | R2 |
|-------|------------|-----|-----|
| A B C | zerlegt in | A B | B C |
| 1 2 3 | | 1 2 | 2 3 |
| 4 2 5 | | 4 2 | 2 5 |

R1 Natural Join R2 liefert nicht R (sondern mehr Zeilen)

- Eine Zerlegung ist verbundtreu, wenn der Durchschnitt der Attributmengen der zerlegten Relationen Schlüssel in einer dieser Relationen ist (hinreichende Bedingung!)
 In Beispiel 1) ist B (Durchschnitt der Attributmengen) Schlüssel in R2, in Beispiel 2) ist dies nicht der Fall.

7.4.2 Abhängigkeitstreue Zerlegung

Dependency preserving Decomposition

- Eine Zerlegung ist abhängigkeitstreu, wenn jede Funktionale Abhängigkeit der ursprünglichen Relation nach der Zerlegung in mindestens einer Relation wieder auftritt
- Wenn Funktionale Abhängigkeiten verloren gehen, dann werden diese Constraints nicht mehr durch die Relationenstruktur gewährleistet (der Constraint 'geht über zwei Relationen'), sondern müssen gesondert überwacht werden (z.B. mit Trigger)

7.4.3 Zusammenfassung

| | Verbundtreue | Abhängigkeitstreue |
|---------|--------------|--------------------|
| bis 3NF | immer | immer |
| BCNF | immer | nicht immer |

- Zu jedem Relationenschema R existiert eine Zerlegung in 3NF-Relationenschemata, die verbundtreu und abhängigkeitstreu ist
- Zu jedem Relationenschema R existiert eine Zerlegung in BCNF-Relationenschemata, die verbundtreu ist

7.5 Übungen zur Relationalen Entwurfstheorie

- 1) $R(A, B, C, D), F = \{A \rightarrow B, C \rightarrow D\}$

Gilt auch $AC \rightarrow BD$?

- Beweis mit Armstrong-Axiomen
- Beweis mit geeigneter Hüllenbildung

- 2) $R(A, B, C, D, E, H), F = \{AB \rightarrow C, BC \rightarrow D, D \rightarrow EH, BE \rightarrow C\}$

- Gilt auch $AB \rightarrow EH$? (Ermittlung von $\{A, B\}^+$)
- Ist AB Superschlüssel in R?
- Ist AB Schlüsselkandidat in R?
- Welche der folgende Mengen ist Superschlüssel, welche ist Schlüsselkandidat?
 $\{A, C, D\}$ $\{A, D\}$ $\{B, C\}$ $\{A, C\}$ $\{A, B, E\}$

- 3) Ein Relationenschema R (A, B, C, D) mit atomaren Attributwerten ist gegeben.

Außerdem ist jeweils eine Menge von FAs angegeben. Ermitteln Sie jeweils:

- Schlüsselkandidat(en)
- höchste NF, in der sich die Relation sicher befindet
- 3NF

a) $\{\}$

b) $\{A \rightarrow B\}$

c) $\{AB \rightarrow C, C \rightarrow D\}$

d) $\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

e) $\{A \rightarrow B, B \rightarrow A, D \rightarrow C\}$

- 4) Folgende Abkürzungen werden benutzt:

- W für Wertpapiere
- Z für deren Zinsen
- H für Händler mit Wertpapieren
- B für deren Büros
- A für Anleger
- M für die Menge der von den Anlegern gekauften Wertpapiere

Folgende Abhängigkeiten wurden ermittelt:

$W \rightarrow Z, A \rightarrow H, AW \rightarrow M, H \rightarrow B$

- Welche(n) Schlüssel(kandidaten) hat die Relation R (W, Z, H, B, A, M)?
- Welche Redundanzen und Anomalien treten auf, wenn R in R1 (H, B, A) und R2 (W, Z, A, M) zerlegt wird?
- Ermitteln Sie eine Zerlegung, die in 3NF ist

- 5) $R(A, B, C, D, E, H), F = \{A \rightarrow BC, E \rightarrow CH, B \rightarrow E, CD \rightarrow EH\}$
- Ist $(BC \rightarrow AE) \in F^+$? (Ermittlung von $\{B, C\}^+$)
 - Ist AB Superkey in R?
 - Ist AE Superkey in R?
 - Ist ACD Superkey in R?
 - Ist ACD Candidate Key in R?
 - Ist AD Candidate Key in R?
- 6) $R(A, B, C, D, E), F = \{B \rightarrow ACD, E \rightarrow C, AD \rightarrow B, D \rightarrow E\}$
- Ist $(D \rightarrow C) \in F^+$? (Ermittlung von $\{D\}^+$)
 - Ist AE Superkey in R?
 - Ist AD Superkey in R?
 - Ist AD Candidate Key in R?
 - Ist ADE Candidate Key in R?
 - Geben Sie alle Candidate Keys an
 - In welcher Normalform ist R (eventuell abhängig vom gewählten Primary Key)?
- 7) $R(A, B, C, D, E, H), F = \{AB \rightarrow EH, D \rightarrow C, E \rightarrow C, EH \rightarrow B\}$
- Geben Sie alle Candidate Keys an
 - In welcher Normalform ist R (eventuell abhängig vom gewählten Primary Key)?
- 8) Die Informationen über Daten eines Telefonbuches wurden in folgender Form abgelegt:

TELEFON

| SVNR | NAME | RAUMNR | RAUMBEZEICHNUNG | KLAPPE |
|------------|------------|--------|-----------------|--------|
| 1485120572 | R.Gross | 12 | Büro | 520 |
| 2711220355 | C.Bergmann | 12 | Büro | 521 |
| 2926240871 | M.Greis | 15 | Labor | 544 |
| 2926240871 | M.Greis | 16 | Büro | 545 |
| 6221051263 | L.Klein | 16 | Büro | 546 |
| 2731080267 | S.Richter | 22 | Sekretariat | 100 |

- Jede Klappe befindet sich in einem Raum und ist höchstens einem Mitarbeiter zugeordnet
 - Jeder Mitarbeiter kann Arbeitsplätze in mehreren Räumen haben
- a) Geben Sie alle funktionalen Abhängigkeiten an (minimale Überdeckung), die in dieser Tabelle zu finden sind
- b) Geben Sie alle möglichen Schlüsselkandidaten für diese Relation an
- c) Zeigen Sie anhand dieser Relation mögliche Anomalien, die bei fehlender Normalisierung auftreten können
- d) Befindet sich diese Relation in 2.Normalform - begründen Sie Ihre Antwort?
- e) Wie sieht die 3.Normalform aus?
- 9) $R(A, B, C, D, E), F = \{A \rightarrow B, AD \rightarrow C, DB \rightarrow E, B \rightarrow C\}$
- Welche der angegebenen funktionalen Abhängigkeiten sind redundant?
 - Geben Sie alle Candidate Keys an
- 10) $R(A, B, C, D, E, H), F = \{ABC \rightarrow D, B \rightarrow E, C \rightarrow H, EH \rightarrow D\}$, welches Attribut in $ABC \rightarrow D$ ist überflüssig?

8 SQL

- Entwicklung:
 - Vorläufer: SEQUEL (Structured English Query Language), 1973
 - SQL (Structured Query Language), 1979, Fa. ORACLE (teilweise immer noch als 'siquel' ausgesprochen)
 - SQL-86, SQL-87 (ANSI X3.135-1986, ISO 9075:1987), ca. 100 Seiten
Enthält ein Integrity Enhancement Feature (IEF) als unverbindlichen Vorschlag im Anhang.
 - SQL-89 (ANSI X3.135-1989, ISO 9075:1989), ca. 120 Seiten
IEF wird als verbindlich erklärt.
 - SQL-92, SQL2 (ANSI X3.135-1992, ISO 9075:1992), über 600 Seiten
Drei Levels: Full SQL-92, Intermediate SQL-92, Entry SQL-92
 - SQL:1999, SQL3 (ISO 9075:1999), 5 Parts, ca. 2200 Seiten
Auch objektorientierte Erweiterungen
 - SQL:2003 (ISO/IEC 9075:2003)
SQL/XML und SQL/MED (Management of External Data)
 - SQL:2006 (ISO/IEC 9075-14:2006)
Im Besonderen XML mit SQL
 - SQL:2008 (ISO/IEC 9075:2008), 9 Parts
- Die Datenbanksprache (Standardschnittstelle) für praktisch alle relationale Datenbanksysteme:
 - Datendefinitionen
 - Datenzugriffsberechtigungen
 - Datenauswahl, Datenabfrage
 - Datenmanipulationen
- Deskriptiv, nicht prozedural: Benutzer spezifiziert nur, was er haben will und nicht, wie es zu ermitteln ist (teilweise hat der Benutzer sehr wohl Einfluss auf die Art der Ermittlung).
- Verwendung von SQL:
 - interaktiv (iSQL)
 - in Hochsprachen (3GL)
 - Unterprogrammaufrufe
 - Embedded SQL (ESQL) mit Precompiler
 - Objektmodell für den Datenzugriff
 - in 4GL-Sprachen als Datenbankschnittstelle
- Syntax-Prinzipien:
 - nicht case-sensitive
 - nur runde Klammern: ()
 - nur einfache Anführungszeichen bei String-Literalen: 'Beispiel'
- Syntax-Notation:
 - BNF-ähnlich
 - Konvention: $\text{xyz-commalist} ::= \text{xyz} \mid \text{xyz} , \text{xyz-commalist}$
 $\text{xyz-list} ::= \text{xyz} \mid \text{xyz} \text{ xyz-list}$

8.1 Datenbank LT: Lieferanten - Teile - Lieferungen

L

| LNR | LNAME | RABATT | STADT |
|-----|--------|--------|--------|
| L1 | Schmid | 20 | London |
| L2 | Jonas | 10 | Paris |
| L3 | Berger | 30 | Paris |
| L4 | Klein | 20 | London |
| L5 | Adam | 30 | Athen |

T

| TNR | TNAME | FARBE | PREIS | STADT |
|-----|----------|-------|-------|--------|
| T1 | Mutter | rot | 12 | London |
| T2 | Bolzen | gelb | 17 | Paris |
| T3 | Schraube | blau | 17 | Rom |
| T4 | Schraube | rot | 14 | London |
| T5 | Welle | blau | 12 | Paris |
| T6 | Zahnrad | rot | 19 | London |

LT

| LNR | TNR | MENGE |
|-----|-----|-------|
| L1 | T1 | 300 |
| L1 | T2 | 200 |
| L1 | T3 | 400 |
| L1 | T4 | 200 |
| L1 | T5 | 100 |
| L1 | T6 | 100 |
| L2 | T1 | 300 |
| L2 | T2 | 400 |
| L3 | T2 | 200 |
| L4 | T2 | 200 |
| L4 | T4 | 300 |
| L4 | T5 | 400 |

MENGE > 0

8.2 Datendefinitionen

- Data Definition Language / Comands, DDL, Schema Definition Language / Commands, Schemaanweisungen
- Datenbank erstellen / löschen:
Erfolgt im Standard über sogenannte Schema-Definitionen, die meisten SQL-Produkte arbeiten mit eigenen Konstruktionen (CREATE / DROP DATABASE, etc.)
- Tabelle erstellen:

```
CREATE TABLE base-table ( base-table-element-commalist )
```

```
base-table-element ::= column-def | table-constraint-def
```

```
column-def ::= column data-type
              [ DEFAULT { literal | niladic-function-ref | NULL } ]
              [ column-constraint-def-list ]
```

```
column-constraint-def ::= NOT NULL |
                          { UNIQUE | PRIMARY KEY } |
                          CHECK ( search-condition ) |
                          REFERENCES base-table [ ( column ) ]
```

```
table-constraint-def ::= { UNIQUE | PRIMARY KEY } ( column-commalist ) |
                          CHECK ( search-condition ) |
                          FOREIGN KEY ( column-commalist )
                          REFERENCES base-table [ ( column-commalist ) ]
```

- search-condition aus CHECK muss so formuliert sein, dass nur eine Zeile (in der die Änderung gemacht wird oder die eingefügt wird) isoliert untersucht werden braucht, z.B.


```
CHECK ( RABATT BETWEEN 10 AND 50 )
```

 als column-constraint-def

```
CHECK ( STADT <> 'London' OR RABATT = 20 )
```

 als table-constraint-def
 - Check-Bedingung ist erfüllt, wenn die search-condition TRUE oder NULL (nicht FALSE) liefert
 - Constraints, die mehrere Zeilen betreffen (aus einer oder mehreren Tabellen), können nicht deklarativ definiert werden (siehe Triggers)
- PRIMARY KEY: Änderung eines Primärschlüsselwertes wird nicht verhindert (außer wenn dadurch eine Verletzung der Referentiellen Integrität erfolgen würde)
- Zusammenfassung Constraints:
 - Unterscheide zwischen Column- und Table-Constraints
 - NOT NULL
Verwendung bei Fremdschlüsseln beachten
 - PRIMARY KEY
 - FOREIGN KEY
Referential Actions
ON { DELETE | UPDATE } { NO ACTION | SET NULL | SET DEFAULT | CASCADE }
 - UNIQUE
Verwendung bei Fremdschlüsseln beachten
 - CHECK
 - DEFAULT-Clause (kein Constraint im engeren Sinn)
- Datentypen (data-type):

| | |
|--------------------------------|--|
| INTEGER | Ganze Zahl |
| DECIMAL[(precision[,scale])] | Festkommazahl |
| NUMERIC[(precision[,scale])] | Festkommazahl |
| | precision=Gesamtanzahl der Stellen (ohne Dezimalpunkt) |
| | scale=Anzahl der Nachkommastellen |
| FLOAT[(precision)] | Gleitkommazahl |
| REAL | Gleitkommazahl |
| DOUBLE [PRECISION] | Gleitkommazahl |
| CHARACTER[(length)] | Zeichenkette fester Länge |
| VARCHAR[(length)] | Zeichenkette variabler Länge |

| | |
|-------------|---|
| DATE | Datum, Format kann angegeben werden (z.B. 24.12.1987 = tt.mm.jjjj), Funktion <code>current_date</code> |
| TIME | Zeit, Genauigkeit und Format können angegeben werden (z.B. 13:46 = hh:mm), auch mit Zeitzone, Funktion <code>current_time</code> |
| TIMESTAMP | Zeitpunkt, Genauigkeit und Format können angegeben werden (z.B. 24.12.1987 13:46:25 = tt.mm.jjj hh:mm:ss), auch mit Zeitzone, Funktion <code>current_timestamp</code> |
| INTERVAL | Zeitintervall, Genauigkeit und Format können angegeben werden (z.B. 17:23:45 = tt:hh:mm) |
| LOGICAL | true / false (optional) |
| BLOB[(n)] | binary large object, Binärdaten (maximal n Bytes Länge) |
| CLOB[(n)] | character large object, Zeichenkette (maximal n Bytes Länge) |

- Teilweise große Unterschiede und Erweiterungen bei den verschiedenen Datenbankprodukten

- Tabelle modifizieren:

```
ALTER TABLE base-table
    [ ADD COLUMN ... | DROP COLUMN ... | { ALTER | MODIFY } COLUMN ... ]
```

- Tabelle löschen:

```
DROP TABLE base-table
```

8.3 Indizes

- Index erstellen:

```
CREATE [ UNIQUE ] INDEX index ON table ( index-specification-commalist )
```

```
index-specification ::= column [ ASC | DESC ]
```

- Achtung: keinen Index als Beginnteil eines anderen Index definieren (unnötig)

- Index löschen:

```
DROP INDEX index
```

8.4 Datenzugriffsberechtigungen

- Data Control Language / Comands, DCL

- Der Benutzer, der eine Tabelle erstellt (CREATE TABLE), ist der Eigentümer dieser Tabelle und hat alle Rechte (SELECT, INSERT, UPDATE, DELETE) mit dieser Tabelle

- Der Eigentümer einer Tabelle kann Berechtigungen an andere Benutzer weitergeben (GRANT)

- Zugriffsberechtigungen erteilen:

```
GRANT { ALL [ PRIVILEGES ] | operation-commalist }
    ON table TO grantee-commalist [ WITH GRANT OPTION ]
```

```
operation ::= SELECT | INSERT | DELETE | { UPDATE [ ( column-commalist ) ] }
```

```
grantee ::= PUBLIC | user
```

- table: base-table oder viewed-table
- WITH GRANT OPTION: die erteilten Rechte dürfen an andere Benutzer weitergegeben werden (grundsätzlich kann ein Benutzer nur solche Berechtigungen weitergeben, die er selber hat)

- Zugriffsberechtigungen entziehen:

```
REVOKE { ALL [ PRIVILEGES ] | operation-commalist }
    ON table FROM grantee-commalist
```

- Dem Eigentümer einer Tabelle können keine Berechtigungen entzogen werden

8.5 Datenauswahl - SELECT

- Data Query Language / Commands, DQL
- Abgeschlossenheit (Closure):
Abfrageergebnis aus Tabellen (Eingabe) ist wieder eine Tabelle (Ausgabe).
Mindestens eine Spalte, gegebenenfalls keine Zeile (wenn Abfragebedingung für alle Zeilen falsch ist)
- SELECT-Anweisung (select-statement) – prinzipieller Aufbau:

```
SELECT [ ALL | DISTINCT ]
      { * | expression-commalist } = Projektion
FROM table-reference-commalist
[ WHERE search-condition ]        = Restriktion
[ GROUP BY column-reference-commalist [ HAVING search-condition ] ]
[ ORDER BY order-specification-commalist ]
```

table-reference ::= table [alias]

column-reference ::= [{ table | alias } .] column

order-specification ::= { column-reference | integer } [ASC | DESC]

8.5.1 Projektion

- Auswahl von Spalten aus einer Tabelle

| S1 | S2 | S2 | S4 | S5 |
|----|----|----|----|----|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

- Alle Spalten, die nach dem SELECT in der expression-commalist angegeben sind
- Alle Spalten: *
- expression:
 - Operatoren: + | - | * | / | (|) | ** | mod | ||
 - Operanden: column-reference | literal | aggregate-function | parameter
 - Funktionen (z.B. Arithmetische Funktionen, Zeichenkettenfunktionen) stehen zur Verfügung
 - Zuweisung eines Spaltennamens / -alias für die Ergebnistabelle: expression [AS] column
 - Wenn ein Operand der expression den Wert NULL hat, so ist auch das Ergebnis der expression NULL
 - In neueren Implementierungen kann als Operand auch (select-statement) angegeben werden, dieses muss eine skalare Tabelle (eine Zeile, eine Spalte) liefern (siehe Sub-Selects in der Projektion)
- Alle Tabellen nacheinander anzeigen


```
select * from l;
select * from t;
select * from lt;
- falsch: select * from l,t,lt;
```
- Welche Namen haben die Teile?


```
select tname from t;
```
- Teile (TNR, TNAME) mit verdoppeltem Preis anzeigen


```
select tnr,tname,preis*2 from t;
```
- Teile (TNR, TNAME) mit Quadratwurzel des Preises (auf zwei Nachkommastellen gerundet) anzeigen


```
select tnr,tname,round(sqrt(preis),2) from t;
select tnr,tname,cast(round(sqrt(preis),2) as decimal(4,2)) from t;
```

- Teile und deren Farben anzeigen, in der Form: MUTTER in rot

```
select upper(tname)||' in '||farbe from t;
select rtrim(upper(tname))||' in '||farbe from t;
```
- Teile (TNR, TNAME) mit verdoppeltem Preis anzeigen (Spaltenname des doppelten Preises: preis_mal_2)

```
select tnr,tname,preis*2 as preis_mal_2 from t;
select tnr,tname,preis*2      preis_mal_2 from t;
```
- Was liefern die folgenden Anweisungen?

```
select 27 'immer 27' from t;
select 27 immer 27      from t;
select 27 immer, 27     from t;
```
- Funktionen
 - Mathematische Funktionen, Zahlen-Funktionen
 - Zeichenketten-Funktionen
 - Datum- und Zeit-Funktionen
 - Typ-Konvertierungs-Funktionen
 - System-Funktionen
 - Sonstige Funktionen

8.5.2 DISTINCT

- Mehrfache Zeilen mit gleichem Inhalt (Duplikatzeilen) werden eliminiert
- Welche verschiedenen Namen haben die Teile?

```
select distinct tname from t;
```

8.5.3 Restriktion (Selektion)

- Auswahl von Zeilen aus einer Tabelle

| S1 | S2 | S2 | S4 | S5 |
|----|----|----|----|----|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

- Alle Zeilen, bei denen in der WHERE-Klausel die search-condition wahr ist
- search-condition:
 - Operatoren: OR | AND | NOT | (|)
 - Operanden (simple predicate):
 - expression comparison expression
 - expression [NOT] BETWEEN expression AND expression)
 BETWEEN: x BETWEEN y AND z ist gleichbedeutend mit y<=x AND x<=z) 'Syntaktischer Zucker'
 - expression [NOT] IN (literal-commalist)) oder
 IN: x IN (a,b,...,z) ist gleichbedeutend mit x=a OR x=b OR ... OR x=z) 'Syntactic Sugar'
 - column-reference [NOT] LIKE character-literal [ESCAPE escape-character]
 character-literal: % beliebige Zeichenkette, _ ein beliebiges Zeichen
 (in Dialekten auch andere Zeichen, z.B. in ACCESS die üblichen * und ?)
 Immer ein Wildcard-Zeichen im character-literal, sonst = (Aufwand!)
 - column-reference IS [NOT] NULL
 - comparison ::= = | <> | < | > | <= | >=
 - Wenn ein Operand (expression) des predicates NULL ist, hat (außer bei NOT, AND, OR) das predicate auch den Wert NULL
- Welche Lieferanten sind aus Paris?

```
select * from l where stadt='Paris';
```
- Welche Lieferanten (LNR, LNAME, RABATT) sind aus Paris?

```
select lnr,lname,rabatt from l where stadt='Paris';
```

- Welche roten Teile haben einen Preis zwischen 13 und 23?

```
select tnr from t where farbe='rot' and preis between 13 and 23;
```

 - Kurzform für

```
select tnr from t where farbe='rot' and preis >= 13 and preis <= 23;
```
- Welche Teile zu einem Preis von 12 oder 14 sind nicht aus London oder Rom?

```
select tnr from t where preis in (12, 14) and stadt not in ('London', 'Rom');
```

 - Kurzform für

```
select tnr from t
where (preis=12 or preis=14) and not (stadt='London' or stadt='Rom');
select tnr from t
where (preis=12 or preis=14) and stadt<>'London' and stadt<>'Rom';
```
- Alle Lieferanten (LNR), die das Teil T1 oder das Teil T2 liefern

```
select distinct lnr from lt where tnr='T1' or tnr='T2';
select distinct lnr from lt where tnr in ('T1','T2');
```
- Welche Teile haben ein 'e' im Namen und sind aus einer Stadt mit 3 Zeichen?

```
select * from t where tname like '%e%' and stadt like '___';
```
- Entwertung von Wildcard-Zeichen

```
insert into t values ('T8','Zang%','blau',20,'Wien');
insert into t values ('T9','Weller'_'l','rot',10,'Wi_en');
select * from t where tname like '%_';
select * from t where stadt like '%_';
select * from t where stadt like '%\_%' escape '\';
select * from t where tname like '%%';
select * from t where tname like '%!%' escape '!';
delete from t where tnr='T8';
delete from t where tnr='T9';
```
- Welche Teile haben keinen Preis?

```
select * from t where preis is null;
```

 - falsch: ... preis=null;

8.5.4 ORDER BY

- Die Ergebnistabelle wird sortiert zur Verfügung gestellt
- Column-reference von ORDER BY muss sich auf die Projektion beziehen und wird daher auch ein Element der expression-commalist nach SELECT sein
- Bei Verwendung von Spaltenalias kann auch ein solcher angegeben werden
- Ein integer n bezieht sich auf das n-te (mit 1 beginnend) Element der Projektion, damit kann nach dem Ergebnis beliebiger Ausdrücke sortiert werden
- NULL-Werte sind implementierungsabhängig entweder alle größer oder alle kleiner als Nicht-NULL-Werte
- Achtung: Ohne Verwendung von ORDER BY wird das Ergebnis (Tabelle ist eigentlich eine Menge) in 'irgendeiner' Reihenfolge geliefert, nicht unbedingt nach dem Primärschlüssel oder der Reihenfolge des Einfügens. Unterschied zu einer Sequentiellen Datei, wo die Reihenfolge des Schreibens auch die Reihenfolge des Lesens ist.
- Welche Namen haben die Teile (sortiert nach Namen)?

```
select tname from t order by tname;
```
- Welche verschiedenen Namen haben die Teile (sortiert nach Namen)?

```
select distinct tname from t order by tname;
```
- Alle Teile mit dem Quadrat des um 16 erniedrigten Preises anzeigen, sortiert in erster Linie nach diesem Ausdruck absteigend, in zweiter Linie nach dem Teilnamen

```
select tnr, (preis - 16)*(preis-16),tname from t order by 2 desc, tname;
select tnr, (preis - 16)*(preis-16) ausdruck,tname from t
order by ausdruck desc, tname;
```

8.5.5 Mengenoperation Vereinigung - UNION

- Die Zeilen zweier SELECT-Ergebnisse kommen ('untereinander') in eine Ergebnistabelle:
`SELECT ... UNION [ALL] SELECT ...`
- Zu vereinigende Ergebnistabellen müssen vereinigungsverträglich sein, d.h.
syntaktisch: gleiche Spaltenanzahl, gleiche Datendefinitionen (Typ, Länge) der Spalten
semantisch: gleiche Bedeutung der Spalten
- ALL: Duplikatzeilen werden nicht eliminiert
- ORDER BY darf nur nach dem letzten SELECT angegeben werden
- Welche Lieferanten (LNR) sind aus London oder liefern das Teil T2?
`select lnr from l where stadt='London'`
`union`
`select lnr from lt where tnr='T2';`

`select lnr from l where stadt='London'`
`union all`
`select lnr from lt where tnr='T2'`
`order by lnr;`
- Was liefern die folgenden Anweisungen?
`select lnr from l`
`union`
`select * from t;`

`select lnr,lname from l`
`union`
`select tnr,preis from t;`

`select lnr,lname from l`
`union`
`select tnr,tname from t`
`order by 2;`
- Nicht Vereinigung statt Restriktion verwenden:
`select * from t where farbe='blau'`
`union`
`select * from t where stadt='Paris';`

`select * from t where farbe='blau' or stadt='Paris';`

8.5.6 Cross-Join

- Kreuz-Verbund, (Karthesisches) Produkt
- Die Zeilen zweier Tabellen kommen ('nebeneinander') in eine Ergebnistabelle:
Jede Zeile der einen Tabelle wird mit jeder Zeile der anderen Tabelle zusammengehängt (konkateniert), alle kommen in die Ergebnistabelle
- Übung: Wieviele Spalten und Zeilen (auf die Spalten- und Zeilenanzahl der Operanden-Tabellen bezogen) hat die Ergebnistabelle eines Cross-Joins?
- Beispiel
`select * from lt, l;`
`select * from lt cross join l;`

8.5.7 Equi-Join

- Gleichheits-Verbund
- Analog Cross-Join, jedoch kommen nur jene Zeilen in die Ergebnistabelle, bei denen eine Gleichheitsbedingung (Spalteninhalte der zwei Operanden-Tabellen) wahr ist.
Bedingung wird formuliert in
 - der Restriktion (SQL-89) – eigentlich CROSS-Join + Restriktion
 - der FROM-Klausel (SQL-92) – eigene JOIN-Operation

- Häufigster Anwendungsfall: Zeilen, die einen Fremdschlüssel enthalten, werden jeweils mit der Zeile, die den entsprechenden Primärschlüsselwert hat, verbunden
- Spalten-Qualifikation mit Tabellennamen oder Tabellenalias meist notwendig (da Spaltennamen von Primär- und Fremdschlüssel üblicherweise gleich sind)
 Spalte einer Tabelle: `table-reference.column-reference`
 Alle Spalten einer Tabelle: `table-reference.*`
- Lieferungen mit Mengen über 100 samt Lieferantennamen anzeigen (LNR, LNAME, TNR, MENGE)

```
select l.lnr, l.lname, lt.tnr, lt.menge
from lt, l
where lt.lnr = l.lnr and menge>100;

select l.lnr, l.lname, lt.tnr, lt.menge
from lt join l on lt.lnr=l.lnr
where menge>100;
```
- Wo (STADT) ist Lieferant und Teil aus der derselben Stadt?

```
select distinct l.stadt
from lt, l, t
where lt.lnr=l.lnr and lt.tnr=t.tnr and t.stadt=l.stadt;

select distinct l.stadt
from (lt join l on lt.lnr=l.lnr) join t on lt.tnr=t.tnr
where t.stadt=l.stadt;
```

 - Klammerung nicht notwendig, erleichtert Lesbarkeit
 - möglich, aber strukturell falsch:

```
select distinct l.stadt
from (lt join l on lt.lnr=l.lnr) join t on lt.tnr=t.tnr and t.stadt=l.stadt;
```
- Equi-Join über mehrere Spalten (z.B. bei zusammengesetztem Fremdschlüssel)

```
select *
from t1,t2
where t1.t1c1=t2.t1c1 and t1.t1c2=t2.t1c2;

select *
from t1 join t2 on t1.t1c1=t2.t1c1 and t1.t1c2=t2.t1c2;
```
- Verwendung von (sinnvollen) Tabellenalias erspart bei langen Tabellennamen Schreibarbeit

8.5.8 Self-Join

- Alle Lieferanten (LNR), die das Teil T1 und das Teil T2 liefern (mit SQL-89-Join)

```
select lt1.lnr
from lt lt1, lt lt2
where lt1.lnr=lt2.lnr and lt1.tnr='T1' and lt2.tnr='T2';
```

 - Verwendung von Tabellenalias notwendig
 - falsch:

```
select distinct lnr from lt where tnr='T1' and tnr='T2';
```
- Alle Lieferanten (LNR,LNAME), die das Teil T1 und das Teil T2 liefern (mit SQL-92-Join)

```
select l.lnr, l.lname
from (lt lt1 join lt lt2 on lt1.lnr=lt2.lnr) join l on lt1.lnr=l.lnr
where lt1.tnr='T1' and lt2.tnr='T2';
```
- Alle Paare von Lieferanten anzeigen, die aus derselben Stadt sind (nicht Paare mit denselben Lieferanten oder nur zu einem anderen Paar vertauschten Lieferanten)

```
select l1.lnr, l2.lnr from l l1, l l2 where l1.stadt=l2.stadt and l1.lnr<l2.lnr;
```
- ORACLE-Demodatenbank: Nummern und Namen der Mitarbeiter mit den Namen der jeweilige Vorgesetzten, sortiert nach Mitarbeitername (1:n – rekursiv)

```
select e.empno,e.ename,m.ename
from emp e join emp m on e.mgr=m.empno -- nicht: m.mgr=e.empno
order by e.ename;
```

mit dem 'Big Boss'

```
select e.empno,e.ename,m.ename from emp e join emp m on e.mgr=m.empno
union
select e.empno,e.ename,'' from emp e where e.mgr is null
order by e.ename;
```

- siehe auch Outer-Join

- SCHULUNGSFIRMA:

Welche Kurse (Bezeichnungen) setzt der Kurs 'Komposition' unmittelbar voraus? (m:n – rekursiv)

```
select kv.bezeichn
from setztvor s, kurs k, kurs kv
where s.knr=k.knr and s.knrvor=kv.knr and -- nicht symmetrisch!
      k.bezeichn='Komposition';
```

- SCHULUNGSFIRMA:

Welche Kurse (Bezeichnungen) setzt der Kurs 'Komposition' alle voraus? siehe Stored Routines

- Ortsentfernungen: Ort (ONr, OName), Entfernt (ONr1, ONr2, km)

```
select o1.ename, o2.ename, km
from entfernt e, ort o1, ort o2
where e.onr1=o1.onr and e.onr2=o2.onr; -- symmetrisch!
```

- Weiteres Anwendungsgebiet: Speicherung und Analyse von Graphen
(ungerichtet / gerichtet, ohne / mit gewichteten Kanten, ohne / mit Mehrfachkanten, etc.)

8.5.9 Sub-Selects in der Search-Condition

- Unterabfragen, Sub-Queries (non-correlated, correlated)

- search-condition:

- predicate with subquery (geschachteltes select-statement, sub-select):

- expression comparison (select-statement) 1)

Skalare Unterabfrage mit Vergleichsprädikat

- expression [NOT] IN (select-statement) 2)

- expression comparison {ALL | ANY | SOME} (select-statement) 2)

Unterabfragen mit quantifiziertem Vergleichsprädikat

ANY und SOME sind gleichbedeutend

- EXISTS (select-statement) 3)

- Notwendige Eigenschaften der Ergebnistabelle des select-statements:

1) Spaltenanzahl: 1, Zeilenanzahl: 1 (Skalares Ergebnis, Skalare Tabelle)

2) Spaltenanzahl: 1, Zeilenanzahl: n

3) Spaltenanzahl: m, Zeilenanzahl: n

- Welche Teile sind teurer als das Teil T4?

```
select * from t where preis > (select preis from t where tnr='T4');
```

- Welche Lieferanten (LNR) liefern ein Teil, das auch von L2 geliefert wird?

```
select distinct lnr
from lt
where tnr in (select tnr from lt where lnr='L2') and (lnr<>'L2');
```

- Alle Lieferanten (LNR,LNAME), die das Teil T1 und das Teil T2 liefern (Alternative)

```
select lnr,lname
from l
where lnr in (select lnr from lt where tnr='T1') and
      lnr in (select lnr from lt where tnr='T2');
```

- Alle Lieferanten (LNR,LNAME), die entweder das Teil T1 oder das Teil T2 (nicht beide) liefern

```
select lnr,lname
from l
where lnr in      (select lnr from lt where tnr='T1') and
      lnr not in (select lnr from lt where tnr='T2') or
      lnr not in (select lnr from lt where tnr='T1') and
      lnr in      (select lnr from lt where tnr='T2');
```

- Welche Lieferanten (LNR) liefern ein Teil in einer Menge, die kleiner ist als jede / eine Liefermenge des Lieferanten L4?

```
select distinct lnr
from lt
where menge < all (select menge from lt where lnr='L4');
```

```
select distinct lnr
from lt
where menge < some (select menge from lt where lnr='L4');
```

- Welche Lieferanten (LNR,LNAME) haben Teile geliefert? (mehrere Varianten)

```
select distinct l.lnr, lname from lt, l where l.lnr=lt.lnr;
select lnr, lname from l where lnr in (select lnr from lt);
select lnr, lname from l where exists (select * from lt where lt.lnr=l.lnr);
- EXISTS: In Projektion immer *, da Spalten nicht relevant
```

- Unterschied:

| | References | Processing | Remark |
|----------------------------|---------------------------|------------------------------|-------------------|
| - Non-Correlated Sub-Query | only local references | inside-out | |
| - Correlated Sub-Query | also non-local references | outside-in, 'nested loop' | always for EXISTS |

- Welche Lieferanten (LNR,LNAME) haben keine Teile geliefert?

```
select lnr, lname from l where lnr not in (select lnr from lt);
select lnr, lname from l where not exists (select * from lt where lt.lnr=l.lnr);
- wenn L einen zusammengesetzten Primärschlüssel hätte (und damit LT einen zusammengesetzten Fremdschlüssel), wäre die Variante mit EXISTS zu bevorzugen
```

- Welche Lieferanten (LNR) haben keine roten Teile geliefert?

```
select lnr from l
where not exists (select * from lt where lt.lnr=l.lnr and lt.tnr in
  (select tnr from t where farbe='rot'));
```

```
select lnr from l
where lnr not in (select lnr from lt where tnr in
  (select tnr from t where farbe='rot'));
```

```
select lnr from l where not exists
  (select * from lt, t where lt.tnr=t.tnr and lt.lnr=l.lnr and farbe='rot');
```

```
select lnr from l where lnr not in
  (select lnr from lt, t where lt.tnr=t.tnr and farbe='rot');
```

- falsch (welche Fragestellung beantwortet diese Anweisung?):

```
select distinct lnr from lt, t where lt.tnr=t.tnr and farbe<>'rot';
```

- Welche Lieferanten (LNR) liefern ausschließlich gelbe Teile?

```
select lnr from lt
where lnr not in (select lnr from lt join t on lt.tnr=t.tnr where farbe<>'gelb');
```

- Welche Lieferanten (LNR, LNAME) liefern ausschließlich gelbe Teile?

```
select lnr,lname from l
where lnr not in (select lnr from lt join t on lt.tnr=t.tnr where farbe<>'gelb')
and lnr in (select lnr from lt); -- sonst auch Lieferanten ohne Lieferungen
```

- Welche Lieferanten (LNR, LNAME) haben welche Teile (TNR) noch nicht geliefert?

```
select lnr,lname,tnr from l cross join t
where not exists (select * from lt where lt.lnr=l.lnr and lt.tnr=t.tnr);
```

8.5.10 All-Quantor

- Welche Lieferanten (alle Spalten) haben alle Teile geliefert? (kein Teil nicht geliefert)

```
select * from l where not exists
  (select * from t where not exists
    (select * from lt where lt.tnr=t.tnr and lt.lnr=l.lnr));
```


- Welche Lieferanten (LNR) haben alle Teile geliefert?

```
select distinct lnr from lt where not exists
  (select * from t where not exists
    (select * from lt lt1 where lt1.tnr=t.tnr and lt1.lnr=lt.lnr));
```
- Hinweis: Kann nicht mit zweifachem NOT IN gebildet werden

```
select * from l where lnr not in
  (select lnr from t ???)
```

 Inneres Sub-Select in diesem Fall mit NOT IN möglich

```
select * from l where not exists
  (select * from t where tnr not in
    (select tnr from lt where lt.lnr=l.lnr))
```

8.5.11 Aggregatfunktionen

- aggregate-function:
 COUNT(*)
 { COUNT | SUM | AVG | MIN | MAX } (expression)
 { COUNT | SUM | AVG | MIN | MAX } (DISTINCT column-reference)
 - Aus der Tabelle (COUNT) oder einer Spalte der Tabelle (SUM, AVG, MIN, MAX) wird ein Wert ermittelt
 - NULL-Werte werden vor Anwendung der Funktion eliminiert, außer bei COUNT(*), wo sie mitgezählt werden
 - Wenn die Tabelle leer ist, liefert COUNT 0 zurück, alle anderen Funktionen NULL
- Von wievielen Lieferanten wird das Teil T2 geliefert, wie groß sind Summe, Maximum, Minimum und Durchschnitt der Mengen?

```
select count(*), sum(menge), max(menge), min(menge), avg(menge)
from lt
where tnr='T2';
```
- Aus wieviel verschiedenen Städten sind die Lieferanten?

```
select count(distinct stadt) from l;
```
- Welche / Wieviele Lieferanten liefern mehr als eine Teilenummer?

```
select * from l
where (select count(*) from lt where lt.lnr=l.lnr)>1;

select count(*) from l
where (select count(*) from lt where lt.lnr=l.lnr)>1;
```
- Alle Lieferanten (LNR,LNAME), die entweder das Teil T1 oder das Teil T2 (nicht beide) liefern (Alternative)

```
select lnr,lname from l
where (select count(*) from lt where lt.lnr=l.lnr and tnr in ('T1','T2'))=1;
```
- Welche Lieferanten (alle Spalten) haben alle Teile geliefert? (Alternative)

```
select * from l
where (select count(*) from lt where lt.lnr=l.lnr)=(select count(*) from t);
```

 - so nur möglich, weil LNr und TNr in LT den PK bilden!
- Der dritt-kleinste / n-kleinste Primärschlüsselwert ist zu ermitteln

```
select tnr
from t
where (select count(*) from t t1 where t1.tnr<=t.tnr) = 3;
```
- Die drei / n teuersten Teile sind zu ermitteln (ohne: TOP n, FIRST n, LIMIT n, ROWNUM<=n, ...)

```
select tnr, tname, preis
from t
where (select count(*) from t t1 where t1.preis>=t.preis) <= 3;
```

8.5.12 Gruppierung

- GROUP BY:
 - Aus jenen Zeilen der Tabelle, die in der (den) bei GROUP BY angegebenen Spalte(n) denselben Wert haben, wird eine Zeile gebildet
 - Column-reference von GROUP BY muss sinnvollerweise auch in der Projektion enthalten sein
 - Die anderen Ausdrücke der Projektion müssen aggregate-functions sein
 - Aggregate-functions bilden in Zusammenhang mit GROUP BY gruppenweise Ergebnisse
 - HAVING: analog WHERE-Klausel, allerdings für Gruppen (in der search-condition muss eine aggregate-functions vorkommen, sonst kann die Bedingung in der WHERE-Klausel effizienter formuliert werden)

- Anzahl der Teile und Durchschnittspreis pro Farbe anzeigen

```
select farbe, count(*) anzahl, avg(preis) durchschnitt from t group by farbe;
```
- nicht

```
select lnr from lt group by lnr;
```

 statt

```
select distinct lnr from lt;
```
- In welchen Städten sind mindestens 2 Lieferanten? (Stadt, Anzahl der Lieferanten, durchschnittlicher Rabattwert und Anzahl der verschiedenen Rabattwerte)

```
select stadt, count(*) anzahl, avg(rabatt) durchschnitt,
       count(distinct rabatt) anz_versch
from l
group by stadt
having count(*)>1
order by stadt;
```
- Alle Lieferanten (LNR) mit ihren Umsätzen anzeigen.

```
select lt.lnr, sum(lt.menge * t.preis) umsatz
from lt join t on lt.tnr=t.tnr
group by lnr;
```
- Alle Lieferanten (LNR) mit ihren Umsätzen anzeigen, auch die ohne Umsatz

```
select lt.lnr, sum(lt.menge * t.preis) umsatz
from lt join t on lt.tnr=t.tnr
group by lnr
union
select lnr,0
from l
where lnr not in (select lnr from lt);
```

 - siehe auch Outer-Join
- Alle Lieferanten (LNR, LNAME), die mehr als zwei verschiedene Nummern roter Teile liefern, mit den (unter Berücksichtigung des Rabattsatzes gebildeten) Umsätzen bei diesen Teilen, nach Umsätzen absteigend sortiert, anzeigen

```
select lt.lnr, lname, sum(menge*preis*(1-rabatt/100)) Umsatz
from (lt join t on lt.tnr=t.tnr) join l on lt.lnr=l.lnr
where farbe='rot'
group by lt.lnr, lname
having count(*)>2
order by umsatz desc;
```
- Wieviele (nicht welche) Lieferanten liefern mehr als eine Teilenummer? (Alternative)

```
select count(*)
from l
where lnr in (select lnr from lt group by lnr having count(*)>1);
```

 - falsch, da count(*) pro Gruppe ausgeführt wird

```
select count(*) from lt group by lnr having count(*)>1;
```

8.5.13 Theta-Join

- θ -Join
- Bis jetzt EQUI-JOIN, d.h. Vergleichsoperator in der Join-Bedingung ist 'gleich'. Beim THETA-JOIN ist der Vergleichsoperator in der Joinbedingung nicht 'gleich'.
- ORACLE-Demodatenbank: Name und Gehalt der Mitarbeiter samt Gehaltsstufe anzeigen

```
select ename,sal,grade
from emp join salgrade on sal between losal and hisal;
```



```
select ename,sal,grade
from emp,salgrade
where sal between losal and hisal;
```
- siehe auch Self-Join

