Datenbanken und Informationssysteme

Release 0.1.215

Kurt Hillebrand, Günter Burgstaller

19.12.2021

Inhalt:

KAPITEL 1

Grundlagen der Datenbanksysteme

1.1 Entwicklung

1.2 Vergleich: Dateien - Datenbanksystem

1.3 Eigenschaften von Datenbanksystemen

1.4 Datenbankarchitektur: Drei-Schichten-Konzept

1.4.1 Konzeptionelle Ebene

- Beschreibung der Realität durch ein *Datenmodell*, ohne Rücksicht auf hardwaremäßige oder applikationsspezifische Aspekte.
- Eine einzige Sichtweise.

1.4.2 Interne Ebene

1.4.3 Externe Ebene

1.4.4 Verbindung der Ebenen (Mapping)

KAPITEL 2

Konzeptionelles Datendesign

KAPITEL 3

Vom konzeptionellen Datenmodell zur Implementierung in Tabellen

KAPITEL 4

Weiterführende ER-Modellierung

KAPITEL 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 IT-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 eindeutigen Kennung 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 Einteilungszwecke 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 Fertigungsvorgang 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 wielange 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, Geschlecht, 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).
- Bei jedem Flug muss auch die *tatsächliche Start- und Landezeit* festgehalten werden können, um Auswertungen über die Pünktlichkeit der einzelnen Flüge erstellen zu können.
- Vorab wird für jeden Flug festgelegt, wieviele *Plätze* welcher *Klasse* zur Verfügung stehen. Die Anzahl der verbleibenden freien Plätze eines Fluges muss laufend zu ermitteln sein. Die vom Passagier gebuchten Flüge werden auf einem *Ticket* (Ticketnummer, Ausstellungdatum, 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.

5.4 Schulinformationssystem

Ein *Schulinformationssystem* für eine HTL soll entwickelt werden:

- Die *abteilungsweise Gliederung* einer HTL ist wiederzugegeben.
- Jeder *Lehrer* ist einer Abteilung als Stammabteilung 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 feststellbar sein, ebenso von welchem Schüler 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 (LVG)* (I bis VI) zugeordnet. Für jede LVG 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:

- In jedem *Kino* (Name, Adresse,…) können mehrere *Säle* untergebracht sein, in denen die Filme gezeigt werden.
- Die Daten eines *Films* umfassen: Titel, Genre (Krimi, Western, Jugendfilm,…), Herstellungsjahr, Land, Sprache, Dauer, Verleih, Altersfreigabe (FSK).
- 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 gleichen 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.

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. Die Beschreibung des Einsatzes, Einsatzort, sowie Art und Zeitpunkt der Einsatzanforderung (z.B. Unfallmeldung) sind festzuhalten.
- Bei geplanten Einsätzen sind Rücksprachemöglichkeiten sowie Planbeginn und Planende wesentlich.
- Jeder Einsatz wird von einem Mitarbeiter geleitet.
- Die freiwilligen *Mitarbeiter* der Rettungsstelle werden beschrieben durch: Name, Wohnadresse, Geburtsdatum, Dienstgrad, Ausbildung.
- Der *Fuhrpark* umfasst verschiedene Fahrzeuge mit: Marke, Modell, Kennzeichen, Ausstattung.
- 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 des Anlasses für den Transport dieses Patienten möglich sein.
- Von den Patienten ist zu speichern: Name, Wohnadresse, Geburtsdatum, Geschlecht, Sozialversicherungsnummer (SVN), Krankenversicherungsanstalt (KVA).
- Bei Mitversicherten soll der eigentliche Versicherungsnehmer ermittelt werden können.
- Um häufig angefahrene *Orte* (z.B. Krankenhäuser, Rehabilitationszentren, etc.) nicht jedesmal wieder eingeben zu müssen, sind diese mit Adresse und GPS-Koordinaten festzuhalten. Jeder Fahrt ist ein Ort als Ausgangs- oder Zielort zuzuordnen. Weiters sind auch die Entfernungen und Soll-Fahrzeiten 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

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 wird die SVNr gespeichert, die 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 Spielergebnis muss 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, wird die Anzahl der Tore gespeichert.
- 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 errichten und die relevanten Daten in einer Datenbank speichern:

- In Kleinkennstenich (Bezirk Hintermberg, 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.
- Die Tankstelle 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.
- Die 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“ mit einer 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.
- 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.
- 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

5.11 Chess Tournaments (CLIL)

5.12 Library (CLIL)

5.13 School Organisation (CLIL)

5.14 Aufträge

5.15 Ableitung ERD in Tabellen 1

5.16 Ableitung ERD in Tabellen 2

5.17 Einfaches Auftrags-Bestell-System (Handelsbetrieb)

KAPITEL 6

Relationales Daten(bank)modell

6.1 Grundbegriffe

6.1.1 Datenbankmodell

Das **Datenbankmodell** legt fest, wie Daten *logisch* dargestellt werden. Beispiele dafür sind:

- *Relationales Datenbankmodell*: Mengen von Tupeln (Tabellen mit Datensätzen in SQL Datenbanken)
- *Graphen-Datenbankmodell*: Graphen bestehend aus Knoten und Kanten
- *XML Datenbankmodell*: Baumstruktur(en) aus Elementen (Start-Tag, Inhalt, End-Tag)

6.1.2 Datenmodell

Ein **Datenmodell** ist eine Abbildung eines Ausschnitts der Realität in einer Datenbank. Abgebildet werden nur jene Informationen, die für die Aufgabenstellung wichtig sind. Beispiele für *Datenmodelle* sind:

- Datenmodell *Lieferanten - Teile*
- Datenmodell *Lieferanten - Teile - Projekte*
- Datenmodell *Schulungsfirma*

6.1.3 RDBMS

- Relationales Datenbank-Management-System (RDBMS)
- SQL als Abfragesprache (wenige Ausnahmen, z.B. *Tutorial D* in Rel)
- Mathematischer Begriff *Relation* als theoretische Grundlage (*Relationales Modell*)

6.1.4 NoSQL-Systeme

- Sammelbegriff für alle anderen Arten von DBMS
- Diverse Abfragesprachen (XQuery, CQL, N1QL, Cypher, SPARQL)
- Diverse Datenbankmodelle, meist ohne formale Grundlagen (Ausnahme: Graphen-DBMS)

6.2 Grundlagen des relationalen Modells

Das *relationale Datenbankmodell* wurde von 1970 von *Edgar Frank „Ted" Codd (1923 - 2003)* entwickelt, *Turing Award* 1981.

- Eine **relationale Datenbank** besteht aus einer Menge von **relationalen Variablen (Relvar)**.
- Eine *Relvar* enthält einen *relationalen Wert (Relation)*.
- Eine *Relation (relationaler Wert)* ist eine *Menge von Tupeln* mit dem gleichen Aufbau.
- Ein **Tupel** besteht aus mehreren *Attributen*.
- Jedes **Attribut** hat einen *Namen*, einen *Datentypen (Domäne)* und einen *Wert*.
- Alle Tupel einer Relation haben die gleiche Anzahl an Attributen (**Grad** der Relation) mit übereinstimmenden Namen und Datentypen.
- Die *Anzahl der Tupel* einer Relation heißt **Kardinalität** der Relation.

6.3 Normalformen von Relationen

Datenmodellierung mittels *funktionaler Abhängigkeiten*. Ziel ist eine *Zerlegung von Relvars (Decomposition)*, um ein *redundanzfreies*, *konsistentes* und *realitätskonformes Modell* der Wirklichkeit zu bekommen.

Begriff Normalisierung:

> **Interviewer:** Where did normalization come from?
>
> **Codd:** It seemed to me essential that some discipline be introduced into database design. I called it normalization because then President Nixon was talking a lot about normalizing relations with China. I figured that if he could normalize relations, so could I.
>
> *„A Fireside Chat: Interview with Dr. Edgar F. Codd" (DBMS Magazine 6, No. 13, December 1993).*

6.3.1 Relationen und Tabellen

6.3.2 Erste Normalform (1NF)

6.3.3 Funktionale Abhängigkeit (FA)

6.3.4 Zweite Normalform (2NF)

6.3.5 Dritte Normalform (3NF)

6.3.6 Boyce-Codd Normalform (BCNF)

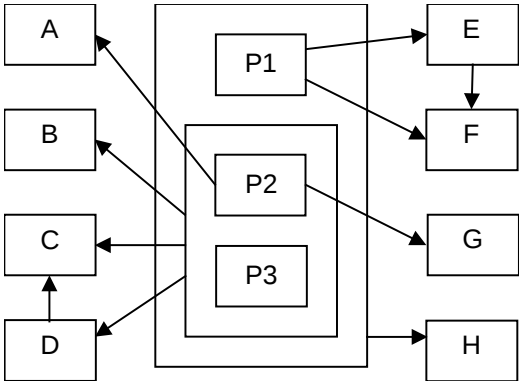
6.3.7 Weitere Normalformen

6.3.8 Denormalisieren

6.4 Übungen zum Normalisieren

6.4.1 Übung 1: Abhängigkeitsdiagramm

Gegeben ist folgende Relvar `R(P1, P2, P3, A, B, C, D, E, F, G, H)`, Schlüsselattribute sind `P1 P2 P3`. Folgendes Abhängigkeitsdiagramm wurde ermittelt:



- Spalten Sie die Relvar `R` in Relvars mit mindestens 2NF auf. Begründen Sie für jede Relvar, warum sie in 2NF ist. Welche Relvars sind bereits in 3NF?
- Spalten Sie alle Relvars auf, die nicht in 3NF sind. Begründen Sie, warum diese nun in 3NF sind.
- Sind alle Relvars auch in BCNF? Begründen Sie Ihre Entscheidungen.

6.4.2 Übung 2: Freigegenstände

In einer Schule soll für ein Schuljahr verwaltet werden, welche Schüler sich für welche Freigegenstände angemeldet haben und wie sie darin beurteilt wurden. Beispiel:

SName	SGebDat	JG	KatNr	JGVor	StKlRaum	Trakt	Stock	FGKurz	FGBez	FGWochStd	SemNote	JahrNote
Maier	2006-06-20	3CHIF	17	Prof. Huber	Z305	Zubau	3	JGL	Jonglieren	4	1	1

Folgende Attribute wurden ermittelt:

- *Schülername* `SName`, *Geburtsdatum* `SGebDat`, *Jahrgang (Klasse)* `JG` sowie *Katalognummer* `KatNr`
- *Jahrgangsvorstand* `JGVor`, *Stammklassenraum* `StKlRaum`, *Trakt* `Trakt` und *Stockwerk* `Stock`
- *Kurzzeichen des Freigegenstands* `FGKurz`, *Bezeichnung* `FGBez`, *Wochenstunden* `FGWochStd`
- Die Wochenstundenzahl des gleichen Freigegenstands kann je nach Jahrgang unterschiedlich sein, z.B. Robotik im 3. Jahrgang mit 4 Wochenstunden, Robotik im 5. Jahrgang mit 3 Wochenstunden.
- *Semesternote* `SemNote`, *Jahresnote* `JahrNote`

Aufgabe:

a. Schreiben Sie eine Relvar `FG` mit diesen Attributen auf.

b. Bestimmen Sie einen Primärschlüssel und unterstreichen Sie die Schlüsselattribute.

c. Erstellen Sie ein entsprechendes Abhängigkeitsdiagramm gemäß obigen Anforderungen.

d. Stellen Sie fest, in welcher Normalform sich die Relvar `FG` befindet. Begründen Sie Ihre Schlussfolgerung.

e. Zerlegen Sie `FG` in Relvars, die mindestens die 2NF aufweisen.

f. Bringen Sie alle Relvars in 3NF.

g. Zeichnen Sie ein ERD, das zum gleichen Resultat führt.

6.4.3 Übung 3: Vorlesungsverwaltung

Eine *Informatik-Fakultät* verwaltet die Prüfungen für Vorlesungen in folgender Tabelle:

MatrNr	Name	LVNr	LVTitel	PruefErg		
				Datum	**Note**	**Saal**
6625337	Maier	116468	Programmieren	20000601	5	2.13
				20060415	5	1.47
				20070219	1	2.13
6625337	Maier	116138	Compilerbau	**Datum**	**Note**	**Saal**
				20021220	1	1.47
7025123	Müller	116138	Compilerbau	**Datum**	**Note**	**Saal**
				20021220	4	1.47
6830478	Hofer	116468	Programmieren	**Datum**	**Note**	**Saal**
				20000601	5	2.13
				20031015	1	E.05

- Eine Tabellenzeile ist der Eintrag für einen Studenten und seine *Prüfungsergebnisse* `PruefErg` in einer Lehrveranstaltung.
- Ein Student hat eine *Matrikelnummer* `MatrNr` und einen *Namen* `Name`.
- Eine Lehrveranstaltung hat eine *Lehrveranstaltungsnummer* `LVNr` sowie einen *Lehrveranstaltungstitel* `LVTitel`.
- Eine Prüfung umfasst das *Prüfungsdatum* `Datum`, die *Note* `Note` und einen *Prüfungssaal* `Saal`.
- An einem Tag findet in einer Lehrveranstaltung höchstens eine Prüfung statt, pro Prüfung gibt es nur einen Saal.
- Ein Student kann an einem Tag jedoch zu mehreren Prüfungen antreten.

Aufgabe:

a. Schreiben Sie den Relvar-Kopf `VorVer` mit den beschriebenen Attributen auf.

b. Geben Sie durch Unterstreichen einen Schlüssel(kandidaten) für die Relvar an.

c. Wieviele Tupel hat die Beispielrelation? Wieviele Attribute?

d. Lösen Sie das *relationswertige Attribut* `PruefErg` auf. Geben sie den neuen Relvar-Kopf an.

e. Wieviele Tupel hat die neue Beispielrelation? Wieviele Attribute?

f. Wie ändert sich der Schlüssel?

g. Geben Sie die Menge der voll funktionalen Abhängigkeiten an.

h. Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm.

i. Warum ist diese Relvar nicht in 2NF?

j. Spalten Sie die Relvar in Relvars mit mindestens 2NF auf. Begründen Sie für alle Relvars, dass sie in 2NF sind.

k. Sind diese Relvars auch bereits in 3NF? Begründen oder widerlegen Sie diese Eigenschaft für jede einzelne Relvar.

6.4.4 Übung 4: Spital

Für ein *Spital* werden folgende Sachverhalte in einer Tabelle festgehalten:

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	**ANr**	**AName**	**AFach**	**Krheit**
					10	Dr. No	Internist	Gelbsucht
1313	Berger	Wien	5	Westtrakt2	**ANr**	**AName**	**AFach**	**Krheit**
					13	Dr. Kurt O.	Kinderarzt	Röteln
					3	Dr. Frank	Internist	Bluthochdruck
1350	Huber	Wien	13	Osttrakt1	**ANr**	**AName**	**AFach**	**Krheit**
					10	Dr. No	Internist	Herzinfarkt
					3	Dr. Frank	Internist	Herzinfarkt
					7	Dr. Mabuse	Hautarzt	Gürtelrose

- Ein *Patient* hat eine *Patientennummer* `PNr`, einen *Namen* `PName` und einen *Wohnort* `POrt`.
- Jeder Patient wird von mehreren Ärzten behandelt. Ein Arzt behandelt höchstens eine *Krankheit* `Krheit` bei einem Patienten.
- Ein *Arzt* hat eine *Nummer* `ANr`, einen *Namen* `AName` und ein *Fachgebiet* `AFach`. Ein Arzt behandelt mehrere Patienten.
- Jeder Patient liegt in einem *Zimmer* `ZNr`, das eine *Traktbezeichnung* `ZBez` hat. In einem Zimmer liegen mehrere Patienten.

Aufgabe:

a. Schreiben den Relvar-Kopf `Spital` mit den genannten Attributen auf.

b. Geben Sie durch Unterstreichen einen Schlüssel(kandidaten) für die Relvar an.

c. Wieviele Tupel hat die Beispielrelation? Wieviele Attribute?

d. Lösen Sie das *relationswertige Attribut* `Behandlung` auf. Geben sie den neuen Relvar-Kopf an.

e. Wieviele Tupel hat die neue Beispielrelation? Wieviele Attribute?

f. Wie ändert sich der Schlüssel?

g. Geben Sie die Menge der voll funktionalen Abhängigkeiten an.

h. Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm.

i. Warum ist diese Relvar nicht in 2NF?

j. Spalten Sie die Relvar in Relvars mit mindestens 2NF auf. Begründen Sie für alle Relvars, dass sie in 2NF sind.

k. Welche dieser Relvars sind bereits in 3NF? Welche nicht? Begründen Sie Ihre Entscheidung.

l. Bringen Sie alle Relvars in BCNF. Begründen Sie für alle Relvars, dass sie in BCNF sind.

m. Zeichnen Sie ein ERD, das zum gleichen Ergebnis führt.

