Vorlesung: Datenbanken

Kapitel (A): Daten / Datenstrukturen

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Daten / Datenstruktur

Daten

(Daten: latein. *dare* geben, *datum* das Gegebene)

Daten sind logisch gruppierte Informationseinheiten, geeignet für eine Speicherung auf Systemen oder zur Informationsübertragung zwischen Systemen.

Daten sind Gebilde aus Zeichen oder kontinuierliche Funktionen, die aufgrund bekannter oder unterstellter Abmachungen Informationen darstellen, vorrangig zum Zweck der Verarbeitung und als deren Ergebnis.

Datenstruktur

Eine Datenstruktur ist ein mathematisches Objekt zur Speicherung von Daten. Es handelt sich um eine Struktur, wenn die Daten in einer bestimmten Art und Weise angeordnet und verknüpft werden. Man strukturiert Daten um den Zugriff auf sie und ihre Verwaltung geeignet zu ermöglichen.

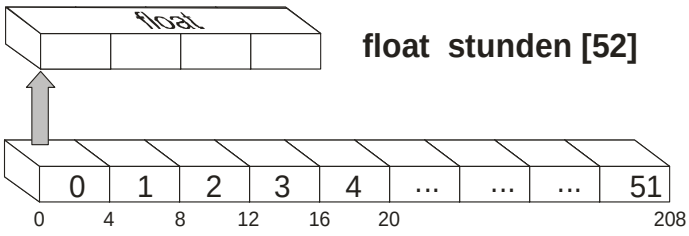
Datenstrukturen sind daher nicht nur durch die enthaltenen Daten charakterisiert, sondern vor allem durch die Operationen auf diesen Daten, die Zugriff und Verwaltung realisieren.

Bei jeder Form von Datenhaltung entsteht die Problematik deren Speicherung.

Allgemein bekannte Spezifikationen zur Datenhaltung sind:

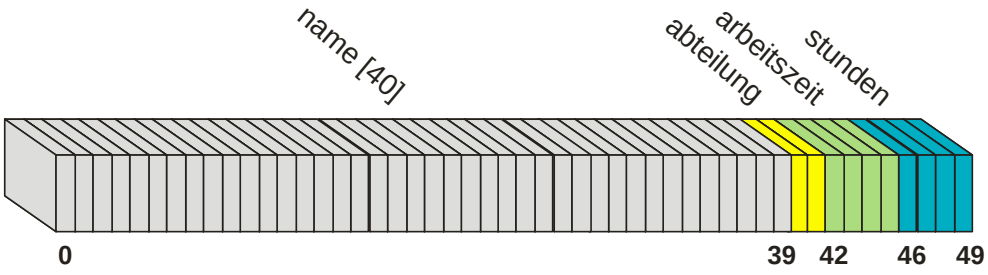
Array Das Array ist die einfachste Art einer Datenstruktur. Das Array beschreibt eine vorher festgelegte Anzahl von homogenen Daten desselben Datentypen.

Typ [Anzahl] ***Name***



Struct In vielen Programmiersprachen lassen sich auch ‚ungleiche' Datentypen als Einheit zusammenfassen. Eine Struct (Structure) wird im Vorfeld definiert und enthält somit ebenfalls eine festgelegte Anzahl von Elementen.

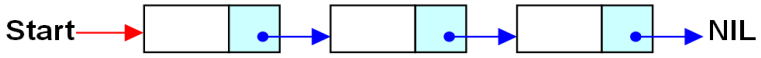
```
typedef struct {  char name [40];
                  int abteilung;
                  float arbeitszeit;
                  float stunden;     } mitarbeiter;
```



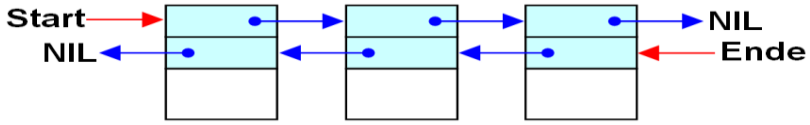
Verkettete Liste

Eine einfach- oder zweifach-verkettete Liste erweitert die Datenhaltung um die dynamische Speicherung von Objekten. Verkettete Listen erlauben eine Speicherung von einer im Vorhinein nicht bestimmten Anzahl von miteinander in Beziehung stehenden Werten einfacher oder zusammengesetzter Datentypen. Bei einer einfach verketteten Liste beinhaltet jedes Listenelement einen Verweis auf das nächste Element. Mit Hilfe dieser Verweise, meist in Form von Speicheradresse, kann man von Element zu Element navigieren. Bei einer einfach-verketteten Liste nur in ‚eine' Richtung. Eine zweifach-verkettete Liste enthält zwei Verweise. Ein Verweis auf das nächste Element und ein Verweis auf das vorherige Element und erlaubt dann die Navigation in ‚beide' Richtungen.

Einfach verkettete Liste



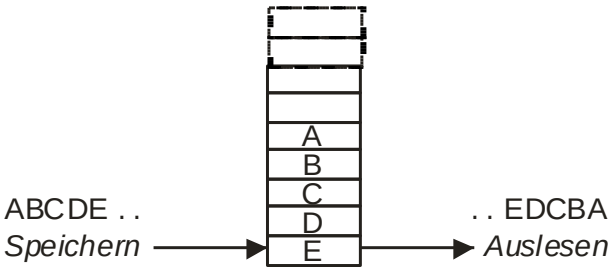
Zweifach verkettete Liste



Stack

In einem Stapelspeicher (Stack) kann eine beliebige Anzahl von Objekten gespeichert werden. Die Besonderheit hierbei ist das die gespeicherten Objekte nur in umgekehrter Reihenfolge zur Speicherung wieder gelesen werden.

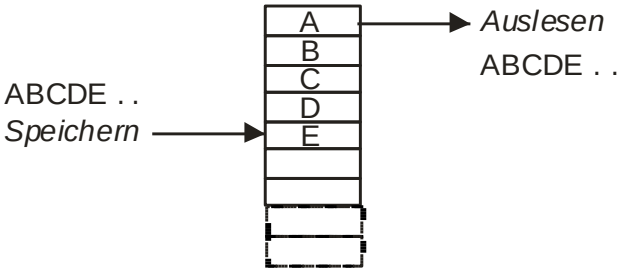
Stack-Speicher
(LIFO - Last In, First Out)



Queue

Eine Warteschlange (Queue) hat das gleiche Wirkprinzip wie die Stack-Speicherung nur mit dem Unterschied, dass Daten in der gleichen Reihenfolge gelesen werden in der sie gespeichert wurden.