8.5.14 Outer-Join

- Bisher Inner-Join (Innerer Verbund):
... FROM table-ref [INNER] JOIN table-ref ON cond-expr ...
- Es müssen nicht alle Zeilen der Tabellen im Ergebnis sein (es fehlen jene, für die die cond-expr nie zutrifft)
- Alle Lieferanten mit jeweils ihren gelieferten Teilen
select l.lnr, lname, tnr, menge from l join lt on l.lnr=lt.lnr;
- Lieferanten, die nichts geliefert haben, scheinen im Ergebnis nicht auf
- Outer Join (Äußerer Verbund, Verlustfreier Verbund):
... FROM table-ref {LEFT | RIGHT | FULL} [OUTER] JOIN table-ref ON cond-expr ...
- Es sind alle Zeilen der linken (LEFT), der rechten (RIGHT) oder beider (FULL) Tabelle(n) im Ergebnis, der fehlende Teil der Zeilen wird mit NULL aufgefüllt
- Alle Lieferanten mit jeweils ihren gelieferten Teilen; auch Lieferanten, die nichts geliefert haben, sollen aufscheinen
select * from l left join lt on l.lnr=lt.lnr;
ohne Outer-Join:
select * from l join lt on l.lnr=lt.lnr
union
select l.*,NULL,NULL,NULL from l where lnr not in (select lnr from lt);
- Alle Lieferanten (LNR, LNAME) mit Anzahl der verschiedenen Teile, die sie liefern (auch Lieferanten, die keine Teile liefern - dort Anzahl gleich 0)
select l.lnr, lname, count(lt.lnr) -- Spalte aus lt, nur diese ist NULL
-- count(*) würde 1 statt 0 liefert
from l left join lt on l.lnr=lt.lnr
group by l.lnr, lname; -- nicht lt.lnr

ohne Outer-Join:
select l.lnr, lname, count(*)
from l join lt on l.lnr=lt.lnr
group by l.lnr, lname
union
select lnr, lname, 0 from l where lnr not in (select lnr from lt);
- Alle Lieferanten (LNR) mit ihren Umsätzen anzeigen, auch die ohne Umsatz (Alternative)
select l.lnr, coalesce(sum(preis*menge),0) umsatz
from (lt right join l on lt.lnr=l.lnr) left join t on lt.tnr=t.tnr
group by l.lnr; -- nicht lt.lnr

select l.lnr, coalesce(sum(preis*menge),0) umsatz
from (lt join t on lt.tnr = t.tnr) right join l on lt.lnr = l.lnr
group by l.lnr; -- nicht lt.lnr
- Welche Lieferanten (LNR,LNAME) haben keine Teile geliefert? (Alternative)
select l.lnr, l.lname from l left join lt on l.lnr=lt.lnr where lt.lnr is null;
- ORACLE-Demodatenbank: Nummern und Namen der Mitarbeiter mit den Namen der jeweilige Vorgesetzten, sortiert nach Mitarbeitername, mit dem 'Big Boss' (Alternative)
select e.empno,e.ename,coalesce(m.ename,'') ename_vorgesetzter
from emp e left join emp m on e.mgr=m.empno
order by e.ename;

8.5.15 Natural-Join

- Natürlicher Verbund
- Equi-Join mit Eliminierung jeweils einer der gemeinsamen (doppelten) Spalten

- Variante 1:
... FROM table-ref JOIN table-ref USING (column-commalist) ...
- wenn table-refs gleich A und B sind und column-commalist aus C1, C2, ..., Cn besteht, dann
... FROM A JOIN B ON A.C1=B.C1 AND A.C2=B.C2 AND ... AND A.Cn=B.Cn ...
- Jede der Spalten C1, C2, ..., Cn tritt nur einmal im Resultat auf (Werte ohnehin immer gleich)
- Die gemeinsamen Spalten scheinen als erste ('links') auf
- Auch mit Outer-Join kombinierbar
- Variante 2:
... FROM table-ref NATURAL JOIN table-ref ...
- Wie Variante 1, die column-commalist wird gebildet aus den gemeinsamen Spalten (gleicher Name) der beiden table-refs
- Wenn es keine gemeinsamen Spalten gibt, dann Degenerierung auf Cross-Join

8.5.16 Semi-Join

- Projektion der Spalten einer Tabelle aus einem Join-Ergebnis:
select A.* from A join B on A.Nr=B.Nr;
select B.* from A join B on A.Nr=B.Nr;

8.5.17 Mengenoperationen Durchschnitt und Differenz - INTERSECT, EXCEPT

- Tabellen müssen 'vereinigungsverträglich' sein
- Alle Lieferanten (LNR), die das Teil T1 und das Teil T2 liefern (Alternative)
select lnr from lt where tnr='T1'
intersect
select lnr from lt where tnr='T2';
- Alle Lieferanten (LNR), die das Teil T2 aber nicht das Teil T1 liefern (Alternative)
select lnr from lt where tnr='T2'
except
select lnr from lt where tnr='T1';
- Welche Lieferanten (LNR) liefern ausschließlich gelbe Teile? (Alternative)
select lnr from lt
except
select lnr from lt join t on lt.tnr=t.tnr where farbe<>'gelb';

8.5.18 Sub-Selects in der Projektion

- Sub-Select (wie üblich) in Klammern, muss Skalares Ergebnis liefern (Tabelle mit einer Zeile und einer Spalte)
- Beispiel
select lnr,(select lname from l where l.lnr=lt.lnr) lname, tnr, menge
from lt;
- nicht sinnvoll, besser Join:
select lnr,tnr,menge,
 (select lname from l where l.lnr=lt.lnr) lname,
 (select stadt from l where l.lnr=lt.lnr) stadt,
 (select tname from t where t.tnr=lt.tnr) tname
from lt;
- Alle Lieferanten (LNR, LName) mit Anzahl der verschiedenen Teile, die sie liefern (auch Lieferanten, die keine Teile liefern - dort Anzahl gleich 0) (Alternative)
select lnr,lname,(select count(*) from lt where lt.lnr=l.lnr) anzahl
from l;
- Alle Lieferanten (LNR) mit ihren Umsätzen anzeigen, auch die ohne Umsatz (Alternative)
select lnr, (select coalesce(sum(preis*menge),0)
 from lt join t on lt.tnr=t.tnr
 where lt.lnr=l.lnr)
from l;

- Ortsentfernungen (Alternative)

```
select o1.ortname von,o2.ortname nach,
       (select km from entfernt e where e.onr1=o1.onr and e.onr2=o2.onr) km
from ort o1, ort o2
where o1.onr<o2.onr
order by von,nach;
```

8.5.19 Sub-Selects in der From-Klausel (Inline View)

- Sub-Select in der From-Klausel: (wie üblich) in Klammern und unbedingt mit Alias-Namen; kann auch als Join-Operand verwendet werden

- Beispiel

```
select *
from (select * from t where farbe='rot') t1 join
     (select tnr,menge from lt where menge<300) lt1 on t1.tnr=lt1.tnr;
```

- Wieviele (nicht welche) Lieferanten liefern mehr als eine Teilenummer? (Alternative)

```
select count(*) from (select lnr from lt group by lnr having count(*)>1) as erg;
```

8.5.20 NULL und dreiwertige Logik

- Wenn ein Operand NULL ist, dann ist die ganze expression NULL
Ausnahme: Bei logischen Operatoren (NOT, AND, OR) gilt eine dreiwertige Logik
- Für die Operatoren NOT, AND, OR gilt folgende Verknüpfungstabelle:

| | NOT | AND | | | OR | | |
|------|------|------|------|---|----|------|------|
| | | T | NULL | F | T | NULL | F |
| T | F | T | NULL | F | T | T | T |
| NULL | NULL | NULL | NULL | F | T | NULL | NULL |
| F | T | F | F | F | T | NULL | F |

- Zeilen, für die die search-condition NULL ist, kommen nicht in das Ergebnis (NULL ist nicht TRUE)
- Abfrage auf NULL mit eigenem Prädikat (IS [NOT] NULL)
- Aggregat Functions
 - Zeilen mit NULL-Werten werden vor der Aggregation eliminiert, Ausnahme: count(*) zählt alle
 - Aggregation auf leere Tabelle liefert NULL, Ausnahme: count liefert 0
- Sortierung von NULL siehe ORDER BY

8.5.21 Zusammenfassung der Abarbeitung

- Reihenfolge der Abarbeitung:
 - 1) Tabellenverknüpfung (nach FROM)
 - 2) Zeilen auswählen gemäß search-condition (nach WHERE)
 - 3) Zeilen zusammenfassen gemäß column-reference-commalist (nach GROUP BY)
 - 4) Zeilen auswählen gemäß search-condition (nach HAVING)
 - 5) Spalten auswählen gemäß expression-commalist (nach SELECT)
 - 6) Mengenoperation(en) durchführen (UNION, INTERSECT, EXCEPT)
 - 7) Zeilen sortieren gemäß order-specification-commalist (nach ORDER BY)
- Merkregel: Abarbeitung erfolgt in der Reihenfolge der Notation der einzelnen Klauseln, außer der Projektion (diese wird am Schluss gemacht)

8.6 Datenmanipulationen

- Data Manipulation Language / Commands, DML, Module Language

8.6.1 INSERT

- Zeile(n) in eine Tabelle einfügen:

```
INSERT INTO table [ ( column-commalist ) ]
      { VALUES ( insert-atom-commalist ) | select-statement | DEFAULT VALUES }
```

insert-atom ::= literal | NULL | DEFAULT | parameter
- Lieferanten L6, namens Maxi aus Graz mit Rabatt 30 einfügen

```
insert into l values ('L6','Maxi',30,'Graz');
```
- Andere Reihenfolge der Spaltenwerte als im CREATE angegeben

```
insert into l (lname,lnr,stadt,rabatt) values ('Moritz','L7','Wien',15);
```
- Standardwerte für nicht angegebene Spalten

```
insert into l (lname,lnr) values ('Pauli','L8');
insert into l values ('L8','Pauli',DEFAULT,DEFAULT);
```
- Lieferungen mit einer Menge kleiner 150 in eine eigene Tabelle auslagern (mit Angabe des Zeitpunktes) und wieder zurückkopieren

```
create table lt_min (lnr char(2), tnr char(2), menge decimal(4), wann timestamp);
-- eventuell auch: wann ... default current_timestamp
insert into lt_min select lt.*,current_timestamp from lt where menge<150;
delete from lt where menge<150;
select * from lt_min;
select * from lt;
insert into lt select lnr,tnr,menge from lt_min;
drop table lt_min;
```
- Sehr schnell sehr viele Zeilen

```
create table x (x integer);
insert into x values (1);
insert into x select * from x; -- öfters!
drop table x;
```

8.6.2 UPDATE

- Zeile(n) in einer Tabelle ändern (searched UPDATE):

```
UPDATE table SET assignment-commalist [ WHERE search-condition ]
```

assignment ::= column = { expression | NULL | DEFAULT }

search-condition ::= siehe select-statement
- Alle Liefermengen um 10% erhöhen

```
update lt set menge=menge*1.1;
```

 - Achtung: keine WHERE-Klausel
- Preise der roten Teile verdoppeln

```
update t set preis=preis*2 where farbe='rot';
```
- Preise der roten Teile verdoppeln und Städte der roten Teile auf Wien setzen

```
update t set preis=preis*2, Stadt='Wien' where farbe='rot';
```
- Alle Athener Lieferanten ziehen nach Rom

```
update l set stadt='Rom' where stadt='Athen';
```

- Rabatt der Lieferanten, die ein gelbes Teil liefern, auf 50 setzen

```
update l set rabatt=50
where lnr in (select lnr from t join lt on t.tnr=lt.tnr where farbe='gelb');
```



```
update l set rabatt=50
where exists (select * from t join lt on t.tnr=lt.tnr
              where l.lnr=lt.lnr and farbe='gelb');
```



```
update l set rabatt=50
where lnr in (select lnr
              from lt where tnr in (select tnr from t where farbe='gelb'));
```

8.6.3 DELETE

- Zeile(n) aus einer Tabelle löschen (searched DELETE):

```
DELETE FROM table [ WHERE search-condition ]
```



```
search-condition ::= siehe select-statement
```
- Alle Lieferungen löschen

```
delete from lt;
```

 - Achtung: keine WHERE-Klausel
- Lieferungen mit kleinerer Menge als 200 löschen

```
delete from lt where menge<200;
```
- Die Lieferungen von Londoner Lieferanten löschen

```
delete from lt where lnr in (select lnr from l where stadt='London');
```

8.7 Transaktionssteuerung

- Transaction Control Language / Commands
- Transaktion beginnt: bei erster ausführbarer SQL-Anweisung
- Transaktion normal abschließen:

```
COMMIT [ WORK ]
```
- Transaktion zurücknehmen und abschließen:

```
ROLLBACK [ WORK ]
```

8.8 Virtuelle Tabellen - VIEW

- Werden aus anderen Tabellen, als Ergebnis einer Select-Anweisung, gebildet
- Haben einen eigenen Namen (View, Pseudo-Tabelle, viewed-table zum Unterschied von base-table) und werden wie reale Tabellen benutzt
- Werden nicht nur einmal statisch beim Erstellen der View gebildet, sondern sind stets am aktuellen Stand der Tabellen, von denen sie abgeleitet sind
- Umgekehrt sind Datenmanipulationen (INSERT, UPDATE, DELETE) in Views, die sich auf die Ursprungstabellen auswirken, nur eingeschränkt anwendbar (updatability)
- Es können auch Views von Views definiert werden
- Zweck:
 - Vereinfachung sich wiederholender Abfragen (Restriktion, Join, etc.)
 - z.B. nur aktive Mitglieder, nicht ausgetretene, nicht verstorbene (Restriktion)
 - z.B. nur weibliche Patienten in Geburtsabteilung (Restriktion)
 - z.B. immer Name des Referenten bei der Kursveranstaltung notwendig (Join über PNr)
 - z.B. pro Sub-Entity-Typ eine View mit allen Attributen definieren (Natural Join)

- Datenschutzmaßnahmen
- Änderung der Tabellenstruktur: wenn möglich View bilden, die auf die alten Strukturen abbildet
z.B. Änderung einer Tabelle mit Person - Gehalt auf
zwei Tabellen mit Person - Gehaltsstufe und Gehaltsstufe – Gehalt.
Mit Join kann auf Struktur der ursprünglichen Tabelle abgebildet werden.
- View erstellen:

```
CREATE VIEW view [ ( column-commalist ) ]
      AS select-statement [ WITH CHECK OPTION ]
```

 - column-commalist: Spaltennamen der View. Müssen verwendet werden, wenn die expression-commalist des select-statements Ausdrücke oder gleiche Spaltennamen enthält
 - select-statement: ohne ORDER BY
 - WITH CHECK OPTION: Änderungen in der View dürfen der search-condition der WHERE-Klausel nicht widersprechen
- View löschen:

```
DROP VIEW view
```
- Implementierung: Select-Anweisung jedes Mal ausführen oder transparentes Abspeichern des Select-Ergebnisses (Materialized View)

8.9 Zeilenweise Verarbeitung - CURSOR

- Verarbeitung von Einzelzeilen und Zuweisung an Programmvariable (parameter) kann in Programmiersprachen (Hochsprachen oder 4GL) auf zwei Arten erfolgen:
 - Singleton Select (Ergebnistabelle darf nur eine Zeile haben):


```
SELECT ...
      { * | expression-commalist }
      INTO parameter-commalist
      FROM table-reference-commalist
      ...
```
 - Cursor-Konzept
- Ein Cursor ist ein Zeiger auf eine Zeile einer Tabelle, die als Ergebnistabelle einer Select-Anweisung ermittelt wird
- Cursor definieren:


```
DECLARE cursor CURSOR FOR select-statement [FOR UPDATE]
```

 - FOR UPDATE notwendig für positioned UPDATE und DELETE
- Cursor öffnen:


```
OPEN cursor
```

 - Wertet das select-statement der Cursordefinition aus und stellt den Cursor vor die erste Zeile der Ergebnistabelle
- Zeile lesen:


```
FETCH cursor INTO parameter-commalist
```

 - Der Cursor wird um eine Zeile in der Ergebnistabelle weiterbewegt und danach der Inhalt der Spalten in entsprechender Reihenfolge den Parametern zugewiesen (Anzahl und Datentyp!)
 - Wenn beim Weiterbewegen des Cursors das Tabellenende überschritten wird
 - Variablen wird bestimmter Wert zugewiesen (z.B. sqlcode=100)
 - Funktion liefert bestimmten Wert
 - Eigenschaft des Cursor-Objekts nimmt einen bestimmten Wert an, etc.
 - etc.
- Zeile ändern (positioned UPDATE):


```
UPDATE table SET assignment-commalist WHERE CURRENT OF cursor
```
- Zeile löschen (positioned DELETE):


```
DELETE FROM table WHERE CURRENT OF cursor
```
- Cursor schließen:


```
CLOSE cursor
```


8.10 Übungen zu SQL

- 1) Erstellen Sie die SQL-Anweisungen zur Definition der Tabellen, die dem in 2.3.3. dargestellten ER-Diagramm entsprechen. Überlegen Sie, in welcher Reihenfolge die Zeilen in die Tabellen Angestellter und Abteilung einzufügen sind. Ergänzen Sie die SQL-Anweisungen auch um das Löschen der Tabellen.
Abfrage: Namen aller Angestellten, Bezeichnung der Abteilung zu der er gehört und Name des Abteilungsleiters

2) Join-Beispiele

```
drop table a;
drop table b;
```

```
create table a (      a1 integer, a2 integer, a3 integer);
insert into a values ( 1,          1,          1);
insert into a values ( 2,          1,          NULL);
insert into a values ( 3,          1,          1);
insert into a values ( 4,          1,          NULL);
```

```
create table b (      b1 integer, b2 integer, b3 integer);
insert into b values ( 1,          2,          1);
insert into b values ( 2,          2,          NULL);
insert into b values ( 3,          2,          1);
```

```
select * from a cross join b;
```

```
select * from a join b on a.a1=b.b1;
select * from a join b on a.a1=b.b2;
select * from a join b on a.a1=b.b3;
select * from a join b on a.a2=b.b1;
select * from a join b on a.a2=b.b2;
select * from a join b on a.a2=b.b3;
select * from a join b on a.a3=b.b1;
select * from a join b on a.a3=b.b2;
select * from a join b on a.a3=b.b3;
```

```
select * from a join b on a.a1=b.b1*2;
select * from a join b on a.a2=b.b2-1;
```

```
select * from a join b on a.a1=b.b1 and a.a2=b.b2;
select * from a join b on a.a1=b.b2 and a.a2=b.b1;
```

```
select * from a a1 join a a2 on a1.a1=a2.a1;
select * from a a1 join a a2 on a1.a2=a2.a2;
select * from a a1 join a a2 on a1.a3=a2.a3;
```

```
select * from a a1 join a a2 on a1.a1=a2.a2;
select * from a a1 join a a2 on a1.a1=a2.a3;
select * from a a1 join a a2 on a1.a2=a2.a3;
```

```
select * from a join b on a.a1<=b.b2;
select * from a join b on a.a2<>b.b1;
select * from a join b on a.a1> b.b3;
select * from a join b on a.a3<>b.b3;
```

```
select * from a join b on a.a1=b.b1 where a.a3 is not null;
select * from a join b on a.a3=b.b3 where a.a1<4;
```

Wieviele Zeilen haben die Ergebnistabellen der einzelnen Joins?

Wie schauen die Ergebnistabellen der einzelnen Joins aus?

3) Eine Tabelle a (a1, a2) mit 3 Zeilen sei gegeben (alles INTEGER-Spalten):

```
drop table a;
create table a (a1 integer primary key, a2 integer not null);
```

| | Zeilenanz. mindestens | Zeilenanz. höchstens |
|---|--------------------------|-------------------------|
| <code>select * from a where a1=7;</code> | | |
| <code>select * from a where a2=7;</code> | | |
| <code>select * from a where a1=a2;</code> | | |
| <code>select * from a x cross join a y;</code> | | |
| <code>select * from a x join a y on x.a1=y.a1;</code> | | |
| <code>select * from a x join a y on x.a1=y.a2;</code> | | |
| <code>select * from a x join a y on x.a2=y.a2;</code> | | |

Wie ändern sich die Ergebnisse, wenn bei a2 nicht NOT NULL angegeben ist?

4) Lösen Sie folgende Aufgabenstellungen mit SQL-Anweisungen (Datenbank LT):

a) Alle Teile (vollständige Information), die von keinem Lieferanten geliefert wurden, der aus derselben Stadt wie das Teil ist.

b) Alle Lieferanten (LNr, LName), die zumindest alle Teile liefern, die auch L2 liefert.

c) Alle Lieferanten (LNr, LName), die das Teil T2 in einer Menge liefern, die größer als die durchschnittliche Liefermenge von T2 ist.

d) Pro Lieferant ist das teuerste Teil, das von ihm geliefert wurde, anzuzeigen.
Form: LNr, TNr, TName, Preis

e) Erhöhen Sie bei allen Lieferungen von Lieferanten aus Paris die Menge um 10%.

f) Löschen Sie alle Lieferanten, die kein Teil geliefert haben.

g) Definieren Sie eine View, die nur Lieferungen (samt Lieferantennamen und Teilnamen) enthält, bei denen der Umsatz größer als 2000 ist.

5) Für alle roten und blauen Teile, bei denen die Gesamtliefermenge größer als 350 ist (wobei in der Gesamtliefermenge Lieferungen in einer Menge kleiner oder gleich 200 nicht berücksichtigt werden sollen), sind die Teilnummer, der Preis in Dollar (1 Euro = 1,2 Dollar), die Farbe und die größte Liefermenge (nach dieser absteigend sortiert) auszugeben (Datenbank LT).

6) Gegeben sind folgende Tabellen mit den daneben stehenden Bedeutungen.

| | | |
|-----------|-------------------|---|
| BESUCHER | (GAST, WIRTSHAUS) | Welcher Gast besucht welches Wirtshaus? |
| ANGEBOT | (WIRTSHAUS, BIER) | Welches Wirtshaus bietet welches Bier an? |
| GESCHMACK | (GAST, BIER) | Welcher Gast mag welches Bier? |

Geben Sie jeweils einen SQL-Befehl an, der die folgenden Fragen beantwortet (*=leicht, **=mittel, ***=schwer):

- * Welche Wirtshäuser bieten ein Bier an, das 'Hans' mag?
- * Welche Gäste besuchen (mindestens) ein Wirtshaus, das (mindestens) ein Bier nach ihrem Geschmack anbietet?
- *** Welche Gäste besuchen nur Wirtshäuser, die (mindestens) ein Bier nach ihrem Geschmack anbieten?
- ** Welche Gäste besuchen kein Wirtshaus, das (mindestens) ein Bier nach ihrem Geschmack anbietet?
- * Welche Biere, die in irgendeinem Wirtshaus angeboten werden, mag kein Gast?
- *** Welche Gäste der 'Braustube' mögen alle Biere, die dort angeboten werden?
- ** Welche Wirtshäuser bieten alle Biere an, die irgendein Gast mag?

7) ORACLE-Demodatenbank:

DEPT

| DEPTNO | DNAME | LOC |
|--------|------------|----------|
| 10 | ACCOUNTING | NEW YORK |
| 20 | RESEARCH | DALLAS |
| 30 | SALES | CHICAGO |
| 40 | OPERATIONS | BOSTON |

EMP

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|-------|--------|-----------|------|------------|---------|---------|--------|
| 7369 | SMITH | CLERK | 7902 | 1980-12-17 | 800,00 | NULL | 20 |
| 7499 | ALLEN | SALESMAN | 7698 | 1981-02-20 | 1600,00 | 300,00 | 30 |
| 7521 | WARD | SALESMAN | 7698 | 1981-02-22 | 1250,00 | 500,00 | 30 |
| 7566 | JONES | MANAGER | 7839 | 1981-04-02 | 2975,00 | NULL | 20 |
| 7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250,00 | 1400,00 | 30 |
| 7698 | BLAKE | MANAGER | 7839 | 1981-05-01 | 2850,00 | NULL | 30 |
| 7782 | CLARK | MANAGER | 7839 | 1981-06-09 | 2450,00 | NULL | 10 |
| 7788 | SCOTT | ANALYST | 7566 | 1982-12-09 | 3000,00 | NULL | 20 |
| 7839 | KING | PRESIDENT | NULL | 1981-11-17 | 5000,00 | NULL | 10 |
| 7844 | TURNER | SALESMAN | 7698 | 1981-09-08 | 1500,00 | 0,00 | 30 |
| 7876 | ADAMS | CLERK | 7788 | 1983-01-12 | 1100,00 | NULL | 20 |
| 7900 | JAMES | CLERK | 7698 | 1981-12-03 | 950,00 | NULL | 30 |
| 7902 | FORD | ANALYST | 7566 | 1981-12-03 | 3000,00 | NULL | 20 |
| 7934 | MILLER | CLERK | 7782 | 1982-01-23 | 1300,00 | NULL | 10 |

SALGRADE

| GRADE | LOSAL | HISAL |
|-------|---------|---------|
| 1 | 700,00 | 1200,00 |
| 2 | 1201,00 | 1400,00 |
| 3 | 1401,00 | 2000,00 |
| 4 | 2001,00 | 3000,00 |
| 5 | 3001,00 | 9999,00 |

8) Ein Fahrtenbuch wird in einer Tabelle geführt:

```
create table fbuch (datum date primary key, kmstand integer, km integer);
```

Am Tagesende wird der Kilometerstand in das Fahrtenbuch geschrieben.

In die Spalte km sollen die am Tag gefahrenen Kilometer eingetragen werden (CURSOR mit POSITIONED UPDATE).

8.11 Datenbank LTP: Lieferanten - Teile - Projekte - Lieferungen

L

| LNR | LNAME | RABATT | STADT |
|-----|--------|--------|--------|
| L1 | Schmid | 20 | London |
| L2 | Jonas | 10 | Paris |
| L3 | Berger | 30 | Paris |
| L4 | Klein | 20 | London |
| L5 | Adam | 30 | Athen |

T

| TNR | TNAME | FARBE | PREIS | STADT |
|-----|----------|-------|-------|--------|
| T1 | Mutter | rot | 12 | London |
| T2 | Bolzen | gelb | 17 | Paris |
| T3 | Schraube | blau | 17 | Rom |
| T4 | Schraube | rot | 14 | London |
| T5 | Welle | blau | 12 | Paris |
| T6 | Zahnrad | rot | 19 | London |

P

| PNR | PNAME | STADT |
|-----|-----------|--------|
| P1 | Flugzeug | Paris |
| P2 | Schiff | Rom |
| P3 | Seilbahn | Athen |
| P4 | Schiff | Athen |
| P5 | Eisenbahn | London |
| P6 | Flugzeug | Oslo |
| P7 | Autobus | London |

LTP

| LNR | TNR | PNR | MENGE |
|-----|-----|-----|-------|
| L1 | T1 | P1 | 200 |
| L1 | T1 | P4 | 700 |
| L2 | T3 | P1 | 400 |
| L2 | T3 | P2 | 200 |
| L2 | T3 | P3 | 200 |
| L2 | T3 | P4 | 500 |
| L2 | T3 | P5 | 600 |
| L2 | T3 | P6 | 400 |
| L2 | T3 | P7 | 800 |
| L2 | T5 | P2 | 100 |
| L3 | T3 | P1 | 200 |
| L3 | T4 | P2 | 500 |
| L4 | T6 | P3 | 300 |
| L4 | T6 | P7 | 300 |
| L5 | T1 | P4 | 100 |
| L5 | T2 | P2 | 200 |
| L5 | T2 | P4 | 100 |
| L5 | T3 | P4 | 200 |
| L5 | T4 | P4 | 800 |
| L5 | T5 | P4 | 400 |
| L5 | T5 | P5 | 500 |
| L5 | T5 | P7 | 100 |
| L5 | T6 | P2 | 200 |
| L5 | T6 | P4 | 500 |

MENGE > 0

- 1) Tabellen anlegen
 - a) Erstellen der Tabellen
 - b) Füllen der Tabellen
- 2) Einfache Abfragen
 - a) Alle Projekte im Detail
 - b) Projekte in London im Detail
 - c) Nummern der Lieferanten, die das Projekt P2 beliefern
 - d) Lieferungen mit einer Menge im Bereich von 300 bis 750
 - e) Verschiedene Kombinationen aus Teile-Farben und Teile-Städte
 - f) Lieferungen mit einer nichtleeren Menge
 - g) Projektnummer und Städte mit einem 'o' als zweiten Buchstaben des Stadtnamens
- 3) UNION
 - a) Liste aller Städte aus denen zumindest ein Lieferant, ein Teil oder ein Projekt ist
 - b) Ergebnis von: `SELECT FARBE FROM T UNION SELECT FARBE FROM T;`
 - c) 3b) mit UNION ALL statt UNION
 - d) Ergebnis von: `SELECT DISTINCT FARBE FROM T UNION ALL SELECT DISTINCT FARBE FROM T;`
- 4) Subqueries
 - a) Namen der Projekte, die vom Lieferanten L1 beliefert werden
 - b) Farben der Teile, die vom Lieferanten L1 geliefert werden
 - c) Nummern der Teile, die an ein Projekt in London geliefert werden
 - d) Nummern der Projekte, die zumindest ein Teil beinhalten, das L1 liefert
 - e) Nummern der Lieferanten, die zumindest ein Teil liefern, das zumindest von einem Lieferant geliefert wird, der zumindest ein rotes Teil liefert
 - f) Nummern der Lieferanten mit einem Rabatt kleiner als der des Lieferanten L1

5) Join

- a) Tripel aus Lieferanten-, Teile- und Projektnummer so, dass die Lieferanten, Teile und Projekte alle aus derselben Stadt sind
- b) Tripel aus Lieferanten-, Teile- und Projektnummer so, dass die Lieferanten, Teile und Projekte nicht alle aus derselben Stadt sind
- c) Tripel aus Lieferanten-, Teile- und Projektnummer so, dass nicht zwei der Lieferanten, Teile und Projekte aus derselben Stadt sind
- d) Nummern der Teile, die von einem Lieferanten aus London geliefert werden
- e) Nummern der Teile, die von einem Lieferanten aus London an ein Projekt in London geliefert werden
- f) Paare von Städtenamen so, dass ein Lieferant aus der ersten Stadt ein Projekt in der zweiten Stadt beliefert
- g) Nummern der Teile, die an ein Projekt von einem Lieferanten geliefert werden, der aus derselben Stadt wie das Projekt ist
- h) Nummern der Projekte, die zumindest von einem Lieferanten beliefert werden, der nicht aus derselben Stadt wie das Projekt ist
- i) Paare von Teilenummern so, dass derselbe Lieferant beide Teile liefert (Self-Join)

6) IN / EXISTS

- a) 4c) mit EXISTS
- b) 4d) mit EXISTS
- c) Nummer der Projekte, die nicht mit roten Teilen von Lieferanten aus London beliefert werden
- d) Nummern der Projekte, die ausschließlich von Lieferant L2 beliefert werden
- e) Nummern der Teile, die an alle Projekte in London geliefert werden
- f) Nummern der Projekte, die zumindest mit allen Teilen beliefert werden, die Lieferant L1 liefert
- g) Welche Lieferanten (alle Spalten) haben alle Teile geliefert?
- h) Welche Lieferanten (alle Spalten) haben alle Teile an ein Projekt geliefert?
- i) Welche Lieferanten (alle Spalten) haben ein Teil an alle Projekte geliefert?
- j) Welche Lieferanten (alle Spalten) haben alle Teile an alle Projekte geliefert?

7) Aggregatfunktionen

- a) Anzahl der Projekte, die vom Lieferant L5 beliefert werden
- b) Gesamtliefermenge des Teiles T1 durch Lieferant L1
- c) Für jedes Teil, das an ein Projekt geliefert wird: Teilenummer, Projektnummer und entsprechende Summe der Liefermengen
- d) Nummern der Projekte, deren Stadt die erste in der alphabetischen Reihenfolge der Projekt-Städte ist
- e) Nummern der Projekte, die mit Teil T1 in einer durchschnittlichen Menge beliefert werden, die größer ist als die größte Liefermenge in der ein Teil an Projekt P1 geliefert wird
- f) Nummern der Lieferanten, die ein Projekt mit Teil T1 in einer Menge beliefern, die größer ist als die Durchschnittsmenge in der Teil T1 an dieses Projekt geliefert wird (Correlated Sub-Query)

8) INTERSECT / EXCEPT

- a) Projekte (nur PNr oder PNr und PName), die keine Teile aus London beinhalten
- b) Lieferanten (nur LNr oder LNr und LName), die Projekte in London und Paris beliefern
- c) 8a) und 8b) ohne Mengenoperationen

9) Änderungen von Tabelleninhalten

- a) Farbe aller roten Teile in rosa ändern
- b) Alle Projekte, die nicht beliefert werden, löschen
- c) Bei allen Lieferungen von Lieferanten, die ein rotes Teil liefern, die Menge um 10 Prozent erhöhen
- d) Alle Projekte in Rom löschen
- e) Neuen Lieferanten (L9) aus Wien namens Schuh aufnehmen, dessen Rabatt noch nicht bekannt ist
- f) Tabelle erstellen mit allen Nummern von Teilen, die von einem Lieferanten aus London oder an ein Projekt in London geliefert werden
- g) Tabelle erstellen mit allen Nummern von Projekten, die in London sind oder von einem Lieferanten aus London beliefert werden
- h) Bei allen Lieferanten, deren Rabatt kleiner als der von Lieferant L4 ist, den Rabatt um 5 erhöhen

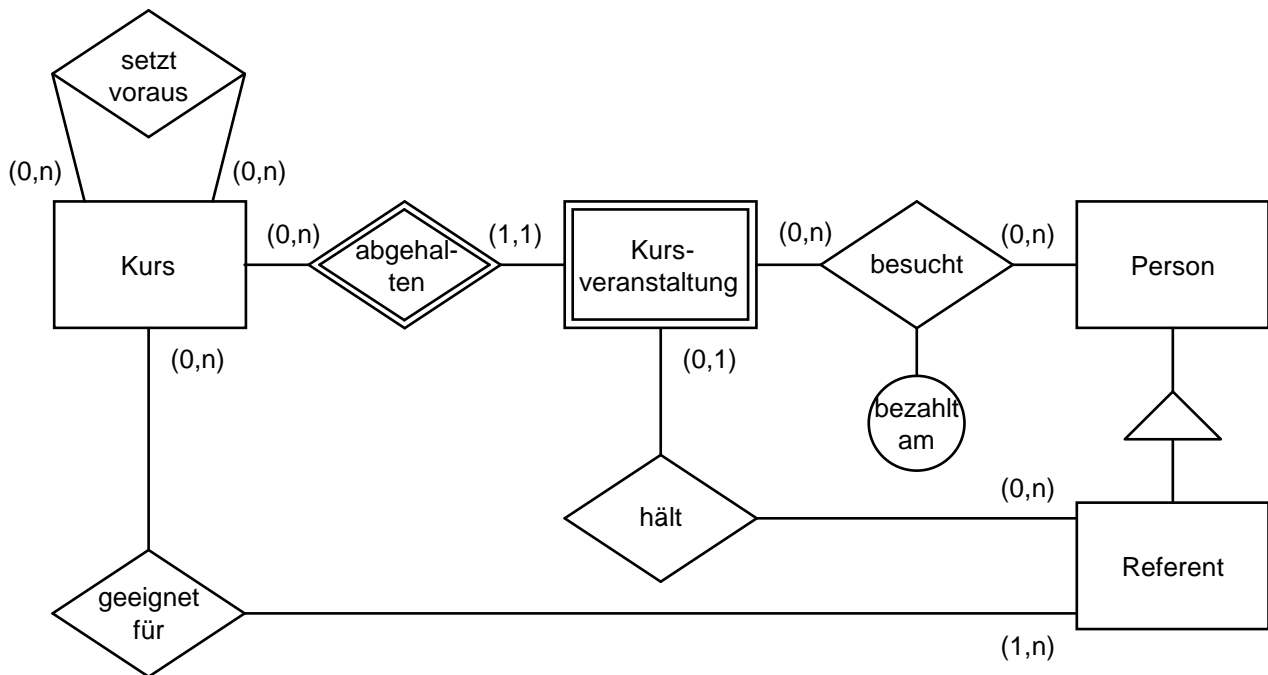
10) Views

- a) Projekte in London (PNR, PNAME, STADT)
- b) Aus LTP abgeleitet: LT (LNR, TNR, MENGE)
- c) Projekte (PNR, STADT), die von Lieferant L1 oder mit Teil T1 beliefert werden
- d) Alle Kombinationen aus Nummern von Teilen und Lieferanten, die nicht aus derselben Stadt sind (LNR, TNR)

11) Tabellen löschen

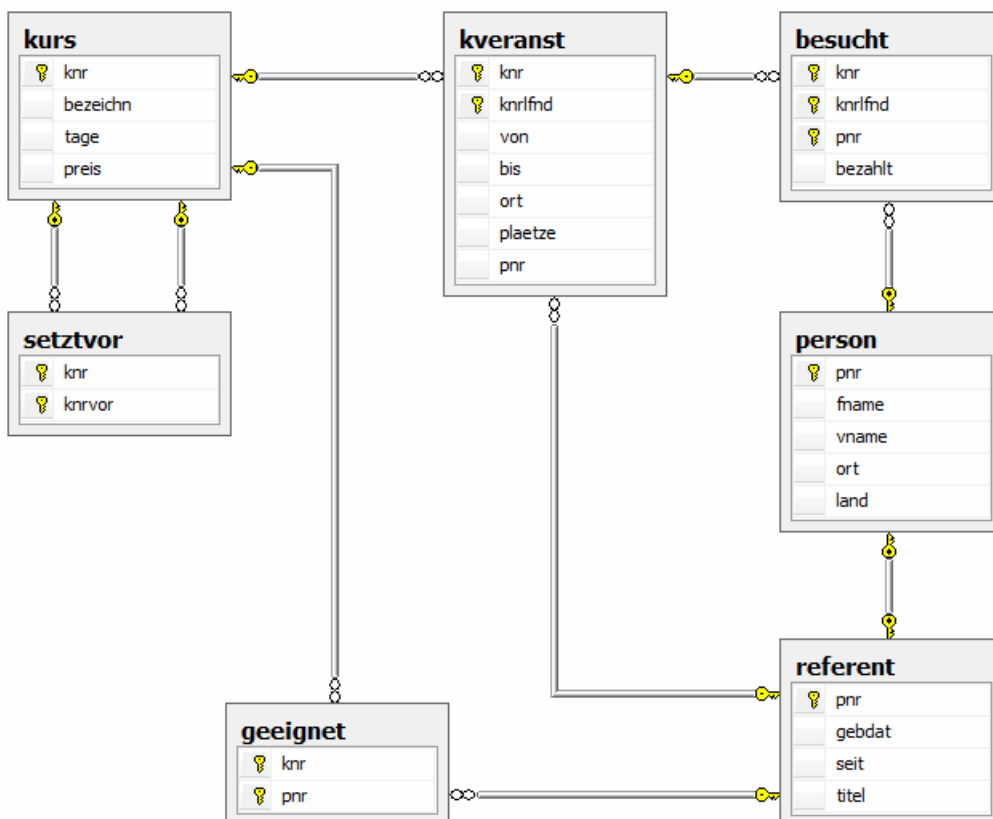
8.12 Datenbank Schulungsfirma

- 1) Erstellen Sie für folgendes ER-Diagramm einer Schulungsfirma die notwendigen SQL-Anweisungen zur Definition der entsprechenden Tabellen



Attribute

- Person: PNr, FName, VName, Ort, Land (A, D, F, GB, I oder RUS)
 - Referent: GebDat, Seit, Titel (GebDat kleiner Seit)
 - Kurs: KNr, Bezeichn, Tage (mindestens 1 und höchstens 10), Preis
 - KVeranst: KNrL_fnd, Von, Bis, Ort, Plätze (von nicht größer als bis, bis kann NULL sein)
- 2) Die Schulungsfirma hat ihre Datenbank in einem RDBMS implementiert. Das folgende Datenbankdiagramm zeigt die Tabellen (mit ihren Spalten und Primärschlüssel) samt den Beziehungen zwischen den Tabellen (Fremdschlüssel)



3) Tabelleninhalte

PERSON

| PNR | FNAME | VNAME | ORT | LAND |
|-----|---------------|------------------|----------|------|
| 101 | Bach | Johann Sebastian | Leipzig | D |
| 102 | Händel | Georg Friedrich | London | GB |
| 103 | Haydn | Joseph | Wien | A |
| 104 | Mozart | Wolfgang Amadeus | Salzburg | A |
| 105 | Beethoven | Ludwig van | Wien | A |
| 106 | Schubert | Franz | Wien | A |
| 107 | Berlioz | Hector | Paris | F |
| 108 | Liszt | Franz | Wien | A |
| 109 | Wagner | Richard | München | D |
| 110 | Verdi | Giuseppe | Busseto | I |
| 111 | Bruckner | Anton | Linz | A |
| 112 | Brahms | Johannes | Wien | A |
| 113 | Bizet | Georges | Paris | F |
| 114 | Tschaikowskij | Peter | Moskau | RUS |
| 115 | Puccini | Giacomo | Mailand | I |
| 116 | Strauss | Richard | München | D |
| 117 | Schönberg | Arnold | Wien | A |

REFERENT

| PNR | GEBDAT | SEIT | TITEL |
|-----|-----------|----------|-------|
| 101 | 21.3.1935 | 1.1.1980 | |
| 103 | 1.4.1932 | 1.1.1991 | |
| 104 | 27.1.1956 | 1.7.1985 | |
| 111 | 4.9.1924 | 1.7.1990 | Mag |
| 114 | 25.4.1940 | 1.7.1980 | |
| 116 | 11.6.1964 | 1.1.1994 | Dr |

KURS

| KNR | BEZEICHN | TAGE | PREIS |
|-----|-------------------|------|---------|
| 1 | Notenkunde | 2 | 1400,00 |
| 2 | Harmonielehre | 3 | 2000,00 |
| 3 | Rhythmik | 1 | 700,00 |
| 4 | Instrumentenkunde | 2 | 1500,00 |
| 5 | Dirigieren | 3 | 1900,00 |
| 6 | Musikgeschichte | 2 | 1400,00 |
| 7 | Komposition | 4 | 3000,00 |

GEEIGNET

| KNR | PNR |
|-----|-----|
| 1 | 103 |
| 1 | 114 |
| 2 | 104 |
| 2 | 111 |
| 3 | 103 |
| 4 | 104 |
| 5 | 101 |
| 5 | 114 |
| 6 | 111 |
| 7 | 103 |
| 7 | 116 |

SETZTVOR

| KNR | KNRVOR |
|-----|--------|
| 2 | 1 |
| 3 | 1 |
| 5 | 2 |
| 5 | 3 |
| 5 | 4 |
| 7 | 5 |
| 7 | 6 |

KVERANST

| KNR | KNRLEFND | VON | BIS | ORT | PLAETZE | PNR |
|-----|----------|------------|------------|----------|---------|-----|
| 1 | 1 | 7.4.2003 | 8.4.2003 | Wien | 3 | 103 |
| 1 | 2 | 23.6.2004 | 24.6.2004 | Moskau | 4 | 114 |
| 1 | 3 | 10.4.2005 | 11.4.2005 | Paris | 3 | |
| 2 | 1 | 9.10.2003 | 11.10.2003 | Wien | 4 | 104 |
| 3 | 1 | 17.11.2003 | 17.11.2003 | Moskau | 3 | 103 |
| 4 | 1 | 12.1.2004 | 13.1.2004 | Wien | 2 | 116 |
| 4 | 2 | 28.3.2004 | 29.3.2004 | Wien | 4 | 104 |
| 5 | 1 | 18.5.2004 | 20.5.2004 | Paris | 3 | 101 |
| 5 | 2 | 23.9.2004 | 26.9.2004 | Wien | 2 | 101 |
| 5 | 3 | 30.3.2005 | 1.4.2005 | Rom | 3 | |
| 7 | 1 | 9.3.2005 | 13.3.2005 | Wien | 5 | 103 |
| 7 | 2 | 14.9.2005 | 18.9.2005 | Muenchen | 4 | 116 |

BESUCHT

| KNR | KNRLEFND | PNR | BEZAHLT |
|-----|----------|-----|------------|
| 1 | 1 | 108 | 1.5.2003 |
| 1 | 1 | 109 | |
| 1 | 1 | 114 | |
| 1 | 2 | 110 | 1.7.2004 |
| 1 | 2 | 112 | 3.7.2004 |
| 1 | 2 | 113 | 20.7.2004 |
| 1 | 2 | 116 | |
| 1 | 3 | 110 | |
| 2 | 1 | 105 | 15.10.2003 |
| 2 | 1 | 109 | 3.11.2003 |
| 2 | 1 | 112 | 28.10.2003 |
| 2 | 1 | 116 | |
| 3 | 1 | 101 | |
| 3 | 1 | 109 | |
| 3 | 1 | 117 | 20.11.2003 |
| 4 | 1 | 102 | 20.1.2004 |
| 4 | 1 | 107 | 1.2.2004 |
| 4 | 1 | 111 | |
| 4 | 2 | 106 | 7.4.2004 |
| 4 | 2 | 109 | 15.4.2004 |
| 5 | 1 | 103 | |
| 5 | 1 | 109 | 7.6.2004 |
| 5 | 2 | 115 | 7.10.2004 |
| 5 | 2 | 116 | |
| 7 | 1 | 109 | 20.3.2005 |
| 7 | 1 | 113 | |
| 7 | 1 | 117 | 8.4.2005 |

- 4) Welche Kurse (KNr) haben einen Kurs als Voraussetzung?
- 5) Welche Kurse (Bezeichnung) dauern zwischen 2 und 4 Tagen und haben einen durchschnittlichen Tagespreis von höchstens 700,--? (aufsteigend nach Bezeichnung sortiert)
- 6) Welche Personen (Familiename) haben ein Leerzeichen im Vornamen und denselben Vokal zweimal im Ort?
- 7) Welche Personen (PNr, aufsteigend sortiert danach) haben noch nicht alle Kursbesuche bezahlt?
- 8) Wieviele Tage dauern die Kursveranstaltungen, die in Wien stattfinden und von Referent 103 oder 104 gehalten werden (KNr, KNrLfnd, Tage - danach absteigend)?
- 9) Welche Referenten (PNr, Alter) sind älter als 75 Jahre?
- 10) Welche Personen (PNr) halten oder besuchen mindestens eine Kursveranstaltung?
- 11) Alle Kursveranstaltungen (KNr, Bezeichnung, KNrLfnd, von) bei denen noch kein Referent festgelegt ist
- 12) Alle Kursveranstaltungen (KNr, Bezeichnung, KNrLfnd, von), die mindestens ein Teilnehmer besucht
- 13) Alle Kursveranstaltungen (KNr, Bezeichnung, KNrLfnd, von), die mindestens ein Teilnehmer besucht und bei denen schon ein Referent festgelegt ist
- 14) Referenten zahlen für Besuche von Kursveranstaltungen nichts.
Zeigen Sie Besuche (samt Kursbezeichnung und Familienname) an, wo dies nicht eingehalten ist.
- 15) Teilnehmerliste pro Kursveranstaltung mit folgenden Spalten (sortiert nach vonDatum und Kursbezeichnung):
 - Kursbezeichnung
 - vonDatum
 - Familien- und Vorname des Referenten
 - Familien- und Vorname des Teilnehmers
- 16) Welche Kursveranstaltungen (Bezeichnung, vonDatum) werden von Referenten besucht?
- 17) Welche Kursveranstaltungen (Bezeichnung, vonDatum) werden von Referenten besucht und halten diese auch? (Fehlerfall!)
- 18) Welche Personen (FName) haben den Kurs Nr 1 und den Kurs Nr 5 besucht?
- 19) Alle Kursveranstaltungen mit durchschnittlichen Tagespreisen zwischen 610,-- und 690,--, die von Referenten ohne Titel gehalten werden (Bezeichnung, von, bis, durchschnittlicher Tagespreis)
- 20) Welche Kursbesuche wurden vor dem Kursbeginn bezahlt?
Welche Kursbesuche wurden während des Kurses bezahlt?
Welche Kursbesuche wurden nach dem Kursende bezahlt?
- 21) Welche Personen (Familiename, danach sortiert) besuchen Kursveranstaltungen, die in ihrem Wohnort abgehalten werden und länger als zwei Tage dauern?
- 22) Welche Kursveranstaltungen (Bezeichnung, laufende Nummer) werden von Referenten gehalten, die für den Kurs auch geeignet sind?
- 23) Alle Referenten (Nummer und Name), die Kursveranstaltungen gehalten haben bevor / nachdem sie in die Firma eingetreten sind (seit).
- 24) Alle Personen (PNr, FName), die einen Kurs in 'Wien' besucht oder gehalten haben.
- 25) Dauer der Kursveranstaltungen im Vergleich mit der im Kurs angegebenen Dauer
(geht die Veranstaltung über ein Wochenende / Sa,So?)
- 26) Welche Referenten (Nummer und Name), haben Kursveranstaltungen in einem Alter von über 60 Jahren gehalten?