6.4.5 Übung 5: Baufirma

Für eine *Baufirma* soll folgender Sachverhalt in einer Tabelle (eine Zeile pro Bauprojekt), möglicherweise mit Untertabellen, festgehalten werden:

- Ein Bauprojekt hat eine *Nummer* `PNr`, einen *Namen* `PName` und findet an einem *Ort* `POrt` statt.
- Für ein Projekt werden Bauteile von Lieferanten zu einem *Lieferdatum* `LDat` in bestimmten *Mengen* `LMenge` geliefert. Ein Projekt kann mehrere Bauteile benötigen.
- Das gleiche Bauteil für ein bestimmtes Projekt wird nie mehrmals vom selben Lieferanten geliefert.
- Das gleiche Bauteil kann in mehreren Bauprojekten benötigt und von verschiedenen Lieferanten geliefert werden.
- Ein *Lieferant* hat eine *Nummer* `LNr`, einen *Namen* `LName` und einen *Standort* `LOrt`.
- Ein Lieferant kann verschiedene Bauteile liefern. Jeder Lieferant hat für ein bestimmtes Bauteil einen bestimmten Preis.
- Ein Lieferant kann für ein bestimmtes Projekt einen *Preisnachlass* `Rabatt` in Prozent gewähren.
- Ein *Bauteil* hat eine eindeutige *Nummer* `TNr` und einen *Typ* `TTyp`.

Aufgabe:

a. Stellen Sie eine Tabelle mit drei Beispieldatensätzen auf.

b. Schreiben Sie den Relvar-Kopf `Baufirma` mit den beschriebenen Attributen auf.

c. Geben Sie durch Unterstreichen einen Schlüssel(kandidaten) für die Relvar an.

d. Lösen Sie *relationswertige Attribute* auf. Geben sie den neuen Relvar-Kopf und den neuen Schlüssel an.

e. Geben Sie die Menge der voll funktionalen Abhängigkeiten an.

f. Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm.

g. Warum ist diese Relvar nicht in 2NF?

h. Spalten Sie die Relvar in Relvars mit mindestens 2NF auf. Begründen Sie für alle Relvars, dass sie in 2NF sind.

i. Wie unterscheidet sich in diesem Beispiel die 3NF von der 2NF?

j. Sind diese Relvars auch in BCNF? Begründen oder widerlegen Sie diese Eigenschaft für jede einzelne Relvar.

6.4.6 Übung 6: Gebäudereinigung

Eine *Reinigungsfirma* hat in einer Stadt mehrere Gebäude zu reinigen. Der Reinigungsplan ist in folgender Tabelle dargestellt:

GebCode	GebAdr	Raumpflege					
		RaumNr	**SitzAnz**	**SitzTyp**	**Pflege**	**PflgrNr**	**PflgrName**
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	Echtholzpflege	P02	Huber
B	Stadtplatz 5	**RaumNr**	**SitzAnz**	**SitzTyp**	**Pflege**	**PflgrNr**	**PflgrName**
		1	38	B	Saugen	P02	Huber
		2	25	C	Echtholzpflege	P02	Huber
		3	46	A	Feucht abwischen	P01	Maier
		3	46	A	Feucht abwischen	P03	Bauer

- Jedes *Gebäude* hat einen *Code* `GebCode` und eine *Adresse* `GebAdr`.

- In jedem Gebäude sind mehrere *Konferenzräume* zu reinigen. Jeder Raum hat eine *Nummer* `RaumNr` und eine *Sitzanzahl* `SitzAnz`.
- Alle Sitze eines Raums haben den gleichen *Sitztyp* `SitzTyp`. Zu jedem Sitztyp gibt es spezielle *Pflegehinweise* `Pflege`.
- Für die Reinigung eines Raumes sind ein oder mehrere *Pfleger* zuständig. Ein Pfleger hat einen *Nummer* `PflgrNr` und einen *Namen* `PflgrName`. Ein Raumpfleger säubert mehrere Räume.

Aufgabe:

a. Schreiben den Relvar-Kopf `GebRein` mit den genannten Attributen auf.

b. Geben Sie durch Unterstreichen einen Schlüssel(kandidaten) für die Relvar an.

c. Lösen Sie das *relationswertige Attribut* `Raumpflege` auf. Geben sie den neuen Relvar-Kopf an.

d. Wie ändert sich der Schlüssel?

e. Geben Sie die Menge der voll funktionalen Abhängigkeiten an.

f. Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm.

g. Warum ist diese Relvar nicht in 2NF?

h. Spalten Sie die Relvar in Relvars mit mindestens 2NF auf. Begründen Sie für alle Relvars, dass sie in 2NF sind.

i. Welche dieser Relvars sind bereits in 3NF? Welche nicht? Begründen Sie Ihre Entscheidung.

j. Bringen Sie alle Relvars in BCNF. Begründen Sie für alle Relvars, dass sie in BCNF sind.

k. Ergänzen Sie in allen Relvars zweckmäßige *FK-Constraints* in der Form `Attr>Relvar`.

l. Zeichnen Sie ein ERD, das zum gleichen Ergebnis führt.

6.4.7 Übung 7: Krankenkasse

Eine *Krankenkasse* verwaltet zur *Abrechnung der Behandlungshonorare* die notwendigen Daten in einer Tabelle:

ANR	ANAME	AFACH	PNR	PNAME	PGESCH	BBEGINN	BCODE	BTEXT	BKOST
5	Dr. Nahtlos	Allgemeinmedizin	8	Unger	W	2017-12-20 13:40	117	Grippeimpfung	21
9	Dr. Sägeberg	HNO	4	Schneider	M	2017-11-05 10:30	398	Tonsillektomie	380

- Jede Tabellenzeile ist eine Behandlungsabrechnung eines Arztes.
- Ein Arzt hat eine *Nummer* `ANR`, einen *Namen* `ANAME` und ein Fachgebiet `AFACH`.
- Für jeden *Patienten* werden die *Nummer* `PNR`, der *Name* `PNAME` und das *Geschlecht* `PGESCH` erfasst.
- Der *Beginnzeitpunkt* (Datum mit Uhrzeit) jeder Behandlung `BBEGIN` wird vermerkt.
- Der *Behandlungscode* `BCODE` bestimmt die Art der medizinischen Leistung, z.B. eine Impfung. Mit diesem Code werden auch ein dazugehöriger *Behandlungstext* `BTEXT` und die *Behandlungskosten* `BKOST` festgelegt.
- Eine *Behandlung* betrifft einen Arzt und einen Patienten. Ein Arzt kann einen Patienten auch öfter behandeln.

Aufgabe:

a. Schreiben Sie eine Relvar `KRAKAS` mit diesen Attributen auf.

b. Geben Sie *zwei* Schlüsselkandidaten für diese Relvar an.

c. Erstellen Sie für beide Schlüsselkandidaten jeweils ein entsprechendes Abhängigkeitsdiagramm gemäß obigen Geschäftsregeln.

d. Analysieren Sie, in welcher Normalform sich die Relvar `KRAKAS` befindet.

e. Zerlegen Sie `KRAKAS` in Relvars, die mindestens die 2NF aufweisen.

f. Bringen Sie alle Relvars in 3NF.

g. Vergleichen Sie die beiden Ergebnisse.

h. Zeichnen Sie ein ERD, das zum gleichen Resultat führt.

i. Was müssen Sie ändern, wenn folgende Geschäftsregeln zusätzlich gelten sollen?

- Zu einem bestimmten Beginnzeitpunkt kann der Arzt nur mit der Behandlung eines einzigen Patienten beginnen.
- Ebenso kann ein Patient zu einem bestimmten Beginnzeitpunkt eine Behandlung nur bei einem Arzt beginnen.

6.4.8 Übung 8: Classic CD World

Die Organisation *Classic CD World* richtet ein Informationssystem über CDs, Plattenfirmen (Labels), Musikstücke, Komponisten, Interpreten und Aufnahmen bezüglich klassischer Musik ein. Eine erster Entwurf ergibt folgende Relvar, mit einem Tupel für jede CD:

```
CDWorld := relation {
  tuple { CDNr '1359X5N7', Titel 'Bach Inventionen & Sinfonias', Preis 21.00,
          LabNr 357, LabName 'Sony', WWW 'www.sony.com',
          Aufnahme relation {
            tuple { StNr 'BWV 796', StBez 'Sinfonia No. 10 in G Dur',
                    AufnDat '2011-03-05', Ort 'Wien', Dauer '1:06',
                    KomNr 197, KomName 'J. S. Bach', KomGeb '1685-03-21',
                    IntNr 325, IntName 'Simone Dinnerstein', IntGeb '1972-09-18',
                    Hono 0.2 },
            tuple { StNr 'BWV 797', StBez 'Sinfonia No. 11 in G Moll',
                    AufnDat '2011-03-05', Ort 'Wien', Dauer '2:56',
                    KomNr 197, KomName 'J. S. Bach', KomGeb '1685-03-21',
                    IntNr 325, IntName 'Simone Dinnerstein', IntGeb '1972-09-18',
                    Hono 0.2 }
          }
  },
  tuple { CDNr '175K42Q', ...
    ...
  },
  ...
}
```

- Eine *CD* ist durch eine *Nummer* `CDNr` identifiziert, hat einen *Titel* `Titel` und einen *Preis* `Preis`.
- Jede CD wird von einer *Plattenfirma (Label)* mit *Labelnummer* `LabNr`, *Labelname* `LabName` und *Webadresse* `WWW` herausgebracht.
- Eine CD enthält mehrere Aufnahmen, wobei klarerweise die gleiche Aufnahme nie mehrmals auf der gleichen CD vorhanden ist.
- Im Rahmen einer *Aufnahme* spielt ein Interpret ein Musikstück zum *Aufnahmedatum* `AufnDat` an einem *Aufnahmeort* `Ort` ein. Die Aufnahme hat eine *Dauer* `Dauer`. Ein Interpret nimmt dasselbe Stück nur einmal an einem Tag auf.
- Ein *Musikstück* hat eine *Stücknummer* `StNr`, eine *Stückbezeichnung* `StBez`.
- Ein Musikstück stammt von einem *Komponisten* mit einer *Komponistennummer* `KomNr`, einem *Komponistennamen* `KomName` sowie dessen *Geburtsdatum* `KomGeb`.
- Von einem *Interpreten* werden eine *Interpretennummer* `IntNr`, ein *Interpretenname* `IntName` und das *Geburtsdatum* `IntGeb` gespeichert.
- Ein Interpret bekommt pro verkaufter CD ein *Honorar* `Hono`.

Aufgabe:

a. Schreiben Sie eine Relvar `CDWorld` mit den angeführten Attributen auf.

b. Ermitteln Sie den Schlüssel(kandidaten) für diese Relvar.

c. Lösen Sie das *relationswertige Attribut* `Aufnahme` auf. Geben sie den neuen Relvar-Kopf an.

d. Wie ändert sich der Schlüssel?

e. Geben Sie die Menge der voll funktionalen Abhängigkeiten an.

f. Zeichnen Sie ein vollständiges Abhängigkeitsdiagramm.

g. Warum ist diese Relvar nicht in 2NF?

h. Spalten Sie die Relvar in Relvars mit mindestens 2NF auf. Begründen Sie für alle Relvars, dass sie in 2NF sind.

i. Welche dieser Relvars sind bereits in 3NF? Welche nicht? Begründen Sie Ihre Entscheidung.

j. Bringen Sie alle Relvars in BCNF. Begründen Sie für alle Relvars, dass sie in BCNF sind.

k. Ergänzen Sie in allen Relvars zweckmäßige *FK-Constraints* in der Form `Attr>Relvar`.

l. Welche weitere *Constraints* sind erforderlich bzw. sinnvoll?

m. Zeichnen Sie ein ERD, das zum gleichen Ergebnis führt.

n. Interpreten und Komponisten sind ähnliche Entity-Typen. Vereinfachen Sie das ERD, in dem Sie diese zusammenfassen.

o. Nachträglich wird erkannt, dass die *Reihenfolge der Aufnahmen* auf einer CD nicht definiert wurde. Es wird ein Attribut `NrAufCD` in der ursprünglichen Relvar ergänzt. Was ändert sich dadurch?

KAPITEL 7

Relationale Entwurfstheorie

7.1 Armstrong-Axiome

Für eine Relvar R ist eine Menge F von *funktionalen Abhängigkeiten (FA)* gegeben:

- Welche zusätzlichen FA können daraus abgeleitet werden?
- Lässt sich zeigen oder widerlegen, dass eine bestimmte FA aus den in F vorhandenen ableitbar ist?

Diese Fragestellungen können durch die **Ableitungsregeln (inference rules)** von Armstrong[1] beantwortet werden.

Die Regeln werden auch als **Armstrong-Axiome** bezeichnet. Ein **Axiom** ist ein Grundsatz, der nicht bewiesen werden muss (gr. axioma = Grundsatz).

Bedeutung verwendeter Symbole:

- Relvars: R, S
- Attribute: A, B, C, …
- Attributmengen: W, X, Y, Z, K
- Mengen funktionaler Abhängigkeiten: F, G

Reflexivität (reflexivity) Wenn $Y \subseteq X$, dann $X \to Y$ (triviale Abhängigkeit).

Erweiterung, Verstärkung (augmentation) Wenn $X \to Y$, dann $XZ \to YZ$ (für alle Attributmengen Z). *Hinweis:* XZ ist verkürzt für $X \cup Z$.

Transitivität (transitivity) Wenn $X \to Y$ und $Y \to Z$, dann $X \to Z$.

Die Axiome sind

- **korrekt (sound)**, d.h. sie leiten von F nur wirklich gültige FA ab.
- **vollständig (complete)**, d.h. sie erzeugen bei wiederholter Anwendung alle von F implizierten FA.

Beispiel:

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

Gilt auch $AB \to EH$?

1. $AB \to C$ gegeben

[1] William Ward Armstrong, kanadischer Mathematiker und Informatiker, 1974.

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 drei *Armstrong-Axiomen* können weitere Ableitungsregeln hergeleitet werden:

Vereinigung (union, additivity) Wenn $X \rightarrow Y$ und $X \rightarrow Z$, $X \rightarrow YZ$.

Zerlegung (decomposition, projectivity) 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

Ist F eine *Menge von Abhängigkeiten (set of dependencies)*, so ist die **transitive Hülle (closure)** F^+ von F die Menge aller funktionalen Abhängigkeiten, die aus F durch Anwendung der Armstrong-Axiome ableitbar sind.

Wann sind zwei Mengen G und F von FA *gleichwertig (äquivalent)*? Genau dann, wenn $G^+ = F^+$. Dafür schreibt man auch $G \equiv F$. Die Hülle einer Menge von FA ist umfangreich, da sie z.B. auch alle *trivialen FA* enthält. Sie wird daher praktisch nie explizit ermittelt.

Wenn unterschiedliche Mengen von FA *äquivalent* sein können, muss es unter diesen eine *minimale äquivalente Menge von FA* geben. Sie enthält in den FA keine überflüssigen Attribute. Diese kleinste gleichwertige Menge von FA heißt **minimale** oder **kanonische Überdeckung**[2] F_* von F. Es gilt: $F_*^+ = F^+$ oder $F_* \equiv F$.

7.3 Hülle einer Menge von Attributen

Gegeben ist eine Menge X von Attributen und eine Menge F von funktionalen Abhängigkeiten. Welche Menge von Attributen ist von X direkt oder indirekt (transitiv) bezüglich F abhängig? Diese Menge X^+ heißt **Attributhülle von X (closure of attribute set X)**.

Formal ist die **Hülle einer Menge von Attributen X** die Vereinigung aller Attributmengen Y, sodass $X \rightarrow Y$ in F^+ enthalten ist:

$$X^+ = \{Y | X \rightarrow Y \in F^+\}$$

Ein Algorithmus zur Hüllenbildung in Pseudocode:

```
result := X
repeat
  temp := result
  for each Y -> Z in F do
    if Y in result then
      result := result union Z
until temp = result
```

Fragestellung: Gegeben eine Menge F von FA, kann $X \rightarrow Y$ daraus abgeleitet werden, d.h. ist $X \rightarrow Y \in F^+$?
Anwort: Genau dann, wenn $Y \subseteq X^+$.