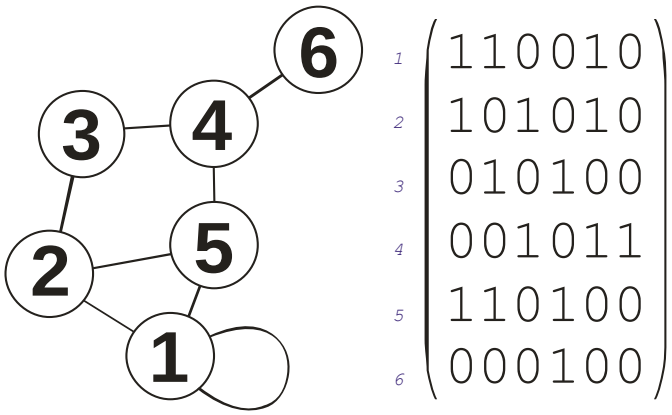
Queue-Speicher
(FIFO - First In, First Out)



Graph Die Datenstruktur Graph ermöglicht es die Unidirektionalität der verketteten Listen zu überwinden. In einem Graphen werden die Verknüpfungen zwischen *n* Datenelementen durch eine *n x n -Matrix* repräsentiert. Bekannte Repräsentation von Graphen im Computer sind beispielsweise die Adjazenzmatrix (Nachbarschaftsmatrix).

Adjazenzmatrix

Ungerichteter Graph **Adjazenzmatrix**



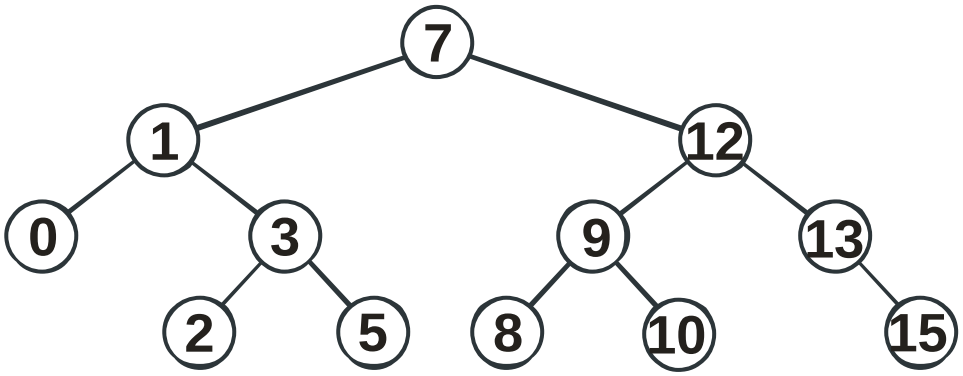
Beispiel für Graphen:

Baum

Die Bäume gehören zu den meist verwendeten Datenstrukturen in der Informatik. Die bekannteste Art dürften die binären Suchbäume darstellen, aber es existieren viele Spezialisierungen. Bei geordneten Bäumen, insbesondere Suchbäumen, werden die Elemente in der an eine Baumstruktur angelehnten, vorher definierten und geordneten Reihenfolge abgelegt. Dies ermöglicht ein schnelles Auffinden der Elemente im Baum und somit einen schnellen Datenzugriff. Eine typische Anwendung sind Indizes bei Datenbanken.

Modelldarstellung eines Binärbaums

Beispiel: Ordne [7, 12, 1, 3, 9, 0, 8, 2, 13, 10, 15, 5]



Der **erste Datenwert** bildet die *Wurzel*. (7)

Der Wert des **nächsten** Listenelements wird mit dem in der Wurzel eingetragenen verglichen.

Ist er kleiner, -> wird ein ***linker Sohn/Tochter*** erzeugt und der Wert dort abgelegt.

Ist er größer, -> erzeugen man einen ***rechten Sohn/Tochter*** in dem der Wert abgelegt wird.

Ist ein Knoten, schon **belegt**, so vergleicht man ihn mit dem einzutragenden Wert auf die bereits beschriebene Weise und steigt den Baum solange ***herunter***, bis der eigentliche Eintrag erfolgen kann. Knoten ohne Kinder nennt man analog einer natürlichen Baumstruktur ‚Blätter'.

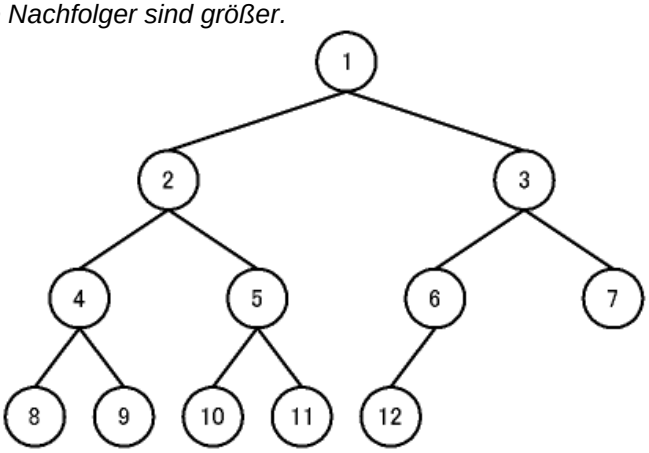
Ergebnis in diesem Beispiel:

Höhe des Baumes = Anzahl der Kanten im längsten Pfad = maximal 4 Vergleiche

Der Binärbaum hat 4 Ebenen. Dies ist gleichbedeutend mit maximal 4 Zugriffen zum Auffinden jeder beliebigen Ziffer.

Heap Der Heap (Halde, Haufen) vereint die Datenstruktur eines Baums mit den Operationen einer Warteschlange (Queue). Der Unterschied zum Baum ist an der festgelegten Priorität und an der ‚totalen' Ordnung erkennbar.

Beispiel Min-Heap: *Alle Nachfolger sind größer.*



Treap Der Treap vereinigt wiederum die Eigenschaften von Bäumen und Heaps in sich. Alle Elemente erfüllen die Eigenschaften des binären Suchbaums. Jeder Knoten besteht aus zwei Elementen –> Schlüssel + Priorität.