- 27) Welche Kursveranstaltungen gibt es, zu denen eine (unmittelbar) vorausgesetzte Kursveranstaltung zeitlich davor und am selben Ort abgehalten wird?
(jeweils alle Daten der Kursveranstaltung und der vorausgesetzten Kursveranstaltung)
- 28) Welche Kursveranstaltungen überschneiden einander terminlich? (je Bezeichnung, laufende Nummer, von, bis)
- 29) Gibt es Personen (Familien- und Vorname), bei denen Kursbesuche einander terminlich überschneiden?
- 30) Gibt es Referenten (Familien- und Vorname), bei denen Kursveranstaltungen, die sie halten, einander terminlich überschneiden?
- 31) Laut ERD muss jeder Referent für mindestens einen Kurs geeignet sein.
Gibt es Referenten (Name), bei denen diese Bedingung nicht eingehalten ist?
- 32) Welche Kurse (Bezeichnung) kosten nicht mehr als der Kurs 'Dirigieren'?
- 33) Welche Kurse (Bezeichnung) sind teurer als alle, die in 'Paris' veranstaltet wurden?
- 34) Welche Kurse (Bezeichnung) setzen keine Kurse voraus?
- 35) Welche Personen (FName, VName) haben im Jahr 2003 oder 2005 keine Kursveranstaltung besucht?
- 36) Welche Personen (FName, VName) haben im Jahr 2003 und 2005 keine Kursveranstaltung besucht?
- 37) Welche Personen (FName, VName) haben einen Kurs besucht, der billiger ist als ein Kurs, der in 'Moskau' veranstaltet wurde?
- 38) Alle Kursveranstaltungen (KNr, Bezeichnung, KNrLfd und von), die von einem Referent (PNr) gehalten werden, der nicht geeignet ist
- 39) Welche Personen (PNr und FName) besuchen Kurse in 'Wien' und in 'Paris'?
Welche Personen (PNr und FName) besuchen Kurse in 'Wien' aber nicht in 'Paris'?
- 40) Welche Kurse (Bezeichnung) fanden in 'Wien' und in 'Paris' statt?
Welche Kurse (Bezeichnung) fanden in 'Wien' aber nicht in 'Paris' statt?
- 41) Welche Personen (PNr) haben mindestens zwei Kursveranstaltungen besucht?
- 42) Alle Kursveranstaltungen (KNr, KNrLfd), die keiner besucht
- 43) Pro Kursveranstaltung den tatsächlichen Preis anzeigen: Normalerweise der Kurs-Preis, wenn an einem Wochenende (Sa oder So enthalten), dann 10% (einmal / pro Wochenend-Tag) teurer
- 44) Alle Kursveranstaltungen (KNr, KNrLfd) mit vorhandenen, belegten und freien Plätzen
(nur Kursveranstaltungen, bei denen noch Plätze frei sind)
- 45) Für wieviele Kurse gibt es noch keinen geeigneten Referenten?
Für wieviele Kursveranstaltungen gibt es noch keinen geeigneten Referenten?
- 46) Welche Referenten halten keine Kursveranstaltung?
- 47) Welche Personen besuchen keinen Kurs?
- 48) Alle Kurse (nach Bezeichnung geordnet) mit jeweils allen Kursen, die unmittelbare Voraussetzung sind
(Bezeichnung), anzeigen.
- 49) Zu welchen Kursen, die länger als einen Tag dauern, gibt es keine Kursveranstaltungen, die in 'Wien' stattfinden?
- 50) Welche Personen (PNr, FName) haben alle Kurse besucht?
- 51) Welche Personen feiern während eines Kursbesuches / Kursreferates Geburtstag?
- 52) Für welche Kurse (KNr, Bezeichnung) sind ausschließlich österreichische Referenten geeignet?

- 53) Alle Personen (PNr, FName), die einen Kurs in 'Wien' besucht und gehalten haben
- 54) In welchen Orten wurden alle Kurse abgehalten?
- 55) Welche Kurse (Bezeichnung) kosten weniger als die Hälfte des teuersten Kurses?
- 56) Wie viele Tage dauern und was kosten die Kurse im Durchschnitt?
- 57) Wie viele Tage dauert die längste und die kürzeste Kursveranstaltung?
- 58) Für jeden Referenten, der mindestens eine Kursveranstaltung gehalten hat, geben Sie an:
Nummer des Referenten, Anzahl der Kursveranstaltungen (nach Anzahl absteigend sortiert)
- 59) Für jeden Referenten, der mindestens zwei Kursveranstaltungen gehalten hat, geben Sie an:
Familien- und Vorname des Referenten, Anzahl der Kursveranstaltungen (nach Name sortiert)
- 60) Für jede Person, die mindestens eine Kursveranstaltung besucht hat, geben Sie an:
Familien- und Vorname der Person, Summe der besuchten Kurstage
(nach Summe der Kurstage absteigend sortiert)
- 61) Wie heißt (Familien- und Vorname) der älteste Referent?
- 62) Wieviele Kursveranstaltungen haben die Personen aus Österreich, die keine Referenten sind, jeweils besucht?
(Personennummer, Familienname, Anzahl Kursveranstaltungen)
- 63) Welche Beträge haben die einzelnen Kursteilnehmer in Summe schon bezahlt?
(Familien- und Vorname, Summe der Zahlungen - danach sortiert)
Nur Personen ausgeben, die in Summe schon mehr als 2000,- bezahlt haben.
- 64) Geben Sie die Familiennamen der Personen aus, die für mehr als zwei Kurse geeignet sind
- 65) Alle Kurse (KNr, Bezeichnung) samt Anzahl der geeigneten Referenten für diesen Kurs und Anzahl der Kursveranstaltungen zu diesem Kurs
- 66) Welche Referenten (PNr, FName) haben Kursveranstaltungen gehalten, die höchstens einen vorausgesetzten Kurs haben?
- 67) Welche Personen haben eine Kursveranstaltung besucht (Daten der Person und der Kursveranstaltung) zu der sie noch nicht alle (unmittelbar) vorausgesetzten Kurse (terminlich vorher) besucht haben?
- 68) Welche Kurse (Bezeichnungen) setzt der Kurs 'Komposition' alle voraus? (Stored Routine)
- 69) Fügen Sie in geeignet die Person 101 für alle Kurse ein
- 70) Alle Personen, die den Kurs 1 noch nicht besucht haben, sollen in besucht für die Kursveranstaltung KNr 1 und KNrLfd 3 eingefügt werden
- 71) Alle ausständigen Bezahlungen der Kursbesuche sind mit aktuellem Tagesdatum zu begleichen
- 72) Erhöhen Sie die Preise aller Kurse um 100,-, bei denen es höchstens 2 Veranstaltungen gibt.
- 73) Die Platzanzahl aller Kursveranstaltungen, die voll ausgebucht sind, soll verdoppelt werden
- 74) Erhöhen Sie die Kursdauer bei allen Kursen, die länger als 2 Tage dauern, um 1 Tag und verlängern Sie auch die entsprechenden Kursveranstaltungen durch Änderung des bis-Datums.
- 75) Löschen Sie möglichst alle Personen, soweit dies unter Einhaltung der referentiellen Integrität möglich ist.
- 76) Löschen Sie alle Kursveranstaltungen, für die es keine Besuche gibt und die mehr als 2 Kurse als Voraussetzung haben

9 Stored Routines und Triggers

9.1 Stored Routines

- Stored Routine (Schema Routine, Persistent Stored Module, Stored Commands)
 - Programm wird in der Datenbank (am Server) abgelegt, es führt (aufwendige) Auswertungen oder Änderungen in der Datenbank durch
 - Prozeduraler Code (SPL=Stored Procedure Language) unter direkter Verwendung von SQL
 - Eingabe: Parameter
 - Rückgabe: skalarer Wert, Ausgabe-Parameter, eine oder mehrere Tabellen
- Procedure (Stored Procedure, STP) oder Function (User-defined Function, UDF)
 - Function: Verwendung in Ausdrücken innerhalb von SQL-Anweisungen
 - Procedure: Aufruf vom Anwendungsprogramm (Client), statt einer SQL-Anweisung
- Vorteile
 - Programm läuft (eventuell sogar compiliert) am Datenbankrechner, mit lokalem Zugriff auf die Daten, ab
 - Netzwerkverkehr zwischen Anwendungsprogramm (Client) und Datenbank (Server) wird minimiert
- Sprache für die Routinen
 - SQL:1999-Norm: SQL/PSM (Persistent Stored Modules)
 - MS SQLServer: Transact-SQL
 - Oracle: PL/SQL (Procedural Language)
 - PostgreSQL: PL/pgSQL (Procedural Language/PostgreSQL)
 - etc.
- Eine Routine kann dem Server auch direkt zur Ausführung geschickt werden (vorwiegend zu Testzwecken)
- Übung 1)


```
del_l (lnr) returning int;
```

 Zeile aus L löschen; dabei eventuell vorher entsprechende Zeilen aus LT löschen. Zurückgeben wieviele Zeilen aus LT gelöscht werden mussten.
- Übung 2)


```
clear_lt (m) returning int;
```

 Solange die Summe der Mengen in LT größer als m ist, die Lieferung mit der jeweils niedrigsten Menge löschen. Zurückgeben, wieviele Lieferungen gelöscht wurden. (Keine rekursive Lösung einsetzen)
- Übung 3)

Der gewissenhafte Angestellte Emil Emsig verwendet eine Tabelle ARBEIT (DATUM, STUNDEN), in die er pro Arbeitstag (DATUM) seine Arbeitszeit (STUNDEN) einträgt. Folgende Auswertungen interessieren ihn und sind mit Stored Routines (Procedures oder Functions) zu realisieren:

- a) Wieviele Tage hat er frei gehabt?
- b) An welchen Tagen hat er frei gehabt? (1 Spalte: Datum)
- c) In welchen Zeiträumen hat er frei gehabt? (2 Spalten: vonDatum, bisDatum)
- d) In welchen Zeiträumen hat er durchgehend gearbeitet? (2 Spalten: vonDatum, bisDatum)
- e) Wieviele Tage hat er maximal durchgehend gearbeitet?
- f) Wie oft hat er mindestens eine Woche frei gehabt?
- g) Wie oft hat er mindestens n Wochen frei gehabt? (n = Parameter)
- h) In welchen Monaten hat er frei gehabt? (1 Spalte: Monatsnummer oder Monatsbezeichnung)
- i) An welchen Tagen im Betrachtungsintervall von, bis hat er frei gehabt?
(von, bis = Parameter / 1 Spalte: Datum)

Variante: Tabelle ohne Primärschlüssel (mehrere Zeilen für denselben Tag)

- Übung 4): Lagerverwaltung

- Datenmodell
 - Artikel (ANr, Bezeichnung)
 - Lager (LNr, Ort, StueckKap)
 - Lieferung (LNr > Lager, LfndNr, ANr > Artikel, Datum, Stueck)
- Stored Procedure: Anlieferung (ANr, Datum, Stueck)
 - Auf Lager so aufteilen, dass keine Kapazitätsüberschreitungen auftreten
 - Rückgabe: Tabelle mit LNr und Stueck. Im Fehlerfall leere Tabelle
- Stored Procedure: Entnahme (ANr, Stueck)
 - Entnahmen werden (einfachheitshalber, aber realitätsfremd) nicht protokolliert, sondern nur die Stück entsprechend reduziert oder die Zeile gelöscht
 - Auf Lager so aufteilen, dass nach Alter (der Lieferung) der Artikel entnommen wird
 - Rückgabe: Tabelle mit LNr und Stueck. Im Fehlerfall leere Tabelle
- Stored Procedure: LagerLoeschen (LNr)
 - Lager-Zeile und alle zugehörigen Lieferungs-Zeilen löschen
- Stored Procedure Bestand
 - Rückgabe: Bestandstabelle mit folgendem Aussehen

| Bezeichnung | Ort | Datum | Stueck |
|-------------|---------|------------|--------|
| ----- | ----- | ----- | ----- |
| Alumat | Halle A | 13.11.2005 | 100 |
| | | 12.12.2005 | 50 |
| | Platz 1 | 29.11.2005 | 200 |
| *** Summe | | | 350 |
| Eltex | Halle B | 30.12.2005 | 150 |
| | | 13.01.2006 | 70 |
| | | 16.02.2006 | 320 |
| *** Summe | | | 540 |

- Innerhalb von Bezeichnung und Ort nach Lieferungs-Datum aufsteigend, gleiche Tage sind möglich
- Zeilennummern der Rückgabetable
 - mit Zähler nummerieren
 - Identity verwenden
- Übung 5):

Es sind Formulare zu erstellen aus denen die in Übung 4) erstellten Stored Procedures aufgerufen werden

 - Formular Anlieferung und Entnahme:
 - Artikel aus Kombinationsfeld auswählen
 - Datum in Textfeld eingeben (nur bei Anlieferung)
 - Stück in Textfeld eingeben
 - Befehlsschaltflächen: Anlieferung und Entnahme
 - Aufteilung auf die Lager (Rückgabe) in Listefeld darstellen
 - Formular Bestand: Bestandstabelle in Listefeld darstellen
 - Formular Lager löschen:
 - Lager aus Kombinationsfeld auswählen
 - Befehlsschaltfläche: Lager löschen
- Übung 6) ORACLE-Demodatenbank:

Nummern und Namen / Anzahl der kompletten Vorgesetzten-Hierarchie des Mitarbeiters mit der Nummer 7876 / eines Mitarbeiter (=Parameter).
- Übung 7) ORACLE-Demodatenbank:

Welche / wieviele Mitarbeiter sind dem Mitarbeiter mit der Nummer 7876 / einem Mitarbeiter (=Parameter) unterstellt (direkt und indirekt)?
- Übung 8) SCHULUNGSFIRMA:

Welche (Bezeichnungen) / wieviele Kurse setzt der Kurs 'Komposition' / ein Kurs (=Parameter) alle voraus? (direkt und indirekt)

9.2 Triggers

- Trigger: Stored Procedure, die nicht explizit (vom Client) aufgerufen wird, sondern beim Eintreten von bestimmten Ereignissen
- Arten:
 - DML-Trigger (Ereignisse: INSERT, UPDATE, DELETE)
 - DDL-Trigger (Ereignisse: CREATE, ALTER, DROP)
 - Database-Events (Ereignisse: Logon / Logoff, Startup / Shutdown, etc.)
- Im Folgenden werden die, am häufigsten gebrauchten, DML-Trigger behandelt
- Ereignisse: INSERT in einer Tabelle, UPDATE in einer Tabelle, DELETE in einer Tabelle - Triggerauslösende Anweisung (Triggering Statement)
- Zeitpunkt der Durchführung der Trigger-Aktionen (Trigger Actions, Trigger Statements)
 - BEFORE-Trigger: Trigger-Aktionen werden durchgeführt bevor die Änderungen der triggerauslösenden Anweisung in der Tabelle ausgeführt werden
 - AFTER-Trigger: Trigger-Aktionen werden durchgeführt nachdem die Änderungen der triggerauslösenden Anweisung in der Tabelle ausgeführt wurden
 - INSTEAD OF-Trigger: Trigger-Aktionen werden statt der triggerauslösenden Anweisung durchgeführt, ist besonders bei Views sinnvoll (sofern man Änderungen in Views zulässt)
- Triggerauslösende Anweisungen können mehrere Zeilen betreffen:
 - Row-Level-Trigger: Trigger-Aktionen werden pro Zeile, die von der triggerauslösenden Anweisung betroffen ist, ausgeführt
 - Statement-Level-Trigger: Trigger-Aktionen werden pro triggerauslösender Anweisung ausgeführt
- In den Trigger-Aktionen muss auf die Datenstände von vor und nach Durchführung der triggerauslösenden Anweisung zugegriffen werden können.
Mögliche Mechanismen: Referenznamen für die Zeilen mit dem alten und neuen Inhalt, inserted- und deleted-Pseudotabellen
- Nested Triggers: Durch die Trigger-Aktionen werden weitere Trigger aufgerufen
- Recursive Triggers (direkt, indirekt): In den Trigger-Aktionen kommt der Typ der triggerauslösenden Anweisung direkt oder indirekt wieder vor
- Anwendungsfälle
 - Überwachung von Geschäftsregeln - Business Rules
Diverse Integritätsbedingungen, die deklarativ (im CREATE TABLE) nicht definiert werden können
Beispiele:
 - vom ERD: 1:1-Beziehungstypen, fixe max-Werte, Disjointness Constraints
 - Bedingungen, die mehrere Zeilen einer Tabelle betreffen (z.B. keine Überschneidung von Intervallen)
 - Bedingungen, die mehrere Tabellen betreffen
(z.B. Fahrer eines Autos muss für entsprechenden Autotyp geschult sein, Lagerbestände dürfen Lagerkapazität nicht überschreiten)
 - Einschränkungen bei Änderungen alter Wert auf neuer Wert (z.B. Familienstand)
 - Struktur von Spaltenwerten, soweit nicht als DOMAIN / UDT definierbar (z.B. Autokennzeichen)
 - Aktualisieren redundanter Daten - Derived Attributes
 - Mitschreiben von Änderungsprotokollen - Audit Trails
 - Implementierung komplexer Sicherheitsmechanismen
Beispiel: Personaldaten dürfen nur zu bestimmten Zeiten geändert werden
- Das Datenbanksystem kann komplett enkapsuliert werden, wenn sämtliche Datenmanipulationen und Datenabfragen über Stored Procedures und Triggers geschehen.
- Richtung Objektorientierte Datenbanken und Aktive Datenbanken: Stored Procedure entspricht in etwa einer Methode, Auslöser eines Trigger entspricht in etwa einem Ereignis.

- Übung 1)
Tabelle LT um Spalte Wert (= Menge * Preis) erweitern. Durch Trigger sicherstellen, dass dieser (redundante) Wert bei diversen Änderungen 'richtig' bleibt. Welche Trigger werden benötigt?
- Übung 2)
Tabelle L um Spalte Gesamtmenge erweitern (= Summe aller Mengen aus LT dieses Lieferanten). Durch Trigger sicherstellen, dass diese (redundante) Gesamtmenge bei diversen Änderungen 'richtig' bleibt. Welche Trigger werden benötigt?
- Übung 3)
Tabelle L um Spalte MaxGesamtmenge erweitern. Die Summe aller Mengen aus LT dieses Lieferanten darf die Gesamtmenge nicht überschreiten. Durch Trigger sicherstellen, dass diese Einschränkung eingehalten wird. Welche Trigger werden benötigt?
- Übung 4)
 - Datenmodell
Artikel (ANr, Bezeichnung, Preis, PreisZeit)
ArtikelPreis (ANr > Artikel, PreisZeit, Preis)
Lager (LNr, Ort, StueckKap, StueckIst)
Geeignet (LNr > Lager, ANr > Artikel)
Lieferung (LNr > Lager, LfndNr, ANr > Artikel, Datum, Stueck)
 - Keine Änderung von Primärschlüsselwerten
 - Business Rules
 - Preis darf nie erniedrigt werden
 - Lieferung von Artikel nur in geeignete Lager
 - Kein Artikel darf in mehr als 2 Lagern auftreten
 - Lagerkapazität darf nicht überschritten werden
 - Derived Attributes
 - StueckIst in Lager
 - Audit Trail
 - PreisZeit in Artikel und Tabelle ArtikelPreis
(beim Löschen von Artikel: zusätzlich Löschezitpunkt mit Preis gleich NULL in Tabelle ArtikelPreis eintragen)
 - Stored Function: Preis (ANr, Zeitpunkt) returns decimal

10 ESQL (Embedded SQL)

10.1 Grundlagen

- Verwendung eines relationalen Datenbanksystems mittels SQL-Anweisungen in einer 3GL-Host-Sprache
- Grundproblem (impedance mismatch): 3GL-Sprache arbeitet satzorientiert, SQL arbeitet mengenorientiert (tabellenorientiert)
- Die meisten relationalen Datenbanksysteme stellen ESQL-Pakete für die gängigsten Hochsprachen zur Verfügung.
- Im Folgenden wird ESQL beispielhaft für die Host-Sprache C mit Informix als Datenbanksystem beschrieben
- Die SQL-Anweisungen können in normaler Syntax im C-Quellprogramm verwendet werden, müssen aber jeweils
 - mit \$ (Anfangs-Token) eingeleitet und
 - mit ; (Terminations-Token) abgeschlossen werden.
- SQL-Anweisungen können sich daher auch über mehrere Zeilen erstrecken.
- Host-Variable: C-Variable, die auch in ESQL-Anweisungen als Parameter verwendet wird. Für sie gilt:
 - Die Deklaration der Host-Variablen muss mit \$ eingeleitet werden.
 - Bei der Verwendung in ESQL-Anweisungen muss dem Namen der Host-Variablen ein \$ vorangesetzt werden.
 - In einer normalen C-Anweisung wird die Host-Variable ohne diese Kennzeichnung verwendet.
- Ein ESQL-Programm muss einem ESQL-Precompiler übergeben werden, der ein C-Programm erzeugt (die ESQL-Anweisungen werden im Wesentlichen in Unterprogrammaufrufe übersetzt).
 ESQL-Precompiler+C-Compiler+Linker aufrufen:
 esql file.ec -o file
 - ESQL-Programm muss das Suffix '.ec' haben
 - Erzeugtes Programm hat denselben Namen, jedoch das Suffix '.c'
 - Nur Precompiler-Lauf mit Schalter -e
- Include-Anweisungen für den ESQL-Precompiler:
 \$include file;
 - Die Datei 'file.h' wird an dieser Stelle eingefügt.
 - #include wird erst vom C-Compiler bearbeitet.
 - 'sqlca.h' soll immer eingefügt werden, um die erfolgreiche Ausführung einer ESQL-Anweisung abprüfen zu können (sqlca.sqlcode).
 - Hinweis: Bei Verwendung von #define muss zunächst der C-Compiler aufgerufen werden (cc -P file.c), das Ergebnis (file.i) umbenannt werden (file.ec) und dieses dem ESQL-Precompiler übergeben werden.
- Datenbank eröffnen:
 \$database name;
- SQL-Datentyp: C-Datentyp:

| | |
|------------|--|
| CHAR(n) | char field[n+1]; oder char *field; |
| SMALLINT | short field; |
| INTEGER | int field; |
| SERIAL | long field; |
| SMALLFLOAT | float field; |
| FLOAT | double field; |
| DATE | long field; /* Tage zum 31.12.1899 */ |
| DECIMAL | dec_t field; oder struct decimal field (#include <decimal.h> ist notwendig); |

 - Für Konvertierungen und Manipulationen der Typen DATE, DECIMAL stehen eine Vielzahl von Funktionen zur Verfügung.
 - Entsprechen einander die Typen der SQL-Spalten und der Host-Variablen nicht, versucht ESQL Typumwandlungen sinnvoll vorzunehmen (z.B. DECIMAL in int, DECIMAL in float).

- Beispiele:

```

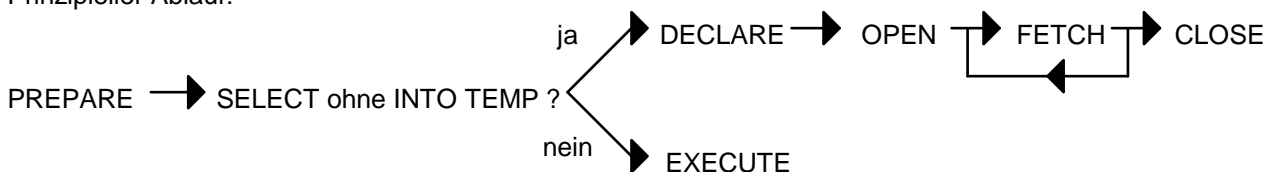
/* ----- Beispiel 1) ----- */
#include sqlca;
main()
{
    $char tnr [3];
    $char tname [9];
    $database t;
    printf ("Teilenummer eingeben: ");
    gets (tnr);
    while (*tnr)
    {
        $select tname into $tname from t where tnr=$tnr;
        if (sqlca.sqlcode == SQLNOTFOUND)
            printf ("Teil nicht gefunden!\n");
        else
            puts(tname);
        printf ("Teilenummer eingeben: ");
        gets (tnr);
    }
}

/* ----- Beispiel 2) ----- */
#include sqlca;
main()
{
    $char tnr [3];
    $char tname [9];
    $int preis;
    $char stadt [7];
    $database t;
    $declare zeiger cursor for select tnr,tname,preis,stadt from t order by tname;
    $open zeiger;
    printf ("Nr T-Name   Pr Stadt\n");
    $fetch zeiger into $tnr,$tname,$preis,$stadt;
    while (sqlca.sqlcode != SQLNOTFOUND)
    {
        printf ("%s %s %d %s\n",tnr,tname,preis,stadt);
        $fetch zeiger into $tnr,$tname,$preis,$stadt;
    }
    $close zeiger;
}

```

10.2 Dynamisches SQL

- Es können SQL-Anweisungen auch erst zur Laufzeit erstellt werden (Dynamic Management).
- In einer Zeichenkette kann mit normalen Sprachmitteln eine SQL-Anweisung aufgebaut und danach ausgeführt werden.
- Folgende Anweisungen dürfen nicht in dieser Zeichenkette enthalten sein: DECLARE, OPEN, FETCH, CLOSE, PREPARE, DESCRIBE, EXECUTE
- Prinzipieller Ablauf:



Nach PREPARE kann mit DESCRIBE festgestellt werden um welche Art von Anweisung es sich handelt.

- Anweisung vorübersetzen:

```
PREPARE statement FROM string
```

- statement: Die in string enthaltene Anweisung wird vorübersetzt und bekommt diesen Namen (Anweisungsbezeichner), unter dem sie dann angesprochen werden kann
- string: Zeichenkettenliteral oder Host-Variable vom Typ Zeichenkette; die Zeichenkette darf keine Host-Variablen enthalten; Anfangs- und Terminations-Token sind nicht notwendig.
- sqlca.sqlcode: ==0, dann ok.
 <0, dann Fehler (Wert ist Fehlerschlüssel)

- Anweisung analysieren:

```
DESCRIBE statement INTO sqlda-ptr
```

- statement: Name, der zuvor im PREPARE vergeben wurde
- sqlda-ptr: Zeiger auf eine 'sqlda'-Struktur (\$include sqlda); ist nicht als Host-Variable anzugeben. Bei SELECT-Anweisungen ohne INTO TEMP kann damit die Struktur der Ergebnistabelle ermittelt werden
- sqlca.sqlcode: ==0, dann SELECT-Anweisung ohne INTO TEMP
 !=0, dann Art der Anweisung (\$include sqltype)

- Anweisung ausführen (keine SELECT-Anweisung ohne INTO TEMP):

```
EXECUTE statement [ USING { parameter-commalist | DESCRIPTOR sqlda-ptr } ]
```

- statement: Name, der zuvor im PREPARE vergeben wurde

- Anweisung mit Cursor ausführen (SELECT-Anweisung ohne INTO TEMP):

```
DECLARE cursor CURSOR FOR statement
```

- statement: Name, der zuvor im PREPARE vergeben wurde

```
OPEN cursor [ USING { parameter-commalist | DESCRIPTOR sqlda-ptr } ]
```

```
FETCH cursor { INTO parameter-commalist | USING DESCRIPTOR sqlda-ptr }
```

```
CLOSE cursor
```

- Verwendung von Host-Variablen in dynamischen SQL-Anweisungen:

- Für jede Host-Variable wird im String, der die SQL-Anweisung enthält, ein ? als Platzhalter verwendet.
- Bei EXECUTE oder OPEN werden nach USING die zu verwendenden Host-Variablen angegeben. Diese ersetzen in der entsprechenden Reihenfolge die ?.

- 'sqlda'-Struktur:

```
struct sqlda
{
    short sqld;          /* Anzahl Elemente der Projektion */
    struct sqlvar_struct *sqlvar; /* Zeiger auf einen Vektor von Strukturen, in dem
                                jedes Element der Projektion beschrieben ist */
}

struct sqlvar_struct
{
    short sqltype;      /* Typ des Elements ($include sqltypes) */
    short sqllen;       /* Länge des Elements in Bytes */
    char *sqldata;      /* Zeiger zum Inhalt des Elements */
                    /* Speicherplatz muss erst angelegt, und
                    Zeiger gesetzt werden! */
    short *sqlind;      /* Zeiger zur Indikator-Variablen */
    char *sqlname;      /* Zeiger zum Namen des Elements */
    char *sqlformat;    /* nicht verwendet */
}
```

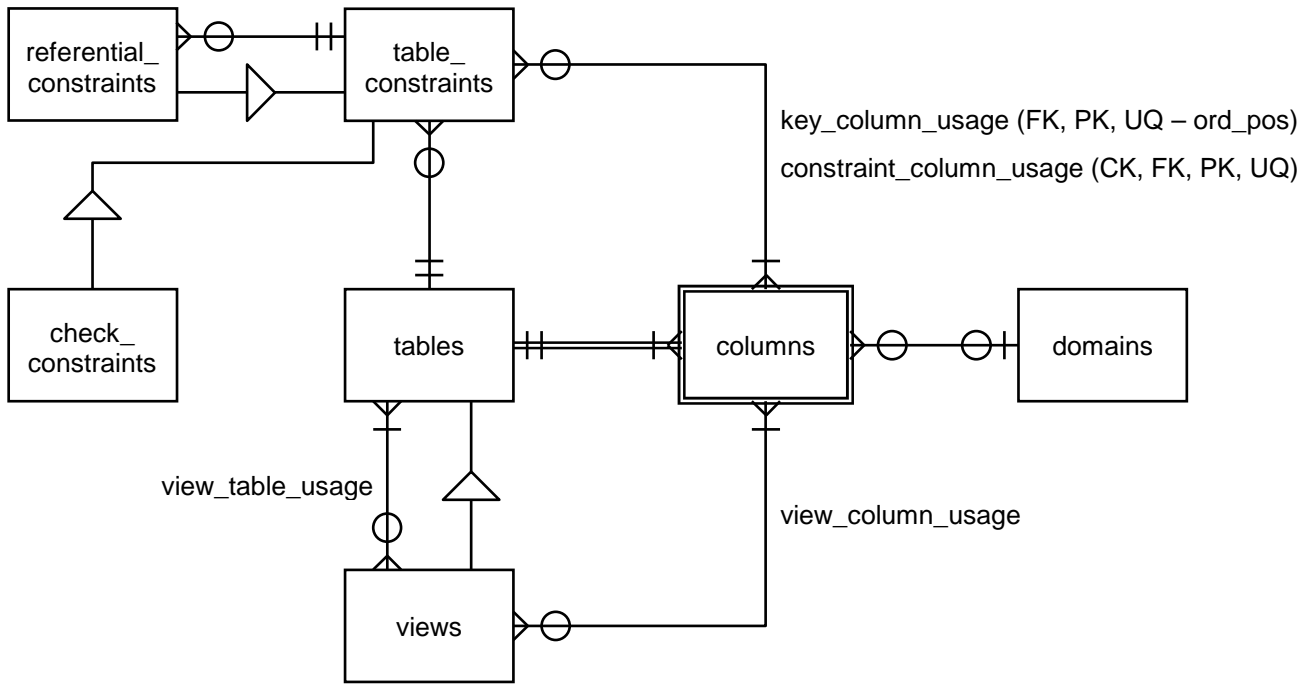
- Beispiel:

```
/* ----- Beispiel 3) ----- */
#include sqlca;
#include sqlda;
main()
{
    struct sqlda *befehl_desc; /* Zeiger auf sqlda-Struktur */
    $char befehl [80];
    $database t;
    printf("Befehl eingeben: ");
    gets(befehl);
    while (strcmp(befehl,"end") != 0)
    {
        $prepare anweis_bez from $befehl;
        if (sqlca.sqlcode != 0)
            printf("Fehlercode nach prepare: %d\n",sqlca.sqlcode);
        else
        {
            $describe anweis_bez into befehl_desc;
            if (sqlca.sqlcode == 0)
                printf ("Keine SELECT-Anweisung ohne INTO TEMP erlaubt!\n");
            else
            {
                printf ("Art der Anweisung: %d\n",sqlca.sqlcode);
                /* Codierung der Anweisungen in sqlstype.h */
                $execute anweis_bez;
                if (sqlca.sqlcode != 0)
                    printf("Fehlercode nach execute: %d\n",sqlca.sqlcode);
            }
        }
        printf("Befehl eingeben: ");
        gets(befehl);
    }
}
```

11 Systemtabellen und Information Schema Views

- Beschreibung der Objekte einer (relationalen) Datenbank (Tabellen, Spalten, Indizes, Primärschlüssel, Fremdschlüssel etc.) wird in Tabellen - Systemtabellen - der Datenbank abgelegt. Bezeichnet auch als: Systemkatalog, Data Dictionary, Datenbank-Metadaten, Daten über Daten, etc.
- Die Systemtabellen können herangezogen werden um (mit Hilfe normaler SELECT-Anweisungen) Informationen über die Struktur der Datenbank zu erhalten
- Eine Veränderung der Systemtabellen mit normalen SQL-Datenmanipulationsanweisungen (INSERT, UPDATE, DELETE) ist nicht möglich.
Die Anweisungen CREATE, ALTER, DROP, etc. verändern (implizit) die Systemtabellen.
- Die Systemtabellen weisen in den verschiedenen Datenbanksystemen unterschiedliche Strukturen auf (Definition Schema)
- Teilweise ist die Information in den Systemtabellen redundant abgespeichert
(Beispiel für sinnvollen Einsatz, Verhältniss Verwendung zu Wartung)
- Mit den Information Schema Views (Informationsschemasichten) der SQL-Norm (ab SQL-92) ist ein einheitlicher Zugriff auf die Systemtabellen möglich (Wrapper around the System Tables)
- Im Information Schema sind auch die Tables / Views des Information Schemas enthalten, es ist daher selbstbeschreibend
- Eigenes Schema: INFORMATION_SCHEMA
- Aufbau der (gebräuchlichsten) Information Schema Views
 - TABLES: alle Tabellen (inklusive Views), TABLE_TYPE = BASE TABLE | VIEW
 - COLUMNS: alle Spalten (inklusive View-Spalten)
 - VIEWS: alle Views, mit Definitions-Text
 - VIEW_TABLE_USAGE: welche Tabellen kommen in den Views vor
 - VIEW_COLUMN_USAGE: welche Spalten kommen in den Views vor
 - TABLE_CONSTRAINTS: alle Constraints, mit den Tabellen zu denen sie gehören,
CONSTRAINT_TYPE = UNIQUE | PRIMARY KEY | FOREIGN KEY | CHECK
 - KEY_COLUMN_USAGE: aus welchen Spalten bestehen UQ, PK, FK, mit Ordinal-Position
 - CONSTRAINT_COLUMN_USAGE: aus welche Spalten bestehen UQ, PK, FK, CK, keine Ordinal-Position
(für UQ, PK, FK siehe auch KEY_COLUMN_USAGE)
 - REFERENTIAL_CONSTRAINTS: alle FK-Constraints mit referenzierten PK-Constraints
 - CHECK_CONSTRAINTS: alle Check Constraints, mit Definitions-Text

- Struktur der Information Schema Views:



- Übung 1):

- Benutzerdefinierten Datentyp Prozent mit Basisdatentyp Integer anlegen, Constraint für Bereich zwischen 0 und 100 zuordnen, Defaultwert 0 zuordnen
- Definition der Datenbank LT erweitern
 - Tabellen löschen, nur wenn Sie tatsächlich vorhanden sind
 - LName in L muss eindeutig sein
 - Rabatt mit Datentyp Prozent
 - In L Stadt ohne Zeichen 'x'
 - TName in T darf nicht NULL sein
 - In T Farbe immer ungleich Stadt
 - In T Index über Preis (absteigend)
 - In T Index über Farbe und TName
 - Standardwert 1 für Menge in LT, Menge größer 0, nicht NULL
 - Definieren Sie eine View LTX, die wie LT aufgebaut ist und zusätzlich LNAME und TNAME darstellt
- Selbstdefinierte (systematisch aufgebaute, aussagekräftige) Constraint-Namen für alle Constraints verwenden
- Anweisungen erstellen, um alle Constraints, Rules und Defaults zu testen
- Anweisungen erstellen, um alle angelegten Objekte wieder zu löschen

- Übung 2):

- a) Welche Spalten enthält die Tabelle L?
- b) Welche Tabellen / Views / Tabellen oder Views enthalten eine Spalte STADT?
- c) Wieviele Spalten enthält die Tabelle LT?
- d) Wieviele Spalten haben die einzelnen Tabellen / Views?
- e) Wieviele Fremdschlüssel sind in der Tabelle LT enthalten?
- f) Welche Tabellen haben einen zusammengesetzten Primärschlüssel?
- g) Welche Tabellen haben einen zusammengesetzten Fremdschlüssel?
- h) Welche Tabellen enthalten keinen Unique-Constraint?
- i) Welche Tabellen kommen in der View LTX vor?
- j) In welchen Views kommt die Tabelle L vor?
- k) In welchen Views kommen die Tabellen L und T vor?
- l) In welchen / wievielen Tabellen kommt eine Spalte mit dem Datentyp DECIMAL vor?
- m) Welche Spalten (Tabellen- und Spaltenname) dürfen nicht NULL sein?
- n) In welchen Check-Constraints kommt die Spalte STADT vor?
- o) Welche Tabellen werden von einem Fremdschlüssel referenziert?
- p) STP hlp_tab (tablename): Ausgabe Spaltennamen und zugehörige Datentypen der Tabelle
- q) STP drp_views: Löschen Sie alle Views (Dynamisches SQL)
- r) STP drp_tabs: Löschen Sie alle Tabellen (vorher alle Fremdschlüssel-Constraints)