[2] Kanon = Norm, Regel, siehe Kemper 6.3.1.

Ein **Schlüsselattribut (key attribute)** einer Relvar R ist ein Attribut, das in mindestens einem Schlüsselkandidaten vorkommt (auch **primes Attribut (prime attribute)**). Andernfalls heißt es **Nichtschlüsselattribut (nonkey attribute)** oder **nicht primes Attribut (non-prime attribute)**.

Fragestellung: Ist die Attributmenge K *Superschlüssel (superkey)* der Relvar R?
Antwort: Genau dann, wenn K^+ alle Attribute von R umfasst, d.h. $K^+ =$ Kopf (heading) von R).

Fragestellung: Ist die Attributmenge K *Schlüsselkandidat (candidate key)* der Relvar R?
Antwort: Genau dann, wenn K^+ alle Attribute von R umfasst und K minimal (nicht reduzierbar) ist. Minimal bedeutet: wird aus K auch nur ein Attribute entfernt, ist die Restmenge *kein* Superschlüssel mehr.

Ermittlung eines Schlüsselkandidaten zu einer Relvar R und einer Menge F von FA: K enthält sicher alle Attribute, die nur auf der linken Seite aller FAs vorkommen sowie alle Attribute, die in keiner FA vorkommen.

7.4 Zerlegung von Relvars

Decomposition of relvars.

7.4.1 Hinreichende und notwendige Bedingungen

Notwendige Bedingung (necessary condition) Notwendige Bedingungen $B_1, B_2, ...$, damit ein Auto fährt K: Räder, Motor, Kraftstoff, jedoch nicht hinreichend, d.h. $K \rightarrow B_1 \wedge B_2 \wedge ... \wedge B_n$ (Pfeil hier für *impliziert*).

Hinreichende Bedingung (sufficient condition) Wenn eine Zahl durch 4 teilbar ist, ist sie gerade (hinreichend, aber nicht notwendig), d.h. $B_1 \vee B_2 \vee ... \vee B_n \rightarrow K$.

Notwendig und hinreichend: Äquivalent, *iff*, $\Leftrightarrow$: Wenn eine Zahl durch 2 teilbar ist, ist sie gerade.

7.4.2 Verbundtreue Zerlegung

Verlustfreie Zerlegung (lossless decomposition): eine Zerlegung ist **verbundtreu**, wenn der *natürliche Verbund (natural join)* zwischen den Relvars wieder die ursprüngliche Relvar 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

R = R1 natural join R2
```

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

R != R1 natural join R2
```

Eine Zerlegung ist *verbundtreu*, wenn der Durchschnitt der Attributmengen der zerlegten Relvars Schlüssel in einer dieser Relationen ist (*hinreichende* Bedingung).

In Beispiel 1 ist $\{B\}$ als Durchschnitt der Attributmengen Schlüssel in R2, in Beispiel 2 nicht.

7.4.3 Abhängigkeitstreue Zerlegung

Dependency-preserving decomposition.

Eine Zerlegung ist **abhängigkeitstreu**, wenn jede FA der ursprünglichen Relvar nach der Zerlegung in mindestens einer Relvar wieder auftritt.

Wenn FA verloren gehen, dann werden diese Constraints nicht mehr durch die Relvar-Struktur gewährleistet (das Constraint geht über mehrere Relvars). Es müssen Relvar-übergreifende Constraints definiert werden bzw. in SQL Datenbanken mit Trigger gewährleistet werden.

7.4.4 Zusammenfassung

- Bis 3NF: immer *verbundtreu*, immer *abhängigkeitstreu*.
- BCNF: immer *verbundtreu*, *nicht* immer *abhängigkeitstreu*.
- Zu jeder Relvar R existiert eine Zerlegung in 3NF Relvars, die *verbundtreu* und *abhängigkeitstreu* ist.
- Zu jeder Relvar R existiert eine Zerlegung in BCNF Relvars, die *verbundtreu* ist.

7.5 Übungen zur relationalen Entwurfstheorie

7.5.1 Übung 1

$R(A, B, C, D), F = \{A \to B, C \to D\}$

Gilt auch $AC \to BD$?

a) Beweis mit Armstrong-Axiomen.

b) Beweis mit geeigneter Hüllenbildung.

7.5.2 Übung 2

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

a) Gilt auch $AB \to EH$? *Hinweis:* Ermittlung von $\{A, B\}^+$.

b) Ist $\{A, B\}$ Superschlüssel in R?

c) Ist $\{A, B\}$ Schlüsselkandidat in R?

d) Welche der folgenden Mengen ist Superschlüssel, welche ist Schlüsselkandidat?

1. $\{A, C, D\}$
2. $\{A, D\}$
3. $\{B, C\}$
4. $\{A, C\}$
5. $\{A, B, E\}$

7.5.3 Übung 3

Eine Relvar $R(A, B, C, D)$ ist gegeben, sowie für jede Teilaufgabe eine Menge von FAs. Ermitteln Sie jeweils:

- Schlüsselkandidat(en)
- Höchste NF, in der sich die Relvar befindet.
- 3NF

a) $\{A \to B\}$

b) $\{AB \to C, C \to D\}$

c) $\{A \to B, B \to C, C \to D\}$

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

e) $\{\}$

7.5.4 Übung 4

Folgende Abkürzungen werden benutzt:

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

Folgende Abhängigkeiten wurden ermittelt:

$F = \{W \to Z, A \to H, AW \to M, H \to B\}$

a) Welche Schlüssel(kandidaten) hat die Relvar $R(W, Z, H, B, A, M)$?

b) Welche Redundanzen und Anomalien treten auf, wenn R in $R1(H, B, A)$ und $R2(W, Z, A, M)$ zerlegt wird?

c) Ermitteln Sie eine Zerlegung, die in 3NF ist.

7.5.5 Übung 5

$R(A, B, C, D, E, H), F = \{A \to BC, E \to CH, B \to E, CD \to EH\}$

a) Ist $BC \to AE \in F^+$? (Ermittlung von $\{B, C\}^+$)

b) Ist AB Superschlüssel in R?

c) Ist AE Superschlüssel in R?

d) Ist ACD Superschlüssel in R?

e) Ist ACD Schlüsselkandidat in R?

f) Ist AD Schlüsselkandidat in R?

7.5.6 Übung 6

$R(A, B, C, D, E), F = \{B \to ACD, E \to C, AD \to B, D \to E\}$

a) Ist $D \to C \in F^+$? (Ermittlung von $\{D\}^+$)

b) Ist AE Superschlüssel in R?

c) Ist AD Superschlüssel in R?

d) Ist AD Schlüsselkandidat in R?

e) Ist ADE Schlüsselkandidat in R?

f) Geben Sie alle Schlüsselkandidaten an.

g) In welcher NF ist R, eventuell abhängig vom Schlüsselkandidaten?

7.5.7 Übung 7

$R(A, B, C, D, E, H), F = \{AB \to EH, D \to C, E \to C, EH \to B\}$

a) Geben Sie alle Schlüsselkandidaten an.

b) In welcher NF ist R, eventuell abhängig vom Schlüsselkandidaten?

7.5.8 Übung 8

Telefonbuchdaten wurden in folgender Form abgelegt:

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.
- Ein Mitarbeiter darf nicht mehrere Klappen im selben Raum haben.
- Jeder Mitarbeiter kann Arbeitsplätze in mehreren Räumen haben.

a) Geben Sie alle FAs an (minimale Überdeckung), die in dieser Tabelle zu finden sind.

b) Geben Sie alle möglichen Schlüsselkandidaten für diese Relvar an.

c) Zeigen Sie anhand der angegebenen Relation Beispiele für mögliche Anomalien.

d) Befindet sich die Relvar in 2NF? Begründen Sie Ihre Antwort.

e) Wie sieht die 3NF aus?

7.5.9 Übung 9

$R(A, B, C, D, E), F = \{A \to B, AD \to C, DB \to E, B \to C\}$

a) Welche der angegebenen FAs sind redundant?

b) Geben Sie alle Schlüsselkandidaten an.

7.5.10 Übung 10

$R(A, B, C, D, E, H), F = \{ABC \to D, B \to E, C \to H, EH \to D\}$

Welches Attribut in $ABC \to D$ ist überflüssig?

7.5.11 Übung 11

Beweisen Sie unter Anwendung der *Armstrong-Axiome* die zusätzlichen Ableitungsregeln:

a) *Vereinigung*

b) *Zerlegung*

c) *Pseudotransitivität*

7.5.12 Exercise 12

a) Watch the Chris Date Master Class video *Normalization: FD axiomatization*.

b) Read chapter 7 „FD Axiomatization“ in Chris Date: „Database Design and Relational Theory“.

KAPITEL 8

SQL

8.1 Datenbank LT: Lieferanten - Teile - Lieferungen

Tabelle l: Lieferanten

lnr	lname	rabatt	stadt
L1	Schmid	20	London
L2	Jonas	10	Paris
L3	Berger	30	Paris
L4	Klein	20	London
L5	Adam	30	Athen

Tabelle t: Teile

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

Tabelle lt: Lieferungen

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 SQL als ISO-Standard

SQL ist die *genormte Datenbanksprache* (Standardschnittstelle) für praktisch alle RDBMS. Hersteller konkreter RDBMS implementieren allerdings nur einen Teil der Norm neben zusätzlichen eigenen Erweiterungen.

SQL ist eine **deklarative** Sprache. Sie beschreibt nicht den schrittweisen Lösungsweg, sondern *das gewünschte Ergebnis*.

Eine wichtiger Bestandteil des RDBMS, der *Abfrageoptimierer (Query Optimizer)*, ist dafür zuständig, mögliche Lösungswege zu finden und zu bewerten. Anhand des besten *Ausführungsplans* wird das Ergebnis zu ermittelt.

Sprachen wie *Java* oder *Python* hingegen sind **prozedural (imperativ)**. Hier muss das Lösungsverfahren (Algorithmus) in genauen Einzelschritten angegeben werden.

8.2.1 Historische Entwicklung

- Vorläufer: **SEQUEL (Structured English Query Language)**, 1973
- **SQL (Structured Query Language)**, 1979, Fa. Oracle (Aussprache: *es-kju-el* oder *siquel*)
- SQL-86, SQL-87 (ANSI X3.135-1986, ISO 9705:1987), ca. 100 Seiten
- SQL-89 (ANSI X3.135-1989, ISO 9075:1989), ca. 120 Seiten
- SQL-92, SQL2 (ANSI X3.135-1992 ISO 9075:1992), 3 Teildokumente, ca. 600 (1996: 1120) Seiten, drei Stufen: Full SQL-92, Intermediate SQL-92, Entry SQL-92
- SQL:1999, SQL3 (ISO/IEC 9075:1999), 5 Teildokumente, 2084 Seiten, u.a. objektorientierte Erweiterungen.
- SQL:2003 (ISO/IEC 9075:2003), 9 Teildokumente, 3606 Seiten
- SQL:2006 (ISO/IEC 9075-14:2006), Erweiterungen für XML mit SQL.
- SQL:2008 (ISO/IEC 9075:2008), 9 Teildokumente, 3906 Seiten
- SQL:2011 (ISO/IEC 9075:2011), 9 Teildokumente, 2456 Seiten
- SQL:2016 (ISO/IEC 9075:2016), 10 Teildokumente, 4348 Seiten

8.2.2 SQL Befehlsgruppen

- Datendefinitionen, DDL (Data Definition Language):

```
CREATE ALTER DROP
```

- Datenmanipulationen, DML (Data Manipulation Language):

```
INSERT UPDATE DELETE
```

- Datenabfrage, DQL (Data Query Language):

```
SELECT
```

- Datenzugriffsberechtigungen, DCL (Data Control Language):

```
GRANT REVOKE
```

- Transaktionssteuerung, TCL (Transaction Control Language):

```
BEGIN␣TRANSACTION COMMIT ROLLBACK SAVEPOINT
```

8.2.3 Verwendung von SQL

- *Interaktive textbasierte Shell (REPL)*: DBMS bieten interaktive Kommandozeilen-Werkzeuge, die SQL-Kommandos und Shell-Kommandos interpretieren, z.B. `sqlite3` (SQLite), `sqlcmd` (SQL Server), `sqlplus` (Oracle), `mysql` (MySQL, MariaDB), `psql` (PostgreSQL)
- *Ausführung über GUI-IDE oder webbasierte GUI*: SQL Server Management Studio (SSMS), Oracle SQL Developer, PostgreSQL pgAdmin, MySQL Workbench, DBeaver, Visual Studio Code usw.
- *Ausführung über Anwendungsprogramm mit Datenbank-API*: JDBC (Java), ADO.NET (C#), Python Database API

- *Indirekte Ausführung über Object-Relational Mapper (ORM)*: Hibernate (Java), Entity Framework (.NET), SQLAlchemy (Python)

8.3 Möglichkeiten zur Syntaxbeschreibung

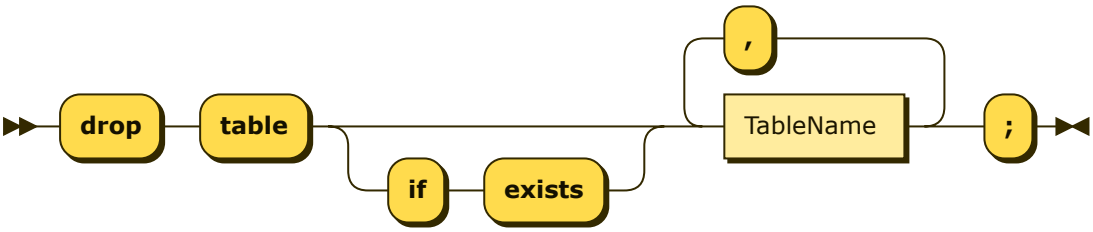
Für alle erlaubten SQL-Kommandos muss definiert werden, welche *Schlüsselwörter* erlaubt sind und welche Operanden bzw. Operatoren erforderlich sind.

Manche Teile (Klauseln) von Anweisungen sind verpflichtend, andere Teile optional. Hier ist ein gültiges `DROP`-Kommando zum Verwerfen der Tabelle *l* in seiner einfachsten Form:

```
drop table l
;                                          -- droptbl.0010.sql
```

Etwaige weitere Klauseln können der *Syntaxbeschreibung* der Herstellerdokumentation entnommen werden. Für die Syntaxbeschreibung von SQL sind bei den RDBMS-Herstellern folgende drei Möglichkeiten verbreitet.

8.3.1 Formale Beschreibung mit Syntaxdiagrammen



- Die Diagramme werden auch *Eisenbahndiagramme (railroad diagrams)* genannt. Alle Wege durch dieses Diagramm ergeben syntaktisch korrekte Anweisungen bzw. Ausdrücke.
- *Abgerundete Rechtecke* heißen **Terminalsymbole** (kurz: Terminale). Sie müssen zeichengetreu eingetippt werden.
- *Rechtecke* heißen **Nonterminalsymbole** (kurz: Nonterminale). Sie bezeichnen *Unterdiagramme*, die getrennt beschrieben werden.

Vorsicht: Manche Hersteller verwenden Rechtecke für Terminale und abgerundete Rechtecke für Nonterminale.

8.3.2 Formale Beschreibung mit EBNF

EBNF (Extended Backus-Naur Form) ist eine **Metasprache** zur Syntaxbeschreibung von Computersprachen. Das obige Syntaxdiagramm in EBNF:

```
DropTable ::= 'drop' 'table' ( 'if' 'exists' )?  TableName ( ',' TableName )* ';'
```

Übung:

- Rufen Sie im Webbrowser den Railroad Diagram Generator[3] auf.
- Geben Sie die obige EBNF-Grammatik für `DROP TABLE` im Reiter *Edit Grammar* ein.
- Erzeugen Sie durch Klicken auf *View Diagram* daraus ein *gleichbedeutendes* Syntaxdiagramm.

- **Terminale** werden in einfache Hochkommas eingeschlossen, z.B. `drop`.