Im Einzelnen bedeutet dies:

Die Elemente y im linken Teilbaum von Baum x erfüllen: key(y) < key(x)
Die Elemente y im rechten Teilbaum von Baum x erfüllen: key(y) > key(x)

Hashtable Die Streuwerttabelle (Hashtable) ist eine spezielle Indexstruktur, bei der die Speicherposition direkt berechnet werden kann. Die Hashtable steht dabei in Konkurrenz zu Baumstrukturen, die im Gegensatz alle Indexwerte in einer Ordnung wiedergeben kann, dafür aber einen größeren Verwaltungsaufwand benötigt. Der Einsatz einer Hashtable zur Suche in Datenmengen erfolgt üblicherweise als Algorithmus zum Suchen von Datenobjekten in großen Datenmengen. Es basiert auf der Idee, dass eine mathematische Funktion die Position eines Objektes in einer Tabelle berechnet. Dadurch erübrigt sich das sequenzielle Durchsuchen vieler Datenobjekte bis das Zielobjekt gefunden wird.

Eine wissenschaftliche Abhandlung der Haschfunktion verlässt das Thema Datenstrukturen. Der Themenbereich Index / Indizes wird an anderer Stelle erneut aufgegriffen, daher hier nur eine naive Erklärung der Anwendungsgebiete. Als Beispiel dient ein Wortindex.

Beim umgekehrten Wortindex wird jedes Wort und seine Fundstelle als Schlüssel gespeichert. Beispielsweise wird der Satz:

„Blaukraut bleibt Blaukraut und Brautkleid bleibt Brautkleid" indiziert.

Als Datenstruktur zur Aufnahme der Fundstellen werden Hashtable oder Binärbäume genutzt.

Bei einer Hashtable ist der Aufwand des Findens *O(1)*, bei Binärbäumen *O(log n)*, wenn *n* die Anzahl der Wörter ist.

Wort	Fundstelle
blaukraut	1, 17
bleibt	10, 42
brautkleid	31, 49
und	27

Vorlesung: Datenbanken

Kapitel (A): Datenbanken Datenschutzgesetz

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Datenschutz

Problemstellung: Was ist Datenschutz heute?
Wer oder Was wird geschützt?
Um welche Daten geht es?

Nach dem Gesetz schützenswert sind ausschließlich Daten mit Personenbezug oder mittelbar auf Personen beziehbare Daten.
Der Schutz dieser Daten leitet sich aus den Persönlichkeitsrechten ab, die im Grundgesetz verankert sind und in weiteren Gesetzen genauer ausgeführt sind. Die Persönlichkeitsrechte gehören zu den höchsten vom Grundgesetz geschützten Werten. Firmen- und Produktionsdaten sind ausdrücklich durch das Gesetz nicht geschützt!

Die Rechte leiten sich vor Allem aus den zwei folgenden Grundgesetzartikeln ab:

Art. 1 Abs.1 GG „Die Würde des Menschen ist unantastbar. …“
Art. 2 Abs.1 GG „Jeder hat das Recht auf freie Entfaltung seiner Persönlichkeit, …“

Darüber hinaus gibt es noch das prominente Volkszählungsurteils des BVerfG (1983) mit dem daraus folgenden Fazit: „Jeder ist Herr seiner Daten“

Jeder kann also über die Verwendung seiner Daten selbst bestimmen!

Weiter aus dem Persönlichkeitsrecht ergeben sich noch zwei Rechte:

Das Recht am gesprochenen Wort

Das Bundesverfassungsgericht entschied am 3. März 2004, dass Abhörmaßnahmen in Privatwohnungen nur durchgeführt werden dürfen, wenn Sie durch einen Richter angeordnet werden und der Verdacht auf eine besonders schwere Straftat besteht. Eine besondere Schwere liegt nach Art. 3 Abs. 3 GG nur vor, wenn der Gesetzgeber die Straftat mit einer maximalen Freiheitsstrafe von mindestens 5 Jahren belegt. Abhören bei weniger schweren Straftaten und ohne Verdacht sahen die Richter dagegen als Verstoß gegen die Menschenwürde an.
In letzter Zeit machten dagegen vermehrt Abhör-Apps für Smartphones Schlagzeilen. Mit denen lassen sich Handys als improvisierte Abhörgeräte nutzen. Das Nutzen solcher Programme von Privatpersonen ist jedoch ein illegaler Lauschangriff nach § 201 II Nr. 1 StGB und kann geahndet werden. Es ist also von solchen Programmen abzuraten, es sei denn man nutzt Sie auf dem eigenen Handy um seine eigenen Gespräche aufzuzeichnen. Dieses wäre nach der aktueller Rechtslage zulässig.

Das Recht am eigenen Bild.

Dabei ist nicht nur das Veröffentlichen, als mögliche Verletzung des Urheberrechts, sondern bereits das Anfertigen von Bildern möglicherweise Verboten, wie das BVerfG beim Fall der Caroline von Monaco klärte (15. Dezember 1999)

Das Gericht führte hierbei aus: „Die von dem allgemeinen Persönlichkeitsrecht aus Art. 2 Abs. 1 in Verbindung mit Art. 1 Abs. 1 GG geschützte Privatsphäre ist nicht auf den häuslichen Bereich beschränkt. Der Einzelne muss grundsätzlich die Möglichkeit haben, sich auch an anderen, erkennbar abgeschiedenen Orten von Bildberichterstattung unbehelligt zu bewegen."

Als Ausnahmen sind geregelt, dass Bilder von einer Persönlichkeit der Zeitgeschichte im öffentlichen Raum gemacht werden dürfen, ein Bild in Zusammenhang mit einer Geschichte oder einem Ereignis von öffentlichem Interesse veröffentlicht werden kann und sofern die Person in anderem Rahmen schon einmal sonst als privat eingestufte Bilder zur eigenen Vermarktung benutzt hat, dürfen ähnliche Bilder auch gemacht und in der Presse veröffentlicht werden.

Verbot mit Erlaubnisvorbehalt

Wie zu Anfang ausgeführt kann jede Person selbst frei über die Verwendung ihrer Daten bestimmen. Wäre jedoch der Staat völlig auf die freiwillige Mithilfe Aller angewiesen, beispielsweise bei der Steuererklärung oder einer Alkoholkontrolle, wäre er in der Ausübung seiner Schutzpflichten gegenüber dem Bürger eingeschränkt.
Der Gesetzgeber hat daher die Möglichkeit im Interesse der Allgemeinheit Gesetze zu erlassen, die dem Recht des Einzelnen auf Selbstbestimmung vorgehen. Geregelt ist dies im § 4 Abs.1 BDSG, der das Erheben von Daten zulässt sofern ein Gesetz es vorschreibt oder der Betroffene einwilligt. Die Datenerhebung durch den Staat muss im Regelfall bei den Betroffenen selbst erfolgen und kann nur in Ausnahmen über andere Stellen laufen, sofern dadurch keine überwiegenden schutzwürdigen Interessen des Betroffenen verletzt werden.
Das Gesetz ist also ein Verbot mit Erlaubnisvorbehalt, welche durch den Betroffenen oder ein entsprechendes Gesetz erfolgen kann. Bezogen auf den Datenschutz bedeutet dies: Es ist Alles, was nicht ausdrücklich per Gesetz erlaubt ist, automatisch verboten.