12 Transaktionen

- Problematik: Logisch zusammengehörige Änderungen von Daten
- Beispiele:
 - Buchungen: Eine Buchung wird unter Angabe von Datum, Betrag, Konto und Gegenkonto gemacht. Es muss bei beiden Konten ein Buchungssatz aufgenommen werden.
 - Personalverwaltung: Die Bezeichnung einer Abteilung wird geändert. In allen Personalsätzen, in denen diese Abteilung vorkommt, muss die Abteilungsbezeichnung geändert werden.
 - Auftragsverwaltung: Eine Auftragsposition wird storniert. Der Auftragspositionssatz muss gelöscht und die Menge der reservierten Einheiten im Artikelsatz vermindert werden.
- Hinweis: Transaktionen sind auch notwendig, wenn keine Redundanzen vorliegen, z.B. Löschen eines Auftrags, der aus einem Kopfsatz und mehreren Positionssätzen besteht.
- Werden nicht alle Änderungen durchgeführt, sind die Daten in einem logisch inkonsistenten Zustand.
- Transaktion (Transaction, Commit Unit, LUW - Logical Unit of Work, Unit of Recovery):
 - Arbeitseinheit (Folge von Anweisungen), die (bezüglich der Auswirkungen auf Dateien) entweder ganz durchgeführt werden muss oder überhaupt nicht durchgeführt werden darf.
 - Prinzip: Alles oder Nichts.
 - Programmabschnitt, in dem die Daten von einem konsistenten Zustand wieder in einen konsistenten Zustand übergeführt werden.
 - Transaktion muss 'atomar' (unteilbar) sein.
- Transaktion wird ordnungsgemäß beendet: Commit
- Transaktion wird nicht ordnungsgemäß beendet: Rollback, Abort, Reset, Backout
- Im Anwendungsprogramm werden Beginn und Ende von Transaktionen
 - entweder explizit angegeben (z.B. BEGIN / COMMIT TRANSACTION, COMMIT WORK für Transaktionsende)
 - oder implizit angenommen (z.B. Prozeduren, bestimmte Blöcke)
- Länge von Transaktionen:
 - so kurz als möglich:
 - geringer Systemoverhead
 - wenige Funktionen sind im Fehlerfall zu wiederholen
 - kurze Sperreinheiten für andere Anwender (siehe 'Synchronisation paralleler Transaktionen')
 - so lang als notwendig:
 - konsistenter Datenzustand muss gewährleistet sein
- Üblicherweise kann immer höchstens eine Transaktion (pro Benutzer / Task) aktiv sein, Transaktionen können nicht geschachtelt sein (flache Transaktionen).
Es gibt auch Konzepte mit geschachtelten Transaktionen, sogenannten Subtransaktionen. Damit ist flexiblere Recovery möglich, denn es kann entweder die ganze Transaktion, oder nur eine bestimmte Subtransaktion zurückgenommen werden.
- Synchronisationspunkt (Sync-Point): Grenze zwischen zwei aufeinanderfolgenden Transaktionen. Alle Datenänderungen müssen permanent gemacht (fixiert) werden, gesperrte Daten freigegeben werden (siehe 'Synchronisation paralleler Transaktionen')
- Ein Programm(system), der Transaktionsmonitor (Transaktionsmanager, Transaction Processing Monitor, TP-Monitor), muss dafür sorgen, dass Datenveränderungen von Transaktionen, die nicht ordnungsgemäß beendet wurden, wieder zurückgenommen werden.
- Der Transaktionsmonitor ist entweder eine Komponente des Datenbanksystems oder ein eigenes Programm (z.B. UTM im BS2000 / SIEMENS, CICS / IBM, MS Transaction Server / Microsoft, Tuxedo / BEA)

- Transaktionen müssen nach dem ACID-Prinzip ablaufen
 - Atomicity (Atomarität, Unteilbarkeit):
 - Veränderung der Daten erfolgt vollständig oder gar nicht (alles oder nichts)
 - wird ein Einzelschritt nicht erfolgreich beendet, so wird der gesamte Vorgang zurückgesetzt
 - Consistency (Konsistenz):
 - Daten sind immer in einem logisch konsistenten Zustand
 - Verarbeitung erfolgt von einem konsistenten Zustand zum nächsten
 - Isolation:
 - Datenänderungen innerhalb der Transaktion sind nur ihr selbst bekannt
 - Andere Transaktionen haben gleichzeitig keinen oder nur beschränkten Zugriff auf diese Daten
 - Transaktion darf keine andere, gleichzeitig mit ihr laufende, beeinflussen
 - Problembereich 'Synchronisation paralleler Transaktionen' (Concurrency)
 - Durability (Persistenz, Dauerhaftigkeit):
 - Nach Transaktionsabschluss müssen die Datenänderungen dauerhaft (gegen Fehler jeglicher Art) bestehen bleiben
 - Problembereich 'Recovery'

13 Recovery

- Recovery: Wiederherstellung eines konsistenten Datenzustands nach Fehlersituationen

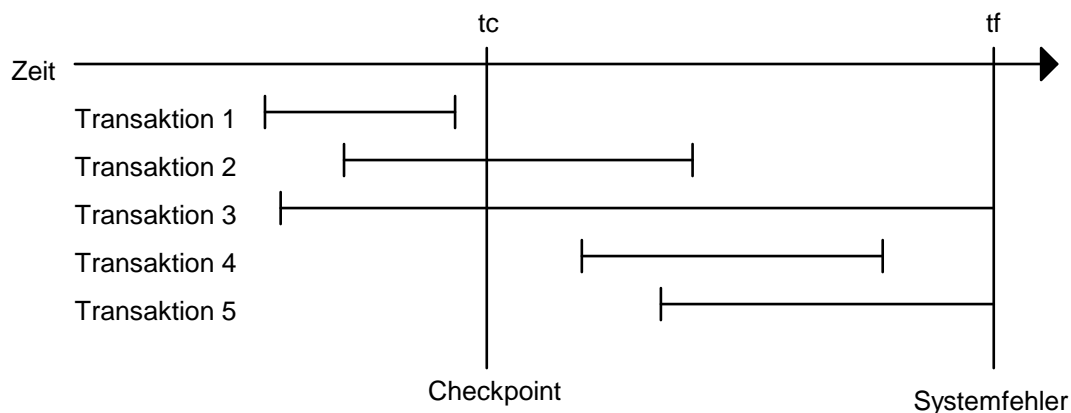
Drei Arten von Fehlersituationen:

- Transaktionsfehler (lokaler Fehler)
 - Eine Transaktion wird nicht ordnungsgemäß beendet
 - Auswirkung: Daten sind in einem inkonsistenten Zustand
 - Beispiele:
 - Laufzeitfehler im Anwenderprogramm (Division durch 0, Variablenüberlauf, Überschreitung der Indexgrenzen einer Tabelle, etc.)
 - Endlosschleife im Anwenderprogramm
 - Abbruch des Anwenderprogramms durch den Systembediener
 - Time-out (Überschreitung einer CPU-Zeitgrenze) des Anwenderprogramms
 - Kein verfügbarer Plattenplatz bei einem Schreibbefehl im Anwenderprogramm
 - Deadlock (siehe 'Synchronisation paralleler Prozesse')
 - Befehl zum Zurücknehmen der Transaktion im Anwenderprogramm selbst (ROLLBACK WORK)
 - Maßnahme: Transaction Recovery, Transaction Rollback, Dynamic Transaction Backout
Alle Änderungen, die die Transaktion bis zum Abbruch gemacht hat, müssen im laufenden Systembetrieb zurückgenommen werden.
- Systemfehler (globaler Fehler, Soft-Crash)
 - Der Transaktionsmonitor wird nicht ordnungsgemäß beendet (damit können in einem Multiuserbetrieb mehrere Transaktionen nicht ordnungsgemäß beendet werden)
 - Auswirkung: Daten sind in einem inkonsistenten Zustand
 - Beispiele:
 - Stromausfall
 - Abbruch des Transaktionsmonitors durch den Systembediener
 - Fehler im Betriebssystem (z.B. Endlosschleife)
 - Verlust von Hauptspeichereinhalten
 - Maßnahme: Crash Recovery
Alle Änderungen der Transaktionen, die zum Zeitpunkt des Systemfehlers aktiv (in flight) waren, müssen beim Systemwiederstart (Restart, Warmstart) zurückgenommen werden.
- Mediumfehler (Speicherfehler, Hard-Crash)
 - Daten sind physikalisch zerstört oder nicht mehr lesbar
 - Auswirkung: Daten sind nicht mehr verfügbar
 - Beispiele:
 - Head-Crash auf Magnetplatte
 - Fehler im Disk-Controller
 - Irrtümliches Löschen von Daten
 - Irrtümliches Formatieren einer Platte
 - Fehler in der Dateiverwaltung des Betriebssystems
 - Katastrophenfälle (Brand, Anschläge, Erdbeben, etc.)
 - Maßnahme: Archive Recovery, Media Recovery, Disaster Recovery
Ein Sicherungsstand (Backup Copy) der Daten wird eingespielt (reloaded, restored) und die Änderungen aller Transaktionen, die seit dem Zeitpunkt der Sicherung beendet wurden, müssen vom System nachvollzogen werden
- Transaction Recovery und Crash Recovery werden auch unter dem Begriff Backward Recovery zusammengefasst
- Archive Recovery wird auch Forward Recovery genannt

- Techniken für Backward Recovery

Zwei mögliche Strategien:

- Logging (Journaling) in Undo-Logs: Vor jeder Veränderung von Daten wird eine Kopie dieser Daten (Before Image) in eine Log-Datei (Undo-Log) eingetragen (WAL = Write Ahead Log - Prinzip). Damit ist für jede Operation (aufnehmen, ändern, löschen) der alte Wert bekannt. Um eine Transaktion rückgängig zu machen, wird das Undo-Log vom Ende beginnend rückwärts gelesen und jeweils der alte Wert wiederhergestellt.
- Verzögertes Update (Deferred Update): Die Änderungen der Daten werden erst geschrieben, wenn die Transaktion beendet ist. Bis dahin wird auf Kopien der eigentlichen Daten gearbeitet. Rollback heißt dann einfach, diese Kopien zu löschen. In diesem Fall enthalten die Daten selbst die Rollback-Information.
- In der Praxis werden im wesentlichen Verfahren mit Log-Dateien verwendet
- Informationen in einer Undo-Logdatei:
Mindestens:
 - Beginn Transaktion
Identifikation der Transaktion
 - Kennzeichen der Operation (aufnehmen, ändern, löschen)
Identifikation der Transaktion
Identifikation der Daten (z.B. Tabellename und Zeilennummer)
Before-Image der veränderten Daten
Bemerkung: Für die Operation Aufnehmen brauchen nicht alle Daten, sondern nur deren Identifikationen protokolliert werden.
 - Ende Transaktion
Identifikation der Transaktion
 Zusätzlich:
 - Benutzeridentifikation
 - Terminal- / Workstationbezeichnung
 - Programmname
 - Datum und Uhrzeit
- Undo-Information braucht nur bis zum erfolgreichen Ende der entsprechenden Transaktion aufgehoben zu werden. Meistens wird das Undo-Log aber fortlaufend geführt und an bestimmten Zeitpunkten als Ganzes gelöscht.
- Bei Transaction Recovery muss das Undo-Log bis zum Beginnkennzeichen der entsprechenden Transaktion nach vorne gelesen werden.
- Bei Crash-Recovery muss das Undo-Log bis zum Anfang durchgelesen werden, es könnten immer noch Before-Images einer nicht beendeten Transaktion gefunden werden.
Um den Abschnitt zu begrenzen, der durchsucht werden muss, werden in gewissen Zeitabständen Sicherungspunkte (Checkpoints) auf das Log-File geschrieben. Dabei werden die Identifikationen aller Transaktionen, die zu diesem Zeitpunkt aktiv sind, abgespeichert.



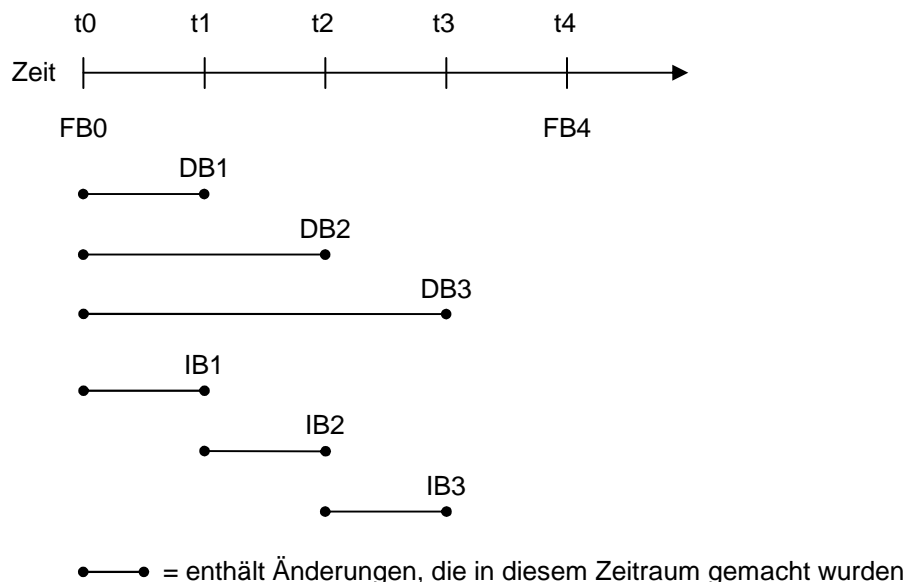
- Logik: Lese das Log-File rückwärts bis zum jüngsten Checkpoint; jede Transaktion, für die keine Endemarke gefunden wurde, wird mittels Before-Image rückgängig gemacht. Vom Checkpoint weiter rückwärts gehend werden die Transaktionen rückgängig gemacht, die im Checkpoint angeführt sind und für die keine Endemarke gefunden wurde.
- An den Sicherungspunkten werden üblicherweise auch alle Puffen auf stabile Speicher (Hintergrundspeicher) geschrieben.

- Techniken für Forward Recovery

- Logging (Journaling) in Redo-Logs: Eine Kopie der veränderten Daten (After Image) wird laufend in eine Log-Datei (Redo-Log) eingetragen. Das Redo-Log muss vom Anfang beginnend vorwärts gelesen und die Änderungen aller beendeten Transaktionen nachvollzogen werden.
- Informationen in einer Redo-Logdatei:
Analog Undo-Logdatei, jedoch mit After-Image für alle veränderten Daten.
Bemerkung: Für die Operation Löschen brauchen nicht alle Daten, sondern nur deren Identifikationen protokolliert werden.
- Redo-Information muss bis zur erfolgreichen Erstellung der nächsten Backup-Copy aufgehoben werden.
- Die Redo-Logs können daher, wenn Backup-Copies aus Gründen des Datenumfanges oder der Verfügbarkeit nur in großen Zeitabständen gemacht werden, ebenfalls sehr umfangreich werden. Mit Dienstprogrammen können ein oder mehrere Redo-Logs komprimiert werden; es werden dabei nur die endgültigen (jüngsten) Änderungsstände abgespeichert (Change Accumulation), damit schnellerer Wiederanlauf.
- Alternative zur Archive Recovery: Gespiegelte Platten (Mirror Disks), Gespiegelte Datenbanken (Mirror Databases, Shadow Databases), RAID-Technologie
Die Daten werden doppelt gehalten und laufend parallel geändert.
Vorteil: schneller Wiederanlauf, weniger Backup-Copies notwendig, hohe Verfügbarkeit ('24 Stunden an 7 Tagen')
Nachteil: große Hardwarekapazität, räumlich ausgelagerte Backups und Logs müssen für Katastrophenfälle trotzdem zusätzlich verwendet werden
- Redo-Logs und Spiegelplatten müssen natürlich auf physisch anderen Disk-Drives als die Basis-Daten gehalten werden.

- Backup-Techniken

- Full Backup (Gesamt-Backup, Voll-Backup) (FB)
- Partial Backup (Teil-Backup)
 - Differential Backup (Differenzielles-Backup) (DB): Alle Änderungen gegenüber dem letzten Full Backup
 - Incremental Backup (Zuwachs-Backup) (IB): Alle Änderungen gegenüber dem letzten Partial Backup



- Vorteile Differential Backup:
 - sicherer (wenn Sicherungsmedien nicht mehr lesbar)
 - kürzere Restorezeiten
- Vorteile Incremental Backup:
 - Weniger Speicherplatzverbrauch
 - kürzere Backupzeiten

- Paralleles Backup (Striped Backup)
Gleichzeitig mit mehreren Sicherungslaufwerken auf mehrere Sicherungsmedien
- Offline Backup (Cold Backup)
- Online Backup (Hot Backup, Dynamic Backup)
Im laufenden Betrieb (Hochverfügbarkeitssysteme, High-Availability-Systems)
Strategie zur Sicherstellung der Konsistenz des gesicherten Datenbestandes:
 - sequentielles Sichern der Zeilen, jede Zeile wird als gesichert gekennzeichnet
 - wenn eine Transaktion eine Änderung machen will, wird das sequentielle Sichern unterbrochen, der alte Stand gesichert, die Änderung durchgeführt, die Zeile als gesichert gekennzeichnet und das sequentielle Sichern wieder fortgesetzt
 - Es wird damit ein konsistenter Stand, wie er am Zeitpunkt des Sicherungsbeginns vorlag, gesichert
- Undo-Log und Redo-Log können auch in einer gemeinsamen Log-Datei (Audit Trail) gespeichert sein. Da das After-Image gleich dem Before-Image der nächsten Änderung ist, können hier Komprimierungen erreicht werden.
- Bei kombinierten System- und Mediumfehlern ist Forward- und Backward-Recovery (in dieser Reihenfolge) durchzuführen.
Bemerkung: Bei gewissen Strategien werden die After-Images erst bei Transaktionsende auf die Log-Datei geschrieben, dann kann Backward-Recovery entfallen.
- Die oben beschriebenen Mechanismen müssen noch genauestens abgestimmt werden mit der meist gepufferten Datenein-/ausgabe zwischen Haupt- und Externspeicher (logisches / physikalisches Schreiben) sowie der Modifikation von Indizes. Außerdem werden die Log-Dateien oft nicht satz-/zeilenweise, sondern block-/seitenweise geführt.

- Übung:

Tabelle P (vor Beginn der Transaktionen)

| Zeilen# | PersNr | PName | Gehalt |
|---------|--------|-------|--------|
| 01 | 27 | Fuchs | 2100 |
| 03 | 15 | Maus | 1500 |
| 05 | 47 | Hase | 800 |
| 06 | 75 | Fisch | 1600 |
| 09 | 32 | Wolf | 1500 |

Transaktionen

| | T17 | T18 |
|----|---|---|
| t1 | BEGIN TRANSACTION | |
| t2 | UPDATE P SET PName='Gans', Gehalt=Gehalt+100 WHERE PName='Hase' | |
| t3 | | BEGIN TRANSACTION |
| t4 | | DELETE FROM P WHERE PersNr=15 |
| t5 | INSERT INTO P VALUES (83,'Adler',1900) | |
| t6 | | UPDATE P SET Gehalt=Gehalt*2 WHERE PName LIKE 'F%' |
| t7 | COMMIT TRANSACTION | |
| t8 | | UPDATE P SET Gehalt=Gehalt+300 WHERE Gehalt<1000 |
| t9 | | COMMIT TRANSACTION |

Undo-Logdatei

| | Aktion | Trans# | Tab | Zeilen# | Before Image |
|----|--------|--------|-----|---------|-------------------|
| 1 | B | 17 | | | |
| 2 | U | 17 | P | 05 | 47 / Hase / 800 |
| 3 | B | 18 | | | |
| 4 | D | 18 | P | 03 | 15 / Maus / 1500 |
| 5 | I | 17 | P | 10 | |
| 6 | U | 18 | P | 01 | 27 / Fuchs / 2100 |
| 7 | U | 18 | P | 06 | 75 / Fisch / 1600 |
| 8 | E | 17 | | | |
| 9 | U | 18 | P | 05 | 47 / Gans / 900 |
| 10 | E | 18 | | | |

Tabelle P (nach Ende der Transaktionen)

| Zeilen# | PersNr | PName | Gehalt |
|---------|--------|-------|--------|
| 01 | 27 | Fuchs | 4200 |
| 05 | 47 | Gans | 1200 |
| 06 | 75 | Fisch | 3200 |
| 09 | 32 | Wolf | 1500 |
| 10 | 83 | Adler | 1900 |

- Was passiert, wenn das UPDATE von t8 zwischen t6 und t7 durchgeführt wird?
- Welchen Inhalt hat für obiges Beispiel die Redo-Logdatei?
- Wieviele Zeilen haben die Undo- und Redo-Logdatei im Verhältnis zueinander?
- Wie sieht die Redo-Logdatei aus, nachdem sie von einem Change-Accumulation Utility bearbeitet wurde?
- Welchen Inhalt hat ein Checkpoint, der zwischen t1 und t2 / t6 und t7 / t7 und t8 / nach t9 geschrieben wurde?
- Zwischen t8 und t9 tritt ein Systemfehler auf, die untere Tabelle soll den Stand nach dem Systemfehler darstellen. Skizzieren Sie P nach erfolgter Recovery.
- Zwischen t6 und t7 tritt ein Systemfehler auf, die untere Tabelle soll den Stand nach dem Systemfehler darstellen. Skizzieren Sie P nach erfolgter Recovery.

14 Concurrency Mehrbenutzerbetrieb

- Synchronisation paralleler Transaktionen

14.1 Problemstellung die Fehler bei unkontrollierten Mehrbenutzerbetrieb

- In einem **Multi-User-Betrieb** (Multi-Tasking-Betrieb) müssen **parallele Zugriffe verschiedener Anwender** (Tasks) auf Daten **synchronisiert** werden (**Concurrency-Control**)

- **Lost Update Problem** (Problem der verlorenen Änderung)

Daten, die soeben geändert wurden werden überschrieben, ohne die Änderungen zu berücksichtigen

| Transaktion A | Zeit | Transaktion B |
|---------------|------|---------------|
| - | | - |
| FETCH R | t1 | - |
| - | | - |
| - | t2 | FETCH R |
| - | | - |
| UPDATE R | t3 | - |
| - | | - |
| - | t4 | UPDATE R |
| | ↓ | |

Änderung der Transaktion A (t3) geht verloren (t4).

- **Inconsistent Analysis Problem** (Incorrect Summary Problem, Problem der inkonsistenten Sicht)

3 Zeilen mit Kontoständen: R1.stand=40, R2.stand=50, R3.stand=30

Transaktion A: Summiert die Kontostände auf

Transaktion B: Bucht 10 Einheiten von R3.stand (-10) auf R1.stand (+10) um

| Transaktion A | Zeit | Transaktion B |
|---------------------------|------|-----------------------|
| - | | - |
| FETCH R1 | t1 | - |
| sum=sum+stand (sum: 40) | | - |
| - | | - |
| FETCH R2 | t2 | - |
| sum=sum+stand (sum: 90) | | - |
| - | | - |
| - | t3 | FETCH R3 |
| - | | stand=stand-10 |
| - | t4 | UPDATE R3 (stand: 20) |
| - | | - |
| - | t5 | FETCH R1 |
| - | | stand=stand+10 |
| - | t6 | UPDATE R1 (stand: 50) |
| - | | - |
| - | t7 | COMMIT |
| - | | - |
| FETCH R3 | t8 | - |
| sum=sum+stand (sum: 110!) | ↓ | - |

Beim Auswerten ändern sich bereits gelesene und nicht gelesene Datensätze

Transaktion A ermittelt eine falsche Summe (t8), richtig wäre 120.

Phantom Read: Beim 2. lese gibt es mehr/weniger Datensätze

None-repeatable read: Beim 2. Lesen hat sich der Inhalt von den Datensätzen geändert

Hinweis: Die folgende Situation stellt kein Inconsistent Analysis Problem dar: Transaktion B liest zwar den alten Wert, der stammt aber von einem konsistenten Datenzustand.

| Transaktion A | Zeit | Transaktion B |
|----------------|------|---------------|
| - | | - |
| FETCH R | t1 | - |
| stand=stand+10 | | - |
| - | | - |
| - | t2 | FETCH R |
| - | | - |
| UPDATE R | t3 | - |
| | ↓ | |

- **Uncommitted Dependency Problem** (Temporary Update Problem, Problem der temporären Änderung)

| Transaktion A | Zeit | Transaktion B |
|---------------|------|---------------|
| - | | - |
| - | t1 | UPDATE R |
| - | | - |
| FETCH R | t2 | - |
| - | | - |
| - | t3 | ROLLBACK |
| | ↓ | |

Daten mit Änderungen werden gelesen die später zurückgerollt werden.

Transaktion A liest Daten (t2), die logisch nie existiert haben (t3).
Einen solchen Lesevorgang bezeichnet man als **Dirty Read**.

| Transaktion A | Zeit | Transaktion B |
|---------------|------|---------------|
| - | | - |
| - | t1 | UPDATE R |
| - | | - |
| UPDATE R | t2 | - |
| - | | - |
| - | t3 | ROLLBACK |
| | ↓ | |

Zweifaches Problem:

- Transaktion A ändert Daten (t2), ausgehend von einem nicht existierenden Stand
- Änderungen werden wieder zurückgenommen (t3). Variante des Lost Update Problems

14.2 Serialisierbarkeit

- **Serialisierbarkeit** von Transaktionen: Der **Zustand** der Datenbank nach einem konkurrierenden Ablauf von Transaktionen und die Ergebnisse von Lesezugriffen während eines **konkurrierenden Ablaufes** von Transaktionen **müssen gleich sein** wie nach/bei einem **beliebigen seriellen** (sequentiellen) Ablauf der Transaktionen (umfangreiches theoretisches Gebiet). **Der Zustand der Datenbank soll nicht gleich aber konsistent sein.**
- **Schedule** (**History**, Ausführung, **Ausführungsplan**) **Historie**
Ein **konkreter** (auch zeitlich verzahnter) **Ablauf** von Datenbank-Abfragen und -Änderungen mehrerer **Transaktionen**

Transaktionen: T1, T2, T3, ..., Tn

Operationen: read, write, commit, abort

Daten: A, B, C, ...

Schedule: S1 = r1(A); r2(A); w1(A); w2(A)
S2 = r1(A); w1(A); r2(A); w2(A)

...

Die Anzahl der Operationen der Transaktion Ti sein Ni,
dann kann die Anzahl der Schedules berechnet werden als:

$$(N_1 + N_2 + \dots + N_n)! / (N_1! * N_2! * \dots * N_n!)$$

Transaktion führt eine Db von einem konsistenten Zustand in den nächsten

Die serielle Ausführung führt DB auch von konsistent zu konsistent

Eine verzahnte Ausführung ist konkret wenn das Ergebnis immer einem Ergebnis einer seriellen Ausführung gleicht

Übung: Wieviele Schedules gibt es beim Lost Update Problem und beim Inconsistent Analysis Problem?

- **Serieller (Serial) Schedule**
Schedule, bei dem die **Transaktionen hintereinander ausgeführt** werden (**Reihenfolge** der Transaktionen **beliebig**)

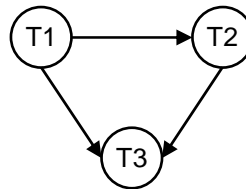
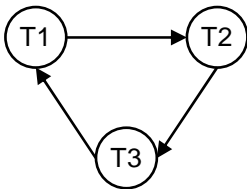
Übung: Wieviele serielle Schedules gibt es beim Lost Update Problem?
Übung: Wieviele serielle Schedules gibt es bei n Transaktionen?
- **Serialisierbarer (serializable, konfliktserialisierbarer) Schedule**
Wenn er **äquivalent zu** irgend einem **Seriellen Schedule** ist
- **Zwei äquivalente (equivalent, konfliktäquivalente) Schedules**
führen **zwei in Konflikt stehende Operationen** jeweils in **derselben Reihenfolge** aus
- **Konfliktoperationen**
 - gehören zu **unterschiedlichen Transaktionen**
 - greifen auf **dasselbe Datenobjekt** zu
 - **mindestens** eine Operation ist eine **Schreiboperation**
- **Algorithmus** zur Ermittlung der **Serialisierbarkeit** eines Schedules mittels **Präzedenzgraph** (Serialisierbarkeitsgraph, Konfliktgraph, Precedence Graph)
 - **Knoten:** alle Transaktionen
 - **Kante:** von T_i nach T_j , wenn es zwei Konfliktoperationen $OP_i \in T_i$ und $OP_j \in T_j$ gibt, wobei OP_i zeitlich vor OP_j liegt (müssen nicht unmittelbar hintereinander sein)
 - Wenn der Graph **azyklisch** ist (**keine Kreise** enthält), dann ist der **Schedule serialisierbar** (eine Angabe des / der äquivalenten seriellen Schedule ist möglich: T_i vor T_j , wenn Kante von T_i zu T_j führt – topologische Sortierung)

- Beispiel:

T_1 : r(A); w(B) T_2 : w(A) T_3 : r(A); r(B)

S_1 : r1(A); w2(A); r3(A); r3(B); w1(B)

S_2 : r1(A); w2(A); r3(A); w1(B); r3(B)



S_1 ist **nicht konfliktserialisierbar**

S_2 ist **konfliktserialisierbar**

Äquivalenter serieller Schedule: T_1, T_2, T_3

Bemerkung: S_1 unterscheidet sich von S_2 nur durch die Reihenfolge der beiden letzten Operationen

- Übung 1):

| | T_1 | T_2 | T_3 |
|---|-------|-------|-------|
| 1 | r(A) | - | - |
| 2 | - | w(A) | - |
| 3 | w(A) | - | - |
| 4 | - | - | w(A) |

- Übung 2):

| | T_1 | T_2 | T_3 |
|---|-------|-------|-------|
| 1 | r(A) | - | - |
| 2 | - | w(B) | - |
| 3 | - | w(C) | - |
| 4 | w(B) | - | - |
| 5 | - | - | r(C) |
| 6 | w(A) | - | - |
| 7 | - | - | w(C) |

- Übung 3): wie 2), zusätzlich

| | T1 | T2 | T3 | T4 |
|---|----|----|----|------|
| 0 | - | - | - | w(A) |

- Übung 4): wie 2), zusätzlich

| | T1 | T2 | T3 | T4 |
|---|----|----|----|------|
| 8 | - | - | - | w(A) |

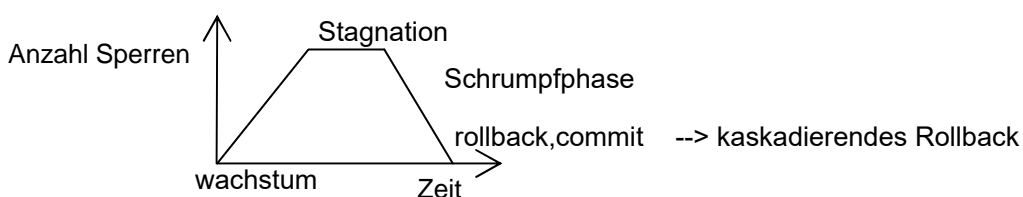
- Vorgangsweise: Einzelne Logs für die beteiligten Objekte aufstellen
- Wenn der Schedule serialisierbar ist, geben sie alle äquivalenten Seriellen Schedules an
- Übung 5): Geben Sie den Präzedenzgraphen für das Lost Update Problem und das Inconsistent Analysis Problem an.
- Übung 6): Gegeben ist folgender Schedule:

r1(D); r2(B); r3(A); w2(B); w2(A); w3(D); r4(B); r1(C); r4(C); r3(C); w4(D)

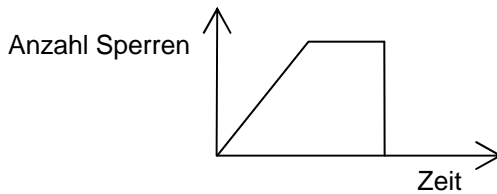
Erstellen Sie den Präzedenzgraph und beurteilen Sie, ob der Schedule serialisierbar ist. Wenn Serialisierbarkeit vorliegt, dann geben Sie alle äquivalenten seriellen Schedules an.

14.3 Lösungsmöglichkeiten

- Methoden zur Lösung von Concurrency-Problemen:
 - **Sperr-Verfahren** (Locking Techniques, **Pessimistische Sperr-Verfahren**, Pessimistic Concurrency Control) Annahme, dass Konflikte auftreten. Daher werden die **Objekte von den Transaktionen gesperrt**, um damit den **Zugriff** anderer Transaktionen auf diese Objekte für eine bestimmte (möglichst kurze) Zeit **einzuschränken** oder zu verhindern.
 - **Optimistische Sperr-Verfahren** (Optimistic Concurrency Control): Annahme, dass **keine Konflikte** auftreten. **Drei Phasen: Lesephase – Validierungsphase – Schreibphase**. Die Objekte werden **möglichst lange nicht gesperrt**, **scheitert** allerdings die **Validierung**, muss **reagiert** werden.
 - **Zeitstempel-Verfahren** (Timestamp Techniques, Timestamp Ordering): Jede **Transaktion** erhält eine **Zeitmarke** (**Startzeitpunkt**). **Synchronisiert** wird so, dass die **Abarbeitung** äquivalent zu einem **seriellen Schedule** ist. Ein **Konflikt** tritt auf, falls eine Transaktion z.B. eine **Leseanforderung** auf ein Objekt absetzt, das **bereits** von einer '**jüngeren**' Transaktion **verändert** wurde. **Konflikte** werden durch einen **Neustart** einer betroffenen Transaktion gelöst. Diese Methode hat im **Zusammenhang** mit **optimistischen Sperrverfahren** und **verteilten Datenbanken** an Bedeutung gewonnen.
- **Two-Phase Locking** (2PL, Zwei-Phasen Sperrprotokoll): **or MVCC (multi version concurrency control) oder multiversioning (Snapshot)**
 - Für jede Transaktion muss gelten
 - Bevor auf ein **Objekt zugegriffen** wird, muss auf das Objekt eine (entsprechende) **Sperre** angelegt werden.
 - Nachdem eine **Sperre aufgehoben** wurde, kann **keine neue** mehr angelegt werden.
 - Die Sperrstrategie besteht somit aus zwei Phasen
 - **Growing Phase (Wachstumsphase)**: **Alle** nötigen **Sperren** werden **gesetzt**, **ohne** eine **Sperre** wieder **aufzuheben**. Wenn **alle** Sperren gesetzt sind, ist der **Locked Point** erreicht.
 - **Shrinking Phase (Schrumpfungsphase)**: **Alle** Sperren werden **aufgehoben**, **keine neuen** gesetzt.
 - Es kann gezeigt werden, dass Two-Phase Locking **Serialisierbarkeit garantiert** (allerdings **Deadlocks nicht verhindert**).



- Strict Two-Phase Locking (S2PL, Striktes Zwei-Phasen Sperrprotokoll):
Alle Sperren werden erst am Ende der Transaktionen freigegeben (strenger als 2PL)



Kein Dirty-Read möglich
keine kaskadierendes Rollback

- Sperrobjekte (Sperrgranulat, Granularity): Granularitätswechsel = Eskalation
 - Hierarchie:
 - Datenbank
 - Tabelle
 - physischer Block / Seite
 - (Zeile) Datensatz/Row
 - Vorteil feiner Sperreinheiten (Fine Granularity): Höhere Parallelität zwischen den Transaktionen ist möglich.
 - Vorteil grober Sperreinheiten (Coarse Granularity): Geringerer Verwaltungsaufwand für die Sperren ist notwendig.
 - In der Praxis stellt Sperren auf Zeilebene (Row-Level Locking) die feinste Sperrform dar.
- Sperrmodi:
 - X-Lock (Exclusive Lock, Schreibsperre, Exklusive Sperre): Wenn eine Transaktion A das Objekt R mit X-Lock gesperrt hat, kann eine andere Transaktion B das Objekt R weder mit X-Lock noch mit S-Lock sperren; in diesem Fall muss Transaktion B solange warten, bis Transaktion A das Objekt R freigegeben hat.
 - S-Lock (Shared Lock, Lesesperre, Gemeinsame Sperre): Wenn eine Transaktion A das Objekt R mit S-Lock gesperrt hat, kann eine andere Transaktion B dasselbe Objekt zwar zusätzlich ebenfalls mit S-Lock, nicht aber mit X-Lock sperren; im zweiten Fall muss Transaktion B solange warten, bis Transaktion A das Objekt R freigegeben hat.
Transaktion A kann ein Objekt R, das ausschließlich von ihr mit S-Lock belegt ist, sehr wohl mit X-Lock sperren (Upgrade, Sperrkonversion). Upgrade != Eskalation
 - Kompatibilitätsmatrix (Compatibility Matrix)

| | | Transaktion A hat Objekt gesperrt mit | | |
|--|-----------|---------------------------------------|--------|-----------|
| | | X-Lock | S-Lock | kein Lock |
| Transaktion B kann Objekt sperrn mit | X-Lock | nein | nein | ja |
| | S-Lock | nein | ja | ja |
| | kein Lock | ja | ja | ja |

- Konkrete Systeme arbeiten mit:
 - nur einem Sperrmodus (X-Lock, binärem Sperren): geringerer Verwaltungsaufwand
 - beiden Sperrmodi (S-Lock und X-Lock): höhere Parallelität
- Für jedes System muss der Sperrmechanismus (Sperrstrategie, Locking-Mechanism) definiert werden:
Wodurch (explizit oder implizit) bei welchem Objekt welche Sperre angebracht wird und wodurch diese wieder aufgehoben wird.
- Die obigen Beispiele werden nun unter Verwendung des folgenden Sperrmechanismus 'SPERR1' behandelt:
 - Sperrobjekte: Zeilen
 - Sperrmodi: Vor FETCH wird ein S-Lock abgesetzt, vor UPDATE wird ein X-Lock durchgeführt
 - Alle Sperren werden bei Transaktionsende aufgehoben

- Lost Update Problem

| Transaktion A | Zeit | Transaktion B |
|-------------------------|------|-------------------------|
| - | | - |
| FETCH R | t1 | - |
| (S-Lock auf R) | | - |
| - | t2 | FETCH R |
| - | | (S-Lock auf R) |
| UPDATE R | t3 | - |
| (Request: X-Lock auf R) | | - |
| warten | t4 | UPDATE R |
| warten | | (Request: X-Lock auf R) |
| warten | | warten |
| warten | ↓ | warten |

Das verwendete Sperrverfahren führt bei diesem zeitlichen Ablauf zu einem Deadlock.

- Inconsistent Analysis Problem

| Transaktion A | Zeit | Transaktion B |
|--------------------------|------|--------------------------|
| - | | - |
| FETCH R1 | t1 | - |
| (S-Lock auf R1) | | - |
| sum=sum+stand (sum: 40) | | - |
| FETCH R2 | t2 | - |
| (S-Lock auf R2) | | - |
| sum=sum+stand (sum: 90) | | - |
| - | | - |
| - | t3 | FETCH R3 |
| - | | (S-Lock auf R3) |
| - | | stand=stand-10 |
| - | t4 | UPDATE R3 (stand: 20) |
| - | | (X-Lock auf R3, Upgrade) |
| - | t5 | FETCH R1 |
| - | | (S-Lock auf R1) |
| - | | stand=stand+10 |
| - | t6 | UPDATE R1 (stand: 50) |
| - | | (Request: X-Lock auf R1) |
| - | | warten |
| FETCH R3 | t7 | warten |
| (Request: S-Lock auf R3) | | warten |
| warten | | warten |
| warten | ↓ | warten |

Das System befindet sich in einer Deadlock-Situation.

- Uncommitted Dependency Problem

| Transaktion A | Zeit | Transaktion B |
|-------------------------|------|-------------------|
| - | | - |
| - | t1 | UPDATE R |
| - | | (X-Lock auf R) |
| FETCH R | t2 | - |
| (Request: S-Lock auf R) | | - |
| warten | t3 | ROLLBACK |
| warten | | (Release: X-Lock) |
| (Resume: FETCH R) | | - |
| (S-Lock auf R) | ↓ | - |

Transaktion A liest konsistente Daten nach dem Abschluss der Transaktion B.

- Realisierung von Locking:

In einer Tabelle werden pro Transaktion alle Sperren und Sperranforderungen gespeichert

- Identifikation der Transaktion
- Identifikation des Sperrobjects (z.B. Tabellename, Zeilennummer)
- Art der Sperre (S, X)
- Kennzeichen, ob Sperre oder Sperranforderung

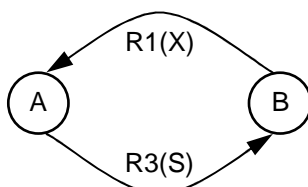
14.4 Deadlocks

- Deadlock (Verklemmung, Deadly Embrace): Wechselseitiges Aufeinanderwarten (geschlossene Kette Aufeinanderwartens)
- Die Transaktion kann von sich aus nicht erkennen, ob sie in eine Deadlock-Situation kommt oder sich in einer Deadlock-Situation befindet:

| Transaktion A | Zeit | Transaktion B |
|-------------------------|------|---------------------------|
| - | | - |
| FETCH R | t1 | - |
| (S-Lock auf R) | | - |
| - | t2 | FETCH R |
| - | | (S-Lock auf R) |
| UPDATE R | t3 | - |
| (Request: X-Lock auf R) | | - |
| warten | | DISPLAY R |
| warten | | DISPLAY 'Weiter ja/nein?' |
| warten | | READ Antwort |
| Warten | t4 | COMMIT |
| Warten | | (Release: S-Lock auf R) |
| (Resume: UPDATE R) | | - |
| (X-Lock auf R) | | - |
| | ↓ | |

Beim Warten nach der UPDATE-Anforderung (t3) ist für Transaktion A nicht ersichtlich, ob ein Deadlock vorliegt (beim Lost Update Problem) oder nicht (in diesem Beispiel)

- Die Entscheidung, ob eine Deadlock-Situation vorliegt, muss ein übergeordnetes System (Datenbanksystem, Transaktionsmonitor) treffen.
- Das Deadlock-Problem wird systemmäßig im Wesentlichen auf zwei Arten behandelt:
 - Deadlock-Vermeidung (avoidance):
Besteht bei einer Sperranforderung die Gefahr, dass ein Deadlock auftreten kann, wird die Transaktion abgebrochen (prevention).
Alle Objekte, die die Transaktion verwenden will, werden zu Beginn der Transaktion gesperrt (preclaiming).
Mit dem Zeitstempel-Verfahren ist ebenfalls eine Deadlock-Vermeidung möglich (timestamping).
 - Deadlock-Erkennung (detection):
Erkennung:
 - Zur Deadlock-Erkennung wird ein (gerichteter) Wartegraph (Wait-For Graph) geführt:
 - Knoten: Im System vorhandene Transaktionen
 - Kanten: Wenn Transaktion A ein Objekt sperren will (Request), Transaktion B aber eine entsprechende Sperre auf dieses Objekt aufrecht hält und Transaktion A daher warten muss, wird eine Kante von A nach B gezeichnet. Zur besseren Übersicht können die Kanten auch mit dem Namen des Objekts und der Art der gewünschten Sperre bezeichnet werden, z.B. R1(X), R3(S)
 - Eine Deadlock-Situation liegt genau dann vor, wenn im Wartegraph Zyklen (Kreise) enthalten sind. Der Wartegraph muss von Zeit zu Zeit (wann?) auf das Vorhandensein von Zyklen geprüft werden.
 - Beispiel: Wartegraph im Inconsistent Analysis Problem

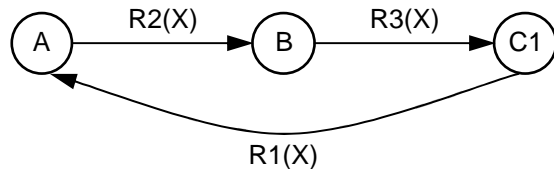


Beseitigung:

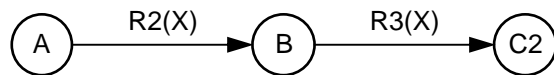
- Eine an der Deadlocksituation beteiligte Transaktion ist als 'Opfer' auszuwählen und zurückzusetzen (diejenige mit den geringsten Rücksetzkosten)
- Timeout: Sperranforderungen werden nach einer gewissen Zeitspanne für unerfüllbar erklärt oder jede Transaktion hat eine bestimmte Zeitdauer vorgegeben, nach deren Ablauf sie abgebrochen wird. Deadlocksituationen werden dadurch zwar aufgelöst, jedoch erfolgen auch Abbrüche von Transaktionen, die in einer normalen Wartesituation sind oder aus irgend einem anderen Grund länger dauern.
- Ein Deadlock kann auch zyklisch mehrere Transaktionen betreffen:

| Zeit | Transaktion A | Transaktion B | Transaktion C1 | Transaktion C2 |
|------|---------------|---------------|----------------|----------------|
| | - | - | - | - |
| | FETCH R1 | - | - | - |
| | - | FETCH R2 | - | - |
| | - | - | FETCH R3 | FETCH R3 |
| | UPDATE R2 | - | - | - |
| | warten | UPDATE R3 | - | - |
| | warten | warten | UPDATE R1 | UPDATE R3 |
| | warten | warten | warten | - |

- Im Fall C1 liegt ein Deadlock vor, Wartegraph:



- Im Fall C2 liegt kein Deadlock vor, Wartegraph:



- Eine Deadlock-Situation kann auch auftreten, wenn nur X-Locks angewendet werden:

| Transaktion A | Zeit | Transaktion B |
|---------------------------------------|------|---------------------------------------|
| - | | - |
| FETCH R1 (X-Lock auf R1) | t1 | - |
| - | t2 | - |
| - | | FETCH R2 (X-Lock auf R2) |
| UPDATE R2 (Request: X-Lock auf R2) | t3 | - |
| warten | t4 | - |
| warten | | UPDATE R1 (Request: X-Lock auf R1) |
| warten | | warten |
| warten | | warten |

- Die meisten Systeme erlauben eine programmgesteuerte Abfrage, ob ein Objekt (Zeile) gesperrt ist. Daher soll zur Deadlock-Vermeidung in der Logik des Anwenderprogramms bei einer Sperranforderung geprüft werden, ob das Objekt nicht bereits gesperrt ist und in diesem Fall die Transaktion programmgesteuert reagieren (z.B. Entsprechende Meldung geben und abbrechen, Rückfragen ob und wie lange gewartet werden soll)
- Durch Angabe entsprechender Optionen beim Datenzugriff kann in den meisten Systemen der Standard-Sperrmechanismus programmspezifisch individuell verändert werden.
Beispiele: Lesen mit No-Lock (Dirty Read), Lesen mit X-Lock, etc.