[3] https://bottlecaps.de/rr/ui

- **Nonterminale** sind Namen, z.B. `TableName`, die in einer eigenen **Regel (Produktion)** später genauer beschrieben werden. Das Symbol `::=` bedeutet *ist definiert als*.
- Eine **Sprachbeschreibung (Grammatik)** besteht aus ein oder mehreren Regeln.
- Die Zeichen `( ) | ? * +` sind Symbole von EBNF (**Metasymbole**).
- Klammern `( )` gruppieren Symbole.
- `|` bedeutet *oder* und trennt Alternativen.
- `?` bedeutet, dass die Gruppe davor einmal oder keinmal vorkommt, also *optional* ist.
- `*` bedeutet, dass die Gruppe davor *keinmal* oder beliebig oft vorkommt.
- `+` bedeutet, dass die Gruppe davor *einmal* oder beliebig oft vorkommt.

Bemerkung: Die Syntax von EBNF ist nicht einheitlich. Häufig werden andere Metasymbole benützt. Hier wird die EBNF-Notation des W3C für XQuery[4] verwendet.

Übung:

- Die Metasprache EBNF kann selbst in EBNF beschrieben werden.
- Rufen Sie im Webbrowser den Railroad Diagram Generator[5] auf.
- Klicken Sie auf den Reiter *Get Grammar* und wählen Sie *EBNF Notation*.
- Sehen Sie sich das Syntaxdiagramm im Register *View Diagram* an.

8.3.3 Informelle Beschreibung mit EBNF-ähnlicher Notation

Beispiel aus der Hersteller-Dokumentation von *SQL Server*:

```
-- Syntax for SQL Server and Azure SQL Database

DROP TABLE [ IF EXISTS ] { database_name.schema_name.table_name | schema_name.
↪table_name | table_name } [ ,...n ]
[ ; ]
```

8.3.4 Übungen

8.4 Datendefinitionen (DDL)

Syntaxbeschreibung für das Anlegen einer Tabelle:

```
CREATE TABLE TableName ( ColumnDef [ ,...n ] [ , { TableConsDef [ ,...n ] } ] ) ;

ColumnDef ::= ColumnName DataType [ ColumnConsDef [ ...n ] ] [ DefaultValue ]

DefaultValue ::= { Literal | NiladicFunctionRef | NULL }

ColumnConsDef ::= { [ NOT ] NULL | UNIQUE | PRIMARY KEY |
                    CHECK ( CheckCondition ) |
                    REFERENCES TableName [ ( ColumnName ) ] [ RefAction [ ...n ] ]}
```

(Fortsetzung auf der nächsten Seite)

[4] https://www.w3.org/TR/2010/REC-xquery-20101214/#id-grammar
[5] https://bottlecaps.de/rr/ui

(Fortsetzung der vorherigen Seite)

```
TableConsDef ::=
  { UNIQUE ColumnNameList } |
    PRIMARY KEY ColumnNameList } |
    CHECK ( CheckCondition ) |
    FOREIGN KEY ColumnNameList REFERENCES TableName [ ColumnNameList ] [ RefAction␣
↪[ ...n ] ] }

ColumnNameList ::= ( ColumnName [ ,...n ] )

RefAction ::= ON { DELETE | UPDATE } { NO ACTION | SET NULL | SET DEFAULT |␣
↪CASCADE }
```

Fallbeispiele für `CREATE TABLE`:

```
create table l (
  lnr    varchar(2) not null primary key
, lname  varchar(6) not null
, rabatt decimal(3) not null
, stadt  varchar(6) not null
);

create table lt (
  lnr    varchar(2) not null references l
, tnr    varchar(2) not null references t
, menge  decimal(4) not null check(menge > 0)
, primary key (lnr, tnr)
);
                                        -- createtbl.0005.sql
```

```
create table Filiale (
  FilialNr  integer     not null primary key
, Strasse   varchar(64) not null
, Ort       varchar(64) not null
, PLZ       varchar(16) not null
, Telefon   varchar(32) not null
, LeiterSVN char(10)    not null unique references Angestellter(SVN)
);
                                        -- createtbl.0008.sql
```

```
create table student (
  studid  varchar(16) not null primary key
, ord     integer     not null check(ord > 0)
, born    date        not null
, sex     varchar(1)  not null check(sex in ('m', 'f')) default 'm'
, last    varchar(32) not null
, first   varchar(32) not null
, class   varchar(8)  not null
, year    varchar(8)  not null
, dropout varchar(1)  not null check (dropout in ('y', 'n'))
);
                                        -- createtbl.0010.sql
```

```
create table mcanswer (
  mcid   integer     not null -- \
, gid    integer     not null --  > mcsolution
, qid    integer     not null -- /
, studid varchar(16) not null
, answer varchar(16) not null
, primary key (mcid, gid, qid, studid)
, foreign key (mcid, gid, qid) references mcsolution
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
, foreign key (mcid, studid) references mcwhen
);
                                        -- createtbl.0020.sql
```

```
create table kveranst (
  knr      integer     not null references kurs
, knrlfnd  integer     not null
, von      date        not null
, bis      date        not null
, ort      varchar(10) not null
, plaetze  integer     not null
, pnr      integer               references referent -- NULL wenn (noch) kein␣
↪Referent
, primary key (knr, knrlfnd)
, check (von <= bis)
);
                                        -- createtbl.0030.sql
```

8.5 Indizes

Ein **Index** (Mehrzahl: Indizes) ist mit dem *Stichwortverzeichnis* eines Buches vergleichbar und kann sowohl Such- als auch Sortiervorgänge um mehrere Größenordnungen beschleunigen. Für *Primary Key*- und *Unique-Constraints* werden automatisch Indizes angelegt.

Hinweis: *Es ist ratsam, auch Fremdschlüssel* immer zu indizieren, da es sonst in einigen Fällen statt zur Sperre einzelner Datensätze zur Sperre der gesamten Tabelle kommt, wenn DML auf der referenzierten Tabelle durchgeführt wird.

Index erstellen:

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

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

Beispiel:

```
create index idx_lt_lnr
          on lt(lnr)
;                                       -- createidx.0010.sql
```

Tipp: Keinen Index als Beginnteil (Präfix) eines anderen Index definieren, da in diesem Fall der bestehende Index verwendet werden kann. Suche nach Spaltenkombinationen, die kein Präfix sind, können den Index nicht verwenden. Es muss ein weiterer Index angelegt werden.

Index löschen:

```
DROP INDEX index_name ;
```

Beispiel:

```
drop index idx_lt_lnr
;                                       -- dropidx.0010.sql
```

Übung:

- Schreiben Sie die Anweisung auf, die einen Index auf die Spalte `tnr` von `lt` erstellt. Achten Sie auf die Namenskonvention.
- Schreiben Sie weiters eine Anweisung auf, die diesen Index wieder verwirft.

8.6 Datenzugriffsberechtigungen (DCL)

Bei *Mehrbenutzersystemen* sind *Benutzerkonten* und *Berechtigungen* erforderlich. Berechtigungen werden mit den Anweisungen der **Data Control Language (DCL)** erteilt oder entzogen.

Der Benutzer, der eine Tabelle erstellt, ist deren Eigentümer und hat die Berechtigungen `SELECT`, `INSERT`, `UPDATE` und `DELETE` auf diese Tabelle. Berechtigungen können an andere Benutzer weitergegeben werden.

Erteilen von Berechtigungen:

```
GRANT { ALL [PRIVILEGES] | permission-commalist }
   ON table-name
   TO user-name-commalist [ WITH GRANT OPTION ] ;

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

`WITH GRANT OPTION`: die erteilten Rechte dürfen vom Empfänger an andere Benutzer weitergegeben werden.

Beispiel:

```
grant select, update
   on l
   to pz
;                                          -- grant.0010.sql
```

Entziehen von Berechtigungen:

```
REVOKE { ALL [ PRIVILEGES ] | operation-commalist }
    ON table-name
  FROM user-name-commalist ;
```

Hinweis:

Da das RDBMS `SQLite` nicht mehrbenutzerfähig ist, gibt es dort keine Benutzerkonten und auch keine DCL-Kommandos.

Übung:

- Erteilen Sie dem Benutzer `pz` das Recht zum Löschen von Datensätzen in `l`.
- Schreiben Sie eine Anweisung auf, die dem Benutzer `pz` das Recht auf `l` entzieht, bestehende Datensätze zu ändern.
- Schreiben Sie eine Anweisung auf, die dem Benutzer `hut` alle Rechte auf `l` entzieht.

8.7 Datenauswahl mit SELECT (DQL)

Der einzige Befehl der Gruppe **Data Query Language (DQL)** ist SELECT. Eine SELECT-Anweisung besteht aus mehreren *Klauseln* (SELECT-Klausel, WHERE-Klausel, usw.). Einige Klauseln sind optional, die Reihenfolge der Klauseln ist jedoch zwingend:

```
SELECT     <-- Auswahl der Spalten (Projektion)
  FROM     <-- Tabellen, die abgefragt werden
 WHERE     <-- opt. Auswahl der Datensätze (Restriktion, Filter)
 GROUP BY  <-- opt. Gruppierung der Datensätze nach gleichen Werten
HAVING     <-- opt. Auswahl nach der Gruppierung
 ORDER BY  <-- opt. Sortierung der Datensätze
```

Das Ergebnis einer Abfrage mit SELECT ist wieder eine Tabelle, die gegenenfalls in Form einer *Unterabfrage* in einer anderen Abfrage (oder auch in DML-Kommandos) verwendet werden kann. Diese Eigenschaft bezeichnet man als **Abgeschlossenheit (Closure)**.

Übung:

- Sind die natürlichen Zahlen $\mathbb{N}$ bezüglich der Addition + abgeschlossen?
- Bezüglich der Subtraktion −? Warum nicht?
- Welche Zahlenmenge ist auch bezüglich der Subtraktion abgeschlossen?

Informelle Syntaxbeschreibung von SELECT:

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

table-reference ::= table-name [ alias-name ]

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

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

Bemerkung: Seit *SQL:1999* darf eine WITH-Klausel vor der SELECT-Klausel stehen, in der für komplexere Abfragen benannte Unterabfragen definiert werden können. Mit dem Schlüsselwort RECURSIVE sind dort auch rekursive Abfragen möglich.

8.7.1 Auswahl von Spalten (Projektion)

Auswahl von Spalten (Attribute) aus einer Tabelle. Der Fachbegriff kommt aus der Mengentheorie und lautet *Projektion*. Mit * werden alle Spalten (Attribute) ausgewählt. Um alle Datensätze der Datenbank LT abzufragen:

```
select *
  from l
;

select *
  from t
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
;

select *
  from lt
;
```

Vorsicht: Diese Abfrage hat eine völlig andere Bedeutung und ergibt sehr viele Datensätze:

```
select *
  from l, t, lt
;
```

Sie kombiniert jeden Datensatz von `l` mit jedem Datensatz von `t` und dieses Ergebnis wiederum mit jedem Datensatz von `lt` (*Kreuzprodukt*, *kartesisches Produkt*, `CROSS JOIN`).

Die Ausführung von

```
select tnr, tname, preis
  from t
;
```

auf der Tabelle

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

ergibt jene Spalten (Attribute), die in der `SELECT`-Klausel angegeben wurden:

tnr	tname	preis
T1	Mutter	12
T2	Bolzen	17
T3	Schraube	17
T4	Schraube	14
T5	Welle	12
T6	Zahnrad	19

Welche Farben haben die Teile?

```
select farbe
  from t
;                                          -- proj.0010.sql
```

Ergebnis:

```
| farbe |
| ----- |
| rot   |
| gelb  |
| blau  |
| rot   |
| blau  |
| rot   |
```

Einige Farben sind mehrfach enthalten. SQL folgt hier nicht dem *relationalen Datenbankmodell*, das als Menge von Tupeln kein Element mehrfach enthält. Werden mehrere gleiche Datensätze nur einmal gewünscht, muss zusätzlich `DISTINCT` (*engl. unterschiedlich*) angegeben werden:

```
select distinct farbe
  from t
;                                          -- proj.0020.sql
```

Ergebnis:

```
| farbe |
| ----- |
| blau  |
| gelb  |
| rot   |
```

Bemerkung: Wird statt `DISTINCT` das optionale Schlüsselwort `ALL` angegeben, ist das Verhalten gleich wie beim ursprünglichen `SELECT`: mehrfache Datensätze sind im Ergebnis enthalten. `ALL` wird grundsätzlich nicht angeführt.

Umbenennung von Spalten (Aliasnamen)

Manchmal ist es erwünscht oder nötig, den Namen einer Spalte zu ändern:

```
select farbe "colour"
     , preis "price"
     , stadt "city"
  from t
;

-- Möglich, aber nicht empfohlen

select farbe as colour
     , preis as price
     , stadt as city
  from t
;                                          -- proj.0030.sql
```

Beide Syntaxvarianten sind zulässig, wir verwenden jedoch *ausschließlich* die Variante *ohne* das Schlüsselwort `AS`.

Geänderte Spaltennamen werden auch als *Aliasnamen* bezeichnet.

Ergebnis:

```
| colour | price | city   |
| ------ | ----- | ------ |
| rot    | 12.00 | London |
| gelb   | 17.00 | Paris  |
| blau   | 17.00 | Rom    |
| rot    | 14.00 | London |
| blau   | 12.00 | Paris  |
| rot    | 19.00 | London |
```

Achtung:

Aliasnamen dürfen *nicht* in der der `WHERE`-Klausel verwendet werden (Ausnahme: `SQLite`), sind aber in der `ORDER BY`-Klausel erlaubt.

Verwendung von Ausdrücken

Anstelle von Spaltennamen können auch Ausdrücke angegeben werden, die verschiedenste Operationen wie etwa Berechnungen durchführen.

Teile (`tnr, tname`) mit verdoppeltem Preis anzeigen:

```
select tnr, tname, preis
     , preis * 2 "dpreis"
  from t
;                                       -- proj.0040.sql
```

Ergebnis:

```
| tnr | tname    | preis | dpreis |
| --- | -------- | ----- | ------ |
| T1  | Mutter   | 12.00 | 24.00  |
| T2  | Bolzen   | 17.00 | 34.00  |
| T3  | Schraube | 17.00 | 34.00  |
| T4  | Schraube | 14.00 | 28.00  |
| T5  | Welle    | 12.00 | 24.00  |
| T6  | Zahnrad  | 19.00 | 38.00  |
```

Wichtig:

Bei Berechnungen *muss* ein Name für die Spalte angegeben werden, da unbenannte Attribute im relationalen Datenbankmodell unzulässig sind und es auch in SQL Datenbanken oft zu schwer auffindbaren Fehlern kommt.

Achtung:

Ist einer der Operanden in einem Ausdruck `NULL`, so ergibt der gesamte Ausdruck `NULL` (Ausnahme: Logikoperatoren `NOT AND OR`). Ein Grund mehr, `NULL` in der Datenbank zu vermeiden.

Gängige Operatoren für Berechnungen:

- Arithmetische Operatoren `+ - * /` und Klammern `( )`.
- Modulo: `%` (*Oracle*: `mod` bzw. `mod(x, y)`).
- Potenzierung: `power(x, y)` `^`
- Zeichenkettenverknüpfung: `||` (*SQL Server*: `+`)

Teile und deren Farben in der From `MUTTER in rot` ausgeben:

```
select upper(tname) || ' in ' || farbe "beschreibung"
  from t
;

-- SQL Server erwartet + statt ||

select upper(tname) + ' in ' + farbe "beschreibung"
  from t
;                                       -- proj.0060.sql
```

Ergebnis:

```
| beschreibung      |
| ----------------- |
| MUTTER in rot     |
| BOLZEN in gelb    |
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
| SCHRAUBE in blau  |
| SCHRAUBE in rot   |
| WELLE in blau     |
| ZAHNRAD in rot    |
```

Verwendung von Funktionen in Ausdrücken

In Ausdrücken können auch die zahlreichen **Funktionen** verwendet werden, die vom System bereitgestellt werden. Funktionen dürfen verschachtelt werden:

Teile (`tnr, tname`) mit Quadratwurzel des Preises, gerundet auf zwei Nachkommastellen, anzeigen:

```
select tnr, tname, preis
     , round(sqrt(preis), 2) "wpreis"
  from t
;

-- Abschneiden auf zwei Nachkommastellen bei SQL Server

select tnr, tname, preis
     , cast(round(sqrt(preis), 2) as decimal(4,2)) "wpreis"
  from t
;                                       -- proj.0050.sql
```

Ergebnis:

```
| tnr | tname    | preis | wpreis|
| T1  | Mutter   | 12.00 | 3.46  |
| T2  | Bolzen   | 17.00 | 4.12  |
| T3  | Schraube | 17.00 | 4.12  |
| T4  | Schraube | 14.00 | 3.74  |
| T5  | Welle    | 12.00 | 3.46  |
| T6  | Zahnrad  | 19.00 | 4.36  |
```

Einige gängige Funktionen (*SQL Server*):

- Typumwandlung: `cast convert`
- Datum: `dateadd datediff datepart datefromparts getdate year day month`
- Zeichen und Zeichenketten: `ascii char replicate upper lower ltrim rtrim substring len space charindex translate replace stringagg left right stuff patindex reverse string_split concat`
- Zahlen: `round count sum avg min max sqrt power`
- Behandlung von NULL: `coalesce`
- Logik: `iif`

Hinweis:

Trotz Normierung von SQL weichen Operatoren, Funktionsnamen und Umfang der verfügbaren Funktionen einzelner RDBMS voneinander ab. Hier ist die Herstellerdokumentation heranzuziehen.

8.7.2 WHERE-Klausel (Restriktion)

Auswahl (*Filter*) von Zeilen aus einer Tabelle, die eine Bedingung (*Prädikat*) erfüllen. Der Fachbegriff *Restriktion* bedeutet *Einschränkung*. Die Ausführung von

```
select *
  from t
 where farbe = 'rot'
;
```

auf der Tabelle

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**

ergibt jene Zeilen (Datensätze), die die WHERE-Bedingung erfüllen:

tnr	tname	farbe	preis	stadt
T1	Mutter	rot	12	London
T4	Schraube	rot	14	London
T6	Zahnrad	rot	19	London

Vorsicht: Ergibt der Ausdruck der WHERE-Bedingung NULL, z.B. weil ein Datensatz als Farbe NULL enthält, ist der Datensatz *nicht* im Ergebnis enthalten. Das Verhalten bei NULL ist hier wie beim Wahrheitswert *Falsch*. Das gilt leider nicht allgemein, siehe z.B. bei CHECK-Constraints.

Die Bedingung kann enthalten:

- Logische Operatoren: OR AND NOT
- Vergleichsoperatoren: = <> != < > <= >=
- Bereiche: BETWEEN 10 AND 50
- Enthalten in: IN bzw. NOT IN
- Platzhalter (*Wildcards*): LIKE bzw. NOT LIKE
  - % für beliebige Zeichenfolge (auch leer), wie * bei *Linux Shell*.
  - _ für genau ein beliebiges Zeichen, wie ? bei *Linux Shell*.
- Prüfen auf fehlende Information: IS NULL, IS NOT NULL

Bemerkung: Die Vergleichsoperatoren != und <> für *Ungleichheit* sind *gleichbedeutend*.

Welche Lieferanten sind aus Paris (alle Spalten)?

```
select *
  from l
 where stadt = 'Paris'
;                                        -- where.0006.sql
```

Welche Lieferanten sind aus Paris (Spalten lnr, lname, rabatt)?

```
select lnr, lname, rabatt
  from l
 where stadt = 'Paris'
;                                       -- where.0008.sql
```

BETWEEN: Welche roten Teile haben eine Preis zwischen 13 und 23?

Wichtig: Die Grenzwerte (hier 13 und 23) sind bei BETWEEN im Bereich *enthalten*.

```
select tnr
  from t
 where farbe = 'rot'
   and preis between 13 and 23
;                                       -- where.0010.sql
```

Bemerkung: Der Ausdruck `between 13 and 23` ist nur eine Kurzform für `preis >=13 and preis <=23`. Kurzformen, die in einer Sprache nicht nötig sind, aber Schreibarbeit ersparen, werden als **syntaktischer Zucker (Syntactic Sugar)** bezeichnet.

IN: 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')
;                                       -- where.0020.sql
```

Das ist wiederum eine Kurzform (*Syntactic Sugar*) für:

```
select tnr
  from t
 where (preis = 12 or preis = 14)
   and not (stadt = 'London' or stadt = 'Rom')
;                                       -- where.0021.sql
```

Weitere Variante:

```
select tnr
  from t
 where (preis = 12 or preis = 14)
   and stadt <> 'London'
   and stadt <> 'Rom'
;                                       -- where.0022.sql
```

IN: Alle Lieferanten (Spalte lnr), die Teil T1 oder T2 oder T4 liefern:

```
select distinct lnr
  from lt
 where tnr in ('T1', 'T2', 'T4')
;                                       -- where.0032.sql
```

Variante mit OR:

```
select distinct lnr
  from lt
 where tnr = 'T1'
    or tnr = 'T2'
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    or tnr = 'T4'
;                                       -- where.0030.sql
```

`LIKE`: Welche Teile haben ein `e` im Namen und sind aus einer Stadt mit drei Zeichen Länge?

```
select *
  from t
 where tname like '%e%'
   and stadt like '___'
;                                       -- where.0040.sql
```

Wichtig: Die Zeichen `%` und _ haben **nur** mit `like` eine besondere Bedeutung. Für alle anderen Vergleichsoperatoren trifft das *nicht* zu.

Enthalten Zeichenketten *Wildcard-Zeichen* und soll mit `LIKE` nach diesen gesucht werden, kann die Sonderbedeutung der Wildcard-Zeichen durch ein frei wählbares *Entwertungs-Zeichen* aufgehoben werden:

```
insert into t
  (tnr , tname      , farbe , preis, stadt  )
values
  ('T8', 'Zang%'    , 'blau',    20, 'Wien' )
, ('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 in ('T8', 'T9')
;                                       -- where.0050.sql
```

Wichtig: Der Eintrag `NULL` für ein Attribut in einem Datensatz ist streng genommen *kein Wert*, sondern eine *Markierung (Flag)*, die bedeutet, dass der Wert *unbekannt* ist, *fehlt* oder *nicht anwendbar* ist. `NULL` ist eine vielfältige Fehlerquelle und sollte nach Möglichkeit vermieden werden.

Übung:

1. Ändern Sie das DDL-Kommando von `t`, sodass `NULL` für das Attribut `preis` erlaubt ist.
2. Setzen Sie alle Preise von roten Teilen auf `NULL`:

```
update t
   set preis = null
 where farbe = 'rot'
;
```

3. Fragen Sie zur Überprüfung alle Teile ab.
4. Fragen Sie ab, bei welchen Teilen der Preis fehlt:

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

5. Fragen Sie mit `NOT NULL` alle Teile ab, bei denen der Preis *nicht* fehlt.
6. Welches Ergebnis hat folgende Abfrage (falls sie überhaupt ausführbar ist)?

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

7. Testen Sie die Abfrage und erklären Sie das Resultat.

8.7.3 `ORDER BY` (Sortierung)

Eine Relation ist eine *Menge* von Tupeln (Datensätzen). Die Reihenfolge der Datensätze ist daher undefiniert. Das gilt auch für SQL Datenbanken. Die Reihung der Datensätze bei einer Abfrage ist *implementierungsabhängig* und kann sich sogar bei wiederholter Ausführung der gleichen Abfrage ändern. Wird eine bestimmte Reihenfolge der Datensätze benötigt, muss die `ORDER BY`-Klausel verwendet werden.

Welche Namen haben die Teile (sortiert nach Namen)?

```
select tname
  from t
 order by tname
;                                       -- order.0010.sql
```

Hinweis:

Sortieroperationen belasten die Systemressourcen. Ist eine Ordnung der Datensätze nicht erforderlich, sollte `ORDER BY` nicht verwendet werden.

- Die Spalte oder der Ausdruck, nach dem sortiert wird, muss in der `SELECT`-Klausel *nicht* unbedingt vorkommen.
- Es kann nach mehreren Kriterien sortiert werden. Jedes Kriterium kann unabhängig mit `ASC` *aufsteigend (ascending)* oder mit `DESC` *absteigend (descending)* gereiht werden.
- *Aliasnamen für Spalten*, die in der Projektion vergeben wurden, dürfen in der `ORDER BY`-Klausel verwendet werden. In der `WHERE`-Klausel ist das bei den meisten RDBMS nicht möglich (Ausnahme: *SQLite*).

Statt Spaltennamen können auch Nummern angegeben werden, z.B. `ORDER BY 2, 5` bedeutet, dass zuerst nach der zweiten und bei gleichen Werten danach nach der fünften Spalte der Projektion sortiert wird. Da dies schlechter lesbar ist und bei Änderungen leicht zu Fehlern führen kann, ist diese Notation *nicht* empfohlen und zu vermeiden. Folgende Abfragen haben die gleiche Bedeutung, die erste Variante ist zu bevorzugen:

```
select *
  from t
 order by stadt, preis desc
;

select *
  from t
 order by 5, 4 desc
;                                       -- order.0030.sql
```

`NULL` wird implementierungsabhängig an den Anfang oder ans Ende gereiht. Bei einigen RDBMS kann eine zusätzliche Klausel angegeben werden, beispielsweise bei *SQLite* und *PostgreSQL* mit `NULLS FIRST` bzw. `NULLS LAST`.

```
select *
  from t
 order by preis nulls last
;                                       -- order.0020.sql
```

8.7.4 Verbundoperationen (`JOIN`)

Kreuzprodukt (`CROSS JOIN`)

Beim **Kreuzprodukt** (**kartesisches Produkt**, `CROSS JOIN`, *Kreuzverbund*) wird jeder Datensatz einer Tabelle mit jedem Datensatz einer anderen Tabelle kombiniert.

```
select *
  from l
  cross join t
;                                       -- crossj.0010.sql
```

Ergebnis (Auszug):

```
| lnr | lname  | rabatt | stadt  | tnr | tname    | farbe | preis | stadt_2 |
| --- | ------ | ------ | ------ | --- | -------- | ----- | ----- | ------- |
| L1  | Schmid |     20 | London | T1  | Mutter   | rot   |    12 | London  |
| L1  | Schmid |     20 | London | T2  | Bolzen   | gelb  |    17 | Paris   |
| L1  | Schmid |     20 | London | T3  | Schraube | blau  |    17 | Rom     |
| L1  | Schmid |     20 | London | T4  | Schraube | rot   |    14 | London  |
| L1  | Schmid |     20 | London | T5  | Welle    | blau  |    12 | Paris   |
| L1  | Schmid |     20 | London | T6  | Zahnrad  | rot   |    19 | London  |
| L2  | Jonas  |     10 | Paris  | T1  | Mutter   | rot   |    12 | London  |
| L2  | Jonas  |     10 | Paris  | T2  | Bolzen   | gelb  |    17 | Paris   |
...
| L5  | Adam   |     30 | Athen  | T1  | Mutter   | rot   |    12 | London  |
| L5  | Adam   |     30 | Athen  | T2  | Bolzen   | gelb  |    17 | Paris   |
| L5  | Adam   |     30 | Athen  | T3  | Schraube | blau  |    17 | Rom     |
| L5  | Adam   |     30 | Athen  | T4  | Schraube | rot   |    14 | London  |
| L5  | Adam   |     30 | Athen  | T5  | Welle    | blau  |    12 | Paris   |
| L5  | Adam   |     30 | Athen  | T6  | Zahnrad  | rot   |    19 | London  |
```

Gleichbedeutende ältere Syntax (nicht empfohlen):

```
select *
  from l, t
;                                          -- crossj.0020.sql
```

Übung:

Die folgenden Abfragen ermitteln, wieviele Datensätze in `l` bzw. `t` enthalten sind.

```
select count(*) "anzahl_l"
  from l
;

select count(*) "anzahl_t"
  from t
;                                          -- count.0010.sql
```

```
| anzahl_l           |
| ------------------ |
| 5                  |

| anzahl_t           |
| --------           |
| 6                  |
```

- Überlegen Sie: wie viele Datensätze enthält das obige Kreuzprodukt somit insgesamt?
- Überprüfen Sie Ihre Vorhersage, indem Sie eine entsprechende Abfrage ausführen.

Equi-Join

Semi-Join

Entspricht einem *Equi-Join*, es werden durch die Angabe von z.B. `l.*` aber nur die Spalten einer der beiden Tabellen ausgegeben. Operator-Symbol ist $\ltimes$, also z.B. $l \ltimes t$:

```
select l.*
  from l
  join t on l.stadt = t.stadt
;                                          -- semij.0010.sql
```

Projektion auf die Spalten der zweiten Tabelle, Operator-Symbol ist $\rtimes$, also z.B. $l \rtimes t$:

```
select t.*
  from l
  join t on l.stadt = t.stadt              -- semij.0020.sql
;
```

Self-Join

Natural-Join

Ursprüngliche und einfachste Form einer Verbundoperation (SQLite, von SQL Server nicht unterstützt):

```
select *
  from l natural join t
;                                        -- natj.010.sql
```

- Verknüpft wird über jene Spalten, die in beiden Tabellen den *gleichen Namen* aufweisen.
- Die Werte *aller* gleichnamigen Spalten müssen übereinstimmen (wie *Equi-Join*).
- Gleichnamige Spalten scheinen im Resultat nur einmal auf (Werte ohnehin gleich).

Ergebnis:

```
|------+--------+--------+--------+-----+----------+-------+--------|
|  lnr | lname  | rabatt | stadt  | tnr | tname    | farbe | preis  |
|------+--------+--------+--------+-----+----------+-------+--------|
|  L1  | Schmid | 20     | London | T1  | Mutter   | rot   | 12     |
|  L1  | Schmid | 20     | London | T4  | Schraube | rot   | 14     |
|  L1  | Schmid | 20     | London | T6  | Zahnrad  | rot   | 19     |
|  L2  | Jonas  | 10     | Paris  | T2  | Bolzen   | gelb  | 17     |
|  L2  | Jonas  | 10     | Paris  | T5  | Welle    | blau  | 12     |
|  L3  | Berger | 30     | Paris  | T2  | Bolzen   | gelb  | 17     |
|  L3  | Berger | 30     | Paris  | T5  | Welle    | blau  | 12     |
|  L4  | Klein  | 20     | London | T1  | Mutter   | rot   | 12     |
|  L4  | Klein  | 20     | London | T4  | Schraube | rot   | 14     |
|  L4  | Klein  | 20     | London | T6  | Zahnrad  | rot   | 19     |
|------+--------+--------+--------+-----+----------+-------+--------|
```

- Bei streng relationalen DBMS ist mit `JOIN` der *Natural-Join* der *einzige* Verbundoperator.
- Operator-Symbol $\bowtie$, z.B. $l \bowtie t$.
- Gibt es keine gemeinsamen Spalten, wird das Kreuzprodukt gebildet (`CROSS JOIN`).

8.7.5 Mengenoperationen

Vereinigung

Durchschnitt

Differenz

Übungen

8.7.6 Unterabfragen in der `WHERE`-Klausel

Vergleiche mit einem einfachen Wert

Welche Teile sind teurer als Teil `T4`?

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                )
;                                          -- subswhere.0010.sql
```

- Die Unterabfrage darf nur ein Attribut (Spalte) umfassen.
- Die Unterabfrage darf höchstens einen Datensatz liefern.

Die Beispielabfrage mit Unterabfrage entspricht einer Kombination zweier Abfragen, mit händischem Einsetzen des Ergebnisses der ersten Abfrage in die zweite Abfrage.

Übung:

Lösen Sie diese Aufgabe mit `JOIN`.

Vergleiche mit Wertemengen

Welcher Teil ist teurer als alle anderen?

```
select *
  from t
 where preis >= all (select preis
                       from t
                     )
;                                          -- subswhere.0020.sql
```

Übung:

Durch welche Unterabfrage unter Verwendung einer Aggregatfunktion müssen Sie die Unterabfrage ersetzen, sodass `all` nicht mehr benötigt wird? Schreiben Sie die geänderte Abfrage auf.

Welche Teile sind teurer als zumindest ein anderer Teil?

```
select *
  from t
 where preis > any (select preis
                      from t
                    )
;                                          -- subswhere.0030.sql
```

Bemerkung: Die Schlüsselwörter `any` und `some` sind gleichbedeutend und somit gegeneinander austauschbar.

Übung:

Durch welche Unterabfrage unter Verwendung einer Aggregatfunktion müssen Sie die Unterabfrage ersetzen, sodass `all` nicht mehr benötigt wird? Schreiben Sie die geänderte Abfrage auf.

Welche Lieferanten (`lnr`) liefern ein Teil in einer Menge, die kleiner ist als *jede* Liefermenge des Lieferanten `L4`?