Erlaubt wird die Datenerhebung aber in § 28 BDSG und § 32 BDSG.
In § 28 BDSG wird ausgeführt, dass Erheben, Speichern, Verändern und Übermitteln in Zusammenhang mit einem bestehenden rechtlichen Schuldverhältnis zulässig sind. Darüber hinaus ist es möglich zur Wahrung berechtigter Interessen Daten zu erheben, sofern keine höherwertigen schutzwürdigen Interessen des Betroffenen verletzt werden.

In § 32 BDSG wird das Erheben von Daten eines Beschäftigten durch seinen Arbeitgeber erlaubt, sofern es im Zusammenhang mit dem Beschäftigungsverhältnis erforderlich ist.

Interessenabwägung

Die Abwägung der Interessen muss nach juristischen Grundsätzen erfolgen. Dabei muss geklärt werden, ob die Datenerhebung, Datenverarbeitung oder Datennutzung für die verantwortliche Stelle stets erforderlich ist und ob dieser Nutzung schutzwürdige Interessen entgegenstehen.

Ein berechtigtes Interesse liegt vor, sofern es durch eine objektive Sachlage gerechtfertigt ist, es kann wirtschaftlicher oder ideeller Natur sein.

Schutzwürdige Interessen beziehen auf die Privat, Intimsphäre oder Vertraulichkeitssphäre oder auf andere Gesichtspunkte wie z.B. zu befürchtende wirtschaftliche oder berufliche Nachteile.
Beispielsweise darf ein Onlinehändler bei dem eine Bestellung eingeht per Mail Rückmeldung geben, sofern der gewünschte Artikel nicht lieferbar ist. In dieser Nachricht darf er aber nicht auf weitere ähnliche Produkte in seinem Sortiment hinweisen, die alternativ gekauft werden könnten, da der Kunde keine entsprechende Anforderung an ihn gestellt hat.

Darüber hinaus ist es auch nicht gestattet Weihnachts- oder Geburtstags-Glückwunschkarten an Kunden oder Lieferanten zu schicken, sofern diese nicht ausdrücklich der Sendung jener Karten zugestimmt haben.

Transparenzgebot

Das Bundesverfassungsgericht führt in seinem Volkszählungsurteil aus:
„Wer nicht mit hinreichender Sicherheit überschauen kann, welche ihn betreffenden Informationen in bestimmten Bereichen seiner sozialen Umwelt bekannt sind, und wer das Wissen möglicher Kommunikationspartner nicht einigermaßen abzuschätzen vermag, kann in seiner Freiheit wesentlich gehemmt werden, aus eigener Selbstbestimmung zu planen oder zu entscheiden. Mit dem Recht auf informationelle Selbstbestimmung wären eine Gesellschaftsordnung und eine diese ermöglichende Rechtsordnung nicht vereinbar, in der Bürger nicht mehr wissen können, wer was wann und bei welcher Gelegenheit über sie weiß. Wer unsicher ist, ob abweichende Verhaltensweisen jederzeit notiert und als Information dauerhaft gespeichert, verwendet oder weitergegeben werden, wird versuchen, nicht durch solche Verhaltensweisen aufzufallen.“

Hat man diesen Satz verstanden, hat man den Datenschutz verstanden.

Verantwortung & Haftung

Verantwortlich für den Datenschutz ist die Unternehmensleitung, welche auch die Haftung für den Datenschutz übernehmen muss. Bei Datenschutzverstößen drohen Bußgelder bis zu 300.000 € (§ 43 Abs.3 BDSG)

Die Verantwortung leitet sich grundsätzlich aus den Regeln der Compliance ab, die z.B. aus §91 Abs.2 AktG oder §43 Abs.2 GmbHG folgen. Es existiert aber auch die Möglichkeit die Verantwortung für den Datenschutz von der Geschäftsleitung auf einen firmeninternen Datenschutzbeauftragten oder einen externen Dienstleister zu übertragen. (Urteil LG Ulm vom 31.10.1990 / Beschluss Düsseldorfer Kreis vom 24./25.11.2010)

Pflichten datenverarbeitender Stellen

Den Rechten der Betroffenen stehen entsprechende Pflichten der datenverarbeitenden Stelle gegenüber:

- Bestellung eines betrieblichen Datenschutzbeauftragten
- Verpflichtung der Mitarbeiter auf das Datengeheimnis
- Erstellung oder Meldung einer Verarbeitungsübersicht
- Vertragliche Absicherung einer Datenverarbeitung im Auftrag
- Löschpflichten alter Daten der Betroffenen
- Treffen technischer und organisatorischer Schutzmaßnahmen

Technischer Datenschutz

Es gelten die 8 Gebote des Datenschutzes (§ 9 BDSG Anlage):

- Zutrittskontrolle
- Zugangskontrolle (Authentifizierung)
- Zugriffskontrolle (Überwachung des Zugriffs auf Ressourcen)
- Weitergabekontrolle (u.a. Verschlüsselungsverordnung)
- Eingabekontrolle (Protokollierung)
- Auftragskontrolle (Datenverarbeitung im Auftrag, Privat-PC)
- Verfügbarkeitskontrolle (Datensicherheit)
- Zweckseparierung

Anforderungen an Passwörter

Das verwenden von Trivialkennwörtern auf dienstlichen PCs ist fahrlässig und kann im Schadensfall dazu führen, dass ein Anspruch geltend gemacht werden kann oder auch eine Versicherung nicht eingelöst werden kann.

Rechtlich nicht zu beanstanden sind Passwörter bestehend aus:

- 8 Stellen
- 1 Zahl
- 1 Großbuchstaben
- 1 Sonderzeichen

Darüber hinaus müssen die Passwörter auch alle 3 Monate ausgewechselt werden. Neben einem sicheren Aufbau des Kennwortes muss auch auf eine sichere Aufbewahrung derselben geachtet werden. Idealerweise hat man seine Passwörter alle im Kopf und nirgendwo physisch aufgeschrieben. Tut man dies doch, dann sollte man Sie an einem sicheren Ort lagern und auf keinen Fall eine Passwortliste in unmittelbarer Nähe zum Rechner aufbewahren oder womöglich noch die Passwörter unter die Tastatur oder an den Bildschirmrand kleben.