- Übung 1): Wie stellen sich die drei, in der Problemstellung beschriebenen, Situationen dar, wenn nur der Sperrmodus X-Lock (vor FETCH und UPDATE) verwendet wird (Sperrmechanismus 'SPERR2')?
- Übung 2): Ergeben sich beim Inconsistent Analysis Problem Änderungen im Verhalten, wenn Transaktion B die Lese- / Updateoperationen (R3,R1) in umgekehrter Reihenfolge (R1,R3) durchführt (für SPERR1 und SPERR2)?
- Übung 3): Ergeben sich beim Inconsistent Analysis Problem Änderungen im Verhalten, wenn Transaktion B zuerst beide FETCH-Operationen (R3, R1 oder R1, R3) und dann beide UPDATE-Operationen ausführt (für SPERR1 und SPERR2)?
- Übung 4): Ergeben sich beim Inconsistent Analysis Problem Änderungen im Verhalten, wenn Transaktion B die Lese- / Updateoperationen bereits nach dem FETCH R1 von A durchführt (für SPERR1 und SPERR2)?
- Übung 5)
Sperrmechanismus:
 - Sperrobjecte sind Zeilen
 - S-Lock vor FETCH
 - X-Lock vor UPDATE
 - Aufheben der Sperren bei Transaktionsende (COMMIT)

| Zeitpunkt | Transaktion | Operation |
|-----------|-------------|-----------|
| t1 | T1 | FETCH A |
| t2 | T5 | FETCH B |
| t3 | T4 | FETCH C |
| t4 | T3 | FETCH A |
| t5 | T5 | UPDATE B |
| t6 | T4 | FETCH A |
| t7 | T1 | FETCH C |
| t8 | T2 | FETCH B |
| t9 | T1 | UPDATE C |
| t10 | T5 | COMMIT |
| t11 | T3 | UPDATE A |
| t12 | ... | ... |

Zeitpunkt t12:

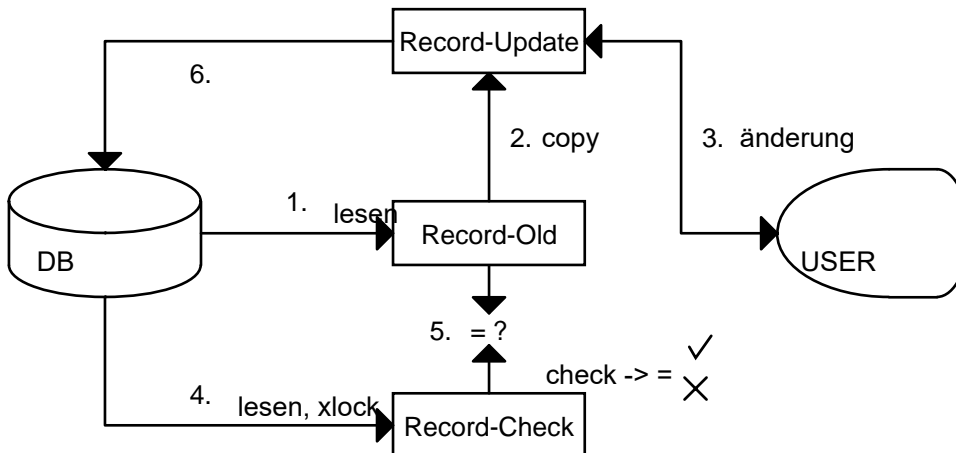
- Welche Transaktionen haben / wollen welche Objekte in welcher Weise gesperrt / sperren?
- Wie sieht der Wartegraph aus?
- Liegt ein Deadlock vor?
 - ja - welche Transaktion(en) muss (müssen) abgebrochen werden?
 - nein - in welcher Reihenfolge können die Transaktionen beendet werden?
- Übung 6)
Sperrmechanismus wie in Übung 5), folgender Schedule ist gegeben:

r2(B); r1(A); w2(B); r1(B); r2(A); r3(C); w2(C); w4(B); w3(A);
- Übung 7)
Folgender Schedule von Sperranforderungen ist gegeben:

s1(A); s3(A); x2(D); x3(C); s1(C); s3(D); x4(A);

14.5 Die Re-Read-Methode

- Am Beispiel der Änderung eines Satzes in einem Multi-Tasking-Betrieb wird das Re-Read-Verfahren (Read-Before-Write-Verfahren) gezeigt, das folgenden Kriterien genügt:
 - Es darf zu keinen Inkonsistenzen kommen
 - Die Objekte dürfen nicht zu lange (im Besonderen über Benutzereingaben hinweg) gesperrt sein (im Besonderen mit X-Lock)
 - Es darf keine Deadlock-Situationen auftreten (wenn das System keinen Mechanismus zur Deadlock-Erkennung hat)
- Graphische Darstellung:



- Beschreibung:
 1. Satz nach Record-Old ohne Sperre einlesen
 2. Satz (oder Teile davon) aus Record-Old nach Record-Update kopieren
 3. Satz Record-Update (oder Teile davon) vom Benutzer ändern lassen
 4. Satz nach Record-Check mit X-Lock noch einmal einlesen (Re-Read)
 5. Ganzen Satz in Record-Old mit ganzem Satz in Record-Check vergleichen:
 6. wenn gleich, dann Satz auf Grund von Record-Update ändern
wenn ungleich, dann Meldung 'Satz wurde mittlerweile geändert' an den Benutzer

14.6 Zusätzlicher Sperrmodus U-Lock

- Update Lock, Promoteable Lock, Incremental Lock, Aktualisierende Sperre
- Wird bei Lese-Operationen angegeben, wenn danach eine Änderungs-Operation beabsichtigt ist
- Geringere Deadlock-Gefahr als bei Lesen mit S-Lock, höhere Parallelität als bei Lesen mit X-Lock
- Kompatibilitätsmatrix

| | X-Lock | U-Lock | S-Lock | kein Lock |
|-----------|--------|--------|--------|-----------|
| X-Lock | nein | nein | nein | ja |
| U-Lock | nein | nein | ja | ja |
| S-Lock | nein | ja | ja | ja |
| kein Lock | ja | ja | ja | ja |

- Sperrmechanismus 'SPERR3':
vor FETCH S-Lock, vor FETCH mit folgendem UPDATE U-Lock, vor UPDATE X-Lock
- Vergleich SPERR1 mit SPERR3
 - Lost Update Problem: kein Deadlock (zweites FETCH muss warten, da zwei U-Locks nicht kompatibel)
 - Inconsistent Analysis Problem: gleiches Verhalten wie bei SPERR1 (Deadlock)
 - Uncommitted Dependency Problem: gleiches Verhalten wie bei SPERR1 (kein Deadlock)
- Lesen mit U-Lock bei konkreten Systemen in folgender Form (beispielhaft)
 - SELECT ... FOR UPDATE ...
 - SQL92-Norm: DECLARE ... CURSOR ... FOR UPDATE
 - MS SQLServer: SELECT ... FROM X (UPDLOCK) ...

14.7 Isolation Levels

- Isolation Level: Grad, zu dem Datenkonsistenz (zu Lasten der Parallelität) eingehalten wird (ursprünglich 'Degree of Consistency').
Definition des Verhaltens einer Transaktion bezüglich Concurrency versus Integrity.
Grad der gegenseitigen Beeinflussung von Transaktionen (Interference of Transactions).
- Es werden nicht für einzelne Operationen Sperrmodi angegeben, sondern für Transaktionen Isolation Levels definiert, die festlegen welche Serialisierungsprobleme auftreten dürfen und welche nicht
- 4 Isolation Levels (Intermediate SQL-92):
 - SERIALIZABLE hoch keine Gefahr von Inkonsistenzen - niedere Parallelität
 - REPEATABLE READ ↓
 - READ COMMITTED ↓
 - READ UNCOMMITTED nieder Gefahr von Inkonsistenzen - hohe Parallelität

Snapshot MVCC

 - SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED |
READ COMMITTED |
REPEATABLE READ |
SERIALIZABLE }
 - Standardannahme: ~~SERIALIZABLE~~ -- Read Committed
- 3 Arten von Serialisierungsproblemen:
 - Dirty Read: Transaktion A liest eine Zeile, die von einer anderen Transaktion B geändert wurde. B führt ein Rollback durch und A arbeitet mit einer Zeile, die so nie existiert hat (Uncommitted Dependency).
 - Nonrepeatable Read (Fuzzy Read): Transaktion A liest eine Zeile; eine andere Transaktion B ändert diese Zeile. A liest diese Zeile nochmals und erhält eine andere Version derselben Zeile.
 - Phantoms: Transaktion A liest mehrere Zeilen, die einer bestimmten Bedingung genügen (Abfrage); eine andere Transaktion B fügt eine Zeile ein, die auch diese Bedingung erfüllt (oder macht in einer existierenden Zeile eine Änderung, sodass sie diese Bedingung dann auch erfüllt). A wiederholt die Abfrage und erhält ein anderes Ergebnis (mehr Zeilen).
- Folgende Tabelle gibt an, ob bei einem bestimmten Isolation Level ein bestimmtes Problem auftreten kann:

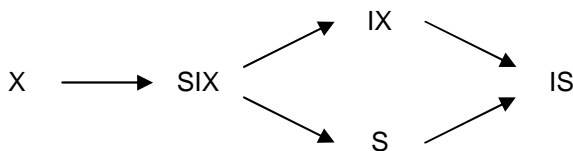
| | Dirty Read | Nonrepeatable Read | Phantoms |
|------------------|------------|--------------------|----------|
| READ UNCOMMITTED | ja | ja | ja |
| READ COMMITTED | nein | ja | ja |
| REPEATABLE READ | nein | nein | ja |
| SERIALIZABLE | nein | nein | nein |

- Realisierung mit Sperren:
 - READ UNCOMMITTED kein Lock beim Lesen (alle Probleme treten auf)
 - READ COMMITTED S-Lock auf Zeile beim Lesen; nicht Two-Phase Locking (z.B. Lost Update tritt auf)
 - REPEATABLE READ S-Lock auf Zeile beim Lesen bis Transaktionsende
 - SERIALIZABLE einfach: S-Lock auf Tabelle(n) beim Lesen bis Transaktionsende
komplex: Predicate Locking; Zugriffspfad (Access Path) für die angegebene Bedingung sperren (z.B. bestimmte Indexeinträge); Sperren 'nicht existenter' Zeilen
- ~~Varianten der Isolation Levels von SQL 92:~~
 - ~~DIRTY READ -- READ UNCOMMITTED (INFORMIX)~~
 - ~~CURSOR STABILITY (INFORMIX, SQL Base)~~
~~zwischen READ COMMITTED und REPEATABLE READ~~
 - ~~RELEASE LOCKS, READ ONLY (SQLBase)~~
 - ~~VERSIONING (ODBC)~~

14.8 Hierarchische Sperrverfahren

Escalation = aufstufen der Granularität

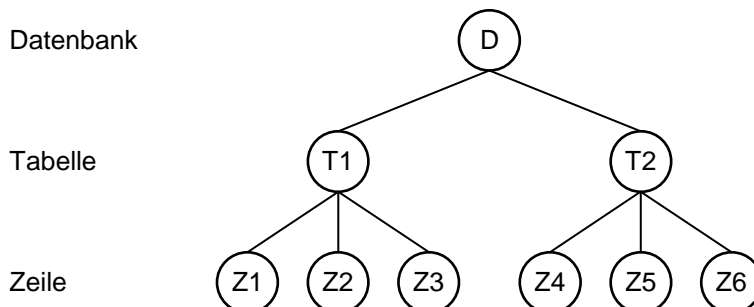
- Sperren von verschiedenen Granulaten / auf verschiedenen Hierarchien
(Datenbank – Tabelle – (Seite) – Zeile)
Multiple-Granularity Locking (MGL) flexible Wahl der Sperrgranularität
- Problem: Wenn eine Transaktion ein Objekt sperren will, muss für alle untergeordneten Objekte überprüft werden, ob ein Sperrkonflikt auftritt
- Beim Sperren eines Objekts muss Top-Down bei allen in der Hierarchie übergeordneten Objekten eine geeignete Sperre erworben werden, das Freigeben erfolgt Bottom-Up
- Einführung von Intent Locks (Anwartschaftssperren) **Intentionssperren**
(nicht sinnvoll für Objekte auf unterster Hierarchiestufe):
 - IS-Lock (Intent Shared Lock): weiter unten in der Hierarchie sind S-Locks beabsichtigt **Intention Share**
 - IX-Lock (Intent Exclusive Lock): weiter unten in der Hierarchie sind X-Locks beabsichtigt **Intention Exclusiv**
 - SIX-Lock (Shared Intent Exclusive Lock): sperrt Objekt mit S-Lock und siehe IX
- Stärke der Sperrmodi (links ist stärker):



- Sperrstrategie
 - Bevor eine Transaktion ein Objekt mit S-Lock sperren kann, müssen alle seine übergeordneten Objekte Top-Down (von oben nach unten) mit IS-Lock oder stärker gesperrt werden
 - Bevor eine Transaktion ein Objekt mit X-Lock sperren kann, müssen alle seine übergeordneten Objekte Top-Down (von oben nach unten) mit IX-Lock oder stärker gesperrt werden
 - Die Sperren werden Bottom-Up (von unten nach oben) freigegeben, d.h. es wird die Sperre nicht freigegeben, wenn die Transaktion noch untergeordnete Objekte gesperrt hat
- Kompatibilitätsmatrix

| | X | SIX | IX | U | S | IS |
|-----|---|-----|----|---|---|----|
| X | - | - | - | - | - | - |
| SIX | - | - | - | - | - | + |
| IX | - | - | + | - | - | + |
| U | - | - | - | - | + | + |
| S | - | - | - | + | + | + |
| IS | - | + | + | + | + | + |

- Beispiel



Transaktion1 sperrt Z1 mit X-Lock
 Transaktion2 sperrt T2 mit S-Lock
 Transaktion2 sperrt Z2 mit S-Lock
 Transaktion3 sperrt Z5 mit S-Lock
 Transaktion3 sperrt Z6 mit X-Lock

Welche Transaktionen bringen auf welchen Objekten welche Sperren an?

15 Vergleich OLTP- und OLAP-Anwendungen

OLTP Online Transaction Processing – System

- Tagesgeschäft, Produktive Geschäftsfälle
- Operative Ebene / Operativ / Geschäft
z.B. Flugbuchungssystem, Order-Entry-System
- Aktuelle, dynamische Informationen
- Standardisierte Abläufe / Abfragen
- (Direkt) lesender und schreibender Zugriff
- Viele Benutzer (Concurrency!)
- Wenige Daten pro Transaktion betroffen
- Kurze Transaktionen
- Moderater Datenumfang (TByte-Bereich)
- Performance (Sekundenbereich) und Verfügbarkeit (rascher Wiederanlauf) wichtig
- tendenziell hohe Normalisierung

OLAP Online Analytical Processing – Systeme

- Analyse, Entscheidungsfindung
- Strategische Ebene / Dispositiv / Unterstützung
z.B. Analyse der Verkäufe im 3.Quartal 2003
- Historische, statische Informationen, 'old news'
- Vielfältige, komplexe Abläufe / Abfragen
- (Analytisch) nur lesender Zugriff
- Eher wenige Benutzer
- Sehr viele Daten pro Anfrage betroffen
- Lange Transaktionen
- Großer Datenumfang (PByte-Bereich)
- Spielraum bei Performance und Verfügbarkeit
- tendenziell Denormalisierung
(auf Auswertungs- und Analysezwecke abgestimmt)

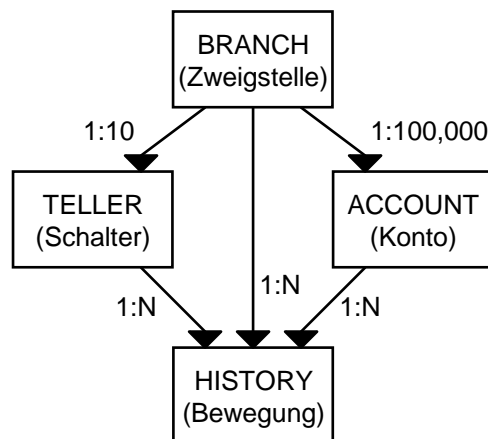
auch:

Data Warehousing, Data Mining
(Analyse von Zusammenhängen / Cluster)
BI (Business Intelligence)
DSS (Decision Support Systeme)
EIS (Executive Informations Systeme)
MIS (Management Informations Systeme)

- Strategie: Physische Trennung von OLTP- und OLAP-Datenbeständen (Information mehrfach / redundant geführt, allerdings in unterschiedlicher Struktur)
- **Data Warehouse:** Aus den OLTP-Informationen erstellte / kodierte Datenbasis für OLAP-Anwendungen; validierte, konsolidierte, teilweise schon vorverdichtete Daten
- **OLAP-Datenstruktur**
 - **Sternschema** (Star Schema): Mehrere Dimensionstabellen (wenige Zeilen, eventuell nicht normalisiert), eine Faktentabelle (sehr viele Zeilen), in der Faktentabelle die Fremdschlüssel zu allen Dimensionstabellen.
 - Beispiel:
Dimensionstabellen: Produkte (samt Produktgruppen- und Herstellerinformation), Filialen (samt Bezirks-, Bundeslands- und Landesinformation), Kunden, Verkäufer, Tag (samt Wochen-, Wochentags- und Quartalsinformation)
Faktentabelle: Verkäufe
 - **Schneeflockenschema** (Snow Flake Schema): Dimensionstabellen sind weiter aufgeteilt / normalisiert
- Die Struktur des Data Warehouses wird auch als **Hypercube** (Datacube, Mehr- / Multidimensionale Datenbank, MDDb, Hyperwürfel) bezeichnet. Der Benutzer untersucht diesen Würfel durch Aufteilen / Schneiden in kleinere Würfel (Dicing) oder Scheiben (Slicing)
- SQL-Erweiterungen für OLAP (z.B. Star-Join, Roll-Up- / Drill-Down-Abfragen, Materialized Aggregates, CUBE-Operator, Slicing and Dicing) werden auch als ROLAP bezeichnet
- Andere Verwendung von Indizes als bei OLTP-Systemen (z.B. Bitmap-Indizes)

16 Performancemessung in Datenbanksystemen

- Transaction Processing Council (TPC): 1988 gegründetes Konsortium von Hard- und Software-Herstellern, das Benchmarking von TP- und DB- Systemen vereinheitlicht.
Sämtliche Spezifikationen verfügbar unter: www.tpc.org
- Benchmark (Benchmarkprogramm, Bewertungsprogramm): Typische Zusammensetzung von Benutzer- und Systemprogrammen, wie sie in der Praxis (für eine konkrete Anwendung) eingesetzt werden. Dient zur Leistungsbeurteilung von Rechnersystemen (Hardware und Software).
- Entscheidend sind die tps- oder tpm-Raten (transactions per second oder minute) der verschiedenen Benchmarks.
Als Variante, in der Kosten berücksichtigt sind, werden auch \$/tps oder \$/tpm angegeben (Kosten: Summe der Hardware-, Software- und Wartungskosten in fünf Jahren)
- TPC-A (1989), TPC-B (1990) - obsolete:
 - Bankanwendung
 - Ein Transaktionstyp
 - Bei TPC-B wird nur der DBMS-bezogene Anteil der Verarbeitung berücksichtigt (keine Terminalverbindungen, E-/A-Nachrichten, etc). In den Kosten sind diese Komponenten ebenfalls nicht erfasst (Terminals, Kommunikationsnetz, etc). TPC-B liefert daher höhere Durchsatzraten mit geringeren Kosten.
 - Bei TPC-A ist noch zu unterscheiden, ob die Verarbeitung in lokalen (tpsA-local) oder in geographisch verteilten Netzen (tpsA-wide) stattfindet.
 - Beschreibung:



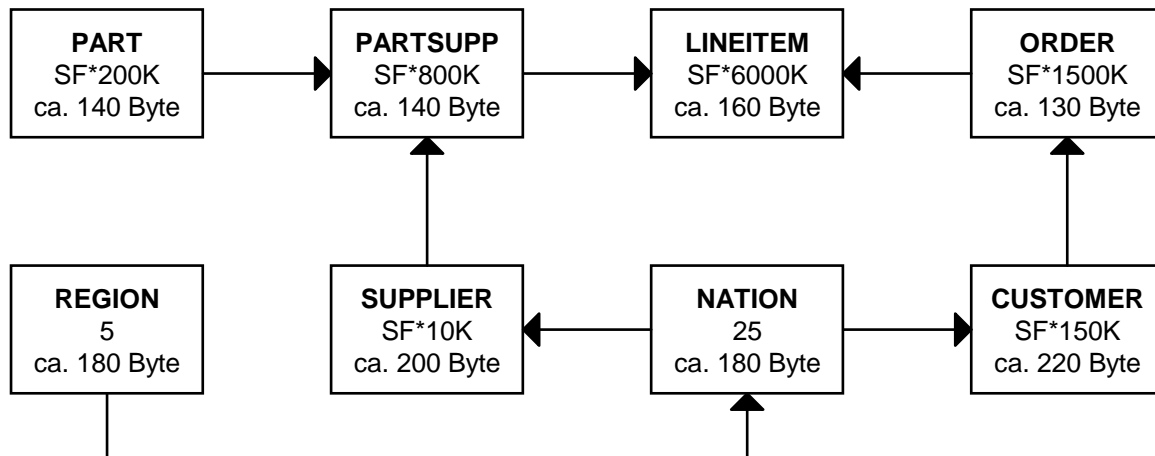
Transaktionsprogramm:

```

Read message from Terminal (acctno, branchno, tellerno, delta);
BEGIN WORK;
UPDATE ACCOUNT SET balance = balance + $delta WHERE acct_no = $acctno;
SELECT balance INTO $abalance FROM ACCOUNT WHERE acct_no = $acctno;
UPDATE TELLER SET balance = balance + $delta WHERE teller_no = $tellerno;
UPDATE BRANCH SET balance = balance + $delta WHERE branch_no = $branchno;
INSERT INTO HISTORY (tid, bid, aid, delta, time) VALUES ($tellerno, $branchno, $acctno, $delta, ...);
COMMIT WORK;
Write message to Terminal (abalance, ...);
  
```

- Pro tps sind vorzusehen: 1 BRANCH-Satz, 10 TELLER-Sätze, 100,000 ACCOUNT-Sätze, 10 Terminals
- HISTORY muss alle Bewegungen von 90 (TPC-A) oder 30 (TPC-B) Tagen aufnehmen können
- 90% der Transaktionen müssen eine Antwortzeit von unter zwei Sekunden haben
- Beispiel: Durchsatzziel 100 tps
BRANCH: 100 Sätze, TELLER: 1000 Sätze, ACCOUNT: 10 Millionen Sätze, Terminals (simuliert): 1000
HISTORY (Achtstundentag angenommen): 260 (TPC-A) oder 86 (TPC-B) Millionen Sätze

- TPC-C (1992):
 - OLTP
 - Arbeitsvorgänge im Großhandel
 - Fünf Transaktionstypen unterschiedlicher Komplexität (Eingabe und Ausführung von Aufträgen, Erfassen von Zahlungen, Statusüberprüfungen eines Auftrages, Kontrolle des Artikelbestandes in den Lagern, etc)
 - Haupttransaktionstyp: Abwicklung eines Bestellvorganges (48 SQL-Anweisungen)
 - 9 Tabellen, 92 Spalten
 - Online- und Batch-Transaktionen
- TPC-E (1992):
 - OLTP, 'Enterprise'-Benchmark
 - Broker und Trader, die an der Börse handeln
 - 33 Tabellen, 188 Spalten
 - Interdependenz einzelner Transaktionen
- TPC-H (1999):
 - Decision Support for Ad-hoc Queries
 - Komplexe, datenintensive Queries
 - Datenbankstruktur und -größe:



1. Zeile: Tabellename
2. Zeile: Anzahl der Zeilen; SF=Scale Factor (erlaubte Werte: 1, 10, 30, 100, 300, 1000)
3. Zeile: Länge einer Zeile; Datenbankgröße damit bei SF=1 ca. 1GB, bei SF=1000 ca. 1TB

- 22 Queries und 2 Update-Funktionen werden in der Datenbank ausgeführt
- Kennwerte:
 - QppH/R (Query processing power H oder R): Kehrwert des geometrischen Mittels aller Antwortzeiten
 - QthH/R (Query throughput H oder R): errechnet aus Anzahl der gleichzeitigen Benutzer und der längsten Gesamtzeit für alle Operationen
 - QphH/R (Query per hour H oder R): geometrisches Mittel aus QppH/R und QthH/R, dieses wird ins Verhältnis zu den Cost of Ownership über 5 Jahre gesetzt
- Kritik an Benchmarking:
 - Praktische Aussagekraft teilweise umstritten
 - Eigentlich müsste die eigene (gewünschte) Anwendung unter verschiedenen DB-Systemen verglichen werden
 - Parallelität ist teilweise zwar berücksichtigt, wie sie jedoch im tatsächlichen Einsatz auftritt ist schwierig zu prognostizieren
 - Transaktionsdurchsatz ist nicht das einzige Kriterium für die Leistungsfähigkeit eines DB-Systems

17 XML (CLIL)

17.1 Introduction

- eXtensible Markup Language
- World Wide Web Consortium (W3C) - www.w3.org
- Structured data: distinction between schema and data, structure of data strictly defined by schema
- Semi-structured data: mainly structured data, also elements which are not liable to the static schema
- Format for universal data exchange of any structured and / or nested data (records, lists, trees, etc.), advantages: simple, text-based, platform independent, extensible
- Meta-informations make data self-explaining

17.2 XML Document Structure

- Elements
 - start-tag – text data – end-tag (tags are case sensitive)
example for text content
`<name>Max Maier</name>`
 - example for empty content
`<name></name>` abbreviated `<name/>`
 - start-tag – elements – end-tag (nesting of elements)
example for element content
`<person>`
 `<name> <firstname>Max</firstname> <lastname>Maier</lastname> </name>`
 `<salary>1345</salary>`
`</person>`
 - example for mixed content
`<statement>`
 being a `<pet>` dog `</pet>` is a `<kindofjob>` full-time `</kindofjob>` job
`</statement>`
- Attributes
 - name-value pairs, in the start-tag of an element (quoted value)
 - example
`<person pid="4711" snr="1234 241281">`
 `<name> <firstname>Max</firstname> <lastname>Maier</lastname> </name>`
 `<salary>1345</salary>`
`</person>`
 - example with empty content
`<image source="picture.jpg" size="80%" />`
 - several attributes per element possible
 - differences to element: not nestable, unique name within one element, order irrelevant
 - element-oriented or attribute-oriented style
- Well-formed XML documents obey the XML-Syntax (checked by every internet-browser):
 - Elements must have a start-tag and an end-tag
 - Elements must be properly nested
 - Documents must have a single root element
 - Element- and attribute-names are case sensitive
 - Attribute names must be unique within an element
 - Attribute values must be quoted (single or double)
- Comments
`<!-- this is a comment -->`
- Prolog (declaration)
 - first line, optional, strongly recommended
 - example
`<?xml version="1.0" encoding="ISO-8859-1" ?>`
 - 3 name value pairs possible:
version (common), encoding (default: UTF-8), standalone (yes | no)

- Why are the following XML documents not well-formed?

a)

```
<employee>
  <name>Frank</name>
  <position>Chef</position>
</employee>
<employee>
  <name>Ronnie</name>
  <Position>Chef</Position>
</employee>
```

b)

```
<employee>
  <kitchen_staff/>
  <name language=en>Frank</name>
  <new_hire />
  <position language=en>Chef</position>
</employee>
```

c)

```
<employee>
  <name>
    <lastname>Kelly</lastname>
    <firstname>Grace</firstname>
  </name>
  <hiredate>October 15, 2005</hiredate>
  <projects>
    <project>
      <id>111<product>Printer</id></product>
      <price>$111.00</price>
    </project>
    <project>
      <id>222<product>Laptop</id></product>
      <price>$989.00</price>
    </project>
  </projects>
</employee>
```

17.3 Document Type Definition (DTD)

<http://www.w3schools.com/dtd>

- Valid XML documents are well-formed and conform to the rules of a schema, 'contract with trading partners', defined in a schema language (DTD or XML Schema), 'an XML document is valid against a specific DTD or XML schema'
- Internal DTD Declaration

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE root-element [
element-declarations
]>
```
- External DTD Declaration

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE root-element SYSTEM "filename.dtd">
```

 - referenced file contains element-declarations
 - also URI (Uniform Resource Identifier) possible
- Elements
 - Elements with text data

```
<!ELEMENT element-name (#PCDATA)>
```

PCDATA: parsed character data
 - Empty Elements

```
<!ELEMENT element-name EMPTY>
```
 - Elements with any Contents

```
<!ELEMENT element-name ANY>
```

- Elements with Children

```
<!ELEMENT element-name (child-name1,child-name2,...)> group - sequence
<!ELEMENT element-name (child-name1|child-name2|...)> group - choice
<!ELEMENT element-name (child-name) > required - 1..1
<!ELEMENT element-name (child-name?)> optional - 0..1
<!ELEMENT element-name (child-name+)> required and repeatable - 1..n
<!ELEMENT element-name (child-name*)> optional and repeatable - 0..n
```

- Nesting is possible

- Attributes

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

| attribute-type (most common) | Description |
|---------------------------------|--|
| CDATA | character data (not to be parsed) |
| (en1 en2 ...) | one from the enumerated list |
| ID | unique id within whole document, not per element, value must begin with a letter |
| IDREF | id of another element, global in document, name of element cannot be specified |
| IDREFS | list of other ids, separated by blank |

| default-value | Explanation |
|---------------|--------------------------------|
| value | default value of the attribute |
| #REQUIRED | attribute is required |
| #IMPLIED | attribute is not required |
| #FIXED value | attribute value is fixed |

- Recursive Definition, e.g.

```
<!DOCTYPE A [
  <!ELEMENT A (B,A?,C)>
  <!ELEMENT B (#PCDATA)>
  <!ELEMENT C (#PCDATA)>
]>
<A><B>text</B><C>text</C></A> or
<A><B>text</B><A><B>text</B><C>text</C></A><C>text</C></A> or ...
```

17.4 Namespaces

http://www.w3schools.com/xml/xml_namespaces.asp

- Unique named elements and attributes in an XML document
- Namespace: global unique name (e.g. URI), names within one namespace are unique
- Syntax for namespaces
 - Qualifying of element- and attributenames
prefix:elementname
prefix:attributename
 - Assigning a prefix to a namespace
xmlns:prefix="URI"
defaultnamespace: xmlns="URI"

```
<root xmlns:h="http://www.w3.org/TR/html4"
      xmlns:f="http://www.trading.com/furniture">
<!-- HTML Table -->
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<!-- Table as a piece of furniture -->
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

17.5 XML Schema (XSD)

<http://www.w3schools.com/schema>

<http://www.w3.org/TR/xmlschema-0>

<http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>

<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>

- Differences to DTD

XML schema language is well-formed, data types, inheritance, presentation, much more complex, order in schema mostly not significant

- XML Schema file (extension .xsd)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- data-structure -->
</xs:schema>
```

- see files <http://www.w3.org/2001/XMLSchema.xsd> and <http://www.w3.org/2001/XMLSchema.dtd>

- XML Document file (extension .xml) has additional attributes in the root element

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file-name.xsd"
```

- also URI (Uniform Resource Identifier) possible

- Simple elements contain only text, no nested elements and no attributes

```
<xs:element name="element-name" type="data-type"/>
- also default="default-value"
- also fixed="fixed-value"
- 'only text' misleading: can be different types, also custom types
- Example: <xs:element name="dateborn" type="xs:date"/>
```

- Primitive types (most common built-in types)

```
xs:string
xs:decimal
xs:integer
xs:boolean
xs:date (yyyy-mm-dd)
xs:time (hh:mm:ss)
```

- Derived by restriction types (restriction constraints also called facets)

```
<xs:restriction base="data type">
  <!-- constraining-facets -->
</xs:restriction>
```

| constraint | description |
|----------------|---|
| enumeration | list of acceptable values |
| fractionDigits | maximum number of decimal places allowed (≥ 0) |
| Length | exact number of characters or list items allowed (≥ 0) |
| maxExclusive | upper bounds for numeric values ($<$ this value) |
| maxInclusive | upper bounds for numeric values (\leq this value) |
| maxLength | maximum number of characters or list items allowed (≥ 0) |
| minExclusive | lower bounds for numeric values ($>$ this value) |
| minInclusive | lower bounds for numeric values (\geq this value) |
| minLength | minimum number of characters or list items allowed (≥ 0) |
| Pattern | exact sequence of characters that are acceptable |
| totalDigits | exact number of digits allowed (> 0) |
| whitespace | handling of white spaces (line feeds, tabs, spaces and carriage returns) is handled |

- Example:

anonymous custom type

```
<xs:element name="month">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="12"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

named custom type

```

<xs:element name="month" type="monthInt"/>
<xs:simpleType name="monthInt">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="12"/>
  </xs:restriction>
</xs:simpleType>

```

- Derived by list types

```

<xs:simpleType name="monthsInt">
  <xs:list itemType="monthInt"/>
</xs:simpleType>

```

- Derived by union types

```

<xs:simpleType name="monthName">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Jan"/>
    <xs:enumeration value="Feb"/>
    <xs:enumeration value="Mar"/>
    <!-- and so on -->
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="monthMixed">
  <xs:union memberTypes="monthInt monthName"/>
</xs:simpleType>

```

- Complex elements

- 4 kinds
 - no content (empty elements)
 - text-only content (elements that contain only text)
 - standard content (elements that contain only other elements)
 - mixed content (elements that contain both other elements and text)
- each may contain attributes

- Attributes (always declared as a simple type)

```

<xs:attribute name="attribute-name" type="data-type"/>

```

- also default="default-value"
- also fixed="fixed-value"
- also use="optional | prohibited | required" (default: optional)
- data-type also xs:ID, xs:IDREF, xs:IDREFS (legacy, constraint definitions: unique, key, keyref)
- Example: <xs:attribute name="lang" type="xs:string" default="EN"/>

- No content

anonymous custom type

```

<xs:element name="element-name">
  <xs:complexType>
    <xs:attribute name="attribute-name" type="data-type"/>
    <!-- more-attributes -->
  </xs:complexType>
</xs:element>

```

named custom type

```

<xs:element name="element-name" type="type-name"/>
<xs:complexType name="type-name">
  <xs:attribute name="attribute-name" type="data-type"/>
  <!-- more-attributes -->
</xs:complexType>

```


- Text-only content

anonymous custom type

```
<xs:element name="element-name">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="data-type">
        <xs:attribute name="attribute-name" type="data-type"/>
        <!-- more-attributes -->
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- also as named custom type
- also restriction instead of extension

- Standard content

anonymous custom type

```
<xs:element name="element-name">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="element-name" type="data-type"/>
      <!-- more-elements -->
    </xs:sequence>
    <xs:attribute name="attribute-name" type="data-type"/>
    <!-- more-attributes -->
  </xs:complexType>
</xs:element>
```

- also as named custom type

- Mixed content

same as standard content,
additional attribute `mixed="true"` in `<xs:complexType>`

- Indicators

- order indicators
 - sequence
 - all
 - choice
- occurrence indicators
 - `minOccurs` (default: 1, optional element: `minOccurs="0"`)
 - `maxOccurs` (default: 1, unlimited number of times: `unbounded`)
- group indicators
 - `group name`
 - `attributeGroup name`

- Referencing global elements and attributes

global: child of root-element

```
<xs:element ref="element-name">
<xs:attribute ref="attribute-name">
```

disadvantage: same type cannot be used with different elements or attributes

- unique / key (in an element, at the end)

unique: values of elements or attributes are unique within the scope

key: elements or attributes are a key (unique, non-nullable, and always present) within the scope