```
select distinct lnr
  from lt
 where menge < all (select menge
                      from lt
                     where lnr = 'L4'
                    )
;                                          -- subswhere.0070.sql
```

Welche Lieferanten (`lnr`) liefern ein Teil in einer Menge, die kleiner ist als *eine* Liefermenge des Lieferanten `L4`?

```
select distinct lnr
  from lt
 where menge < any (select menge
                      from lt
                     where lnr = 'L4'
                   )
;                                          -- subswhere.0080.sql
```

Enthaltensein in einer Wertemenge

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'
;                                          -- subswhere.0040.sql
```

Alle Lieferanten (`lnr`, `lname`), die sowohl das Teil `T1` als auch das Teil `T2` liefern. Lösung mit *Unterabfrage*:

```
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'
               )
;                                          -- subswhere.0050.sql
```

Alle Lieferanten (`lnr`, `lname`), die entweder das Teil `T1` oder das Teil `T2` liefern, aber nicht beide. Lösung mit *Unterabfrage*:

```
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'
                )
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      )
;                                          -- subswhere.0060.sql
```

Übung:

Lösen Sie die gleiche Aufgabenstellung mit Mengenoperationen.

Welche Lieferanten (`lnr`, `lname`) haben keine Teile geliefert?

```
select lnr, lname
  from l
 where lnr not in (select lnr
                     from lt
                   )
;                                          -- subswhere.0090.sql
```

Korrelierte Unterabfragen mit `EXISTS`

Korrelierte Unterabfragen verwenden Werte aus der umgebenden Abfrage. Lösung als *korrelierte* Unterabfrage mit `EXISTS`-Operator: Welche Lieferanten (`lnr`, `lname`) haben keine Teile geliefert?

```
select lnr, lname
  from l
 where not exists (select *
                     from lt
                    where lt.lnr = l.lnr
                   )
;                                          -- subswhere.0100.sql
```

Die Unterabfrage verwendet mit `l.lnr` einen Wert der umgebenden äußeren Abfrage, die für jeden Datensatz von `l` die Unterabfrage einmal ausführt.

Übung:

Wenn die Tabelle `l` einen *zusammengesetzten Primärschlüssel* hätte (und damit `lt` einen *zusammengesetzten Fremdschlüssel*), warum wäre nur die Variante mit `EXISTS` möglich?

Mehrere Varianten: Welche Lieferanten (`lnr`, `lname`) haben Teile geliefert?

```
select distinct l.lnr, lname
  from lt
  join l on lt.lnr = l.lnr
;                                          -- subswhere.0102.sql
```

```
select lnr, lname
  from l
 where lnr in (select lnr
                 from lt
               )
;                                          -- subswhere.0104.sql
```

```
select lnr, lname
  from l
 where exists (select *
                 from lt
                where lt.lnr = l.lnr
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                )
;                                       -- subswhere.0106.sql
```

Bemerkung: Unterabfragen mit `EXISTS` verwenden üblicherweise `*` in der `SELECT`-Klausel, da die Spalten (Attribute) unerheblich sind.

Welche Lieferanten (`lnr`) haben keine roten Teile geliefert?

```
select lnr
  from l
 where lnr not in (select lnr
                     from lt
                    where tnr in (select tnr
                                    from t
                                   where farbe = 'rot'
                                  )
                   )
;                                       -- subswhere.0110.sql
```

```
select lnr
  from l
 where not exists (select lnr
                     from lt
                    where lt.lnr = l.lnr
                      and tnr in (select tnr
                                    from t
                                   where farbe = 'rot'
                                  )
                   )
;                                       -- subswhere.0120.sql
```

```
select lnr
  from l
 where lnr not in (select lnr
                     from lt
                     join t on lt.tnr = t.tnr
                    where farbe = 'rot'
                   )
;                                       -- subswhere.0130.sql
```

```
select lnr
  from l
 where not exists (select *
                     from lt
                     join t on lt.tnr = t.tnr
                    where lt.lnr = l.lnr
                      and farbe = 'rot'
                   )
;                                       -- subswhere.0140.sql
```

Übung:

Die folgende Abfrage löst nur *scheinbar* die gleiche Aufgabenstellung:

```
select distinct lnr
  from lt
  join t on lt.tnr = t.tnr
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
 where farbe != 'rot'
;                                       -- subswhere.0150.sql
```

Warum ist sie falsch und welche Fragestellung beantwortet sie tatsächlich?

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'
                  )
;                                       -- subswhere.0160.sql
```

Welche Lieferanten (`lnr`, `name`) liefern ausschließlich gelbe Teile (ohne Lieferanten, die nichts liefern)?

```
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
               )
;                                       -- subswhere.0170.sql
```

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 l.lnr = lt.lnr
                      and t.tnr = lt.tnr
                  )
;                                       -- subswhere.0180.sql
```

8.7.7 Aggregatfunktionen

Aggregatfunktionen[6] fassen die Werte aus mehreren Datensätzen (Zeilen) zu *einem* Wert zusammen. Als Beispiel der Durchschnittspreis aller Teile:

```
select avg(preis) "durchschnittspreis"
  from t
;                                       -- agg.0010.sql
```

Ergebnis:

```
| durchschnittspreis |
| ------------------ |
| 15.166666          |
```

[6] engl. *aggregate* = Anhäufung, Gesamtheit, Gesamtgröße

Die gängigsten Aggregatfunktionen sind:

- `SUM`: Summieren
- `AVG`: Durchschnitt (arithmetischer Mittelwert)
- `MIN`: Minimum (kleinster Wert)
- `MAX`: Maximum (größter Wert)
- `COUNT`: Anzahl
- `STRING_AGG`: Zeichenketten verknüpfen

Wichtig:

- Bei *leeren* Tabellen liefern alle Aggregatfunktionen `NULL` zurück. Ausnahme ist `COUNT`, das den Wert `0` zurückgibt.
- Alle Datensätze, die in der Spalte für die Aggregatfunktion `NULL` enthalten, werden *vor* der Berechnung entfernt. Enthalten *alle* Spalten `NULL`, ist das Gesamtergebnis `NULL` (Verhalten wie bei leeren Tabellen).

Sonderfall `COUNT`

- Bei `count(*)` werden immer *alle* Datensätze gezählt, unabhängig von etwaigen `NULL`.
- Bei Verwendung von z.B. `count(preis)` werden nur jene Datensätze gezählt, die in `preis` *nicht* `NULL` enthalten.
- Ist bei `count(preis)` in *allen* Datensätzen bei `preis` ein `NULL` enthalten, ist das Ergebnis `0` (*nicht* `NULL`).

Von wie vielen Lieferanten wird das Teil `T2` geliefert? Wie groß sind Summe, Maximum, Minimum und Durchschnitt der Mengen?

```
select count(*)    "#lieferanten"
     , sum(menge)  "gesamtmenge"
     , max(menge)  "maxmenge"
     , min(menge)  "minmenge"
     , avg(menge)  "mengenschnitt"
  from lt
 where tnr = 'T2'
;                                        -- agg.0020.sql
```

Zusatz `distinct`

Nach der öffnenden Klammer der Aggregatfunktion kann das Schlüsselwort `distinct` angegeben werden. Dadurch wird jeder Wert nur *einmal* berücksichtigt. So wird etwa bei `count(distinct stadt)` jede Stadt nur einmal gezählt.

Aus wieviel verschiedenen Städten sind die Lieferanten?

```
select count(distinct stadt) "#städte"
  from l
;                                        -- agg.0030.sql
```

Welche Lieferanten (alle Spalten) liefern mehr als eine Teilenummer?

```
select *
  from l
 where (select count(*)
          from lt
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
         where lt.lnr = l.lnr) > 1
;                                        -- agg.0040.sql
```

Wieviele Lieferanten liefern mehr als eine Teilenummer?

```
select count(*) "#mehrals1"
  from l
 where (select count(*)
          from lt
         where lt.lnr = l.lnr) > 1
;                                        -- agg.0050.sql
```

Alle Lieferanten (`lnr`, `lname`), die entweder das Teil `T1` oder das Teil `T2`, aber *nicht* beide liefern (Variante):

```
select lnr
     , lname
  from l
 where (select count(*) "#"
          from lt
         where lt.lnr = l.lnr
           and tnr in ('T1', 'T2')
       ) = 1
;                                        -- agg.0060.sql
```

Welche Lieferanten (alle Spalten) haben alle Teile geliefert? (Variante)

```
select *
  from l
 where (select count(*) "#"
          from lt
         where lt.lnr = l.lnr
       ) =
       (select count(*) "#"
          from t
       )
;                                        -- agg.0070.sql
```

Die dritt-kleinste (*n*-kleinste) Teilenummer ist zu ermitteln:

```
select tnr
  from t
 where (select count(*) "#"
          from t t1
         where t1.tnr <= t.tnr
       ) = 3
;                                        -- agg.0080.sql
```

Die drei (allgemein *n*) teuersten Teile (`tnr`, `tname`, `preis`) sind zu ermitteln (ohne Verwendung von `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
;                                        -- agg.0090.sql
```

8.7.8 Gruppierung mit GROUP BY

- Datensätze (Zeilen) mit *gleichen Werten* können mit der `GROUP BY`-Klausel in Töpfe (Gruppen) eingeteilt (gruppiert) werden, z.B. nach der Spalte `farbe`.
- Aggregatfunktionen werden dann nicht mehr auf alle Datensätze angewendet, sondern auf alle Datensätze jeder Gruppe. So wird etwa mit `max(preis)` der Datensatz mit dem höchsten Preis innerhalb der roten Teile, der gelben Teile, usw. bestimmt.
- Es kann auch nach mehreren Eigenschaften gruppiert werden, z.B. erst nach Farbe und dann nach Herkunftsort.
- Mit einer optionalen `HAVING`-Klausel können die Ergebnis-Datensätze noch eingeschränkt werden, z.B. nur jene Maximalpreise, die größer als 30 sind.
- Im Gegensatz zur `WHERE`-Klausel sind in der `HAVING`-Klausel Aggregatfunktionen erlaubt. `WHERE` ermöglicht Filtern *vor dem Gruppieren*, `HAVING` nochmaliges Filtern *nach dem Gruppieren*.

Vorsicht: In der `SELECT`-Klausel dürfen nur Spalten angegeben werden, die entweder auch in der `GROUP BY` Klausel stehen oder auf die eine Aggregatfunktion angewendet wird. Andernfalls ist die Abfrage nicht ausführbar.

Anzahl der Teile und Durchschnittspreis pro Farbe anzeigen:

```
select farbe
     , count(*) "anzahl"
     , avg(preis) "durchschnitt"
  from t
 group by farbe
;                                       -- groupby.0010.sql
```

Achtung: `GROUP BY` soll *nicht* als Ersatz für `DISTINCT` verwendet werden, also nicht

```
select lnr
  from lt
 group by lnr
;                                       -- groupby.0020.sql
```

statt

```
select distinct lnr
  from lt
;                                       -- groupby.0030.sql
```

In welchen Städten sind mindestens zwei Lieferanten? Von diesen Stadt, Anzahl der Lieferanten, Durchschnittsrabatt, Anzahl verschiedener Rabattwerte:

```
select stadt
     , count(*) "anzahl"
     , avg(rabatt) "durchschnitt"
     , count(distinct rabatt) "anz_versch"
  from l
 group by stadt
having count(*) > 1
 order by stadt
;                                       -- groupby.0040.sql
```

Alle Lieferanten (Spalte `lnr`) mit ihren Umsätzen:

```
select lnr
     , sum(menge * preis) "umsatz"
  from lt
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  join t on lt.tnr = t.tnr
 group by lnr
;                                          -- groupby.0050.sql
```

Alle Lieferanten (Spalte `lnr`) mit ihren Umsätzen, auch jene ohne Umsatz:

```
select lnr
     , sum(menge * preis) "umsatz"
  from lt
  join t on lt.tnr = t.tnr
 group by lnr
union
select lnr
     , 0 "umsatz"
  from l
 where lnr not in (select lnr
                     from lt
                   )
;                                          -- groupby.0060.sql
```

Alle Lieferanten (`lnr`, `lname`), die mehr als zwei verschiedene Nummern roter Teile liefern. Berechnung von Umsätzen dieser Teile unter Einbeziehung des Rabatts. Sortierung nach Umsätzen absteigend:

```
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
;                                          -- groupby.0070.sql
```

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

```
select count(*) "anzahl"
  from l
 where lnr in (select lnr
                 from lt
                group by lnr
               having count(*) > 1
              )
;                                           -- groupby.0080.sql
```

Übung:

Warum ist die folgende Abfrage keine zielführende Lösung dafür?

```
select count(*) "anzahl"
  from lt
 group by lnr
having count(*) > 1
;                                -- groupby.0090.sql
```

8.7.9 Varianten der Verbund-Operation

Theta-Join

Outer-Join

8.7.10 Unterabfragen in der Projektion

8.7.11 Unterabfragen in der `FROM`-Klausel (Inline View)

8.7.12 `NULL` und dreiwertige Logik

8.7.13 Abarbeitung von Abfragen

8.8 Datenänderungen (DML)

Der Tabelleninhalt (Datensätze) wird durch **DML-Kommandos (Data Manipulation Language)** geändert:

- `INSERT`: Einfügen von Datensätzen
- `UPDATE`: Ändern bestehender Datensätze
- `DELETE`: Löschen von Datensätzen

8.8.1 INSERT

8.8.2 UPDATE

8.8.3 DELETE

8.9 Transaktionssteuerung (TCL)

8.10 Zeilenweise Verarbeitung (Cursor)

8.11 Übungen zu SQL

8.12 Datenmodell LTP

8.13 Datenmodell Schulungsfirma

1. Erstellen Sie für folgendes ER-Diagramm einer *Schulungsfirma* die notwendigen DDL-Anweisungen zur Definition der entsprechenden Tabellen.
2. Die Schulungsfirma hat das Datenmodell in einem RDBMS implementiert. Das folgende Diagramm (Microsoft-Notation) zeigt die Tabellen, deren Attribute (Spalten), die Schlüsselattribute sowie die Beziehungen zwischen den Tabellen (Foreign-Key-Constraints).
3. Daten (Tabelleninhalte)
4. Welche Kurse (`knr`) haben einen anderen Kurs als Voraussetzung?
    a. Nur mit `knr`:

```
| knr |
| --- |
|   2 |
|   3 |
|   5 |
|   7 |
```

b. Zusätzlich mit Kursname (`bezeichn`):

```
| knr | bezeichn      |
| --- | ------------- |
|   2 | Harmonielehre |
|   3 | Rhythmik      |
|   5 | Dirigieren    |
|   7 | Komposition   |
```

c. Schreiben Sie die Abfrage auch mit der älteren Syntax (ohne Schlüsselwort `JOIN`).

5. Welche Kurse (`bezeichn`) dauern zwischen 2 und 4 Tagen und haben einen durchschnittlichen Tagespreis von höchstens 700,00? Sortierung aufsteigend nach Bezeichnung.

```
| bezeichn        |
| --------------- |
| Dirigieren      |
| Harmonielehre   |
| Musikgeschichte |
| Notenkunde      |
```

6. Welche Personen (`fname`, `vname`, `ort`) haben ein Leerzeichen im Vornamen und denselben Vokal zweimal im Ort?

```
| fname  | vname            | ort     |
| ------ | ---------------- | ------- |
| Bach   | Johann Sebastian | Leipzig |
| Händel | Georg Friedrich  | London  |
```

7. Welche Personen (`pnr`) haben noch nicht alle Kursbesuche bezahlt? Sortierung nach `pnr` aufsteigend.

```
| pnr |
| --- |
| 101 |
| 103 |
| 109 |
| 110 |
| 111 |
| 113 |
| 114 |
| 116 |
```

8. Über wieviele Tage (`tage`) erstrecken sich Kursveranstaltungen, die in Wien stattfinden und von Referent `103` oder `104` gehalten werden (`knr`, `knrlfnd`, `tage`)? Sortierung nach `tage` absteigend.

Hinweis:

Eine Kursveranstaltung kann über einen längeren Zeitraum gehen, als der Kurs Tage hat, z.B. wegen Feiertagen dazwischen.

```
| knr | knrlfnd | tage |
| --- | ------- | ---- |
|   7 |       1 |    5 |
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
|   2 |       1 |     3 |
|   4 |       2 |     2 |
|   1 |       1 |     2 |
```

9. Welche Referenten (`pnr`, `alter`) sind älter als 75 Jahre?

   Hinweis:

   Abhängig vom aktuellen Ausführungsdatum kann das Alter der Personen von nachfolgendem Beispielergebnis abweichen.