Auch der Systemadministrator darf die Passwörter nicht kennen. Er kann aber ein Initialpasswort vergeben, das vom Benutzer unverzüglich in ein nur ihm Bekanntes zu wechseln ist. Dies ist wichtig, da das eigene Passwort wie eine Unterschrift behandelt wird. Durch das Benutzen des Accounts versichert man seine Identität und unbefugtes Einloggen in ein Profil wird ähnlich geahndet wie Urkundenfälschung.

§ 42a Informationspflicht bei unrechtmäßiger Kenntniserlangung von Daten

Stellt eine nichtöffentliche Stelle im Sinne des § 2 Absatz 4 oder eine öffentliche Stelle nach § 27 Absatz 1 Satz 1 Nummer 2 fest, dass bei ihr gespeicherte
1.besondere Arten personenbezogener Daten (§ 3 Absatz 9),
2.personenbezogene Daten, die einem Berufsgeheimnis unterliegen,
3.personenbezogene Daten, die sich auf strafbare Handlungen oder Ordnungswidrigkeiten oder den Verdacht strafbarer Handlungen oder Ordnungswidrigkeiten beziehen, oder
4.personenbezogene Daten zu Bank- oder Kreditkartenkonten
unrechtmäßig übermittelt oder auf sonstige Weise Dritten unrechtmäßig zur Kenntnis gelangt sind, und drohen schwerwiegende Beeinträchtigungen für die Rechte oder schutzwürdigen Interessen der Betroffenen, hat sie dies nach den Sätzen 2 bis 5 unverzüglich der zuständigen Aufsichtsbehörde sowie den Betroffenen mitzuteilen. Die Benachrichtigung des Betroffenen muss unverzüglich erfolgen, sobald angemessene Maßnahmen zur Sicherung der Daten ergriffen worden oder nicht unverzüglich erfolgt sind und die Strafverfolgung nicht mehr gefährdet wird. Die Benachrichtigung der Betroffenen muss eine Darlegung der Art der unrechtmäßigen Kenntniserlangung und Empfehlungen für Maßnahmen zur Minderung möglicher nachteiliger Folgen enthalten. Die Benachrichtigung der zuständigen Aufsichtsbehörde muss zusätzlich eine Darlegung möglicher nachteiliger Folgen der unrechtmäßigen Kenntniserlangung und der von der Stelle daraufhin ergriffenen Maßnahmen enthalten. Soweit die Benachrichtigung der Betroffenen einen unverhältnismäßigen Aufwand erfordern würde, insbesondere aufgrund der Vielzahl der betroffenen Fälle, tritt an ihre Stelle die Information der Öffentlichkeit durch Anzeigen, die mindestens eine halbe Seite umfassen, in mindestens zwei bundesweit erscheinenden Tageszeitungen oder durch eine andere, in ihrer Wirksamkeit hinsichtlich der Information der Betroffenen gleich geeignete Maßnahme. Eine Benachrichtigung, die der Benachrichtigungspflichtige erteilt hat, darf in einem Strafverfahren oder in einem Verfahren nach dem Gesetz über Ordnungswidrigkeiten gegen ihn oder einen in § 52 Absatz 1 der Strafprozessordnung bezeichneten Angehörigen des Benachrichtigungspflichtigen nur mit Zustimmung des Benachrichtigungspflichtigen verwendet werden.

Anforderungen an Beschäftigte und Stellenleitungen

- Umsetzung der geltenden Datenschutzgesetze
- Sensibilisierung von Verantwortlichen und Mitarbeitern

Vorlesung: Datenbanken

Kapitel (B): Grundlagen relationaler Datenbanken

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Grundlagen

Daten

(Daten: latein. *dare* geben, *datum* das Gegebene)

Daten sind logisch gruppierte Informationseinheiten, geeignet für eine Speicherung auf Systemen oder zur Informationsübertragung zwischen Systemen[1].

Analogie: Daten <> Informationen <> Wissen

Eine kleine Analogie soll helfen die Begriffe:

Daten, Informationen und Wissen

voneinander abzugrenzen.
Als Beispiel dient eine Verkehrsampel als ein bekanntes Informationselement aus dem Alltag.

Die Farben Rot und Grün stellen für sich alleine genommen keine Information sondern nur ein *Datum* dar (Singular von Daten). Ein grünes T-Shirt erfährt in der Regel keine besondere Beachtung, eher sein Inhalt.

Eine Ampel zeigt Rot für ‚stehen' Grün für ‚gehen'. Die Farbe Rot oder Grün in Form eines Ampellichtes wird zu einer *Information*. Diese Information führt normalerweise zu einer Reaktion.

Erst nach einer Reaktion und der folgerichtigen Interpretation der Information kann man dies als *Wissen* bezeichnen.

Leuchtet an einer Ampel Rot und Grün gleichzeitig, so stellt dieser Informationszustand ein Fehlen der Datenintegrität[2] dar. Mit andern Worten, die durch den Benutzer festgelegten Regeln für die Interpretation der in der Datenbank enthaltenen Informationen werden verletzt.

Datenbank (DB)

Unter einer Datenbank versteht man eine systematische Sammlung von Informationen zu einem bestimmten Thema oder Zweck. Datenbanken stellen Systeme zur Beschreibung, Speicherung und Wiedergewinnung von umfangreichen Datenmengen zur Verfügung.

Eine Datenbank ist eine Zusammenstellung von Daten samt ihrer Beschreibung (Metadaten), die persistent im Datenbanksystem (DBS) abgelegt werden.

Datenbankmanagementsystem (DBMS)

Ein Datenbankmanagementsystem ist die eingesetzte Software, mit der die Daten in eine Datenbank eingegeben, verwaltet und ausgewertet werden.