```
<xs:unique/key name="unique/key-name">
  <xs:selector xpath="XPath-expression">
  <xs:field xpath="XPath-expression">
  <!-- more-fields -->
</xs:unique/key>
```

- keyref (in an element, at the end)

values of attributes or elements correspond to values of the specified key

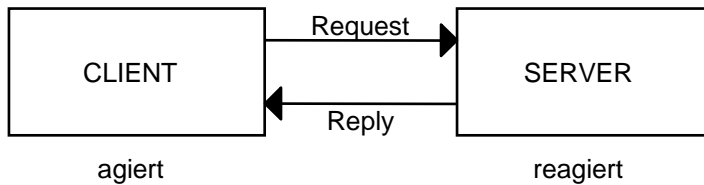
```
<xs:keyref name="keyref-name" refer="key-name">
  <xs:selector xpath="XPath-expression">
  <xs:field xpath="XPath-expression">
  <!-- more-fields -->
</xs:keyref>
```

- XPath – selecting nodes

| expression | description |
|------------|--|
| nodename | selects all nodes with the name "nodename" |
| / | selects from the root node |
| // | selects nodes in the document from the current node that match the selection, no matter where they are |
| . | selects the current node |
| .. | selects the parent of the current node |
| @ | selects attributes |

18 Client-Server-Architektur und Verteilte Datenbanken

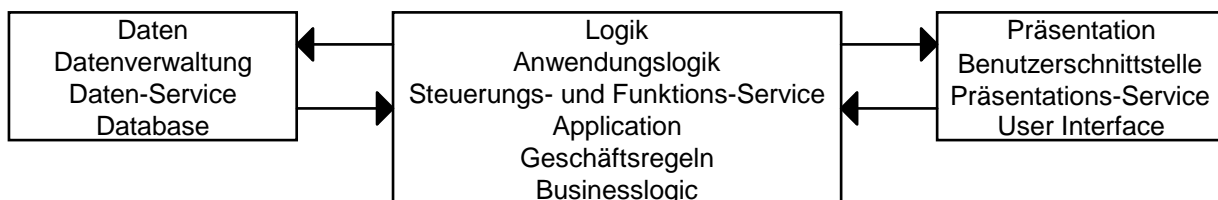
- Client (Diensteanforderer, Service Consumer, Kunde):
Programmmodul, das Dienste (Ressourcen) eines Servers anfordert (in Anspruch nimmt).
- Server (Dienstbringer, Service Provider, Lieferant):
Programmmodul, das Dienste (Ressourcen) einem / vielen Client(s) zur Verfügung stellt (die Anforderung erfüllt).



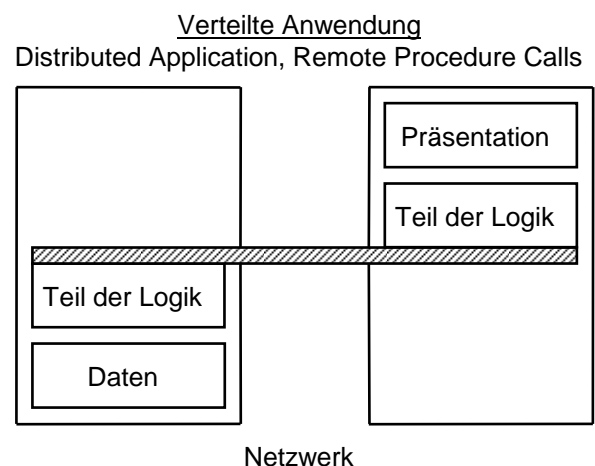
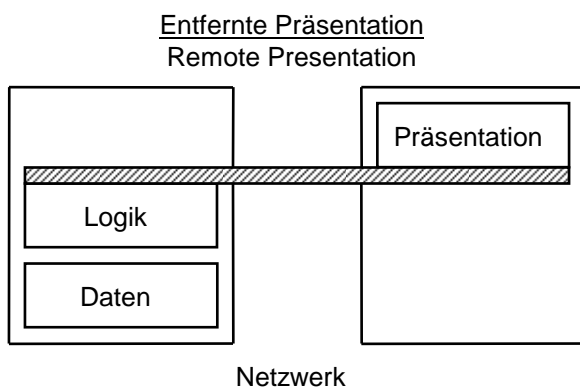
typischerweise:

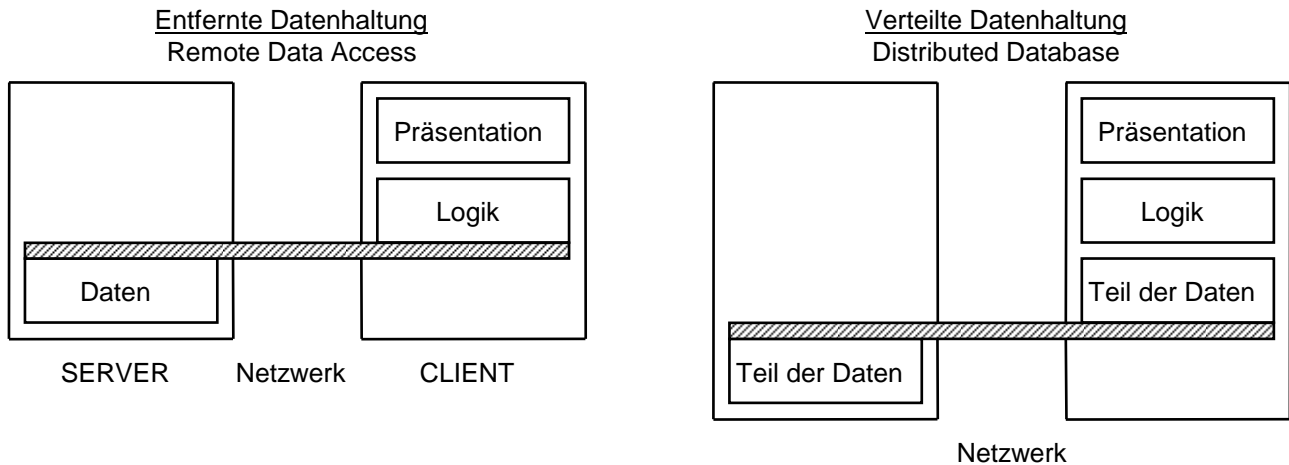
- eigene Rechner (zumindest eigene Prozesse / Tasks)
- ein Server für viele Clients

- Die Schnittstelle zwischen Client und Server (Anforderung des Clients - Reaktion des Servers) muss natürlich exakt definiert sein (für spezielle Client-Server-Schnittstellen liegen auch Normen vor)
- Distributed Processing (Parallel Processing): Client und Server laufen auf verschiedenen (durch ein Netzwerk geeignet verbundenen) Rechnern (= mindestens Prozessor + Hauptspeicher)
- Meist bedienen sich mehrere Clients eines Servers (Multi-User-Server), z.B. Print-Server, Fax-Server, Boot-Server
- In jedem Anwendungssystem werden 3 Komponenten (Funktionsblöcke) in entsprechenden Softwareteilen realisiert (Dreischicht-Modell, Three-Tier Architecture):



- In einer Client-Server-Architektur (Dreischicht-Architektur, Three-Tier-Architecture) werden diese Komponenten (Schichten) in getrennten, als Clients und Server zusammenarbeitenden, Softwaremodulen realisiert. Darüber hinaus können die einzelnen Komponenten auch noch in mehrere Programmodule aufgeteilt sein.
 - Host-Based-Solution: Anwendung läuft auf einem Rechner, in einem Programm (oder sogar Prozess) ab.
 - Client-Server-Based-Solution, Service-oriented Architecture, SOA: Für jede Komponente (Schicht) der Anwendung laufen ein oder mehrere Programme (Prozesse) im Rahmen eines Client-Server-Betriebs ab. Diese können
 - auf einem Rechner konzentriert sein (eher selten) oder
 - auf verschiedenen Rechnern verteilt sein (diese Rechner können auch unter verschiedenen Betriebssystemen laufen; Bezeichnung: Heterogene Systeme).
- Möglichkeiten der Aufteilung:





- Die Aufteilung in oben beschriebener Art kann natürlich auch kombiniert eingesetzt werden, z.B. Verteilte Datenhaltung und Verwendung von Presentationsservers (verbunden durch dieselbe Netzhard- und -software)
- Bei der Entfernten Datenhaltung können wieder verschiedene Konzepte unterschieden werden:
 - Platten- / Disk-Server (selten gebräuchlich)
Client → Server: Liefere von der Platte 'Volume-1' die Sektoren 5 bis 9
Client ← Server: 5 Sektoren
 - Datei- / File-Server
Client → Server: Liefere aus der Datei 'Bestand' Bytes 25 bis 130
Client ← Server: 106 Bytes
 - Datensatz- / Record-Server
Client → Server: Liefere aus der Datei 'Kunden' den Satz, der im Index 'Name' den Wert 'Maier' hat
Client ← Server: Satz
 - Datenbank- / Database-Server
Client → Server: Liefere den Durchschnitt des Gehalts aller Mitarbeiter aus Abteilungen in 'Wien'
Client ← Server: Durchschnittswert
Beispiel: Oracle Server von Oracle, SQLServer von Microsoft
- Vorteile von Client-Server-Architektur:
 - Herstellerunabhängigkeit bei genormten Schnittstellen
 - Einsatz von Standard-Hardware und Standard-Betriebssystemen (Kostenvorteil)
 - Jede Komponente ist für ihren Einsatz optimal geeignet (Datenbankrechner, Präsentationsrechner, etc.)
 - Schrittweise Kapazitätsanpassung ist möglich, bei Erhaltung der getätigten Investitionen
- Nachteile von Client-Server-Architektur:
 - Viele Komponenten (Ausfallsicherheit)
 - Teilweise komplizierte Abläufe und Overhead durch Einhaltung der Schnittstellen

18.1 Datenbank-Server

- Im Datenbankbereich wird unter dem Begriff Client-Server generell diese Konfiguration verstanden
- Am häufigsten eingesetzt als SQL-Server: Verringerung der Netzbelastung gegenüber File-Server-Architektur bei tabellenweiser Bearbeitung (wegen des bestehenden Overheads unter Umständen Verschlechterung bei intensiver satzweiser Bearbeitung).
- Client → Server: SQL-Anweisung
- Client ← Server: Ergebnistabelle und/oder Statuscode
- Begriffe:

| | |
|-------------------------------|---|
| Client | Server |
| Frontend | Backend |
| Workstation | Database-Server, Database-Engine, Datenbankmaschine |
| Requester | Responder, Engine |
| Arbeitsplatzrechner / -system | Hintergrundrechner / -system |

- In diesem Zusammenhang ist auch die Verwendung von Stored Procedures sehr effektiv

18.2 Verteilte Datenbanken

- Wenn die Objekte einer Datenbank auf mehrere Rechner (Systeme, Sites, Nodes) eines Netzwerks verteilt sind, spricht man von einer Verteilten Datenbank (Distributed Database).
- Möglichkeiten der Aufteilung (Fragmentation) einer relationalen Datenbank
 - Komplette Tabellen sind auf verschiedenen Systemen gespeichert.
 - Horizontale Aufteilung (Horizontal Fragmentation)
Die Zeilen einer Tabelle sind auf verschiedene Systeme verteilt.
Beispiel: In einer Firma mit mehreren Standorten sind in jedem Standortrechner die Mitarbeiter gespeichert, die an diesem Standort tätig sind.
 - Vertikale Aufteilung (Vertical Fragmentation)
Die Spalten einer Tabelle sind auf verschiedene Systeme verteilt (Primärschlüssel muss in jedem Teil enthalten sein).
Beispiel: Die Entlohnungsinformationen der Mitarbeiter sind im Rechner der Personalverwaltung gespeichert.
 - Kombinierte Aufteilung (Mixed Fragmentation)
Kombination aus Horizontaler und Vertikaler Aufteilung.
- Jede Anwendung muss einen Transparenten Zugriff auf die Verteilte Datenbank haben, d.h. von der Verteilung nichts merken und so arbeiten können, als handle es sich um eine zentrale Datenbank. Diese Forderung an Verteilte Datenbanksysteme wird Distribution Transparency genannt.
- Bei den oben beschriebenen Aufteilungen werden teilweise Redundanzen (Data Replication) zugunsten von Verfügbarkeit der Daten und Antwortzeiten in Kauf genommen (No Replication - Partial Replication - Fully Replication). Dabei darf nicht die Anwendung, sondern muss das Datenbanksystem diese Redundanzen verwalten. Diese Eigenschaft von Verteilten Datenbanksystemen wird als Replication Transparency bezeichnet. Die Verwaltung erfolgt durch Replication Server.
- Vorteile von Verteilten Datenbanken:
 - Das Datenelement ist dort gespeichert (und daher schnell zugreifbar) wo es am häufigsten verwendet wird, steht aber auch anderen Anwendungen im Netz zur Verfügung.
 - Wenn ein Rechner im Netz ausfällt, so ist nur ein Teil der Datenbank nicht verfügbar.
 - Schrittweise (rechnerweise) Erweiterung ist möglich.
- Probleme bei Verteilten Datenbanken:
 - Verwaltung des Datenkatalogs (Catalog Management)
Im Datenkatalog (Systemkatalog, Systemverzeichnis) wird festgehalten welche Tabellen es gibt, welche Attribute welchen Typs diese haben, Zugriffsberechtigungen, etc. Im Fall einer Verteilten Datenbank muss darüberhinaus abgelegt sein auf welchem Rechner welches Objekt gespeichert ist.
Es ist zu entscheiden wo und wie der Systemkatalog gespeichert ist.
Möglichkeiten:
 - auf einem Rechner der ganze Katalog (centralized)
 - auf jedem Rechner der ganze Katalog (fully replicated)
 - auf jedem Rechner ein Teil des Katalogs (partitioned)
 - diverse Kombinationen (z.B. jeder Rechner hat seinen eigenen Katalog, zusätzlich ein Rechner den ganzen Katalog)
 - Änderung der mehrfach gespeicherter Daten (Update Propagation)
Mechanismen müssen vorgesehen werden, dass Datenkopien immer auf demselben Stand sind.
Beispiele:
 - Wenn ein Rechner blockiert ist (z.B. durch einen Netzausfall) muss ein nachträgliches Update geschehen. Pending Updates müssen unmittelbar nach Ende der Blockade durchgeführt werden.
 - Es wird zwischen Primary-Kopien und Normal-Kopien unterschieden. Im Zweifelsfall müssen sich alle an der Primary-Kopie orientieren (wobei in Fehlersituationen die Verantwortung für die Primary-Kopie auch von einem anderen Rechner übernommen werden können muss).

- Abarbeitung von Abfragen (Query Processing)

Je nachdem, wo die Daten gespeichert sind und wie eine Abfrage abgearbeitet wird, kann es zu völlig unterschiedlichen Antwortzeiten kommen (wobei im Vorhinein nicht entschieden werden kann, welche Strategie die beste ist).

Beispiel: Datenbank Lieferanten-Teile-Lieferungen

L: 10,000 Zeilen Rechner A; T: 100,000 Zeilen Rechner B; LT 1,000,000 Zeilen Rechner A;

Abfrage:

```
SELECT L.LNR FROM L, LT, T WHERE L.STADT='London' AND T.FARBE='rot' AND  
LT.LNR=L.LNR AND LT.TNR=T.TNR
```

Es gäbe 10 Teile mit der Farbe rot und 100,000 Lieferungen von Londoner Lieferanten.

Die Antwortzeiten können (je nach Abarbeitung der Abfrage) um ca. einen Faktor 200,000 variieren.

- Recovery

Weil eine Transaktion Änderungen in der Datenbank vornehmen kann, die auf verschiedenen Rechnern gespeichert sind (Verteilte Transaktion), wird meist das Zwei-Phasen-Commitprotokoll (Two-Phase-Commit) eingesetzt:

- Die beteiligten Transaktionsmanager werden als Teilnehmer bezeichnet, einer von ihnen ist der Koordinator
- Phase 1 (Pre-Commit Phase, Wahlphase):
 - Koordinator sendet eine Vote-Request-Nachricht (prepare to commit) an jeden Teilnehmer
 - Jeder Teilnehmer antwortet mit Ja (ready to commit) oder Nein; bei Nein bricht der Teilnehmer die Transaktion autonom ab.
- Phase 2 (Post-Commit Phase, Entscheidungsphase):
 - Koordinator sammelt die Votes. Laufen alle (seines eingeschlossen) Ja, entscheidet er auf Commit und sendet entsprechende Commit-Nachrichten an alle Teilnehmer. Andernfalls (auch wenn nur ein Nein gemeldet wurde) entscheidet er auf Abort und teilt dies den Teilnehmern mit, von denen Ja empfangen wurde.
 - Jeder Teilnehmer, der mit Ja votiert hat wartet auf die Commit- oder Abort-Nachricht des Koordinators. Bei Eintreffen der Antwort wird die entsprechende Aktion ausgeführt.

Mit dieser Vorgangsweise wird erreicht, dass entweder alle Teilnehmer die Transaktion ordnungsgemäß beenden (commiten) oder alle sie zurücknehmen (aborten).

Kritik: Die Zeit während Absetzen des Ja-Votums und Erhalt der Entscheidung vom Koordinator ist 'unsicher' (Probleme, wenn in dieser Zeit ein Kommunikationsfehler auftritt oder der Rechner ausfällt). Als Lösung wird ein Drei-Phasen-Commitprotokoll oder die Verwendung eines Replication-Servers vorgeschlagen.

- Steuerung des Parallelen Zugriffs (Concurrency Control)

Locking- und Timestamping-Strategien sind möglich.

- Vorteil Locking: Höhere Parallelität
- Vorteil Timestamping: Keine Deadlock-Situationen

Da Deadlocks in Verteilten Datenbanken besonders unangenehm zu behandeln sind (Globale Deadlocks über mehrere Rechner) wird eher dem Timestamping-Verfahren der Vorzug gegeben.

19 Datenkomprimierung

- Datenkomprimierung, Datenkompression: Daten(codierung) verkürzen, ohne dass Information verloren geht.
- Vorteile:
 - Platzersparnis bei Speicherung
 - Zeitersparnis bei Übertragung
 - Zusammenfassung von Daten (mehrere Dateien in einem Archiv)
- Nachteile:
 - Zeit für Komprimierung
 - Zugriff langsamer

19.1 Lauflängencodierung

- Run-Length-Encoding; Packing
- Prinzip: Mehrfaches Auftreten des gleichen Zeichens unmittelbar hintereinander wird durch einmalige Angabe des Zeichens, zusammen mit einem Zähler, codiert (Sequenz z.B. Steuerzeichen - Zeichen - Anzahl).
- Beispiel

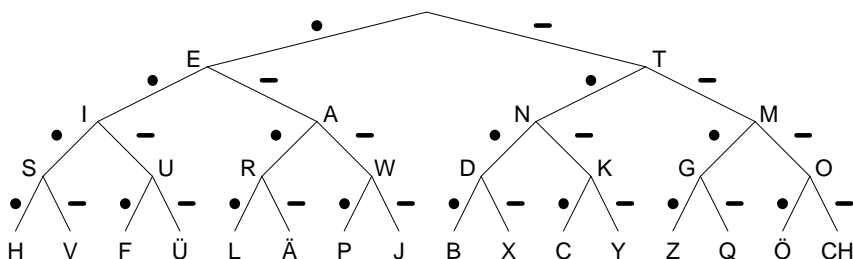
| | | |
|--------------|---|--------------|
| | A A B B B B B B B B B B C C C D D D D | (19 Zeichen) |
| verkürzt zu: | A A 0x01 B 0x0A C C C 0x01 D 0x04 | (11 Zeichen) |
| oder: | A A 0x01 B 0x0A 0x01 C 0x03 0x01 D 0x04 | (11 Zeichen) |
- Sinnvoll erst ab mehr als 3-maligem Auftreten des gleichen Zeichens.
- Codierung des Steuerzeichnes als Datenzeichen benötigt mehr Platz (z.B. 0x01 0x01 1)

19.2 Codieren in variabler Länge

- Variable-Length-Encoding; Squeezing
- Prinzip: Häufig vorkommende Zeichen werden mit einer kurzen Bitfolge, selten vorkommende mit einer langen Bitfolge codiert
- Beispiel: SCHUBIDUBIDU

| Zeichen | Häufigkeit | Code |
|---------|------------|------|
| S | 1 | 100 |
| C | 1 | 11 |
| H | 1 | 000 |
| U | 3 | 0 |
| B | 2 | 1 |
| I | 2 | 10 |
| D | 2 | 01 |

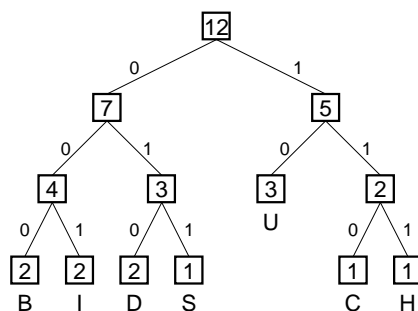
- Problem: wo endet die Codierung eines Zeichens, wo beginnt die Codierung des nächsten Zeichens?
Lösung:
 - Trennzeichen
 - Fano-Bedingung, Präfix-Bedingung: keine Bitfolge (kein Code) eines Zeichens entspricht der Anfangsbitfolge (dem Codebeginn) eines anderen Zeichens.
- Beispiel: Im Morse-Code ist die Fano-Bedingung nicht eingehalten (die zu verschlüsselnden Zeichen stehen nicht nur an den Blättern des Code-Baums). Hier wird die Trennung der einzelnen Zeichen durch eine Pause vorgenommen.



- **Huffman-Codierung:** Verfahren zur Codierung der Zeichen in variabler Länge mit Einhaltung der Fano-Bedingung
 - 1) Ordne die im Text vorkommenden Zeichen absteigend nach der Häufigkeit des Vorkommens in einer Tabelle an.
 - 2) Erzeuge über den beiden Zeichen mit der geringsten Häufigkeit einen Knoten des Code-Baums, der die summierte Häufigkeit der beiden Zeichen enthält.
 - 3) Lösche die beiden Zeichen aus der Tabelle und sortiere den neuen Knoten nach seiner Summenhäufigkeit in die Tabelle ein.
 - 4) Führe Schritt 2 und 3 solange fort, bis nur mehr ein Eintrag in der Tabelle verbleibt (Baumwurzel).

- Beispiel: SCHUBIDUBIDU

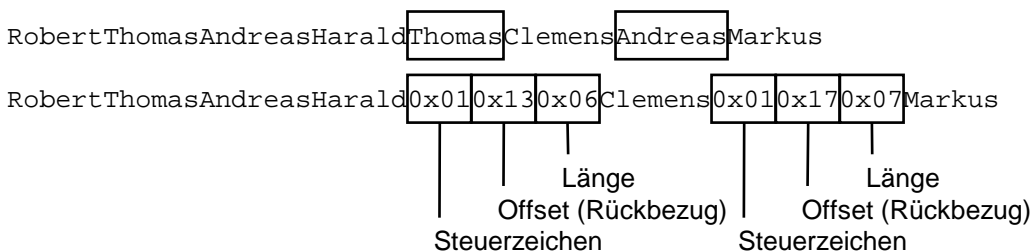
| Zeichen | Häufigkeit | Code |
|---------|------------|------|
| U | 3 | 10 |
| B | 2 | 000 |
| I | 2 | 001 |
| D | 2 | 010 |
| S | 1 | 011 |
| C | 1 | 110 |
| H | 1 | 111 |



- Zur Festlegung der jeweiligen Codierung sind die Zeichenhäufigkeiten zu ermitteln. Dafür gibt es drei Vorgangsweisen.
 - Statisch: Häufigkeiten sind fix festgelegt.
 - Dynamisch: Daten werden einmal durchgelesen und die Häufigkeiten ermittelt (Codetabelle muss bei den Daten mitabgespeichert werden).
 - Adaptierend: Am Beginn werden Häufigkeiten angenommen (z.B. alle Zeichen kommen gleich oft vor), während der Codierung wird der Code angepasst (berücksichtigt damit auch verschiedene Arten von Textteilen).

19.3 LZW-Verfahren

- nach den Erfindern Lempel, Ziv und Welch; Crunching
- Prinzip: Wiederholt vorkommende Zeichenfolgen (Phrasen) werden mit eigenen Codes versehen und unter Angabe dieses Codes abgespeichert (oder Verweis auf schon einmal aufgetretene Zeichenfolgen).
- Beispiel:



20 Kryptologie Cryptographie

- Die **Kryptologie** ist die **Wissenschaft** der **Verschlüsselung** und der **Entschlüsselung** von **Informationen**.
Unterteilung:

- **Entwurf** von Kryptosystemen: **Kryptographie**
- **Entschlüsselung** von Informationen (ohne Kenntnis des Schlüssels): **Kryptoanalyse**
- **Verstecken** von Informationen: **Steganographie** **Verschleierung**

- Anwendungen (Beispiele):

- Verschlüsselung des Passworts beim Abspeichern
- Verschlüsselung von Dateiinhalten oder Backup-Daten (auf Backup-Medien)
- Verschlüsselung bei der Datenübertragung (in Netzen)

- **Grundbegriffe:**

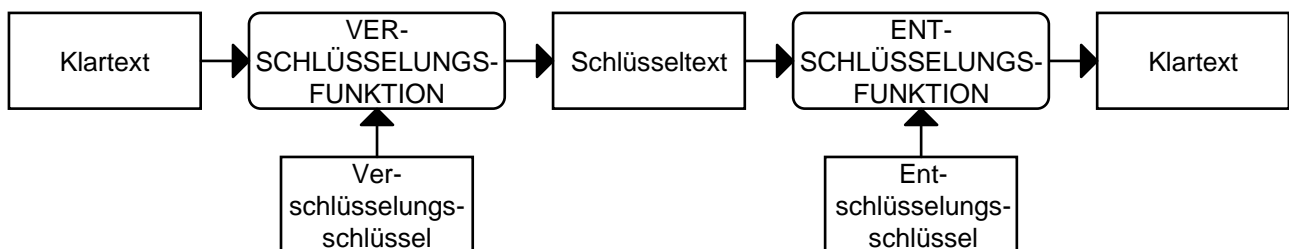
Die zu verschlüsselnden Daten werden **Klartext** (Plaintext) genannt. Dieser wird durch eine **Funktion f** (Abbildung, Verschlüsselungsfunktion) in einen **Schlüsseltext** (**Chiffretext**, **Ciphertext**) umgewandelt (Vorgang: Verschlüsseln, Chiffrieren, Encryption). Die Funktion wird von einem **Schlüssel** (Key, Verschlüsselungsschlüssel, Encryption Key) **parametrisiert**:

$f(\text{Klartext}, \text{Verschlüsselungsschlüssel}) = \text{Schlüsseltext}$

Unter Anwendung der **Umkehrfunktion** f^{-1} (Entschlüsselungsfunktion) **kann** (bei bekanntem Schlüssel, Entschlüsselungsschlüssel, Decryption Key) **aus** dem **Schlüsseltext** **wieder** der **Klartext** **abgeleitet** werden (Vorgang: Entschlüsseln, Dechiffrieren, Decryption):

$f^{-1}(\text{Schlüsseltext}, \text{Entschlüsselungsschlüssel}) = \text{Klartext}$

- Graphische Darstellung:



- Meist werden die Funktionen so gewählt, dass Ver- und Entschlüsselungsschlüssel dergleiche sein kann
- **Block-Verschlüsselungsverfahren** (Block Cipher): Jeweils ein **Block fester Größe** des Klartexts wird mit Hilfe der, durch den Schlüssel parametrisierten, Verschlüsselungsfunktion in einen Block des Schlüsseltexts transformiert. Problem: Wenn der **Klartext** **Muster** mit einer **Periode** enthält, die ein **ganzzahliges Vielfaches** oder ein **ganzzahliger Bruchteil** der **verwendeten Blocklänge** ist, ist eine **Analyse** der **Häufigkeitsverteilung** der Blöcke möglich.
- **Strom-Verschlüsselungsverfahren** (Stream Cipher): Der ganze **Klartext** wird als **Zeichenstrom** aufgefasst und **insgesamt verschlüsselt**. Aus dem **Schlüssel** wird nach einem **definierten Verfahren** ein **Schlüsselstrom** (Key Stream), **derselben Länge** wie die des Klartexts, erzeugt. Die **Verschlüsselung** erfolgt **kontinuierlich**, aus jeweils **einem Teil** des **Klartexts** und des **Schlüsselstroms**. Eine **Untersuchung** von **Häufigkeitsverteilungen** ist damit **wesentlich erschwert**.
Problem: **Synchronisation** des **Schlüsselstroms** bei der Ver- und Entschlüsselungsfunktion (Sender und Empfänger). Geht durch einen **Fehler** (bei einer Übertragung) ein **Teil** des **Schlüsseltexts** **verloren**, wird der **Rest** mit einem **anderen Schlüsselstrom** **entschlüsselt** als **verschlüsselt**.
- **Anzahl** der **Klartextzeichen**, die (in einem Verschlüsselungsschritt) verwendet werden
Eines: **Monographische Verschlüsselung**
Mehrere: **Polygraphische Verschlüsselung**
- **Einfachstes** Verfahren der Kryptoanalyse ist die **Brute-Force-Methode**:
Durchprobieren aller **Schlüssel**. Schlüssel muss **so lang gewählt** werden, dass dieses Verfahren wegen der vielen Möglichkeiten viel **zu lange** dauert.

20.1 Einfache Methoden

- **Substitution (Ersetzungsverfahren):**
Jedes Zeichen (oder jede Zeichenfolge) des Klartexts wird durch ein anderes Zeichen (oder andere Zeichenfolge) ersetzt.
- **Monoalphabetische Substitution:**
Gleiche Zeichen des Klartexts werden durch gleiche Zeichen im Schlüsseltext ersetzt.
 - **Sicherheit:** Durch Analyse der Häufigkeitsverteilungen von einzelnen Zeichen, Zeichenpaaren und -tripel relativ einfach zu entschlüsseln.
 - **Beispiel: Cäsar-Code**
Jeder Buchstabe wird durch den drittnächsten Buchstaben im Alphabet (zyklisch) ersetzt (es wird a zu d, b zu e, ..., z zu c).
 - **Beispiel: Übersetzungstabelle**
 Klartext: a b c d e f g h i j k l m n o p q r s t u v w x y z
 Schlüsseltext: q w e r t z u i o p a s d f g h j k l y x c v b n m
- **Polyalphabetische Substitution:**
Gleiche Zeichen des Klartexts werden durch verschiedene Zeichen im Schlüsseltext ersetzt.
 - **Sicherheit:** Gegenüber der monoalphabetischen Variante werden die Häufigkeitsverteilungen besser verschleiert, sind durch statistische Analysen aber trotzdem zu entschlüsseln.
 - **Beispiel: Vigenère-Verschlüsselung (Blaise de Vigenère 1523-1596)**
Verschlüsselungstabelle besteht aus 26, verschieden angeordneten, Alphabeten. Jedes Zeichen des Verschlüsselungsschlüssels wählt eine Zeile aus, jedes Zeichen des Klartexts eine Spalte, im Kreuzungspunkt steht das Schlüsseltextzeichen. Wenn der Verschlüsselungsschlüssel kürzer als der Klartext ist (trifft üblicherweise zu), muss der Verschlüsselungsschlüssel entsprechend oft wiederholt werden.

| | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| b | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a |
| c | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b |
| d | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c |
| e | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d |
| f | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e |
| g | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f |
| h | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g |
| i | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h |
| j | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i |
| k | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j |
| l | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k |
| m | m | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l |
| n | n | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m |
| o | o | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n |
| p | p | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| q | q | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
| r | r | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q |
| s | s | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r |
| t | t | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s |
| u | u | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t |
| v | v | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u |
| w | w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v |
| x | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w |
| y | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x |
| z | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y |

Klartext: d e r w u n s c h n a c h g e h e i m h a l t u n g
 Schlüssel: i n f o r m a t i k i n f o r m a t i k i n f o r m
 Schlüsseltext: l r w k l z s v p x i p m u v t e b u r i y y i e s

- Durch Substitution von ganzen Zeichenfolgen anstatt einzelner Zeichen können Häufigkeitsanalysen zusätzlich erschwert werden (Beispiel: Porta-Verschlüsselung)

- **Transposition** (Permutation, Versetzungsverfahren):

Die Reihenfolge der Zeichen (oder Zeichengruppen) des Klartexts wird verändert.

- **Sicherheit:** Da Transpositionen nur jeweils auf (relativ kurze) Zeichenfolgen (Teilfolgen) des Klartexts angewendet werden können, ist durch entsprechendes Analysieren eine Entschlüsselung möglich.

- **Beispiel:** Skytala der Spartaner (ca. 500 v.Chr.)

Ein fingerbreiter Pergamentstreifen wurde auf einen Holzstab (die Skytala) gewickelt und mit dem Klartext in Längsrichtung des Stabes beschrieben. Der Klartext konnte nur wieder von einem Empfänger gelesen werden, der einen Holzstab des gleichen Durchmessers besaß.

- Beispiel: Der Verschlüsselungsschlüssel ist ein Wort ohne Buchstabenwiederholungen (z.B. megabuck). Die relativen Stellungen der Buchstaben dieses Wortes im Alphabet stellen die Transposition dar.

m e g a b u c k

7 4 5 1 2 8 3 6

d e r w u n s c

h n a c h a u f

l o e s u n g x

Klartext: d e r w u n s c h n a c h a u f l o e s u n g x

Schlüsseltext: w c s u h u s u g e n o r a e c f x d h l n a n

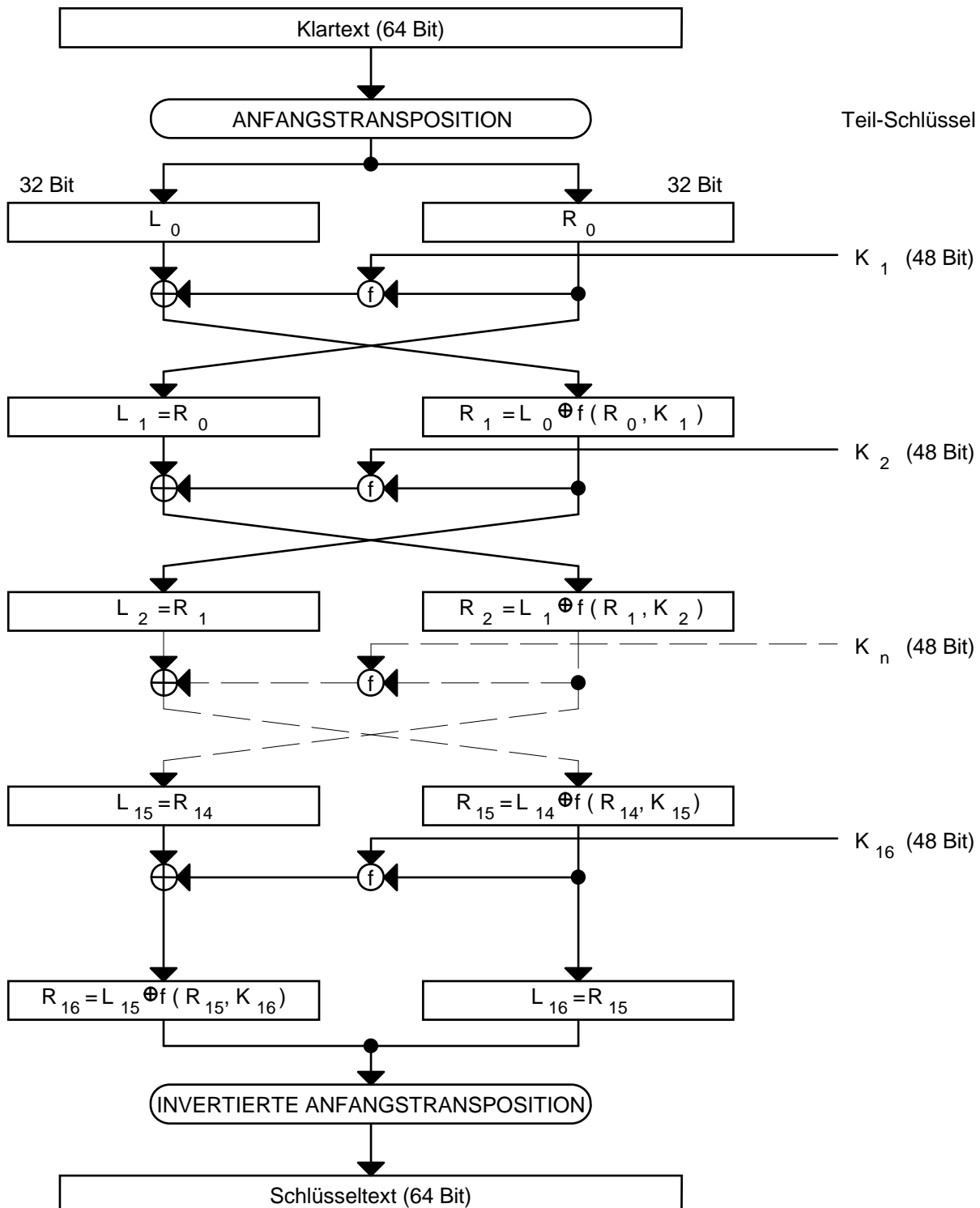
- **Vernam-Verfahren** (Einmalschlüssel, One-Time-Pad) ca. 1925, Gilbert Vernam (1890-1960):

Verschlüsselungsverfahren, das theoretisch absolut sicher ist.

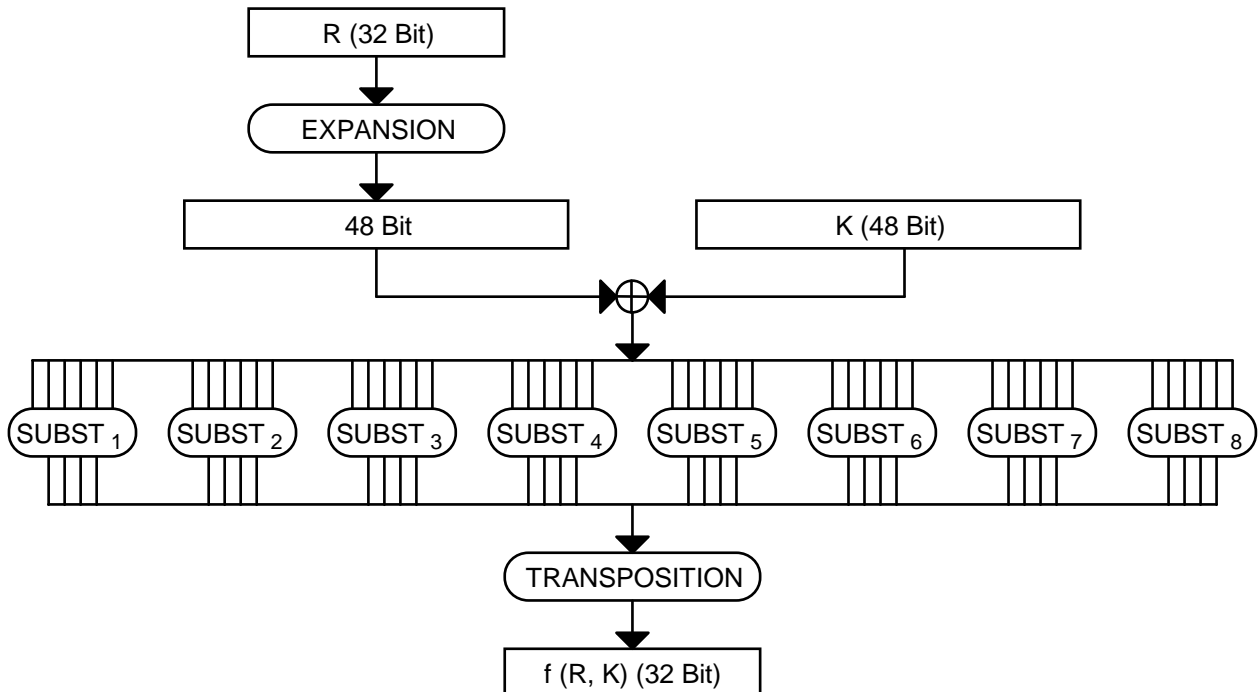
Es wird ein Schlüssel gewählt, der so lang wie der Klartext ist und der aus einer zufälligen Bitfolge besteht. Der Schlüsseltext wird durch bitweises Exklusiv-Oder von Klartext und Schlüssel gebildet. Die Bitfolge im entstandenen Schlüsseltext stellt wieder eine reine Zufallsfolge dar und kann daher nicht aufgebrochen werden. In der Praxis besteht die Schwierigkeit in der Verwaltung von beliebig langen, zufälligen Bitfolgen. Pseudo-Zufallszahlenfolgen, wie sie von Rechnern algorithmisch erzeugt werden, haben eine (wenn auch große, aber trotzdem) endliche Periode und sind daher nicht absolut sicher. Um nicht immer die gleiche Bit-Folge vom Pseudo-Zufallszahlengenerator erzeugen zu lassen, kann der Startwert des Generators mit einem Schlüssel parametrisiert werden.

20.2 Die Datenverschlüsselungsnorm DES (Data Encryption Standard)

- Von IBM entwickelt, 1977 vom National Bureau of Standards veröffentlicht, auch gut für hardwaremäßige Implementierung geeignet (En-/Decryption-Chips, Geschwindigkeit!)
- Besteht im Wesentlichen aus wiederholten Anwendungen von Substitutionen und Transpositionen
- Jeweils 8 Zeichen (64Bit) des Klartexts werden mit einem, für die Verschlüsselung des gesamten Klartexts gültigen, Schlüssel von 7 Zeichen (56Bit) in 8 Zeichen (64Bit) Schlüsseltext umgesetzt (der Schlüssel wird bei seiner Anwendung um 8 Paritäts-Bits verlängert)
- Schema:



- Zunächst erfolgt eine, vom Schlüssel unabhängige, Transposition des Klartexts
- Dann werden 16 Transformationen durchgeführt, die (bis auf die letzte) in ihrer Funktionsweise identisch sind; für jede Stufe wird allerdings ein, aus dem Schlüssel abgeleiteter, unterschiedlicher Teilschlüssel verwendet. Die Komplexität liegt in der Funktion $f(R, K)$, die aus dem rechten Teil der Vorstufe (R , 32Bit) und dem entsprechenden Teilschlüssel (K , 48Bit) ein Ergebnis mit 32Bit liefert.
- Zum Schluss wird die, zur Transposition des ersten Schrittes, inverse Transposition angewendet
- Berechnung der $f(R, K)$:



- Folgende Informationen sind im DES tabellenmäßig definiert:
 - 1 Anfangstransposition (64Bit)
 - Auswahl der 16 Teilschlüssel aus dem Schlüssel (48Bit aus 64Bit)
 - 1 Expansion in f (32Bit auf 48Bit)
 - 8 Substitutionen in f (6Bit in 4Bit)
 - 1 Transposition in f (32Bit)
- Die Entschlüsselung erfolgt mit demselben Schlüssel und demgleichen Verfahren wie die Verschlüsselung, jedoch wird mit der invertierten Anfangstransposition begonnen und werden die 16 Teilschlüssel in umgekehrter Reihenfolge angewendet
- Betriebsarten des DES:
 - ECB-Modus (Electronic Code Book, Elektronisches Codebuch): Der Klartext wird in Blöcke von je 8 Zeichen eingeteilt und jeder Block separat verschlüsselt (Block-Verschlüsselung). Diese Methode produziert identische Schlüsseltextblöcke bei gleichen Klartextblöcken.
 - CFM-Modus (Cipher Feedback Mode, Schlüsselrückführung): 8-Bit (oder mehr) eines jeden Schlüsseltextblocks werden durch eine Exklusiv-Oder-Funktion mit dem nächsten Klartextblock vor dessen Verschlüsselung verknüpft. Derselbe Klartext erscheint damit jedesmal als unterschiedlicher Schlüsseltext.
 - CBC-Modus (Cipher Block Chain, Schlüsselblockkette): Jeder Schlüsseltextblock wird als Ganzes zum nächsten Klartextblock rückgekoppelt. Damit ist jeder verschlüsselte Datenblock eine auch Funktion des vorhergegangenen Datenblocks.
- Zeitverhalten: 2^{56} (ca. 70 Milliarden) verschiedene Schlüssel gibt es; wenn pro Sekunde 1 Million Überprüfungen durchgeführt werden können und im Durchschnitt die Hälfte der möglichen Schlüssel ausprobiert werden, dann dauert dies mehr als 1000 Jahre (10000 solcher Chips, in einem Parallelcomputer vereinigt, würden ca. 40 Tage brauchen)
- Kritik:
 - Geringe Länge des Schlüssels. Ursprünglicher IBM-Vorschlag war 128 Bits / 16 Zeichen, dieser wurde vom US-Geheimdienst NSA (National Security Agency) verhindert.
 - Blocklänge nur 64 Bits
 - Es sind eine Anzahl 'Schwacher Schlüssel' bekannt, bei deren Anwendung eine Entschlüsselung möglich ist.

- 3DES (Triple-DES): DES mehrmals hintereinander mit zwei verschiedenen Schlüsseln (Schlüssel praktisch 112 Bit lang)
- Als neuer Standard im Jahr 2000 festgelegt:
AES (Advanced Encryption Standard), nach seinen Entwicklern Joan Daemen und Vincent Rijmen (Belgier) auch Rijndael-Algorithmus genannt. Symmetrisches Blockverschlüsselungsverfahren.
Schlüssellänge: 128, 192 oder 256 Bit; Blocklänge: 128 Bit

20.3 Systeme mit öffentlichen Schlüsseln (Public-Key Systems)

- Die bisher besprochenen Verfahren haben den Nachteil, dass die verwendeten Schlüsseln zwischen dem, der den Klartext verschlüsselt (Sender) und dem, der den Schlüsseltext entschlüsselt (Empfänger), ausgetauscht werden müssen. Dieser Schlüsselaustausch stellt naturgemäß ein Sicherheitsrisiko dar, vor allem, wenn die Schlüssel zur Gewährleistung der Geheimhaltung häufig geändert werden müssen.
- Im Prinzip funktioniert ein Public-Key Verfahren folgendermaßen (Idee von Diffie und Hellman, 1976): Jedem Teilnehmer sind zwei verschiedene, jedoch nach mathematischen Regeln zusammengehörige, Schlüssel zugeordnet:
 - ein öffentlicher Schlüssel: Steht jedem zur Verfügung, der dem betreffenden Teilnehmer eine Nachricht zukommen lassen möchte. Die öffentlichen Schlüssel aller Teilnehmer sind in einem allgemein zugänglichen Verzeichnis vermerkt ('Telefonbuch'). Dieser Schlüssel wird üblicherweise zum Verschlüsseln benutzt.
 - ein privater Schlüssel (Geheimschlüssel): Kennt nur der Teilnehmer selbst. Dieser Schlüssel wird üblicherweise zum Entschlüsseln benutzt.
 Aufgrund dieser Tatsache werden Public-Key Systeme auch als Zweischlüssel-Systeme bezeichnet (die bisher besprochenen Verfahren können demnach als Einschlüssel-Systeme bezeichnet werden).
- Alle Methoden, die aus der mathematischen Klasse der Falltür-Probleme (Trap-Door-Problems) stammen, sind bei der Anwendung von Public-Key Systemen geeignet. In der Folge wird das verbreitetste Verfahren, der RSA-Code, beschrieben.
- RSA-Code:
 - Benannt nach den Anfangsbuchstaben seiner drei Erfinder: Ronald Rivest, Adi Shamir, Leonard Adleman (1978)
 - Das System beruht auf der Multiplikation zweier sehr großer Primzahlen (je ca. 100 Stellen). Es ist vergleichsweise einfach, zwei große Primzahlen zu multiplizieren, jedoch sehr aufwendig, die beiden Primzahlen zu ermitteln, wenn nur das Produkt (ca. 200 Stellen) bekannt ist (Faktorisierungsproblem). Beispiel: Um 29083 manuell in seine beiden Primfaktoren zu zerlegen wird man etwa eine halbe Stunde brauchen. Die Multiplikation von 127 mit 229 dauert manuell etwa eine Minute.
 - Zwei verschiedene, möglichst (je ca. 100 Stellen) große, Primzahlen p und q werden ausgewählt:
 $n = p \cdot q$, $z = (p-1) \cdot (q-1)$
 - Eine möglichst große Zahl e wird ausgewählt, die teilerfremd zu z ist (z.B. eine Primzahl, die sowohl größer als p , als auch größer als q ist)
 - d wird bestimmt als das Multiplikative Inverse von e modulo z ($d \cdot e \text{ modulo } z = 1$)
Methode: Erweiterter Euklidischer Algorithmus (günstiges Zeitverhalten)
 - Öffentlicher Schlüssel: e, n
 - Privater Schlüssel: d, n (n nicht notwendig, ist ohnehin öffentlich bekannt)
 - Verschlüsselung: $C = P^e \text{ modulo } n$
 - Entschlüsselung: $P = C^d \text{ modulo } n$
 - Die Sicherheit der Methode hängt damit zusammen, dass die beiden Faktoren p und q des bekannten n (innerhalb eines brauchbaren Zeitrahmens) nicht ermittelt werden können (sonst könnten auch z , sowie aus z und e auch d berechnet werden)

- Beispiel (mit sehr kleinen Zahlen):

$p=3, q=5$; daher $n=15, z=8; e=11$; daher $d=3; P=13$

$C = 13^{11} \text{ modulo } 15 = 1792160394037 \text{ modulo } 15 = 7$

$P = 7^3 \text{ modulo } 15 = 343 \text{ modulo } 15 = 13$

Rechengang: Zerlegen des Exponenten in Zweierpotenzen, es gilt

$(a*b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$

Module der Faktoren multiplizieren, statt Modul des Produkts ermitteln

$C = 13^{11} \text{ modulo } 15 = 13^8 * (13^4) * 13^2 * 13^1 \bmod 15 =$
 $1 * (1) * 4 * 13 \bmod 15 = 7$

$P = 7^3 \text{ modulo } 15 = 7^2 * 7^1 \bmod 15 =$
 $4 * 7 \bmod 15 = 13$

- Beispiel (mit kleinen Zahlen):

$p=73, q=151$; daher $n=11023, z=10800; e=11$; daher $d=5891$;

$P = \text{H o w _ a r e _ y o u ?}$

33 14 22 62 00 17 04 62 24 14 20 66

$a,b,...,z \rightarrow 00, 01, ..., 25$

$A,B,...,Z \rightarrow 26, 27, ..., 51$

Sonderzeichen $\rightarrow 52, ..., 66$

Block: 2 Zeichen, da 6666 kleiner 11023, jedoch 666666 nicht mehr kleiner 11023 ist.

$C1 = 3314^{11} \text{ modulo } 11023 = 10260$

$C2 = 2262^{11} \text{ modulo } 11023 = 9489$

$C3 = 17^{11} \text{ modulo } 11023 = 1782$

$C4 = 462^{11} \text{ modulo } 11023 = 727$

$C5 = 2414^{11} \text{ modulo } 11023 = 10032$

$C6 = 2066^{11} \text{ modulo } 11023 = 2253$

$P1 = 10260^{5891} \text{ modulo } 11023 = 3314$

$P2 = 9489^{5891} \text{ modulo } 11023 = 2262$

$P3 = 1782^{5891} \text{ modulo } 11023 = 17$

$P4 = 727^{5891} \text{ modulo } 11023 = 462$

$P5 = 10032^{5891} \text{ modulo } 11023 = 2414$

$P6 = 2253^{5891} \text{ modulo } 11023 = 2066$

- Zeitverhalten: Berechnung von p, q und d aus n und e

Stellen von n Rechenoperationen Rechenzeit (1 Operation in 10^{-6} Sekunden)

50 $1,4 \cdot 10^{10}$ 3,9 Stunden

70 $9,0 \cdot 10^{12}$ 104 Tage

100 $2,3 \cdot 10^{15}$ 74 Jahre

200 $1,2 \cdot 10^{23}$ $3,8 \cdot 10^9$ Jahre

- Anzahl großer Primzahlen: Es gibt ca. $10^{60} / \ln(10^{60})$ Primzahlen mit 60 Stellen (Gaußsches Primzahlentheorem). Damit hätte jedes Atom der Erde (ca. 10^{50}) seine eigenen zwei Primzahlen.

- Digitale Unterschrift (Authentifikation):

- Bei Verwendung von Public-Key Systemen kann (auf Grund der Eigenschaften der verwendeten mathematischen Funktionen) die Nachricht P vom Sender A so verschlüsselt werden ('unterschrieben werden'), dass der Empfänger B sicher sein kann, die verschlüsselte Nachricht C tatsächlich vom Sender A erhalten zu haben (A kann im Streitfalle nicht leugnen, dass die Nachricht von ihm stammt).
- Sei $E_X(Y)$ das Ergebnis des Verschlüsselungsvorgangs mit dem öffentlichen Schlüssel des Teilnehmers X angewendet auf den Text Y und $D_X(Y)$ das Ergebnis des Entschlüsselungsvorgangs mit dem privaten Schlüssel des Teilnehmers X angewendet auf den Text Y.
- Verschlüsselung durch A: $C = E_B(D_A(P))$; wendet vor der eigentlichen Verschlüsselung die Entschlüsselung mit seinem privaten Schlüssel an.
- Entschlüsselung durch B: $P = E_A(D_B(C))$; wendet nach der eigentlichen Entschlüsselung die Verschlüsselung mit dem öffentlichen Schlüssel von A an.
- Beispiel (RSA-Code):
Öffentlicher Schlüssel von A: $(3, 55=5*11)$ Privater Schlüssel von A: $(27, 55)$
Öffentlicher Schlüssel von B: $(5, 119=7*17)$ Privater Schlüssel von B: $(77, 119)$

Verschlüsselung durch A: $P = 19$; $C' = 19^{27} \text{ modulo } 55 = 24$; $C = 24^5 \text{ modulo } 119 = 96$

Entschlüsselung durch B: $P' = 96^{77} \text{ modulo } 119 = 24$; $P = 24^3 \text{ modulo } 55 = 19$

- Anwendungsgebiete: Telebanking, Juristische Anwendungen
- Kritik: Public-Key Systeme sind relativ langsam, da es sich um reine Softwareimplementierungen handelt (z.B. 80 Zeichen zu verschlüsseln dauert mit RSA 100 mal so lange als die entsprechende DES-Verschlüsselung)
- PGP - Pretty Good Privacy: Konkretes System, das auch die RSA-Methode verwendet
- Neueres Verfahren für Public-Key Verschlüsselung:
Elliptic Curve Cryptography (ECC), Neal Koblitz und Victor Miller (1986):
 - Angriff erfordert noch wesentlich mehr Rechenaufwand als bei RSA
 - Es können kleinere Zahlen als beim RSA eingesetzt werden (Smartcards, PDAs, etc.)
 - 1024 Bit RSA entspricht 160 Bit ECC (allerdings höherer Rechenaufwand)

21 Grundlagen der Datenorganisation

21.1 Daten und deren Organisationseinheiten

- Daten (data): Darstellung von Informationen (d.h. Angaben über Sachverhalte oder Vorgänge) in einer maschinell verarbeitbaren Form (nach DIN 44 300).
- Datenelement (item): Kleinste logische Dateneinheit, die bei der Verarbeitung (meist) nicht weiter zerlegt wird.
Datenfeld (Feld, field): Platz zur physischen Speicherung eines Datenelements.
 Die Begriffe Datenelement und Datenfeld werden oft synonym verwendet.
 Der Wert (Inhalt) eines Datenelements / Datenfeldes stellt eine bestimmte Information dar.
 Logisch besteht ein Datenelement aus einem oder mehreren Zeichen (characters), physisch wird das entsprechende Feld durch ein oder mehrere Bytes (oder Worte) dargestellt.
- Für jedes Feld wird festgelegt
 - ein (eindeutiger) Name
 - ein Typ (Datentyp)
 Durch den Typ wird definiert:
 - Wertebereich (möglicher Inhalt) des Feldes
 - Art der Darstellung des Feldes
 - mögliche Operationen (und Operatoren)
 - gegebenenfalls eine Länge (z.B. bei Zeichenketten)
- Datentypen:
 - Zeichenkette (String, Text) mit fester oder variabler Länge
 - Zahl (Numerisch, Numeric) in verschiedensten (internen) Darstellungen
 - Datum (Date)
 - Logisches Feld (Logical, Boolean, ja/nein, männlich/weiblich)
 - Zeit (Time)
 - Zeitpunkt (Timestamp)
 - beliebig langer Text (Memo)
 - Währung (Currency)
 - Zähler (Serial)
 - BLOB (Binary Large Object)
 - OLE-Objekt
 - Bild, Film
 - Ton, Sprache
- Feldlänge: Anzahl der Zeichen (Bytes) eines Feldes.
- Beispiel für ein Datenfeld (beschreibt die Eigenschaft *Bezeichnung* eines bestimmten Artikels):

| Bezeichnung |
|-------------|
| Hammer |

- Name des Feldes: Bezeichnung
- Typ des Feldes: Zeichenkette mit fester Länge
- Feldlänge: 10 Zeichen
- Inhalt des Feldes: Hammer◇◇◇◇ [◇ steht hier für das Leerzeichen (Blank)]

- Datensatz (Satz, record): Objekt, in dem logisch zusammengehörige Datenfelder zusammengefasst sind.
- Satzlänge: Anzahl der Zeichen (Bytes) eines Satzes
- Beispiel für einen Datensatz (beschreibt alle Eigenschaften eines bestimmten Artikels):

| ArtikelNr | Bezeichnung | Preis | Letzter Verkauf |
|-----------|-------------|-------|-----------------|
| 235 | Hammer | 6.00 | 2003-07-13 |

- Typ / Länge des Feldes ArtikelNr: Numerisch, ganzzahlig / 4 Zeichen
 - Typ / Länge des Feldes Preis: Numerisch, 3 Vorkomma, 2 Nachkomma / 6 Zeichen
 - Typ / Länge des Feldes Letzter Verkauf: Datum / 8 Zeichen
 - Satzlänge: 28 Zeichen
- Datei (file, data set): Objekt, in dem gleichartige, logisch zusammengehörige Datensätze zusammengefasst sind. Jedes Feld eines Satzes einer Datei hat einen bestimmten Wert (Inhalt).
 - Für jede Datei ist ein eindeutiger Name festzulegen.
 - Beispiel für eine Datei (beschreibt alle Eigenschaften aller Artikel):

Artikel

| ArtikelNr | Bezeichnung | Preis | Letzter Verkauf |
|-----------|-------------|-------|-----------------|
| 104 | Beißzange | 14.50 | 2003-08-20 |
| 235 | Hammer | 6.00 | 2003-07-13 |
| 105 | Kombizange | 18.90 | 2004-02-18 |
| 135 | Säge | 23.00 | 2003-07-24 |
| 28 | Bohrer | 2.40 | 2004-01-30 |
| 217 | Messer | 4.20 | 2003-11-21 |

- Name der Datei: Artikel
- Auf Basis obiger tabellarischer Darstellung können für die Bezeichnungen Datei - Satz - Feld auch die Begriffe Tabelle - Zeile - Zelle verwendet werden.
- Name und Typ der Felder kann
 - durch die Stellung des Feldes im Satz definiert sein (Position und Länge jedes Feldes = Satzspiegel)
 - in der Datei vermerkt sein
- Datenbank (data base): Zusammenfassung mehrerer 'Dateien', zwischen denen logische Abhängigkeiten und Verknüpfungen bestehen.
- Datenhierarchien:

| logisch | physisch |
|-----------|----------------------------|
| Datenbank | |
| Datei | Inhalt eines Datenträgers |
| Satz | Block |
| Feld | Wort, Halbwort, Doppelwort |
| Zeichen | Byte |
| | Bit |

21.2 Dateien: Organisations- und Zugriffsformen

Dateiorganisation (file organization):

- Strukturen und Verfahren zur (physischen) Speicherung der Datensätze auf externen Speichermedien.
- Physische Anordnung der Datensätze in der Datei (den Dateien) am Speichermedium.
- Zweck: Die für die Anwendung notwendige(n) Zugriffsform(en) soll(en) ermöglicht werden.

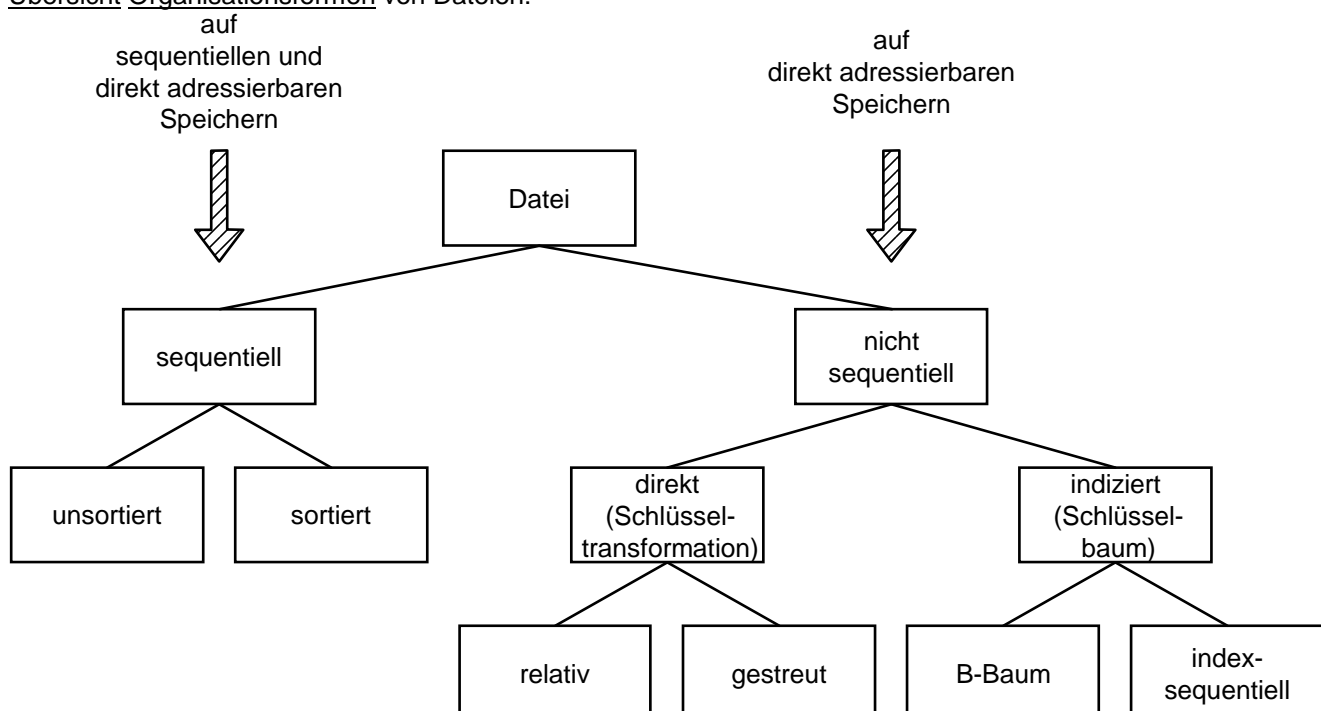
Zugriffsformen auf Dateien (Verarbeitungsformen von Dateien):

- Sequentieller Zugriff (fortlaufender Zugriff, sequential access): Auf die einzelnen Datensätze wird in einer (physisch oder logisch) vorgegebenen Reihenfolge (ein Satz nach dem anderen) zugegriffen (gelesen, geschrieben).
- Direkter Zugriff (wahlfreier Zugriff, random access): Auf die einzelnen Datensätze wird direkt, unter Angabe eines Schlüsselwerts (Key) zugegriffen (gelesen, geschrieben), ohne dass vorher andere Sätze verarbeitet werden müssen.
 - Schlüssel (Ordnungsbegriff, Suchbegriff, Key): Besteht aus einem oder mehreren Feldern; über einen Schlüsselwert kann auf einen Datensatz direkt zugegriffen werden.
 - Primärschlüssel (Basisordnungsbegriff, Primary Key): Identifizierender Schlüssel; Primärschlüsselwerte über alle Sätze sind eindeutig (es gibt keine zwei Sätze mit demselben Wert im Primärschlüssel). Sollte es mehrere identifizierende Schlüssel in einer Datei geben, muss einer als Primärschlüssel definiert werden.
Beispiel: ArtikelNr
 - Sekundärschlüssel (Sekundärordnungsbegriff, Secondary Key): Jeder vom Primärschlüssel verschiedene Schlüssel (üblicherweise sind die Werte der Sekundärschlüssel nicht eindeutig).
Beispiel: Bezeichnung, Preis

Speichermedien (Datenträger):

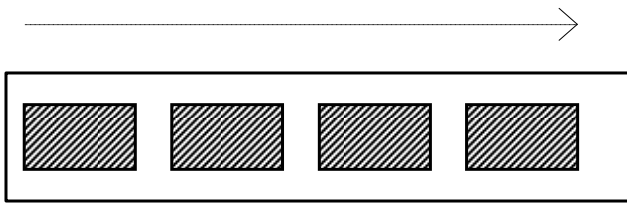
- Sequentielle Speicher (sequential access device): Datenblöcke werden unmittelbar nacheinander gespeichert und können nur in dieser Reihenfolge verarbeitet (gelesen, geschrieben) werden.
Beispiel: Magnetbandrolle, Magnetbandkassette
- Direkt (wahlfrei) adressierbare Speicher (random access device): Jeder Datenblock wird an einer Adresse gespeichert und kann unter Angabe dieser Adresse direkt verarbeitet (gelesen, geschrieben) werden.
Adresse: Fortlaufende Nummer am Datenträger, Zylinder-Spur-Sektor des Datenträgers
Beispiel: Festplatte, Diskette, MO-Speicher
- Datenblock (Block): Dateneinheit, die auf einmal (als Ganzes) zwischen dem externen Speichermedium und dem Hauptspeicher transferiert wird.

Übersicht Organisationsformen von Dateien:



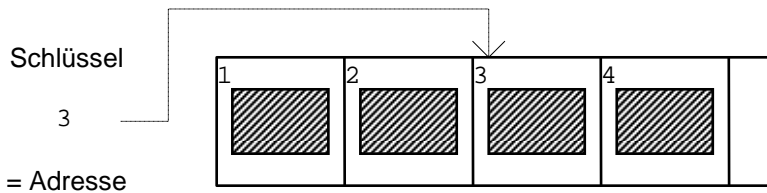
Organisationsformen (Speicherungsformen) von Dateien:

- Sequentielle Organisation:



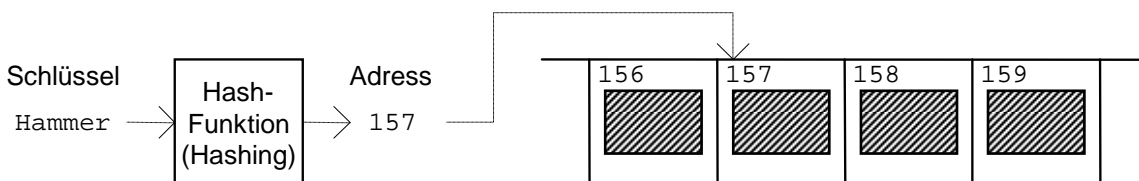
- Die Sätze der Datei stehen nacheinander auf dem Datenträger

- Relative Organisation:



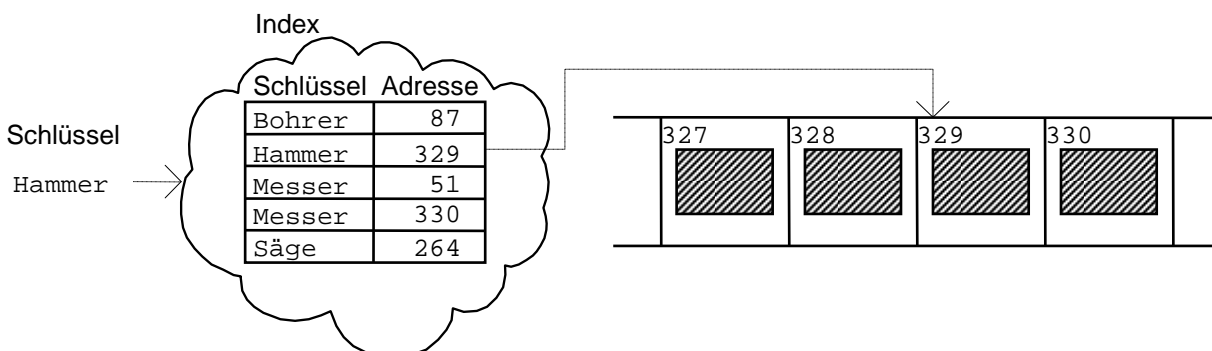
- Jeder Satz steht an einer Adresse (mit 1 beginnend und durchnummeriert) in der Datei.
- Mit dieser Adresse (Positionsnummer) kann direkt auf einen Satz zugegriffen werden.
- Beginn des Satzes mit Schlüsselwert n = Beginn der Datei + $(n - 1) * \text{Satzlänge}$

- Gestreute Organisation (Hashing):



- Aus dem Schlüsselwert wird eine bestimmte Adresse ermittelt (errechnet) an der der Satz in der Datei abgespeichert ist.
- Wenn verschiedene Schlüsselwerte dieselbe Adresse liefern (Synonyme), muss eine entsprechende Kollisionsbehandlung durchgeführt werden.

- Indizierte Organisation (baumartige Organisation):



- Ein Index (eigene Struktur, eventuell eigene Datei) enthält für jeden Schlüsselwert die Adresse des Satzes in der Datei.
- Für eine Datei sind mehrere Indizes, für verschiedene Schlüssel, möglich.
- Wenn in einem Index dieselben Schlüsselwerte mehrmals auftreten dürfen, spricht man von einem Mehrdeutigen (Duplicate) Index; andernfalls von einem Eindeutigen (Unique) Index.
- Für den Zugriff muss (bei Zeichenketten) nicht der ganze Schlüsselwert bekannt sein, es genügt auch ein Beginnteil (Anfangsteil) des Schlüsselwerts.

- Verschiedene Verfahren der Indizierung:
 - Lineare Indizes (Suchen binär, Änderungen aufwendig)
 - Binärbäume (AVL-Bäume)
 - B-Bäume (B-, B⁺-, B^{*}-Bäume)
 - Indexsequentielle Strukturen

Übersicht: Organisationsformen von und Zugriffsformen auf Dateien

| | | Organisationsform | | | |
|--------------|-------------|------------------------------------|-------------------------------|----------|-----------------------------|
| | | sequentiell | relativ | gestreut | indiziert |
| Zugriffsform | sequentiell | Reihenfolge: physische Speicherung | Reihenfolge: Positionsnummern | - | Reihenfolge: Schlüsselwerte |
| | direkt | - | ja | ja | ja |

| | | | | |
|--|---|---|-----------------|------------------------------------|
| verwendbar für Schlüssel | - | Primärschlüssel | Primärschlüssel | Primär- und/oder Sekundärschlüssel |
| Voraussetzungen bezüglich des Schlüssels | - | ganzzahlig, wenig 'Lücken' (Beginn mit 1) | keine | keine |

Übersicht: Gebräuchliche Operationen bei den verschiedenen Organisationsformen

| | sequentiell unsortiert | sequentiell sortiert | relativ | gestreut | indiziert |
|--|------------------------|----------------------|-----------------------|---------------------------------------|--|
| Satz einfügen | am Ende *) | umkopieren | an Adresse | an errechneter Adresse (Kollisionen!) | an freier Adresse, Indizes aktualisieren |
| Satz ändern | überschreiben *) | überschreiben *) | überschreiben | überschreiben | überschreiben, ev. Indizes aktualisieren |
| Satz löschen | kennzeichnen *) | kennzeichnen *) | kennzeichnen (Lücken) | errechnete Adresse freigeben | Adresse freigeben, Indizes aktualisieren |
| Satz lesen mit Schlüsselwert | - | - | an Adresse | an errechneter Adresse (Kollisionen!) | über Index |
| Satz lesen mit Schlüsselwert-Beginnteil | - | - | - | - | über Index |
| Sätze lesen in Schlüsselwert-reihenfolge | - | durchlesen | durchlesen (Lücken) | - | über Index durchlesen |
| Reorganisation | umkopieren | umkopieren | - | Überlaufbereich! | Indexaufbau! |

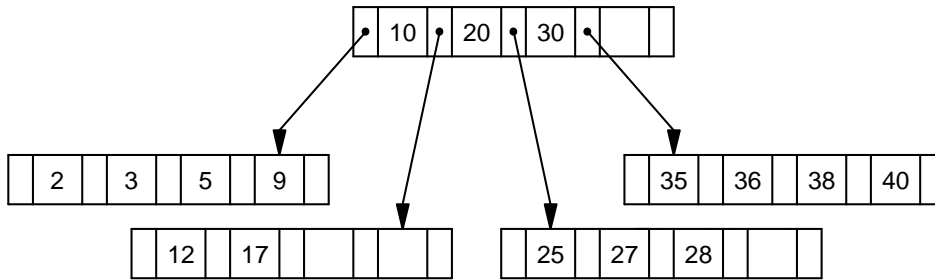
*) wenn der Datenträger diese Operation erlaubt, sonst ganze Datei auf eine neue umkopieren.

21.3 B-Bäume

- Spezielle Mehrwege-Suchbäume (entwickelt von R. Bayer und E. McCreight, 1970), Realisierung von Indizes
- Kriterien für einen B-Baum der Ordnung n
 - Jede Seite enthält höchstens $2 \cdot n$ Elemente (Informationskomponenten, Schlüssel)
 - Jede Seite (mit Ausnahme der Wurzel) enthält mindestens n Elemente
 - Eine Seite mit k Elementen hat entweder genau $k+1$ Nachfolger (Söhne) oder (bei Blattseiten) keinen Nachfolger
 - Alle Blattseiten haben das gleiche Niveau
- Da es sich um Suchbäume (sortierte Schlüsselbäume) handelt muss natürlich auch gelten
 - Die Elemente sind in einer Seite aufsteigend sortiert
 - Sind e_1, e_2, \dots, e_k die Elemente einer Seite, dann sind alle Elemente des ersten Nachfolgers kleiner als e_1 , alle Elemente des zweiten Nachfolgers größer als e_1 und kleiner als e_2 , ..., alle Elemente des $(k+1)$ -ten Nachfolgers größer als e_k
- Suchen eines Elements im B-Baum
 - Von der Wurzel aus beginnend wird geprüft, ob das gesuchte Element in der aktuellen Seite ist (eventuell Binäres Suchen in der Seite bei großen n)
 - Wenn es nicht in der aktuellen Seite ist, dann wird geprüft zwischen welchen Elementen das gesuchte Element liegen müsste; in dem Teilbaum, auf den der entsprechende Zeiger verweist, wird (wie oben beschrieben) weitergesucht; wenn der entsprechende Zeiger den NULL-Wert hat (kein Teilbaum vorhanden ist), dann existiert das gesuchte Element nicht
 - Seitenzugriffe bei einem B-Baum der Ordnung n mit insgesamt N Elementen: $\log_n(N)$
 - Beispiele:

| | | |
|----------|------------------|----------------------------------|
| $n=10,$ | $N=1\ 000$ | $\rightarrow 3$ Seitenzugriffe |
| $n=10,$ | $N=10\ 000\ 000$ | $\rightarrow 7$ Seitenzugriffe |
| $n=100,$ | $N=10\ 000\ 000$ | $\rightarrow 3-4$ Seitenzugriffe |
- Einfügen eines Elements in den B-Baum
 - Ein Element wird zunächst prinzipiell in den Blättern eingefügt
 - wenn in der Seite noch Platz ist, dann ist der Vorgang mit diesem Einfügen beendet
 - wenn die Seite bereits voll ist (und mit dem Einfügen überläuft), dann wird die Seite geteilt
 - die ersten n Elemente kommen in eine erste Seite
 - die letzten n Elemente kommen in eine zweite Seite
 - das mittlere Element wird in die Vorgängerseite (wie oben beschrieben) eingefügt
 - Wenn die Wurzelseite überläuft, wird diese analog geteilt. Es entsteht damit eine neue Wurzelseite und der Baum wächst um ein Niveau (dies ist die einzige Möglichkeit wie ein B-Baum wachsen kann)
- Löschen eines Elements aus dem B-Baum
 - wenn das Element in einem Blatt steht, kann es einfach gelöscht werden
 - wenn es nicht in einem Blatt steht, wird es durch das größte Element des linken Teilbaums oder das kleinste Element des rechten Teilbaums ersetzt
 - durch das Löschen kann es vorkommen, dass eine Seite unterläuft (weniger als n Elemente enthält)
 - wenn eine Nachbarseite mehr als n Elemente enthält, wird entweder das größte Element der linken Nachbarseite oder das kleinste Element der rechten Nachbarseite über die gemeinsame Vorgängerseite rotiert
 - wenn beide Nachbarseiten genau n Elemente enthalten, wird die Seite mit einer Nachbarseite und dem mittleren Element der gemeinsamen Vorgängerseite zu einer Seite verschmolzen; ein eventuell auftretender Unterlauf in der Vorgängerseite wird (wie oben beschrieben) entsprechend weiterbehandelt
 - Wenn das einzige Element der Wurzelseite mit seinen beiden Nachfolgerseiten verschmolzen wird, schrumpft der Baum um ein Niveau
- Beispiel: B-Baum der Ordnung $n=2$
 Einfügen der Knoten: 20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5, 42, 13, 46, 27, 8, 32, 38, 24, 45, 25
 Löschen der Knoten: 25, 45, 24, 38, 32, 8, 27, 46, 13, 42, 5, 22, 18, 26, 7, 35, 15

- Beispiel: Folgender B-Baum der Ordnung $n=2$ ist gegeben:



Erklären Sie die Vorgangsweise bei den nachstehenden Operationen und zeichnen Sie den jeweils entstehenden Baum:

- Einfügen von 19: in Seite einfügen
- Einfügen von 8: Teilen der Seite
- Einfügen von 45: Teilen der Seite, fortgesetzt bis zur Wurzelteilung
- Löschen von 17: Löschen aus Blatt
- Löschen von 30: Löschen und Ersetzen durch Blattelement (ev. Unterlauf durch Rotieren ausgleichen)
- Löschen von 38: Löschen und Ersetzen durch Blattelement (Unterlauf nicht ausgleichbar, Verschmelzen von Seiten, fortgesetzt bis zur Wurzel)

22 Überblick Graphentheorie

- Einleitung, Anwendungsgebiete
 - Graph besteht aus Knoten und Kanten (diese verbinden je zwei Knoten)
 - Kanten können Eigenschaften haben (z.B. Kosten, Länge)
 - Beispiel
 - Knoten: Städte; Kanten: Straßen zwischen den Städten
 - Problemstellungen: Suche eines / des kürzesten Weges von einer Stadt zu einer anderen; Suche einer möglichen / der kürzesten Rundreise, die jede Stadt mindestens / genau einmal passiert
 - Beispiel
 - Knoten: Versorgungspunkte; Kanten: Strom- / Gasleitungen
 - Problemstellungen: Kostengünstigster Bau eines Strom- / Gasnetzes, mit dem alle Versorgungspunkte erreicht werden
 - Beispiel
 - Knoten: elektrische Bauteile; Kanten: Leitungen
 - Problemstellungen: Verdrahtungsprobleme (kreuzungsfreier Verlauf eines Schaltplans oder einer Printplatte)
 - Sonstige Beispiele
 - Königsberger Brückenproblem, Netzpläne, chemische Verbindungen, soziologische Modelle (Verwandtschaften, etc.), Syntaxdiagramme, Struktur von Programmen (Anweisungen, Aufrufe, etc.), Wartegraphen
- Grundlegende Definitionen
 - Knotenmenge, Kantenmenge
 - gewichteter Graph
 - ungerichteter (undirected), gerichteter (directed), gemischter Graph
 - einfacher Graph (ungerichtet, keine Schlingen und Mehrfachkanten)
 - Teilgraph, gesättigter Teilgraph
 - Kantenfolge, Kantenzug, Weg
 - Kreis (Zyklus, cycle)
 - zusammenhängender Graph (connected)
 - ungerichteter Baum (undirected tree): zusammenhängend, azyklisch
 - Gerichteter Graph
 - mit Zyklen (zyklisch)
 - ohne Zyklen (azyklisch), topologische Sortierung ist möglich
 - zyklisch bei Ignorieren der Richtungen
 - azyklisch bei Ignorieren der Richtungen: gerichteter Baum (directed tree)
 - Wurzelbaum (Arboreszenz, rooted tree) / Hierarchie (hierarchy):
 - genau ein Anfangsknoten (Wurzel)
 - jeder Knoten (außer der Wurzel) hat genau einen Vorgänger
 - jeder Knoten (außer den Endknoten, Blättern) hat mindestens einen Nachfolger

- Eulersche Graphen
 - es gibt geschlossenen Kantenzug, der jede Kante genau einmal enthält
 - Kriterium: zusammenhängend und Grad jedes Knotens gerade
 - Hamiltonscher Graph (es gibt Kreis, der jeden Knoten genau einmal enthält; kein Kriterium bekannt)
- Ebene, geometrische, plättbare Graphen
 - geometrischer Graph: Knoten sind Punkte im Raum; Kanten sind Kurvenbögen, die einander nicht schneiden
 - ebener Graph: geometrischer Graph im \mathbb{R}^2
 - Isomorphie: gleiche Struktur
 - plättbarer Graph: wenn es einen isomorphen, ebenen Graphen gibt
 - Graph ist genau dann plättbar, wenn er weder C_5 noch $C_{3,3}$ als Teilgraphen enthält
 C_5 : vollständiger mit 5 Knoten; $C_{3,3}$: vollständiger paarer mit zwei mal drei Knoten
- Länder und Färbungen
 - Jeder ebene Graph zerlegt die Ebene in einzelne Gebiete (sogenannte Länder); dabei auch ein unbeschränktes Gebiet (äußeres Land).
 - Euler'sche Polyederformel: Anzahl Länder = Anzahl Kanten - Anzahl Knoten + 2
 - Vier-Farben-Problem (Francis Guthrie, ca. 1850): Wieviele Farben werden benötigt um die Länder eines beliebigen ebenen Graphen so zu färben, dass jeweils zwei Nachbarländer (haben eine Kante gemeinsam) unterschiedliche Farben haben?
Es wurde erst 1976 (fehlerfrei 1989) bewiesen, dass vier Farben ausreichen.
- Speichern von Graphen
 - Adjazenz-Matrix
 - Listen
 - Komprimierte Arrays
 - Speicherung in Datenbanken
- Algorithmen
 - Test, ob Graph zusammenhängend ist
 - Suche nach dem kürzesten Weg
 - Test, ob Graph Zyklen enthält
 - Suche nach Teilgraphen
 - Traveling Salesman - Problem
 - Minimum spanning trees

23 Literatur

/BAUE94 /

Bauer F.:
Kryptologie;
Springer (1994)

/CELK99 /

Celko J.:
Data & Databases: Concepts in Practice;
Morgan Kaufmann (1999)

/DATE93 /

Date C., Darwen H.:
A Guide to the SQL Standard (3rd ed., covers SQL2);
Addison-Wesley (1993)

/DATE04 /

Date C.:
An Introduction to Database Systems (8th edition);
Addison-Wesley (2004)

/ELMA02 /

Elmasari R., Navathe S.:
Grundlagen von Datenbanksystemen (3. Auflage);
Pearson Studium (2002)

/GULU99 /

Gulutzan P., Pelzer T.:
SQL-99 Complete, Really;
R&D Books Miller Freeman (1999)

/HARB88 /

Harbronn T.:
File Systems;
Prentice Hall (1988)

/HEUE00 /

Heuer A., Saake G.:
Datenbanken: Konzepte und Sprachen (2. Auflage);
Mitp-Verlag (2000)

/KEMP09 /

Kemper A., Eickler A.:
Datenbanksysteme (7. Auflage);
Oldenburg (2009)

/KUDR07 /

Kudraß T.:
Taschenbuch Datenbanken;;
Fachbuchverlag Leipzig (2007)

/MELT93 /

Melton J., Simon A.:
Understanding the new SQL;
Morgan Kaufmann (1993)

/PANN00 /

Panny W.:
Einführung in den Sprachkern von SQL-99;
Springer (2000)

/TAVO88 /

Tavolato P., Kleissner O., Tavolato M.:
Datenbanken Teil1 und Teil2;
Eigenverlag (1988)

/ULLM02 /

Ullman J., Widom J.:
A First Course in Database Systems (2nd edition);
Prentice Hall (2002)

/VOSS93 /

Vossen G., Groß-Hardt M.:
Grundlagen der Transaktionsverarbeitung;
Addison-Wesley (1993)

/WIRT83 /

Wirth N.:
Algorithmen und Datenstrukturen;
Teubner (1983)

TEIL II: SPEZIFISCHE SYSTEME

24 Microsoft SQLServer

24.1 SQLServer Management Studio

- Graphischer Client unter Windows
 - Abfrage ausführen: rotes Rufzeichen oder {Strg}+E
Wenn ein Teil markiert ist, dann wird nur dieser markierte Teil ausgeführt
 - Skript laden / speichern (.sql-Textdatei mit Transact-SQL-Anweisungen)
 - Achtung: Jedes neue Abfragefenster, jede neue Verbindung, jeder neuer Aufruf vom Management Studio erstellt eine neue Sitzung (Connection) zum Server (in Summe begrenzt)
- Stapel (Batch): Folge von Transact-SQL-Anweisungen, die zusammen als Gruppe abgearbeitet werden.
Stapel-Ende-Kennzeichen:
go oder Skript-Ende
Zwischen den einzelnen Transact-SQL-Anweisungen Semikolon empfohlen.
Bei Fehlern wird mit dem nächsten Stapel fortgesetzt.
- Skript: Folge von Stapelprogrammen, die nacheinander abgearbeitet werden (.sql-Textdatei)
- Kennwort ändern:
sp_password old, new
kein Kennwort: NULL
- Objekt Explorer sehr brauchbar
- Verschiedene Informationen anzeigen (Auswahl)

| | |
|------------------------|---|
| select @@version | Version des aktuell installierten SQLServers |
| sp_help [objname] | Informationen zu einem Datenbankobjekt, auch {Alt}+{F1} |
| | (Default: Alle Objekte der aktuellen Datenbank) |
| sp_helpdb dbname | Informationen zur Datenbank |
| sp_spaceused [objname] | Information zum Platzbedarf eines Datenbankobjekts |
| | (Default: Aktuelle Datenbank) |
| sp_who | Aktuelle Benutzer anzeigen |
- Bei Syntaxfehlern im Meldungsfenster auf die Fehlermeldung doppelklicken, die falsche Zeile wird im Abfragefenster markiert
- Speziell wegen Verwendung einheitlicher Datumsliteralen als Sprache 'Deutsch' wählen
 - set language 'Deutsch' (nur für die aktuelle Sitzung)
 - für den Server (Änderungen als sa auszuführen)


```
select * from master..syslanguages          -- Welche Sprachen gibt es?
select @@langid                            -- Aktuelle Sprache anzeigen
sp_configure @configname='default language' -- Aktuelle Sprache anzeigen

use master
sp_configure @configname='default language', @configvalue='1'
-- Aktuelle Sprache setzen, dann
reconfigure with override
```
- User-Optionen anzeigen
DBCC USEROPTIONS
 - 'Database Console Commands' haben außerdem eine Vielzahl von Funktionen

24.2 Transact-SQL

- SQL-Dialekt von SQLServer, viele (auch prozedurale) Erweiterungen gegenüber der Norm
- Kommentar:
-- oder /* */
- Identitätsspalte einer Tabelle (im Create Table)
IDENTITY [(seed , increment)]
- Einige Abweichungen von der Norm
 - Statt Datentyp `TIMESTAMP` ist `DATETIME` zu verwenden (`TIMESTAMP` wird individuell anders verwendet)
 - Eigene Funktionen für Temporal Datatypes, z.B. `datediff`, `dateadd`
 - `getdate()` statt `current_date`
 - `set dateformat dmy`
 - Datentyp `MONEY` nicht Norm
 - String-Concatenation: `+` statt `||`
 - kein `trim`, nur `ltrim` und `rtrim`
 - Prädikat `A=NULL` wird syntaktisch toleriert, liefert aber nie `TRUE`
 - Prädikat `DATUM LIKE '%2004%'` funktioniert
 - Natural Join wird nicht unterstützt
 - Computed Columns (auch Verwendung von UDFs)
`CREATE TABLE mytable (low int, high int, myavg as (low+high)/2)`
- Variable deklarieren
`DECLARE @Variablenname Datentyp [,@Variablenname Datentyp]...`
 - Auch bei Verwendung immer mit `@` kennzeichnen (keine Verwechslung mit Spaltennamen)
 - Globale Variable sind vom System vorgegeben und werden mit `@@` gekennzeichnet
- Variable zuweisen
`SET @Variablenname = {Ausdruck | (SELECT-Anweisung)}`
oder
`SELECT @Variablenname = {Ausdruck | (SELECT-Anweisung)}`
`[,@Variablenname = {Ausdruck | (SELECT-Anweisung)}]...`
`[Fortsetzung der SELECT-Anweisung mit FROM ...]`
 - `SELECT`-Anweisung darf nur eine Zeile liefern
 - (`SELECT`-Anweisung) darf nur eine Zeile und eine Spalte liefern
- Variable ausgeben
`SELECT @Variablenname [,@Variablenname]...`
`PRINT @Variablenname` (nur Zeichenketten)
- Konvertierungsfunktionen
`CAST` und `CONVERT`
- Ablaufsteuerung
 - `BEGIN...END`
 - (`GOTO marke`)
 - `IF...[ELSE]`
 - `RETURN`
 - `WAITFOR`
 - `WHILE` (mit `BREAK` - Abbruch, mit `CONTINUE` - Neustart der `WHILE`-Schleife)
 - Achtung: `CASE` ist eine Funktion
- Cursor
 - `DECLARE cursorname [INSENSITIVE] [SCROLL] CURSOR FOR SELECT ... [FOR UPDATE]`
`FOR UPDATE` für `UPDATE / DELETE ... WHERE CURRENT OF cursorname`
 - `OPEN cursorname`
`@@CURSOR_ROWS` enthält die Anzahl der Zeilen des (zuletzt geöffneten) Cursors
 - `FETCH [[NEXT | PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n] FROM] cursorname`
`[INTO @variablenname1, @variablenname2, ...]`
ok, wenn globale Variable `@@FETCH_STATUS` gleich 0
 - `CLOSE cursorname`
 - `DEALLOCATE cursorname` (nicht Norm, aber notwendig)

- Temporäre Tabelle: Name beginnt mit einem Nummernzeichen (#)
`SELECT ... INTO neuer_Tabellenname FROM ...`
 neuer_Tabellenname: permanente oder temporäre Tabelle
- Tabellen-Variable
`DECLARE @Variablenname
 TABLE ({ < column_definition > | < table_constraint > } [,...])`
 - Bei Qualifizierung unbedingt Aliasnamen verwenden (sonst Fehlermeldung 'Variable nicht deklariert')

24.3 Stored Procedures

- Einfaches Beispiel

```
create procedure del_l (@lnr char(2)) as
  declare @cnt integer
  select @cnt=count(*) from lt where lnr = @lnr
  if @cnt>0
    delete from lt where lnr = @lnr
  delete from l where lnr = @lnr
  select @cnt Anzahl
go
```

Aufruf
`del_l 'L2'`
- Stored Procedure erstellen
`CREATE PROCEDURE Prozedurname [(Parameter1 [,Parameter2]...[,Parameter255])] AS
 SQL-Anweisungen`
`Parameter ::= @Parametername Datentyp [= Standardwert] [OUTPUT]`
- Prozedur-Text anzeigen
`sp_helptext objname`
- Stored Procedure löschen
`DROP PROCEDURE Prozedurname`
- Stored Procedure ausführen
`[EXECUTE] [@Rückgabestatus =] Prozedurname
 [[@Parametername =] {Wert | @Variable [OUTPUT]}]
 [,[@Parametername =] {Wert | @Variable [OUTPUT]}]...`
 - Entweder lauter Stellungsparameter oder lauter Kennwortparameter
 - Nicht versorgte Parameter haben als Standardwert NULL (eventuell alle Parameter auf <> NULL abfragen)
 - Verschachtelung bis 16 Stufen möglich (globale Variable @@NESTLEVEL)
- Keine Nachricht für jede Anweisung ausgeben (unbedingt am Prozedur-Anfang verwenden)
`SET NOCOUNT ON`
- Rückgabe
 - `SELECT` Ausdruck [Spaltenüberschrift] [,Ausdruck [Spaltenüberschrift]]...
 [Fortsetzung der SELECT-Anweisung mit FROM ...]
 damit auch Rückgabe mehrerer Zeilen (einer ganzen Tabelle) möglich
 - OUTPUT-Parameter
 - RETURN (Aufruf mit EXECUTE @Rückgabestatus = Prozedurname)
 - PRINT (Meldungen, nur für Testzwecke)
- Dynamisches Transact-SQL (Befehl als Zeichenfolge aufbauen und ausführen)
`EXECUTE (string_expression)`
 - () wichtig, sonst Ausführen einer Stored Procedure
 - string_expression: nur Literale, Variable und Concatenation (+)
 - auch SP_EXECUTESQL ...
 Vorteil: parametrisiert, aktuelle Parameter können von Aufruf zu Aufruf geändert werden
 (besser gegen SQL-Injection)