```
| pnr | alter |
| --- | ----- |
| 101 |    84 |
| 103 |    87 |
| 111 |    95 |
| 114 |    79 |
```

10. Welche Personen (`pnr`) halten oder besuchen mindestens eine Kursveranstaltung?

```
| pnr |
| --- |
| 101 |
| 102 |
  ...   (pnr 103 - 114 übersprungen)
| 115 |
| 116 |
| 117 |
```

11. Alle Kursveranstaltungen (`knr`, `bezeichn`, `knrlfnd`, `von`), bei denen noch kein Referent festgelegt ist.

```
| knr | bezeichn    | knrlfnd |        von |
| --- | ----------- | ------- | ---------- |
|   1 | Notenkunde  |       3 | 2005-04-10 |
|   5 | Dirigieren  |       3 | 2005-03-30 |
```

12. Alle Kursveranstaltungen (`knr`, `bezeichn`, `knrlfnd`, `von`), die mindestens ein Teilnehmer besucht.

```
| knr | bezeichn           | knrlfnd |        von |
| --- | ------------------ | ------- | ---------- |
|   1 | Notenkunde         |       1 | 2003-04-07 |
|   1 | Notenkunde         |       2 | 2004-06-23 |
|   1 | Notenkunde         |       3 | 2005-04-10 |
|   2 | Harmonielehre      |       1 | 2003-10-09 |
|   3 | Rhythmik           |       1 | 2003-11-17 |
|   4 | Instrumentenkunde  |       1 | 2004-01-12 |
|   4 | Instrumentenkunde  |       2 | 2004-03-28 |
|   5 | Dirigieren         |       1 | 2004-05-18 |
|   5 | Dirigieren         |       2 | 2004-09-23 |
|   7 | Komposition        |       1 | 2005-03-09 |
```

13. Alle Kursveranstaltungen (`knr`, `bezeichn`, `knrlfnd`, `von`), die mindestens ein Teilnehmer besucht und bei denen schon ein Referent festgelegt ist.

```
| knr | bezeichn           | knrlfnd |        von |
| --- | ------------------ | ------- | ---------- |
|   1 | Notenkunde         |       1 | 2003-04-07 |
|   1 | Notenkunde         |       2 | 2004-06-23 |
|   2 | Harmonielehre      |       1 | 2003-10-09 |
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
|   3 | Rhythmik          |       1 | 2003-11-17 |
|   4 | Instrumentenkunde |       1 | 2004-01-12 |
|   4 | Instrumentenkunde |       2 | 2004-03-28 |
|   5 | Dirigieren        |       1 | 2004-05-18 |
|   5 | Dirigieren        |       2 | 2004-09-23 |
|   7 | Komposition       |       1 | 2005-03-09 |
```

14. Referenten zahlen für Besuche von Kursveranstaltungen nichts. Zeigen Sie Besuche an, wo dies nicht eingehalten ist.

    - Machen Sie Person `109` zum Referenten, damit die Abfrage Ergebnisse liefert.

    ```
    | knr | knrlfnd | pnr |    bezahlt |
    | --- | ------- | --- | ---------- |
    |   2 |       1 | 109 | 2003-11-03 |
    |   4 |       2 | 109 | 2004-04-15 |
    |   5 |       1 | 109 | 2004-06-07 |
    |   7 |       1 | 109 | 2005-03-20 |
    ```

    - Zeigen Sie zusätzlich die Kurszeichnung und den Familiennamen an.

    ```
    | knr | bezeichn          | knrlfnd | pnr | fname  |    bezahlt |
    | --- | ----------------- | ------- | --- | ------ | ---------- |
    |   2 | Harmonielehre     |       1 | 109 | Wagner | 2003-11-03 |
    |   4 | Instrumentenkunde |       2 | 109 | Wagner | 2004-04-15 |
    |   5 | Dirigieren        |       1 | 109 | Wagner | 2004-06-07 |
    |   7 | Komposition       |       1 | 109 | Wagner | 2005-03-20 |
    ```

    - Entfernen Sie Person `109` wieder aus den Referenten.

15. Liste aller Kursteilnehmer (`bezeichn`, `von`, `fname`, `vname`). Sortierung nach `von`, `bezeichn`, `fname` und `vname`.

    ```
    | bezeichn          |        von | fname          | vname           |
    | ----------------- | ---------- | -------------- | --------------- |
    | Notenkunde        | 2003-04-07 | Liszt          | Franz           |
    | Notenkunde        | 2003-04-07 | Tschaikowskij  | Peter           |
    | Notenkunde        | 2003-04-07 | Wagner         | Richard         |
    | Harmonielehre     | 2003-10-09 | Beethoven      | Ludwig van      |
    | Harmonielehre     | 2003-10-09 | Brahms         | Johannes        |
    | Harmonielehre     | 2003-10-09 | Strauss        | Richard         |
    | Harmonielehre     | 2003-10-09 | Wagner         | Richard         |
    ...
    | Dirigieren        | 2004-09-23 | Puccini        | Giacomo         |
    | Dirigieren        | 2004-09-23 | Strauss        | Richard         |
    | Komposition       | 2005-03-09 | Bizet          | Georges         |
    | Komposition       | 2005-03-09 | Schönberg      | Arnold          |
    | Komposition       | 2005-03-09 | Wagner         | Richard         |
    | Notenkunde        | 2005-04-10 | Verdi          | Giuseppe        |
    ```

    - Fügen Sie als Spalten am Ende noch den Familien- und Vornamen des Referenten hinzu. Die Spalten für Teilnehmer sollen nun `tn_fname` und `tn_vname` bzw. `ref_fname` und `ref_vname` heißen. Sortierung wie zuvor.
    - Welche Kursveranstaltungen scheinen nun nicht mehr auf? Wie können Sie das ändern?

16. Welche Kursveranstaltungen (`bezeichn`, `von`) werden von Referenten besucht?

    ```
    | bezeichn          |        von |
    | ----------------- | ---------- |
    | Dirigieren        | 2004-05-18 |
    | Dirigieren        | 2004-09-23 |
    ```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
| Harmonielehre     | 2003-10-09 |
| Instrumentenkunde | 2004-01-12 |
| Notenkunde        | 2003-04-07 |
| Notenkunde        | 2004-06-23 |
| Rhythmik          | 2003-11-17 |
```

17. Welche Kursveranstaltungen (`bezeichn`, `von`) werden von Referenten besucht, die gleichzeitig diesen Kurs auch halten?

    - Ändern Sie den Referenten der Kursveranstaltung mit `knr = 3` und `knrlfnd = 1` auf `pnr = 101`, um Ergebnisse zu erhalten.

    ```
    | bezeichn |        von |
    | -------- | ---------- |
    | Rhythmik | 2003-11-17 |
    ```

    - Machen Sie die Änderung wieder rückgängig.

18. Welche Personen (`fname`) haben den Kurs `knr = 1` *und* den Kurs `knr = 5` besucht?

    ```
    | fname   |
    | ------- |
    | Wagner  |
    | Strauss |
    ```

    Schreiben Sie als Zusatzaufgabe folgende Varianten:

    - Mit Self-`JOIN`.
    - Mit `INTERSECT`.
    - Mit Abfrage auf `person` sowie Unterabfrage in der `WHERE`-Klausel mit `INTERSECT`.
    - Mit Unterabfrage in der `FROM`-Klausel (mit `INTERSECT`) und `JOIN` auf `person`.
    - Vorige Variante mit `WITH`-Klausel.

19. Alle Kursveranstaltungen mit Tagespreisen zwischen `610` und `690`, die von Referenten ohne Titel gehalten werden (`bezeichn`, `von`, `bis`, `tagespreis`). *Tagespreis* ist der Kurspreis geteilt durch die Kurstage.

    ```
    | bezeichn      |        von |        bis | tagespreis |
    | ------------- | ---------- | ---------- | ---------- |
    | Harmonielehre | 2003-10-09 | 2003-10-11 |     666,67 |
    | Dirigieren    | 2004-05-18 | 2004-05-20 |     633,33 |
    | Dirigieren    | 2004-09-23 | 2004-09-26 |     633,33 |
    ```

20. Wieviele Kursbesuche gibt es?

    ```
    | #kursbesuche |
    | ------------ |
    |           27 |
    ```

    Welche Kursbesuche wurden vor dem Kursbeginn bezahlt?

    - Um Ergebnisse zu erhalten: ändern Sie in `besucht` den Wert von `bezahlt` auf `2004-01-07` für den Datensatz `knr = 4 knrlfnd = 1 pnr = 111`.

    ```
    | knr | knrlfnd | pnr | bezahlt    | von        | bis        |
    | --- | ------- | --- | ---------- | ---------- | ---------- |
    | 4   | 1       | 111 | 2004-01-07 | 2004-01-12 | 2004-01-13 |
    ```

    - Machen Sie die Änderung wieder rückgängig.

    Welche Kursbesuche wurden während des Kurses bezahlt?

- Um Ergebnisse zu erhalten: ändern Sie in `besucht` den Wert von `bezahlt` auf `2003-04-08` für den Datensatz `knr = 1 knrlfnd = 1 pnr = 109`.

```
| knr | knrlfnd | pnr | bezahlt    | von        | bis        |
| --- | ------- | --- | ---------- | ---------- | ---------- |
| 1   | 1       | 109 | 2003-04-08 | 2003-04-07 | 2003-04-08 |
```

- Machen Sie die Änderung wieder rückgängig.

Welche Kursbesuche wurden nach dem Kursende bezahlt?

```
| knr | knrlfnd | pnr |    bezahlt |        von |        bis |
| --- | ------- | --- | ---------- | ---------- | ---------- |
|   1 |       1 | 108 | 2003-05-01 | 2003-04-07 | 2003-04-08 |
|   1 |       2 | 110 | 2004-07-01 | 2004-06-23 | 2004-06-24 |
|   1 |       2 | 112 | 2004-07-03 | 2004-06-23 | 2004-06-24 |
... (10 Datensätze weggelassen)
|   5 |       2 | 115 | 2004-10-07 | 2004-09-23 | 2004-09-26 |
|   7 |       1 | 109 | 2005-03-20 | 2005-03-09 | 2005-03-13 |
|   7 |       1 | 117 | 2005-04-08 | 2005-03-09 | 2005-03-13 |
```

Welche Kursbesuche wurde noch nicht bezahlt?

```
| knr | knrlfnd | pnr | bezahlt |        von |        bis |
| --- | ------- | --- | ------- | ---------- | ---------- |
|   1 |       1 | 109 |         | 2003-04-07 | 2003-04-08 |
|   1 |       1 | 114 |         | 2003-04-07 | 2003-04-08 |
|   1 |       2 | 116 |         | 2004-06-23 | 2004-06-24 |
|   1 |       3 | 110 |         | 2005-04-10 | 2005-04-11 |
|   2 |       1 | 116 |         | 2003-10-09 | 2003-10-11 |
|   3 |       1 | 101 |         | 2003-11-17 | 2003-11-17 |
|   3 |       1 | 109 |         | 2003-11-17 | 2003-11-17 |
|   4 |       1 | 111 |         | 2004-01-12 | 2004-01-13 |
|   5 |       1 | 103 |         | 2004-05-18 | 2004-05-20 |
|   5 |       2 | 116 |         | 2004-09-23 | 2004-09-26 |
|   7 |       1 | 113 |         | 2005-03-09 | 2005-03-13 |
```

Schreiben Sie eine Abfrage, die alle Kursbesuche auflistet mit einer Spalte `status`, die folgenden Code enthält:

- <: Kurs vor Kursbeginn bezahlt
- =: Kurs während des Kurses bezahlt
- >: Kurs nach Kursende bezahlt
- !: Kurs nicht bezahlt

Ändern Sie die oben beschriebenen Datensätze *vorübergehend*, um Datensätze für alle möglichen Codes zu erhalten.

```
| knr | knrlfnd | pnr |    bezahlt | status |        von |        bis |
| --- | ------- | --- | ---------- | ------ | ---------- | ---------- |
|   1 |       1 | 108 | 2003-05-01 | >      | 2003-04-07 | 2003-04-08 |
|   1 |       1 | 109 | 2003-04-08 | =      | 2003-04-07 | 2003-04-08 |
|   1 |       1 | 114 |            | !      | 2003-04-07 | 2003-04-08 |
|   1 |       2 | 110 | 2004-07-01 | >      | 2004-06-23 | 2004-06-24 |
... (13 Datensätze weggelassen)
|   4 |       1 | 111 | 2004-01-07 | <      | 2004-01-12 | 2004-01-13 |
... (5 Datensätze weggelassen)
|   5 |       2 | 116 |            | !      | 2004-09-23 | 2004-09-26 |
|   7 |       1 | 109 | 2005-03-20 | >      | 2005-03-09 | 2005-03-13 |
```

21. Welche Personen (`fname`, danach sortiert) besuchen Kursveranstaltungen, die in ihrem Wohnort abgehalten werden und länger als zwei Kurstage dauern?

```
| fname     |
| --------- |
| Beethoven |
| Brahms    |
| Schönberg |
```

22. Welche Kursveranstaltungen (`bezeichn`, `knrlfnd`) werden von Referenten gehalten, die für den Kurs auch geeignet sind?

```
| bezeichn          | knrlfnd |
| ----------------- | ------- |
| Notenkunde        |       1 |
| Notenkunde        |       2 |
| Harmonielehre     |       1 |
| Rhythmik          |       1 |
| Instrumentenkunde |       2 |
| Dirigieren        |       1 |
| Dirigieren        |       2 |
| Komposition       |       1 |
| Komposition       |       2 |
```

23. Alle Referenten (`pnr`, `fname`, `vname`), die Kursveranstaltungen gehalten haben, *nachdem* sie in die Firma eingetreten sind (in Attribut `seit`). Sortierung nach `pnr`.

```
| pnr | fname         | vname           |
| --- | ------------- | --------------- |
| 101 | Bach          | Johann Sebastian |
| 103 | Haydn         | Joseph          |
| 104 | Mozart        | Wolfgang Amadeus |
| 114 | Tschaikowskij | Peter           |
| 116 | Strauss       | Richard         |
```

Alle Referenten (`pnr`, `fname`, `vname`), die Kursveranstaltungen gehalten haben, *bevor* sie in die Firma eingetreten sind (in Attribut `seit`). Sortierung nach `pnr`.

Um Ergebnisse zu erhalten: ändern Sie *vorübergehend* in `referent` den Wert von `seit` auf `2006-01-01` für den Datensatz `pnr = 116`.

```
| pnr | fname   | vname   |
| --- | ------- | ------- |
| 116 | Strauss | Richard |
```

24. Alle Personen (`pnr`, `fname`), die einen Kurs in `Wien` besucht oder gehalten haben. Sortierung nach `pnr`.

```
| pnr | fname         |
| --- | ------------- |
| 101 | Bach          |
... (102 - 109 weggelassen)
| 111 | Bruckner      |
| 112 | Brahms        |
| 113 | Bizet         |
| 114 | Tschaikowskij |
| 115 | Puccini       |
| 116 | Strauss       |
| 117 | Schönberg     |
```

Zusatzübung: verschiedene Lösungsmöglichkeiten mit

- Mengenoperationen
- Unterabfragen
- Nur mittels `LEFT JOIN`-Operationen.

25. Zeitraum, auf den die Kurstage einer mehrtägigen Kursveranstaltung verteilt sind, im Vergleich mit der im Kurs angegebenen Kursdauer, d.h. geht die Veranstaltung über ein Wochenende / Sa, So?

    - Führen Sie vor der Abfrage `set language German;` aus, um deutsche Namen für die Wochentage zu erhalten.
    - Verwenden Sie die Funktionen `datediff` und `datename`.

    Abfrageergebnis:

```
| knr | bezeichn    | tage | knrlfnd | von        | von_wotag  | bis        | bis_wotag | von_bis_tage |
| --- | ----------- | ---- | ------- | ---------- | ---------- | ---------- | --------- | ------------ |
| 5   | Dirigieren  | 3    | 2       | 2004-09-23 | Donnerstag | 2004-09-26 | Sonntag   | 4            |
| 7   | Komposition | 4    | 1       | 2005-03-09 | Mittwoch   | 2005-03-13 | Sonntag   | 5            |
| 7   | Komposition | 4    | 2       | 2005-09-14 | Mittwoch   | 2005-09-18 | Sonntag   | 5            |
```

26. Welche Referenten (`pnr`, `fname`, `vname`) haben Kursveranstaltungen in einem Alter von über 60 Jahren gehalten? Sortierung nach *pnr*.

```
| pnr | fname          | vname            |
| --- | -------------- | ---------------- |
| 101 | Bach           | Johann Sebastian |
| 103 | Haydn          | Joseph           |
| 114 | Tschaikowskij | Peter            |
```

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. Alle Attribute der vorausgesetzten Veranstaltung erhalten das Präfix `v`.