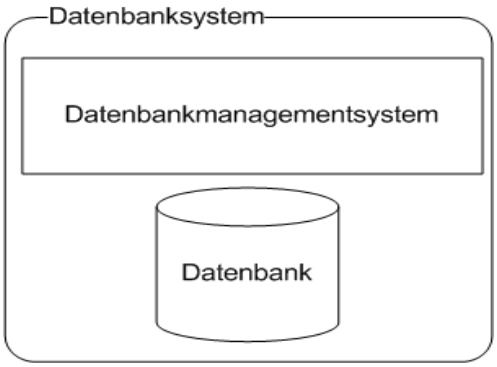
Das DBMS bildet die Schnittstelle zwischen den Datenbanken und dient den Benutzern zur Datenverwaltung und Veränderung. Die zentralen Aufgaben eines DBMS sind im Wesentlichen die Bereitstellung verschiedener Sichten auf die Daten (Views), die Konsistenzprüfung der Daten (Integritätssicherung), die Autorisationsprüfung, die Behandlung gleichzeitiger Zugriffe verschiedener Benutzer (Synchronisation) und das Bereitstellen einer Datensicherungs-möglichkeit, um im Falle eines Systemausfalls zeitnah Daten wiederherstellen zu können.

[1] Siehe auch Kapitel: Daten, Datenstrukturen
[2] Siehe auch Kapitel: Begriffe

Datenbanksystem (DBS)

Datenbanksysteme (DBS) sind ein weithin genutztes Hilfsmittel zur rechnergestützten Organisation, Erzeugung, Veränderung und Verwaltung großer Datensammlungen und stellen in vielen Unternehmen und Organisationen die zentrale Informationsbasis zu ihrer Aufgabenerfüllung bereit. Ein DBS besteht aus dem so genannten Datenbankmanagement-System (DBMS) und einer oder mehrerer Datenbanken[3].

Ein Datenbanksystem ist die Kombination aus Datenbank und Datenbankmanagementsystem.



Bei der Organisation kleiner oder großer Datensammlungen sind grundlegende Faktoren zu beachten. Beispielsweise versucht man eine mehrfache Datenhaltung, die sogenannte **Redundanz**, zu vermeiden, da sie Speicherbedarf und Zugriffszeiten unnötig erhöhen jedoch keinen Informationszuwachs bedeutet.

Das mehrfache Vorhandensein der gleichen Information birgt eine weitere Gefahr in sich. So ist zu jedem Zeitpunkt sicherzustellen, dass bei einer Änderung einer Information ALLE weiteren Kopien dieser Information ebenfalls geändert werden. Sollten gleiche Informationen unterschiedliche Ergebnisse anzeigen, so spricht man von einer Dateninkonsistenz. Die **Datenkonsistenz** ist eines der zentralen Probleme jeder Datenhaltung.

Natürlich muss man seine Daten vor unberechtigtem Zugriff schützen. Alle Maßnahmen, die dies gewährleisten sollen, fasst man unter dem Oberbegriff **Datenschutz** zusammen.

Neben dem Datendiebstahl sind auch Manipulationen eine ernst zu nehmende Bedrohung. Hier spricht man vom Problem der **Datensicherheit**.

Eine weitere Form der Datensicherheit ist eine regelmäßige **Sicherung** der Daten. Diese dient zum einen der Reparatur und der Wiederherstellung defekter Systeme und zum andern auch der Rekonstruktion von Daten, beispielsweise nach deren versehentlichem Löschen durch den Anwender. Leider wird oft missachtet, dass für die Sicherung die gleichen Sicherheitskriterien wie für die Originaldaten gelten müssen

Gleich ob manueller Zugriff auf einen Karteikasten oder der rechnergesteuerte Zugriff im Netzwerk, zusammengehörige oder zeitgleiche Datenzugriffe müssen erkannt, gesteuert und verwaltet werden. Dies soll das Datenbankmanagementsystem gewährleisten. Zu nennen wäre hier die **Transaktion**, die zusammengehörige Anweisungen nur komplett zulässt oder auf den Zustand vor der ersten Anweisung zurückkehrt. Wenn mehrere Systeme zeitgleich um die gleichen Daten konkurrieren, so ist es die Aufgabe der Transaktionssteuerung diese zu verwalten und zu serialisieren.

[3] Quelle: IT-Grundstruktur; Bundesamt für Sicherheit in der Informationstechnik

Man zieht zur Erstellung einer Datenbank ein Konzept heran, dass die Datenunabhängigkeit von vornherein garantieren soll. Es handelt sich um das sogenannte **Drei-Ebenen-Konzept**.

Diese drei Ebenen der Datenbank sind die konzeptionelle Ebene, die externe Ebene und die interne Ebene. In der konzeptionellen Ebene werden sämtliche Daten beschrieben, die in der Datenbank gespeichert werden sollen. Die externen Schemata beschreiben die Ausschnitte, die für die einzelnen Benutzergruppen relevant sind. Die interne Ebene beschäftigt sich mit der physischen Anordnung der Daten auf den peripheren Speicher. Ziel ist es, bei Änderungen auf einer Ebene, die nächsthöhere Ebene nicht ebenfalls ändern zu müssen.

Datenintegrität

Alle vorangegangenen Maßnahmen dienen der Integrität der gespeicherten Daten.

Datenintegrität der Datenbank bedeutet die Gewährleistung einer korrekten und widerspruchsfreien Speicherung von Daten, deren Korrektheit und Vollständigkeit und die Vermeidung aller möglichen Störungen beziehungsweise eine Reaktion zur Herstellung eines integeren Zustandes auf diese Störungen

Das Bundesamt für Sicherheit in der Informationstechnik hat spezielle Verhaltensregeln[4] im Falle des nach Verlustes der Datenbankintegrität im Internet veröffentlicht.

Folgende Schritte sind für die Damen und Herren des Bundesamtes unumgänglich:

- Ruhe bewahren!
- Benachrichtigen Sie den Datenbankadministrator
- Greifen Sie nicht mehr auf die Datenbank zu
- Befolgen Sie die Anweisungen des Datenbankadministrators

Nur Pech, wenn Sie gerade der Datenbankadministrator sind. In diesem Fall sollten Sie sich vorsorglich Zuhause beim Abendessen entschuldigen, es könnte etwas später werden.