```

- Beispiel:
create procedure vartab (@anzcol integer) as
declare @c varchar(255), @i integer
set @c='create table x ('
set @i=1
while @i<=@anzcol begin
    set @c=@c+'f'+convert(varchar,@i)+' int,'
    set @i=@i+1
end
set @c=substring(@c,1,datalength(@c)-1)
set @c=@c+')'
if exists (select *
           from information_schema.tables
           where table_name='x' and table_type='BASE TABLE')
    drop table x
execute (@c)
go
exec vartab 9
go
select * from x
exec sp_help x

```

24.4 Stored Functions

- User-Defined Function (UDF), zum Unterschied zu den Built-in Functions
- Keine Änderungen in den Tabellen der Datenbank erlaubt (Nebenwirkungen)
- Arten
 - Scalar functions
... RETURNS scalar_return_data_type
 - Inline table functions (single SELECT-Statement)
... RETURNS TABLE
 - Multi-statement table functions
... RETURNS @return_variable TABLE < table_type_definition >
- Der Aufruf von Funktionen mit Skalarrückgabe kann in Expressions erfolgen, es muss der Funktionsname mit dem Eigentümernamen (z.B. dbo), wenn notwendig zusätzlich auch mit dem Datenbanknamen, qualifiziert werden
owner_name.function_name ([argument_expression][,...])
- Der Aufruf von Funktionen mit Tabellenrückgabe kann überall erfolgen wo eine Tabelle verwendet wird, z.B.
select * from Grosskunden(10000)
- Übung 1)
Scalar function: Summe
2 Parameter: integer; Rückgabe: Summe der beiden Parameter (integer)
- Übung 2)
Scalar function: LetzterMonatstag
1 Parameter: date; Rückgabe: LetzterTag im Monat des Parameters (date)
- Übung 3)
Inline table-value function: Lieferungen
1 Parameter: Lieferatennummer; Rückgabe: alle Lieferungen des Lieferaten
Joinen Sie zum Funktionsergebnis auch die Teilennamen dazu
- Übung 4)
Multi-statement table-value function: LiefUms
1 Parameter: Stadt; Rückgabe: Lieferanten der Stadt (LNr, LName) mit ihrem Gesamtumsatz
(in diesem soll allerdings das billigste und teuerste Teil nicht enthalten sein)

24.5 Triggers

- Einfaches Beispiel:


```
alter table l add wann datetime null;
go
create trigger l_insupd on l for insert,update as
    update l set wann=current_timestamp
        where lnr in (select lnr from inserted);
go
insert into l (Lnr,LName,Rabatt,Stadt) values ('L9','Meier',25,'Graz');
update l set rabatt=rabatt*1.1 where stadt='London';
```
- Trigger erstellen


```
CREATE TRIGGER Triggername ON { Tabellename | Viewname }
{ FOR | AFTER | INSTEAD OF } {INSERT, UPDATE, DELETE} [WITH ENCRYPTION] AS
[ IF UPDATE (Spaltenname) [{AND | OR} UPDATE (Spaltenname)]... ]
SQL-Anweisungen
```

 - FOR = AFTER
 - Achtung: Wenn ein Trigger für eine Funktion erstellt wird, für die schon ein Trigger existiert, dann wird der existierende Trigger ohne Fehlermeldung überschrieben
- Trigger-Text anzeigen


```
sp_helptext objname
```
- Trigger löschen


```
DROP TRIGGER Triggername
```
- Ausführung:
 - Einmal pro INSERT-, UPDATE-, DELETE-Befehl (auch wenn mehrere Zeilen betroffen sind)
 - Constraints werden vor der Ausführung von AFTER-Trigger geprüft
- INSERTED- und DELETED-Tabellen
 - Pseudo-Tabellen, nur Lesen möglich
 - INSERTED-Tabelle: bei INSERT und UPDATE (neuer Stand)
 - DELETED-Tabelle: bei UPDATE (alter Stand) und DELETE
 - Anzahl der Zeilen in INSERTED und DELETED bei UPDATE ist gleich
 - Gleicher Aufbau wie Tabelle für die der Trigger definiert ist
 - @@ROWCOUNT: Anzahl der Zeilen, die von der Datenmodifikation betroffen waren
- Fehlermeldung liefern - abbrechen


```
RAISERROR ({Meldungs-Id | Meldungstext}, Schweregrad, Zustand [, Argument]...)
```

 - Meldungs-Id: mit sp_addmessage in sysmessages eintragen (nur Administrator)
 - Meldungstext: als Literal angeben, Formatkennzeichen für Argumente, Meldungs-Id 14000 (in @@ERROR)
 - Schweregrad: z.B. 16
 - Zustand: z.B. 1
 - Argumente: ersetzen die Formatkennzeichen im Meldungstext
 - danach ROLLBACK TRANSACTION

24.6 Transaktionen

- 3 Modi
 - Autocommit-Transaktionen (default): Anweisung=Transaktion
 - Explizite Transaktionen:


```
BEGIN TRANSACTION
{COMMIT | ROLLBACK} TRANSACTION
```
 - Implizite Transaktionen (Norm):
 - Umschalten auf den impliziten Modus


```
SET IMPLICIT_TRANSACTIONS {ON | OFF} auch in SET ANSI_DEFAULTS {ON | OFF}
```
 - Verwendung nur von


```
{COMMIT | ROLLBACK} {TRANSACTION | WORK}
```


- Übung
 - Lieferungen von L1 aus LT und Lieferant L1 aus L in einer Transaktion löschen, nach löschen in LT:
 - `select * from lt` - in eigener / anderer Transaktion
 - `rollback / commit`
 - Management Studio beenden (normal, Task beenden mit Task-Manager)
 - Server beenden (normal mit Management Studio – Object Explorer, Rechner ausschalten)
- Verschachtelung möglich, @@TRANSCOUNT enthält Verschachtelungstiefe
- Transaktionen können mit Transaktionsnamen versehen werden
- Sicherungspunkt innerhalb einer Transaktion setzen:
`SAVE TRANSACTION Sicherungspunktname`
mit `ROLLBACK TRANSACTION ...` kann auch auf einen Sicherungspunkt zurückgesetzt werden

24.7 Concurrency

- Isolation Levels wie in SQL-92 mit folgenden Unterschieden:
 - gilt während der ganzen Verbindung
 - Standardannahme: `READ COMMITTED`
 - Sperrgranulat: Zeile (default), Seite (8KB), Tabelle, Datenbank
 - Aktuellen Isolation Level anzeigen:
`DBCC USEROPTIONS`
 - Zusätzlicher Isolation Level:
`SET TRANSACTION ISOLATION LEVEL SNAPSHOT;`
Änderungen, die nach Beginn der aktuellen Transaktion von anderen Transaktionen vorgenommen wurden, sind in der aktuellen Transaktion nicht sichtbar. Es entsteht der Eindruck, als ob eine Momentaufnahme der Daten, wie sie zu Beginn der Transaktion vorhanden waren, vorliegt.
Kein Sperren von Daten beim Lesen, kein Warten auf gesperrte Daten (row versioning instead of locking).
Wenn dieser Isolation Level verwendet wird (Aufwand!), müssen vorher Datenbankoptionen entsprechend gesetzt werden:
`ALTER DATABASE Datenbankname SET ALLOW_SNAPSHOT_ISOLATION ON;`
`ALTER DATABASE Datenbankname SET READ_COMMITTED_SNAPSHOT ON;`
- UPDATE und INSERT mit X-Lock bis Transaktionsende
- DELETE mit X-Lock
- Sperroptionen explizit angeben (in den Table Hints)
`SELECT ... FROM Tabellennamen [WITH] (Sperroptionen) ...`
 - überschreibt Sperrverhalten aus Isolation Level
 - auch mehrere angebbbar (sinnvolle Kombinationen), dann WITH zwingend
 - `NOLOCK` kein Lock
 - `ROWLOCK` Zeile S-Lock (Standardannahme)
 - `HOLDLOCK` Lock bis Transaktionsende
 - `PAGLOCK` Seite S-Lock
 - `TABLOCK` Tabelle S-Lock
 - `UPDLOCK` Zeile U-Lock (bis Transaktionsende)
 - `XLOCK` Zeile X-Lock (bis Transaktionsende)
 - `TABLOCKX` Tabelle X-Lock (bis Transaktionsende)
 - etc.
`UPDATE Tabellennamen [WITH] (Sperroptionen) SET ...`
 - `PAGLOCKX` Seite X-Lock
 - `TABLOCKX` Tabelle X-Lock

- Sperrinformationen anzeigen
`sp_lock [spid1][,spid2] oder sp_lock1`
 - Type: DB | TAB | EXT | PAG | KEY | RID
 - Mode: S | U | X | IS | IX | SIX | Sch-M | Sch-S | BU
 - Status: GRANT | WAIT | CNVT
- Festlegen der maximalen Wartezeit auf eine geperzte Ressource
`SET LOCK_TIMEOUT milliseconds`
 - n: bricht nach maximal n Millisekunden ab
 - 0: bricht sofort ab
 - -1: bricht nie ab
 - Anzeigen: `select @@lock_timeout`
- Deadlockbehandlung
 - Geeignete Transaktion wird mit Meldung 1205 abgebrochen
 - `SET DEADLOCK_PRIORITY { LOW | NORMAL | HIGH | -10 to 10 }`
- Realisierung der Isolation Levels mit Sperren:
 - READ UNCOMMITTED Select mit NOLOCK
 - READ COMMITTED Select mit ROWLOCK
 - REPEATABLE READ Select mit ROWLOCK bis Transaktionsende
 - SERIALIZABLE
- Außerdem werden auch Intent Locks (IS, IX, SIX) im Rahmen eines Hierarchischen Sperrkonzepts verwendet
- Sperrenausweitung (Lock Escalation):
Bei Abfragen, die sehr viele Zeilen einer Tabelle betreffen, ist es ökonomischer von Zeilensperren auf Tabellensperren überzugehen (Sperrübergang). Wann das geschehen soll kann konfiguriert werden (`sp_configure 'LE threshold maximum | minimum | percent'`, LE = Lock Escalation).

25 ORACLE

- Am weitesten verbreitetes DBMS
- Plattformunabhängig (vom PC zum Großrechner, auf verschiedensten Betriebssystemen)
- Auch Client-, Design- und Development-Software
- ab ORACLE9: Objekt-Relationale Datenbank
Relational erweitert um objektorientierte Konzepte
(Abstrakte Datentypen, Geschachtelte Tabellen, Variable Arrays) alles implementiert in Tabellen
- Die wichtigsten Manuals

| | |
|--|------|
| - Oracle Database Concepts | CON |
| - Oracle Database Globalization Support Guide | GSG |
| - Oracle Database SQL Quick Reference | SQLQ |
| - Oracle Database SQL Reference | SQL |
| - Oracle Database Application Developer's Guide – Fundamentals | FUN |
| - Oracle Database PL/SQL User's Guide and Reference | PLS |
| - Oracle Database Application Developer's Guide - Object-Relational Features | OBJ |
| - SQL*Plus Quick Reference | PLUQ |
| - SQL*Plus User's Guide and Reference | PLU |

25.1 Grundlagen

siehe CON

- Schema: Benannte Zusammenfassung von Schemaobjekten innerhalb einer Datenbank
- Schemaobjekte: clusters, database links, database triggers, external procedure libraries, index-only tables, indexes, packages, sequences, stored functions, stored procedures, synonyms, tables, views.
Wenn distributed functionality installiert ist: snapshots, snapshot logs.
Wenn objects option installiert ist: object tables, object types, object views.

Non-Schemaobjekte: directories, profiles, roles, rollback segments, tablespaces, users
- Jedem Datenbank-User ist ein Schema zugeordnet. Beim Anlegen eines Users wird (automatisch) ein entsprechendes Schema mit demselben Namen angelegt. Standardmäßig hat jeder User Zugriff auf die Objekte seines (gleichbenannten) Schemas

25.2 Datentypen

siehe SQL

- Gebräuchlichste Datentypen: VARCHAR2, NUMBER, INTEGER, DATE

| Datentyp | Beschreibung | Länge (Bytes) | Konstante | Bemerkung |
|--|---|------------------|-----------------------------------|---|
| CHAR[(size)] | Zeichenkette (fix lang) | max. 2000 | 'Hansi' keine " | ohne (size): 1 Concatenation: |
| VARCHAR2(size) | Zeichenkette (variabel lang) | max. 4000 | w.o. | |
| NUMBER[(p[,s])] | Zahl | | 123.45 | p: 1 bis 38, s: -84 bis 127 ohne (p,s): floating point INTEGER=NUMBER(38) |
| DATE | Datum und Zeit (bis Sekunden) | | '13.11.1998' oder TO_DATE() | jetzt: SYSDATE, CURRENT_DATE |
| TIMESTAMP | Zeitpunkt (bis Millisekunden) | | | jetzt: CURRENT_TIMESTAMP |
| TIME | Zeit | | | nicht unterstützt! |
| INTERVAL YEAR TO MONTH or DAY TO SECOND | Zeitintervall | | | |
| VARCHAR | synonym mit VARCHAR2, für zukünftige Erweiterungen | | | nicht verwenden! |
| LONG | Zeichendaten (variabel lang) | max. $2^{31}-1$ | w.o. | nicht in Ausdrücken, nicht in Indizes, etc. |
| RAW(size) | Binärdaten (variabel lang) | max. 2000 | | |
| LONG RAW | Binärdaten (variabel lang) | max. $2^{31}-1$ | | Graphiken, Sound, etc. siehe auch LONG |
| ROWID | Hex. Zeilenadresse (physisch, block.row.file) | | | Pseudospalte ROWID |

- Datentypen in Zusammenhang mit Large Objects
 - CLOB Character Large Object
 - BLOB Binary Large Object
 - BFILE Locator to a Large Binary File (außerhalb der Datenbank gespeichert)
- Datentypen in Zusammenhang mit National Character Set (wird bei Datenbankdefinition angegeben)
 - NVARCHAR2(size)) entsprechende Datentypen,
 - NCHAR[(size)]) jedoch mit National Character Set
 - NCLOB) (auch multibyte Characters)
- Datentypen in Zusammenhang mit Trusted Oracle
 - MLSLABEL Binary Format of an Operating System Label
- Angabe von ANSI-Datentypen möglich, jedoch Konvertierung in ORACLE-Datentypen
- Außerdem Benutzerdefinierte Datentypen (User-Defined Types) möglich

25.3 SQL

siehe SQL

- Unterschiede zu Standard SQL: siehe SQL - Appendix B)
- Character-Literale mit ' begrenzen (" nicht möglich)
- Outer Join (ab 9.2 auch Norm möglich)
`select * from l,lt where l.lnr=lt.lnr(+);`
(+) als Zusatz auf der Seite, wo mit NULL aufgefüllt wird
- Like mit Erkennung eines regulären Ausdrucks
`... WHERE REGEXP_LIKE(first_name, '^Ste(v|ph)en$')`
- Subselect-IN mit Tupel
`select * from l cross join t where (lnr,tnr) not in (select lnr,tnr from lt);`
- Pseudotabelle DUAL (eine Zeile, eine Spalte)
`select sqrt(2) from dual;`
`select sysdate from dual;`
- Pseudocolumns
ROWNUM
ROWID
`select x.*, rownum`
`from (select * from l order by rabatt desc) x`
`where rownum<=3;`
- Hierarchical Queries
Pseudocolumn LEVEL
- Sequences
CREATE SEQUENCE ..., ALTER SEQUENCE ..., DROP SEQUENCE
Pseudocolumns CURRVAL, NEXTVAL
- Funktionen: siehe SQL – Functions etc.
Auch benutzerdefinierte Funktionen verwendbar (siehe Stored Subprograms)
- Zusätzlich zu UNION: INTERSECT und MINUS (Norm wäre EXCEPT!)
- Aktueller Username
USER
- Keine Temporären Tabellen, aber CREATE TABLE ... AS SELECT ...
- Data Warehousing-, OLAP-Erweiterungen
ROLAP, CUBE, GROUPING, TOP-N, SAMPLE, Analytic Functions
- Tabellen löschen
`DROP TABLE ... CASCADE CONSTRAINTS`

25.4 SQL*Plus

siehe PLUQ, PLU

- Eingabe und Abarbeitung von SQL-Befehlen
 - Vorteile: Auf allen Plattformen, Gestaltung von Auswertungen, diverse Variable und Automatisierungen
 - Nachteile: Einfaches zeichenorientiertes Interface, einfacher Kommandozeilen-Editor
- Verbindung zu Datenbank herstellen


```
CONNECT username/password@connect_identifizier [AS {SYSOPER | SYSDBA}]
```

connect_identifizier: tns_name
(z.B. xe vordefiniert für //localhost:1521/xe = Hostname:Port/ServiceId)
- Aktuellen Usernamen anzeigen


```
SHOW USER
```
- Password ändern


```
ALTER USER user IDENTIFIED BY password
```
- Object beschreiben


```
DESCRIBE object
```
- Verschiedene Informationen anzeigen (Auswahl)


```
SELECT * FROM V$VERSION;
SELECT * FROM DUAL;
SELECT * FROM USER_OBJECTS | OBJ;
SELECT * FROM SESSION_PRIVS;
SELECT * FROM USER_TABLESPACES;
SELECT * FROM USER_TS_QUOTAS;
```
- 3 Arten von Kommandos
 - SQL Kommando, beenden mit `return` (in eigener Zeile) oder `;` (auch ausführen)
 - PL/SQL Block, beenden mit `.` (in eigener Zeile)
 - SQL*Plus Kommando

Auf Grund des ersten Schlüsselwortes wird entschieden, um welche Art von Kommando es sich handelt
- SQL Buffer
 - Enthält das letzte SQL Kommando oder den letzten PL/SQL Block
 - Ausführen mit `/` oder `RUN` (darf immer nur ein Kommando oder ein Block sein)
- SQL Buffer editieren

| | |
|--|--|
| <pre>L[IST] n I[MPUT] 0 text C[HANGE] /old/new A[PPEND] text n text DEL DEL m n CL[EAR] BUFF[ER]</pre> | <pre>Buffer listen Zeile n zur aktuellen machen und listen nach aktueller Zeile einfügen Zeile mit text vor erster Zeile einfügen in aktueller Zeile old auf new ändern in aktueller Zeile text anhängen Zeile n mit text überschreiben aktuelle Zeile löschen von Zeile m bis Zeile n löschen alle Zeilen löschen</pre> |
|--|--|
- SQL Buffer mit Betriebssystemeditor editieren (Editor festlegen über User Variable `_EDITOR`)


```
ED[IT]
```
- Command File (Betriebssystemdatei)
 - SQL Buffer als Command File abspeichern


```
SAV[E] file_name
```
 - Command File in SQL Buffer laden (sinnvollerweise nur ein SQL Kommando oder PL/SQL Block)


```
GET file_name
```
 - Command File in SQL Buffer laden und ausführen (auch mehrere SQL Kommandos oder PL/SQL Blöcke)


```
START file_name | @ file_name
@@ file_name (für geschachtelte Command Files)
```
 - Kommentar in Command File


```
-- oder /* */
```

- System Variable (beeinflussen das Environment der aktuellen SQL*Plus-Verbindung)
 - System Variable setzen
SET system_variable value
 - Wert der System Variablen anzeigen
SHOW system_variable | ALL
- User Variable (Makro-Variable, nur textlich)
 - User Variable anlegen
DEF[INE] variable = text
 - User Variable mit Wert anzeigen
DEF[INE] [variable]
 - User Variable als Substitution Variable (Ampersand Variable) mit & beginnend verwenden
 - Parameter von START als Substitution Variable verwenden mit &1, &2, usw.
 - User Variable löschen
UNDEF[INE] variable ...
- User Communication
 - Ausgabe
PROMPT ... oder PAU[SE] ...
CLEAR SCREEN
 - Eingabe (in User Variable)
ACC[EPT] ...
- Gestaltung von Auswertungen (Reports)
 - Kopf- und Fußzeilen ausgeben: TTITLE ..., BTITLE ...
 - Spalten formatieren: COLUMN ...
 - Zwischensummen ausgeben: BREAK ..., COMPUTE ...
- Bind Variable (in SQL*Plus angelegt, in PL/SQL verwendet)
 - Bind Variable vereinbaren
VAR[IABLE] variable type
 - Bind Variable anzeigen
VAR[IABLE] [variable]
 - Bind Variable in PL/SQL mit : beginnend verwenden
 - Wert der Bind Variablen anzeigen
PRINT [variable ...]
- Beispiel für ein Command File (Ausgabe wird wieder als Command File verwendet)

```
set echo      off
set termout   off
set feedback  off
set pagesize  0
spool desctmp.sql
select 'DESC ' || table_name FROM user_tables;
spool off
set feedback  on
set termout   on
set echo      on
start desctmp
host del desctmp.sql
```
- Beispiele zu National Language Support

```
select * from V$NLS_PARAMETERS where parameter='NLS_TERRITORY';
select * from V$NLS_PARAMETERS where parameter='NLS_DATE_FORMAT';
alter session set NLS_DATE_FORMAT='YYYY/MM/DD HH24:MI:SS';
```
- Verbindung zu Datenbank beenden
DISCONNECT
- SQL*Plus beenden
EXIT | QUIT

25.5 PL/SQL

siehe PLS, FUN

- Procedural Language / SQL: Erweiterung von SQL um prozedurale Elemente, vollwertige Programmiersprache
- Zwei Kategorien
 - Anonymous Block: Nicht bezeichnet, nicht in Datenbank gespeichert; kann (meist) dort stehen, wo eine SQL-Anweisung stehen kann
 - Stored Subprogram: Bezeichnet, in Datenbank gespeichert; wird beim Speichern übersetzt und in übersetzter Form gespeichert; wird über den Namen aufgerufen
 - Procedures CREATE PROCEDURE
 - Functions CREATE FUNCTION
 - Packages CREATE PACKAGE / PACKAGE BODY (Gruppe von Procedures und Functions)
 - Triggers CREATE TRIGGER
- Um PL/SQL-Code ausführen zu können, ist eine PL/SQL Engine notwendig. Diese ist im Backend (Oracle Database-Server), aber auch in einigen Frontend-Produkten (z.B. Oracle Forms, Oracle Reports) enthalten
- Kommentar
-- oder /* */
- Blockstruktur

```
[ DECLARE
  -- Declarations ]
BEGIN
  -- Statements          <- hier Sub-Blocks möglich
[ EXCEPTION
  -- Exception_handlers ] <- hier Sub-Blocks möglich
END;
```

 - Block kann mit einem <<label_name>> gekennzeichnet werden
- Declarations (mit ; abzuschließen)
 - type_definition
 - item_declaration
collections, constants, cursors, cursor_variables, exceptions, objects, records, variables
 - function_declaration
 - procedure_declaration
 - Verwendung von %TYPE und %ROWTYPE (Attribute)
- Statements (mit ; abzuschließen)
 - assignment_statement :=
 - exit_statement EXIT ...
 - (goto_statement) GOTO ...
 - if_statement IF ... THEN ... [ELSE ...] END IF
 - loop_statement
 - basic loop LOOP ... END LOOP
 - WHILE loop WHILE ... LOOP ... END LOOP
 - FOR loop FOR index_name IN ... LOOP ... END LOOP
 - cursor FOR loop FOR record_name IN ... LOOP ... END LOOP
 - null_statement NULL
 - plsql_block siehe oben
 - raise_statement RAISE ...
 - return_statement RETURN ...
 - sql_statement (im besonderen keine DDL-Statements)

| | | |
|------------------|--------------------|---------------------------|
| select_statement | open_statement | set_transaction_statement |
| insert_statement | open-for_statement | savepoint_statement |
| update_statement | fetch_statement | commit_statement |
| delete_statement | close_statement | rollback_statement |
| | | lock_table_statement |

- Exception handlers

```
WHEN { exception_name [ OR exception_name ]... | OTHERS } THEN
    statement; [statement;]...
[WHEN { exception_name [ OR exception_name ]... | OTHERS } THEN
    statement; [statement;]...]
```

- Fortsetzung nach Ausführten der Exception: Block, der den Block in dem sich die auslösende Anweisung befindet, umgibt
- Auch benutzerdefinierte Exceptions möglich (Declaration mit Typ EXCEPTION, Ausführung mit RAISE)
- Benutzerdefinierte Fehlermeldung erzeugen (im Package DBMS_STANDARD)
`RAISE_APPLICATION_ERROR(message_number, message_text [, { TRUE | FALSE }])`
`-20999 <= message_number <= -20000, length(message_text) <= 2048`

- Cursors

- Impliziter Cursor: für eine SQL-Datenmanipulation (inklusive One-Row-Query), Cursor_Name ist SQL
- Expliziter Cursor: für eine Multiple-Row-Query, im Deklarationsteil zu deklarieren
- Cursor Attributes
 - Impliziter Cursor: %FOUND, %NOTFOUND, %ROWCOUNT
 - Expliziter Cursor: %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT
- Cursor Variable
 - Deklaration mit REF CURSOR
 - `OPEN cursor_variable FOR select_statement`

- BildschirmAusgabe (Testzwecke)

- Verwendung des Packages DBMS_OUTPUT
- In SQL*Plus: `SET SERVEROUTPUT ON`
- Befehle

| | |
|--|------------------------------|
| <code>DBMS_OUTPUT.PUT(...)</code> | ohne Zeilenvorschub |
| <code>DBMS_OUTPUT.PUT_LINE(...)</code> | mit Zeilenvorschub (nachher) |

- Praktische Hinweise

- `END IF, END LOOP` mit Blank
- Auch bei `FOR` mit `REVERSE` `lower_bound` kleiner `upper_bound` (sonst kein Schleifendurchlauf)

- Testing

- `DBMS_OUTPUT`
- Tabelle `test_results (results varchar2(nn))` und `insert into test_results ...`

25.6 Stored Subprograms

siehe SQL, PLS, FUN

- Stored Procedure erstellen

```
CREATE [ OR REPLACE ] PROCEDURE procedure
    [ ( argument [ IN | OUT | IN OUT ] datatype
      [ , argument [ IN | OUT | IN OUT ] datatype ]... ) ]
    { AS | IS } plsql_subprogram_body
```

- Stored Function erstellen

```
CREATE [ OR REPLACE ] FUNCTION function
    [ ( argument [ IN | OUT | IN OUT ] datatype
      [ , argument [ IN | OUT | IN OUT ] datatype ]... ) ] RETURN datatype
    { AS | IS } plsql_subprogram_body
```

Rückgabe des Funktionswerts

```
RETURN [ ( ] expression [ ) ]
```

- Datentypen der formalen Parameter ohne Längen, etc. angeben
- Name Resolution: Name von lokalen Variablen und formalen Parametern haben Vorrang gegenüber Namen von Tabellen-Spalten (diese mit Tabellennamen qualifizieren)
- Stored Procedure ausführen: Mit dem Namen (und den aktuellen Parametern) als PL/SQL-Statement aufrufen
`SQL*Plus: EXEC procedure` erzeugt einen kleinen Block und ruft die Procedure auf

- Stored Function ausführen: Mit dem Namen (und den aktuellen Parametern) in Expressions aufrufen
- Übersetzungsfehler (in SQL*Plus) anzeigen
SHO[W] ERR[ORS]
- Stored Procedures / Functions werden immer im System-Tablespace gespeichert
- Stored Procedure / Function löschen
DROP PROCEDURE procedure
DROP FUNCTION function

25.7 Packages

siehe SQL, PLS, FUN

- Zusammengehörige Procedures und Functions; Vorstufe für Objektorientierung (Datenkapselung); ungefähr analog Klassendefinition und Klassenimplementierung in C++
- Zwei Teile
 - Package Specification (CREATE PACKAGE), Interface (Schnittstelle, Signatur, Prototyp)
 - Package Body (CREATE PACKAGE BODY), kann ausgetauscht werden ohne das Interface zu ändern

```
CREATE PACKAGE name AS          -- specification (visible part), interface
    -- public type and item declarations
    -- subprogram specifications
END [name];
```

```
CREATE PACKAGE BODY name AS     -- body (hidden part)
    -- private type and item declarations
    -- subprogram bodies
[BEGIN
    -- initialization statements]
END [name];
```

- Bei Zugriff auf Variable, Procedures oder Functions diese mit dem Packagenamen qualifizieren
- Deklaration einer Variablen entweder in der Specification (dann direkt zugreifbar - public) oder im Body (dann nicht zugreifbar - privat, Packaged Variable)
- Bodiless package: In Specification nur Types, Constants, Variables und Exceptions (keine Subprograms, Cursors), daher auch kein Body. Damit globale Variable innerhalb einer Session möglich
- Initialization: Wird bei erster Referenzierung des Packages innerhalb einer Session ausgeführt
- Overloading von Procedures und Functions möglich (je nach Aufbau des Interfaces)
- Package löschen
DROP PACKAGE [BODY] package
ohne BODY werden Specification und Body, mit BODY wird nur der Body gelöscht

25.8 Triggers

siehe SQL, FUN

- Drei Teile
 - Triggering Statement (auslösendes INSERT, UPDATE, DELETE)
 - Trigger Restriction (WHEN ...)
 - Trigger Action (plsql_block oder Procedureaufruf)
- Trigger-Typen
 - Row- oder Statement-Trigger (Default: Statement-Trigger);
keine definierte Zeilenreihenfolge bei Row-Trigger
 - BEFORE- oder AFTER-Trigger (Trigger-Timing)
 - INSTEAD OF-Trigger (in Zusammenhang mit Views)
- Trigger erstellen


```
CREATE [ OR REPLACE ] TRIGGER trigger
  { BEFORE | AFTER | INSTEAD OF }
  { DELETE | INSERT | UPDATE [ OF column [ , column ]... }
  [ OR { DELETE | INSERT | UPDATE [ OF column [ , column ]... } ]...
  ON table
  [ [ REFERENCING { OLD [ AS ] old | NEW [ AS ] new }... ]
    FOR EACH { ROW | STATEMENT } ]
  [ WHEN ( condition ) ]
  plsql_block
```
- Unterscheiden des Triggering Statements bei kombinierten Triggers


```
IF { INSERTING | UPDATING | DELETING } THEN ...
```
- Keine rekursiven Trigger verwenden!
- Es können mehrere Trigger für dieselbe Tabelle, dasselbe Statement und dieselbe Trigger-Type definiert werden (Willkürliche Reihenfolge!, Konflikte?)
- Default-Namen für old und new im PL/SQL-Block
:old und :new (in WHEN noch ohne Doppelpunkt)
- Bei Abbruch des Triggering Statements mit RAISE_APPLICATION_ERROR (...) wird automatisch Rollback durchgeführt, explizites ROLLBACK ist nicht notwendig/möglich.
Problem bei Row-Trigger:
Alle fehlerhaften Zeilen abarbeiten / anzeigen, nur einmal am Schluss ROLLBACK anwenden.
- Alle Trigger anzeigen


```
SELECT * FROM USER_TRIGGERS
```
- Trigger aktivieren / deaktivieren


```
ENABLE / DISABLE clause
```
- Trigger löschen


```
DROP TRIGGER trigger
```
- Es werden nicht nur DML-Trigger, sondern auch DDL-Trigger und Database-Events unterstützt

25.9 Collections und Records

siehe PLS

- 3 Collection Types
 - Index-by Tables / Assoziative Arrays (Menge von Schlüssel-Wert Paaren, Schlüssel: Integer oder String)
Können nicht in einer Datenbank (persistent) gespeichert werden
 - Nested Tables – siehe Objektorientierte Erweiterungen
 - VArrays – siehe Objektorientierte Erweiterungen
- Multilevel Collection möglich
- Collection-Methoden
 - EXISTS(n) – Existiert Element mit Indexwert n?
 - COUNT – Aktuelle Anzahl der Elemente
 - (LIMIT)
 - FIRST | LAST – Erster und letzter Indexwert (NULL wenn nicht vorhanden)
 - PRIOR(n) | NEXT(n) – Vorheriger und nächster Indexwert zum Indexwert n, NULL wenn nicht vorhanden
 - (EXTEND)
 - (TRIM)
 - DELETE[(n)] – Löscht Element mit Indexwert n oder löscht alle Elemente
- Records


```
TYPE type_name IS RECORD (field_declaration [,field_declaration] ...);
field_declaration ::= field_name field_type [[NOT NULL] {:= | DEFAULT} expression]
```
- Referencing Records


```
record_name.field_name
```
- Verwendung oft mit table_name%ROWTYPE

25.10 Dynamisches SQL

siehe PLS, FUN

- EXECUTE IMMEDIATE


```
EXECUTE IMMEDIATE dynamic_string
[INTO {define_variable[, define_variable]... | record}]
[USING [IN | OUT | IN OUT] bind_argument
[, [IN | OUT | IN OUT] bind_argument]...]
[{RETURNING | RETURN} INTO bind_argument[, bind_argument]...];
```
- AUTHID CURRENT_USER in Procedure- / Package-Kopf (vor AS)

25.11 Transaktionen

siehe CON, SQL

- Verhalten analog SQL-Norm
- Beginn


```
Erste ausführbare SQL/DML-Anweisung
```
- Ende


```
COMMIT [WORK]
ROLLBACK [WORK]
Implizit (z.B.: DDL-Anweisung, Fehler, Logoff)
```
- Mit Savepoints kann eine Transaktion auch noch unterteilt werden
- in SQL*Plus


```
SET AUTO[COMMIT] { ON | OFF | IMM[EDIATE] | n }
```

25.12 Concurrency

siehe CON, SQL

- Transaktionseigenschaften setzen (vereinfacht)


```
SET TRANSACTION
{ READ { ONLY | WRITE } |
  ISOLATION LEVEL { SERIALIZABLE | READ COMMITTED } };
```

 - muss erste Anweisung einer Transaktion sein (vor erster DML-Anweisung)
 - Default: READ WRITE und ISOLATION LEVEL READ COMMITTED
 - Keine Abfrage liest uncommittete (dirty) Daten
- Isolation Level für ganze Verbindung setzen


```
ALTER SESSION SET ISOLATION_LEVEL=READ COMMITTED;
ALTER SESSION SET ISOLATION_LEVEL=SERIALIZABLE;
```
- Read committed

Jede Abfrage der Transaktion sieht die Daten, die am Beginn der Abfrage committet waren (Statement-Level Read Consistency).
- Serializable

Jede Abfrage der Transaktion sieht die Daten, die am Beginn der Transaktion committet waren (Transaction-Level Read Consistency), alle Abfragen der Transaktion lesen daher denselben Stand.
- Read Only

Wie Serializable, jedoch INSERT, UPDATE und DELETE nicht erlaubt.
- Multiversion Concurrency Control (MVCC), Oracle: Rollback Segments

Zu einem Zeitpunkt kann ein Datenobjekt für verschiedene Transaktionen in verschiedenen Versionen vorliegen.
- Snapshot Isolation (SSI), Oracle: bei Serializable

MVCC kann für jede Transaktion die Datenbank in einem eigenen Zustand (Snapshot) darstellen. Die Änderungen werden von anderen Transaktionen nicht gesehen, frühestens bei Transaktionsende werden die Änderungen öffentlich.
- Timestamp-based concurrency control (Zeitstempel-Verfahren), Oracle: System Change Number (SCN)

Jede Transaktion hat einen Zeitstempel nach Startzeitpunkt (ältere einen kleineren, jüngere einen größeren). Gültige Schedules müssen äquivalent zum seriellen Schedule der Zeitstempel-Reihenfolge sein. Die Datenbankobjekte bekommen einen Lese- und/oder Schreib-Zeitstempel. Das ist der Zeitstempel jener Transaktion, die das Datenbankobjekt als letzte (erfolgreich) gelesen und/oder geschrieben hat. Das Lesen und/oder Schreiben eines Objekts ist einer Transaktion nur möglich, wenn der Zeitstempel der Transaktion in einem bestimmten Ordnungsverhältnis mit dem Lese- und/oder Schreib-Zeitstempel des Objekts steht.
- Pessimistic Concurrency Control (konservativ) - Optimistic Concurrency Control (aggressiv)

Pessimistic: Sperren möglichst früh, bis Transaktionsende (lange Sperrzeiten, einfache Logik)
 Optimistic: Sperren möglichst vermeiden, erst am Transaktionsende oder bei Update (inklusive Validierung) (kurze Sperrzeiten, komplexe Logik, Lese-Phase – Validierungsphase – Schreibphase)
- Fehlersituationen

ORA-01453: SET TRANSACTION muss erste Anweisung der Transaktion sein

ORA-00054: Versuch, mit NOWAIT eine bereits belegte Ressource anzufordern

ORA-00060: Deadlock beim Warten auf Ressource festgestellt

ORA-08177: Zugriff für diese Transaktion kann nicht serialisiert werden
 Beim Ändern von Daten merkt die Transaktion, dass seit ihrem Beginn die Daten schon (von jemandem anderen) geändert wurden. Damit wird ein Lost-Update verhindert.
 Kann nur auftreten, wenn die Transaktion in ISOLATION LEVEL SERIALIZABLE läuft.
- Generell kein automatisches Rollback der Transaktion, sondern nur Laufzeitfehler

- Leseoperationen sperren grundsätzlich nie (außer bei `SELECT ... FOR UPDATE`), dadurch Erhöhung der Parallelität (nonblocking queries).
- Leseoperationen warten grundsätzlich nie (außer bei `SELECT ... FOR UPDATE`), lesen daher eventuell einen nicht aktuellen, aber konsistenten, Zustand. Die in der eigenen Transaktion durchgeführten Änderungen sind beim Lesen sehr wohl berücksichtigt.
- UPDATE und INSERT mit X-Lock bis Transaktionsende
- DELETE mit X-Lock
- Explizites (manuelles) Sperren von Daten
 - Beim Lesen
`SELECT ... FOR UPDATE [OF column [, column]...] [NOWAIT | WAIT integer];`
 X-Lock (statt U-Lock)!
 - Ganze Tabelle
`LOCK TABLE { table | view } IN { ROW SHARE |
 ROW EXCLUSIVE |
 SHARE |
 SHARE ROW EXCLUSIVE |
 EXCLUSIVE } MODE [NOWAIT];`
- Sperren werden grundsätzlich am Ende der Transaktion freigegeben
- Übung:
 - a) Dirty Read (Uncommitted Dependency) mit den 2 verschiedenen Isolation Levels
Preis bei Teil T1 um 1 erhöhen
 - b) Nonrepeatable Read mit den 2 verschiedenen Isolation Levels
Preis bei Teil T1 um 1 erhöhen
 - c) Phantoms mit den 2 verschiedenen Isolation Levels
Teil einfügen: 'T9','Bolzen','rot',20,'Sidney'; Teile mit Farbe='rot' lesen
Teil einfügen: 'T9','Bolzen','blau',20,'Sidney'; Teile mit Farbe='rot' lesen
 - d) Lost Update
 - i) Preis von T1 lesen und anzeigen
 - ii) Preis von T1 um 1 erhöhen (basierend auf den in i) gelesenen Wert)

| | | |
|--|--|-------------------------------------|
| A) read committed serializable serializable | B) read committed serializable read committed | (Reihenfolge der updates variieren) |
|--|--|-------------------------------------|

Wie ändert sich das Verhalten, wenn mit `SELECT ... FOR UPDATE` gelesen wird?

 - e) Inconsistent Analysis mit den 2 verschiedenen Isolation Levels
 - A) Summiert die Preise von T1 und T2 auf
 - B) Vermindert den Preis von T2 um 1, Erhöht den Preis von T1 um 1

Wie ändert sich das Verhalten, wenn mit `SELECT ... FOR UPDATE` gelesen wird?
 - f) Deadlock erzeugen
 - mit Update - Befehlen
 - mit Lock Table - Befehlen
 - g) Unterschied, wenn (in a), b), c), e)) mit Isolation Level READ ONLY gelesen wird?
 - h) Beispiel, bei dem eine Zeile in drei verschiedenen Versionen vorliegt.

25.13 Object-Relational Features (CLIL)

see CON, OBJ, SQL

- Features
 - Object Types (user-defined types) with
 - Attributes (built-in types or other user-defined types)
 - Methods (written in PL/SQL)
 - Object-Tables
 - Object Identifiers (OIDs)
 - Object References (REFs)
 - Collection Types (may contain further collections – Multilevel Collections)
 - VArrays (Varying Arrays)
 - Nested Tables
 - Subtypes
 - Inheritance
 - Object Views
- Comparison OID – Key

| | |
|------------------------------|------------------------|
| OID | Key |
| - set by the system | - set by the user |
| - fix | - changeable |
| - not reuseable | - reuseable |
| - 'not readable' by the user | - readable by the user |
- Comparison: VArray - Nested Table

| | |
|--|--|
| VArray | Nested Table |
| - ordered (may be important) | - unordered |
| - max number of elements (usually small) | - no limitation for number of elements |
| - stored in same table | - stored in separate table |
| - dense (consecutive subscripts) | - sparse (nonconsecutive subscripts) |
| - elements may not be queried (returned as whole) | - elements may be queried |
- Collection Methods

| | |
|--|--|
| <ul style="list-style-type: none"> - LIMIT - COUNT - FIRST LAST - PRIOR NEXT - EXTEND - DELETE - EXISTS - TRIM | <ul style="list-style-type: none"> Tables: always NULL VArrays: COUNT = LAST VArrays: FIRST=1, LAST=COUNT returns index-number always at the end VArrays: only all Elements (without Parameter) mainly with Tables always from the end |
|--|--|
- Create Type - VArray


```
CREATE TYPE type_name AS VARRAY (limit) OF datatype;
```
- Create Type - Nested Table


```
CREATE TYPE type_name AS TABLE OF datatype;
```
- Create Type - Object


```
CREATE TYPE type_name AS OBJECT (element_list);
element ::= attribute datatype |
           {MEMBER | STATIC} {procedure_spec | function_spec} |
           {MAP | ORDER} MEMBER function_spec

CREATE TYPE BODY type_name AS ...
```

- Incomplete Object Type
CREATE TYPE type_name;
- Object Table
CREATE TABLE table OF object_type;
- Expressions and Functions
 - Constructor: name of object-type
 - Function: REF (correlation_variable)
correlation_variable = table alias; returns OID of the object instance
 - Attribute-Reference via Object-Reference (OID)
object_reference.attribute_name
 - Function: Deref (e)
e = expression, that returns a REF to an object; returns the object instance
 - Table Expression: TABLE (collection_expression) [(+)]
collection unnesting (in SELECT, INSERT, UPDATE, DELETE)
 - Function: VALUE (correlation_variable)
correlation_variable = table alias; returns object instances from an object table
 - Expression: CURSOR (subquery)
returns a nested cursor
 - Expression: CAST (...)
converting between built-in datatype and / or collection-typed values
 - UTL_REF.SELECT_OBJECT ...
reading an object only with its oid, no matter in which table stored,
SELECT ... INTO ... needs the tablename
 - UTL_REF.UPDATE_OBJECT ...
updating an object only with its oid, no matter in which table stored, UPDATE needs the tablename
 - UTL_REF.DELETE_OBJECT ...
deleting an object only with its oid, no matter in which table stored, DELETE needs the tablename
 - UTL_REF.LOCK_OBJECT ...
locking an object only with its oid
- Various
 - Use Table-Aliases
 - Dependent Join
... FROM dept d, TABLE (d.emp) e ...
No join-condition!
 - Dangling refs
Row object, which is referenced by OID can be deleted!
- Naming Conventions
 - Object Type names o_
 - Varray Type names v_
 - Nested Table Type names n_
 - Object Tables ot_
 - Relational Tables rt_
 - Nested Tables nt_
 - Object Views ov_
 - Relational Views rv_
 - Parameter names p_
 - Local Variables l_

- Example (from OBJ - A Sample Application Using Object-Relational Features)