    Abfrageergebnis:

```
| knr | knrlfnd | von        | bis        | ort  | plaetze | pnr | vknr | vknrlfnd | vvon       | vbis       | vort | vplaetze | vpnr |
| --- | ------- | ---------- | ---------- | ---- | ------- | --- | ---- | -------- | ---------- | ---------- | ---- | -------- | ---- |
| 2   | 1       | 2003-10-09 | 2003-10-11 | Wien | 4       | 104 | 1    | 1        | 2003-04-07 | 2003-04-08 | Wien | 3        | 103  |
| 5   | 2       | 2004-09-23 | 2004-09-26 | Wien | 2       | 101 | 2    | 2        | 2003-10-09 | 2003-10-11 | Wien | 4        | 104  |
| 5   | 2       | 2004-09-23 | 2004-09-26 | Wien | 2       | 101 | 4    | 4        | 2004-01-12 | 2004-01-13 | Wien | 3        | 116  |
| 5   | 2       | 2004-09-23 | 2004-09-26 | Wien | 2       | 101 | 4    | 4        | 2004-03-28 | 2004-03-29 | Wien | 4        | 104  |
| 7   | 1       | 2005-03-09 | 2005-03-13 | Wien | 5       | 103 | 5    | 5        | 2004-09-23 | 2004-09-26 | Wien | 2        | 101  |
```

28. Fehlt.

```
-- sql.028.out
```

29. Gibt es Personen (Familien- und Vorname), bei denen Kursbesuche einander terminlich überschneiden?

```
| knr | knrlfnd | pnr | bezahlt    | knr_2 | knrlfnd_2 | pnr_2 | bezahlt_2  |
| --- | ------- | --- | ---------- | ----- | --------- | ----- | ---------- |
| 1   | 1       | 109 |            | 2     | 1         | 109   | 2003-11-03 |
| 1   | 1       | 109 |            | 3     | 1         | 109   |            |
| 1   | 1       | 109 |            | 4     | 2         | 109   | 2004-04-15 |
| 1   | 1       | 109 |            | 5     | 1         | 109   | 2004-06-07 |
| 1   | 1       | 109 |            | 7     | 1         | 109   | 2005-03-20 |
| 1   | 2       | 110 | 2004-07-01 | 1     | 3         | 110   |            |
| 1   | 2       | 112 | 2004-07-03 | 2     | 1         | 112   | 2003-10-28 |
| 1   | 2       | 113 | 2004-07-20 | 7     | 1         | 113   |            |
| 1   | 2       | 116 |            | 2     | 1         | 116   |            |
| 1   | 2       | 116 |            | 5     | 2         | 116   |            |
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
| 1    | 3        | 110 |            | 1      | 2          | 110    | 2004-
↪07-01 |
| 2    | 1        | 109 | 2003-11-03 | 1      | 1          | 109    |       ␣
↪      |
| 2    | 1        | 109 | 2003-11-03 | 3      | 1          | 109    |       ␣
↪      |
| 2    | 1        | 109 | 2003-11-03 | 4      | 2          | 109    | 2004-
↪04-15 |
| 2    | 1        | 109 | 2003-11-03 | 5      | 1          | 109    | 2004-
↪06-07 |
| 2    | 1        | 109 | 2003-11-03 | 7      | 1          | 109    | 2005-
↪03-20 |
| 2    | 1        | 112 | 2003-10-28 | 1      | 2          | 112    | 2004-
↪07-03 |
| 2    | 1        | 116 |            | 1      | 2          | 116    |       ␣
↪      |
| 2    | 1        | 116 |            | 5      | 2          | 116    |       ␣
↪      |
| 3    | 1        | 109 |            | 1      | 1          | 109    |       ␣
↪      |
| 3    | 1        | 109 |            | 2      | 1          | 109    | 2003-
↪11-03 |
| 3    | 1        | 109 |            | 4      | 2          | 109    | 2004-
↪04-15 |
| 3    | 1        | 109 |            | 5      | 1          | 109    | 2004-
↪06-07 |
| 3    | 1        | 109 |            | 7      | 1          | 109    | 2005-
↪03-20 |
| 3    | 1        | 117 | 2003-11-20 | 7      | 1          | 117    | 2005-
↪04-08 |
| 4    | 2        | 109 | 2004-04-15 | 1      | 1          | 109    |       ␣
↪      |
| 4    | 2        | 109 | 2004-04-15 | 2      | 1          | 109    | 2003-
↪11-03 |
| 4    | 2        | 109 | 2004-04-15 | 3      | 1          | 109    |       ␣
↪      |
| 4    | 2        | 109 | 2004-04-15 | 5      | 1          | 109    | 2004-
↪06-07 |
| 4    | 2        | 109 | 2004-04-15 | 7      | 1          | 109    | 2005-
↪03-20 |
| 5    | 1        | 109 | 2004-06-07 | 1      | 1          | 109    |       ␣
↪      |
| 5    | 1        | 109 | 2004-06-07 | 2      | 1          | 109    | 2003-
↪11-03 |
| 5    | 1        | 109 | 2004-06-07 | 3      | 1          | 109    |       ␣
↪      |
| 5    | 1        | 109 | 2004-06-07 | 4      | 2          | 109    | 2004-
↪04-15 |
| 5    | 1        | 109 | 2004-06-07 | 7      | 1          | 109    | 2005-
↪03-20 |
| 5    | 2        | 116 |            | 1      | 2          | 116    |       ␣
↪      |
| 5    | 2        | 116 |            | 2      | 1          | 116    |       ␣
↪      |
| 7    | 1        | 109 | 2005-03-20 | 1      | 1          | 109    |       ␣
↪      |
| 7    | 1        | 109 | 2005-03-20 | 2      | 1          | 109    | 2003-
↪11-03 |
| 7    | 1        | 109 | 2005-03-20 | 3      | 1          | 109    |       ␣
↪      |
| 7    | 1        | 109 | 2005-03-20 | 4      | 2          | 109    | 2004-
↪04-15 |
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
| 7   | 1       | 109 | 2005-03-20 | 5     | 1         | 109   | 2004-
↪06-07 |
| 7   | 1       | 113 |            | 1     | 2         | 113   | 2004-
↪07-20 |
| 7   | 1       | 117 | 2005-04-08 | 3     | 1         | 117   | 2003-
↪11-20 |
```

30. Fehlt.

```
-- sql.030.out
```

31. Fehlt.

```
-- sql.031.out
```

32. Fehlt.

```
-- sql.032.out
```

33. Fehlt.

```
-- sql.033.out
```

34. Fehlt.

```
-- sql.034.out
```

35. Fehlt.

```
-- sql.035.out
```

36. Fehlt.

```
-- sql.036.out
```

37. Fehlt.

```
-- sql.037.out
```

38. Fehlt.

```
-- sql.038.out
```

39. Fehlt.

```
-- sql.039.out
```

40. Welche Kurse (`bezeichnung`) fanden in `Wien` und in `Paris` statt?

```
| bezeichn   |
| ---------- |
| Notenkunde |
| Dirigieren |
```

Welche Kurse (`bezeichnung`) fanden in `Wien`, aber nicht in `Paris` statt?

```
| bezeichn          |
| ----------------- |
| Harmonielehre     |
| Instrumentenkunde |
| Komposition       |
```

41. Welche Personen (`pnr`) haben mindestens zwei Kursveranstaltungen besucht?

pnr
109
110
112
113
116
117

42. Alle Kursveranstaltungen (`knr`, `knrlfnd`), die keiner besucht.

knr	knrlfnd
5	3
7	2

43. Den Preis einer Kursveranstaltung (`bezeichn`, `von`, `bis`, `preis`, `we_preis`) mit Wochenendaufschlag ermitteln:

    - Sind nur Wochentage enthalten, entspricht der Preis dem Kurspreis.
    - Für jeden Wochenend-Tag erhöht sich der Preis um 10%.
    - Verwenden Sie die Funktionen `datediff` und `datepart`.

bezeichn	von	bis	preis	we_preis
Notenkunde	2003-04-07	2003-04-08	1.400	1.400
Harmonielehre	2003-10-09	2003-10-11	2.000	2.200
Rhythmik	2003-11-17	2003-11-17	700	700
Instrumentenkunde	2004-01-12	2004-01-13	1.500	1.500
Instrumentenkunde	2004-03-28	2004-03-29	1.500	1.650
Dirigieren	2004-05-18	2004-05-20	1.900	1.900
Notenkunde	2004-06-23	2004-06-24	1.400	1.400
Dirigieren	2004-09-23	2004-09-26	1.900	2.280
Komposition	2005-03-09	2005-03-13	3.000	3.600
Dirigieren	2005-03-30	2005-04-01	1.900	1.900
Notenkunde	2005-04-10	2005-04-11	1.400	1.540
Komposition	2005-09-14	2005-09-18	3.000	3.600

44. Alle Kursveranstaltungen (`knr`, `knrlfnd`) mit vorhandenen, belegten und freien Plätzen.

knr	knrlfnd	plaetze	belegt	frei
1	1	3	3	0
1	2	4	4	0
1	3	3	1	2
2	1	4	4	0
3	1	3	3	0
4	1	3	3	0
4	2	4	2	2
5	1	3	2	1
5	2	2	2	0
5	3	3	0	3
7	1	5	3	2
7	2	4	0	4

    Nur Kursveranstaltungen, bei denen noch Plätze frei sind.

knr	knrlfnd	plaetze	belegt	frei

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
|   1 |       3 |       3 |       1 |     2 |
|   4 |       2 |       4 |       2 |     2 |
|   5 |       1 |       3 |       2 |     1 |
|   5 |       3 |       3 |       0 |     3 |
|   7 |       1 |       5 |       3 |     2 |
|   7 |       2 |       4 |       0 |     4 |
```

45. Für wieviele *Kurse* (`anzahl`) gibt es noch keinen geeigneten Referenten?

```
| anzahl |
| ------ |
|      0 |
```

Für wieviele *Kursveranstaltungen* (`anzahl`) gibt es noch keinen geeigenten Referenten?

```
| anzahl |
| ------ |
|      3 |
```

46. Welche Referenten (`pnr`) halten keine Kursveranstaltung?

```
| pnr |
| --- |
| 111 |
```

Warum ist folgende Abfrage *keine* korrekte Lösung? Achten Sie auf `pnr` in `kveranst`.

```
select pnr
  from referent
 where pnr not in (select pnr
                     from kveranst
                   )
;
```

```
| pnr |
| --- |
```

47. Welche Personen besuchen keinen Kurs?

```
| pnr |
| --- |
| 104 |
```

48. Alle Kurse (`bezeichn`, danach geordnet) mit jeweils jenen Kursen (`bezeichn` als `voraussetzung`), die umittelbare Voraussetzung sind. Hat ein Kurs keine Voraussetzung, soll − angegeben werden.

```
| bezeichn          | voraussetzung     |
| ----------------- | ----------------- |
| Dirigieren        | Harmonielehre     |
| Dirigieren        | Rhythmik          |
| Dirigieren        | Instrumentenkunde |
| Harmonielehre     | Notenkunde        |
| Instrumentenkunde | -                 |
| Komposition       | Dirigieren        |
| Komposition       | Musikgeschichte   |
| Musikgeschichte   | -                 |
| Notenkunde        | -                 |
| Rhythmik          | Notenkunde        |
```

49. Zu welchen Kursen (`knr`, `bezeichn`, `tage`, `preis`), die länger als einen Tag dauern, gibt es keine Kursveranstaltungen, die in `Wien` stattfinden?

```
| knr | bezeichn         | tage | preis |
| --- | ---------------- | ---- | ----- |
|   6 | Musikgeschichte  |    2 | 1.400 |
```

50. Welche Personen (`pnr`, `fname`) haben alle Kurse besucht?

```
| pnr | fname |
| --- | ----- |
```

- Da es keine solche Person gibt: ermitteln Sie, wieviele unterschiedliche Kurse (`pnr`, `anzahl`, danach absteigend sortiert) einzelne Personen besucht haben (ohne Personen, die überhaupt keinen Kurs besucht haben).

```
| pnr | anzahl |
| --- | ------ |
| 109 |      6 |
| 116 |      3 |
| 117 |      2 |
| 112 |      2 |
| 113 |      2 |
| 114 |      1 |
| 115 |      1 |
... (9 Datensätze weggelassen)
```

- Fügen Sie für jene Person, die die meisten Kurse besucht hat, einen entsprechenden Datensatz in den Tabellen `kveranst` und `besucht` ein, sodass die ursprüngliche Abfrage nun ein Ergebnis hat.

```
| pnr | fname  |
| --- | ------ |
| 109 | Wagner |
```

- Entfernen Sie die eingefügten Datensätze wieder (mit `DELETE` oder mit `ROLLBACK` durch Einbettung in eine Transaktion).

51. Welcher Referent feiert während eines Kurses *als Vortragender* Geburtstag?

- Um für diese Abfrage Ergebnisse zu erhalten: ändern Sie innerhalb einer Transaktion das Geburtsdatum des Referenten `103` auf `1932-04-07`.
- Betten Sie die Änderung in eine Transaktion ein.
- Abfrageergebnis:

```
| pnr |
| --- |
| 103 |
```

- Machen Sie die Änderung mit `ROLLBACK` rückgängig.

52. Für welche Kurse (`knr`, `bezeichn`) sind ausschließlich österreichische Referenten geeignet?

```
| knr | bezeichn          |
| --- | ----------------- |
|   2 | Harmonielehre     |
|   3 | Rhythmik          |
|   4 | Instrumentenkunde |
|   6 | Musikgeschichte   |
```

53. Alle Personen (`pnr`, `fname`), die einen Kurs in `Wien` besucht und gehalten haben.

```
| pnr | fname   |
| --- | ------- |
| 116 | Strauss |
```

54. In welchen Orten wurde alle Kurse abgehalten?
    - Um für diese Abfrage Ergebnisse zu erhalten: fügen Sie für `Berlin` Kursveranstaltungen für alle Kurse ein.
    - Betten Sie Änderungen und Abfrage in eine explizite Transaktion ein.
    - Abfrageergebnis:

```
| ort    |
| ------ |
| Berlin |
```

    - Machen Sie die Änderungen mit `ROLLBACK` rückgängig.
55. Fehlt.

```
-- sf.055.out
```

56. Fehlt.

```
-- sf.056.out
```

57. Fehlt.

```
-- sf.057.out
```

58. Fehlt.

```
-- sf.058.out
```

59. Fehlt.

```
|----------+------------------+---------|
|  fname   | vname            | anzahl  |
|----------+------------------+---------|
|  Bach    | Johann Sebastian | 2       |
|  Haydn   | Joseph           | 3       |
|  Mozart  | Wolfgang Amadeus | 2       |
|  Strauss | Richard          | 2       |
|----------+------------------+---------|
```

60. Fehlt.

```
-- sf.060.out
```

61. Fehlt.

```
-- sf.061.out
```

62. Fehlt.

```
-- sf.062.out
```

63. Fehlt.

```
-- sf.063.out
```

64. Fehlt.

```
-- sf.064.out
```

65. Fehlt.

```
-- sf.065.out
```

66. Fehlt.

```
-- sf.066.out
```

67. Fehlt.

```
-- sf.067.out
```

68. Fehlt.

```
-- sf.068.out
```

69. Fehlt.

```
-- sf.069.out
```

70. Fehlt.

```
-- sf.070.out
```

71. Fehlt.

```
-- sf.071.out
```

72. Fehlt.

```
-- sf.072.out
```

73. Fehlt.

```
-- sf.073.out
```

74. Fehlt.

```
-- sf.074.out
```

75. Fehlt.

```
-- sf.075.out
```

76. Fehlt.

```
-- sf.076.out
```

KAPITEL 9

Transaktionen

KAPITEL 10

Recovery

KAPITEL 11

Mehrbenutzerbetrieb (Concurrency)

KAPITEL 12

Verzeichnisse

- genindex
- search