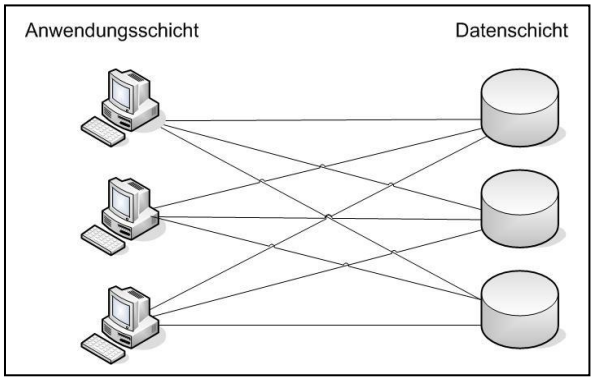
[4] Bundesamt für Sicherheit in der Informationstechnik, Maßnahmenkatalog, M 6.48: Verhaltensregeln nach Verlust der Datenbankintegrität

Schichtenarchitektur

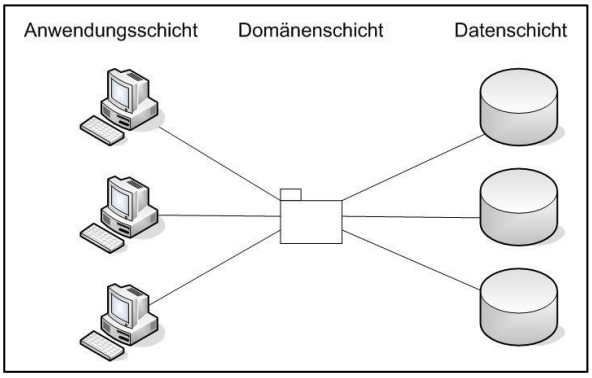
Eine Schichtenarchitektur oder Schichtenmodell ist ein häufig angewandtes Strukturierungsprinzip für die Architektur von Softwaresystemen. Werden Anwendungs- und Datenschicht unterschieden, so bezeichnet man dies auch als ***zweischichtige Architektur***[5]. Ein typischer Vertreter dieses Software-Modells ist eine klassische Client-Server-Anwendung.

Bei zweischichtigen Architekturen wird die Rechenkapazität weitestgehend auf die Client-Rechner (Front-End, FAT-Client) ausgelagert, um den Server zu entlasten. Auf dem Server (Back-End) läuft eine Datenbankanwendung nur als zentrale Datenhaltung ohne selbständige Rechenoperationen. Die Clients übernehmen dabei die Logik, die Darstellung und die Benutzerschnittstelle. Microsoft-Access arbeitet nach diesem Prinzip.

Zwei-Schichten-Architektur



Drei-Schichten-Architektur



Moderne Datenbanksysteme sind überwiegend Bestandteil einer ***dreischichtigen Architektur***.

Als Erweiterung der zweischichtigen Architektur wird hier zwischen Client und Server als dritte Ebene ein Applikationsserver, in unserem Fall ein Datenbankmanagementsystem**,** zur Bereitstellung der Datenbank-Anwendungen eingeführt.

Üblicherweise läuft das Datenbankmanagementsystem auf dem Server und kanalisiert die Benutzerabfragen. Durch diese Verarbeitung und dann das zur Verfügung stellen lediglich der Abfrageergebnisse reduzieren sich die Netzbelastungen erheblich.

Mehrschichtige Systeme sind auch besser skalierbar, da die einzelnen Schichten logisch voneinander getrennt sind. So kann z.B. bei verteilten Systemarchitekturen die Datenschicht auf einem zentralen Datenbank-Server laufen, die Logikschicht auf Workgroup-Servern, und die Präsentationsschicht befindet sich auf der jeweiligen Workstation des Benutzers.

Durch diese Architektur kann eine Kosteneinsparung aufgrund einer vereinfachten Datenbankadministration erreicht werden.

Client-Server-Architekturen müssen nicht notwendigerweise mittels unterschiedlicher Rechner realisiert sein, vielmehr kann der Client auch als ein Software-Modul verstanden werden, das auf ein zweites Software-Modul auf demselben Rechner, meist innerhalb derselben Anwendung zugreift. Gemäß den Abstraktionsregeln[6] aus dem objektorientierten Softwareentwurf darf nur die höhere auf die niedrigere Schicht zugreifen. Bei Client-Server-Modell ist die niedrigere Schicht der Dienstanbieter (engl. Server) der höheren.

[5] Schichtenarchitektur manchmal auch 2-Tier-Architektur genannt. (two tier architecture, englisch: Tier = Schicht)

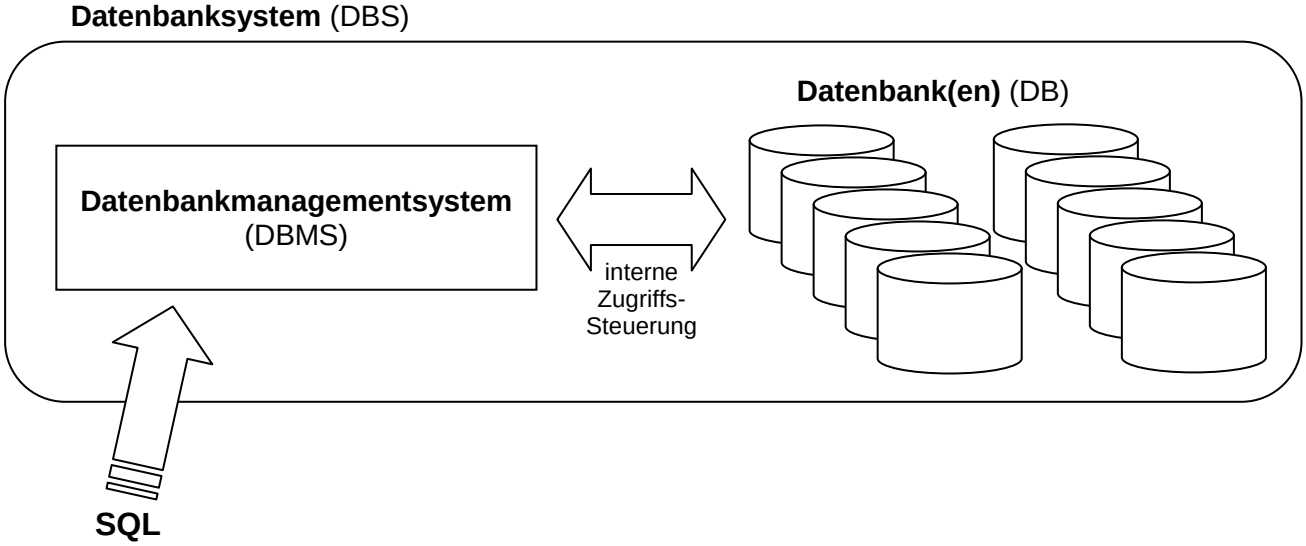
[6] Dependency Inversion Principle (DIP; dt. "Abhängigkeits-Invertierungs-Prinzip") ist ein Prinzip beim Objektorientierten Entwurf von Software. Es beschäftigt sich mit der Abhängigkeit von Modulen.

Datenbankstrukturen

Ein Datenbanksystem (DBS) ist ein System zur elektronischen Datenverwaltung. Die wesentliche Aufgabe eines DBS ist es, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern. Dazu gehören bedarfsgerechte Darstellungsformen und das Bereitstellen von Informations-(Teil-)mengen für Benutzer und Anwendungsprogramme.

Ein **Datenbanksystem** (DBS) besteht aus zwei Teilen:

- der Verwaltungssoftware, genannt **Datenbankmanagementsystem** (DBMS)
- und der Menge der zu verwaltenden Daten, der/die eigentliche(n) **Datenbank**(en) (DB).



Begriffe relationaler Datenbanken

Relationen

- Das relationale Datenmodell wurde von Edgar F. Codd, entwickelt und 1970 in seiner Arbeit „A relational model of data for large shared data bases“[7] vorgestellt.

  Im relationalen Datenmodell werden alle Entitäten[8] der realen Welt (Beispiel: Name, Anschrift, Geburtstag) durch Relationen dargestellt. Relationen sind mehrspaltige Tabellen, deren Spalten man als Attribute bezeichnet.

[7] Die Grundlagen der Theorie der relationalen Datenbank wurden von Edgar F. Codd in den 60ern und 70ern gelegt und in seiner Arbeit A Relational Model of Data for Large Shared Data Banks beschrieben. Theoretisch basieren alle Operationen auf der relationalen Algebra.

[8] Eine Entität (Entity) ist ein Objekt der Realität. Siehe auch Kapitel: Entity-Relationship-Modell

Relationale Datenbanken bestehen aus beliebig vielen Einzeltabellen (Relationen), die in beliebiger Art und Weise miteinander verknüpft werden können (Beziehungen).

In diesen Tabellen werden die Informationen in Spalten und Reihen gespeichert, wobei die Reihen die zusammenhängenden Datensätze, z.B. eine Adresse eines Kunden aufnehmen und die Spalten die Attribute z.B. Name, Ort und Strasse beinhalten.

Die gespeicherten Daten einer Spalte sind immer vom gleichen Attributstyp (long, integer, double, string, blob..).

Relationen

Relationenname / Tabellenname		***↓ Spalten / Felder / Attribute ↓***				***← Relationenschema***
		Spalte1	*Spalte2*	*...*	*SpalteN*	
Relation / Tabelle	*Reihe1*					***← Reihen / Zeilen / Sätze / (Tupel)***
	Reihe2					
	Reihe3					
	Reihe4					
	...					
	...					
	...					
	ReiheN					

↑
Datum / Feld / Attribut

Datenbanksystem • z.B. Oracle, IBM-DB2, Interbase, Firebird, MS-SQL, MySQL, Informix

Datenbank-managementsystem • Die Verwaltungssoftware organisiert intern das strukturierte Speichern der Daten gemäß einem vorgegebenen Datenbankmodell (z.B. relationales Datenbankmodell) und kontrolliert alle lesenden und schreibenden Zugriffe auf die Datenbank.

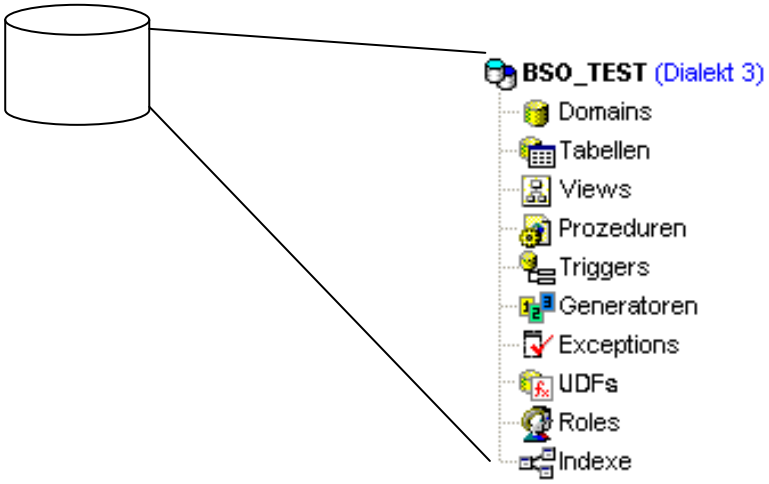
Als externe Schnittstelle stellt es eine Datenbanksprache (**SQL**) zur Formulierung von Abfragen, zum Einfügen und Ändern von Daten und für administrative Befehle (**DDL / DCL**) zur Verfügung. Die Datenbank enthält zusätzlich zu den eigentlichen Daten noch die Beschreibung der Datenstruktur, den sogenannten Datenkatalog (Metadaten).

Metadaten • "Daten über Daten". Daten, die einem bestimmten Zweck dienen. Per Definition sind Metadaten "maschinenlesbare Informationen über elektronische Ressourcen oder andere Dinge." Im DBS liefern die Metadaten Informationen über die Tabellen, die Views, die Stored Procedures usw.

Datenbank

- allgemein für (auch nicht-) elektronische Sammlungen von Daten.

Typischer Aufbau einer Datenbank



Domäne (Domain)

- Eine Domäne ist ein die Datenbank umfassender, benutzerdefinierter Datentyp. Eine Domäne wird verwendet, um Formate und Umfang von Spalten zu definieren, auf denen aktuelle Spaltendefinitionen in Tabellen basieren können.
  Dies ist sinnvoll, wenn Spalten gleich in mehreren Datenbanktabellen die gleichen Eigenschaften haben sollen Die Spalteneigenschaften (z.B. Feldlänge, Typ, Not Null, Constraints, Arrays etc.) müssen so nur einmal in der Domäne definiert werden. Bestimmte, in der Domäne festgelegte Eigenschaften können in der Tabellenfeld-definition überschrieben werden, d.h. eine Spalte kann auf einer Domäne basieren; es können jedoch immer noch kleine Änderungen für diese Spalte vorgenommen werden.

Tabellen

- Eine Tabelle ist ein Datenspeicherobjekt bestehend aus einer zweidimensionalen Matrix oder Gitter aus Spalten und Zeilen, theoretisch bekannt als eine mathematische Relation. Die Tabelle ist ein fundamentales Element der Datenspeicherung.

  Relationale Datenbanken speichern alle ihre Daten in Tabellen (Relationen). Eine Tabelle besteht aus einem ungeordneten Satz an horizontalen Zeilen (Tuples). Jeder dieser Zeilen enthält die gleiche Anzahl an vertikalen Spalten für die individuellen einzelnen Informationstypen.

  Die Schnittfläche einer einzelnen Zeile und einer Spalte ist ein Feld, das ein bestimmtes, unteilbares, atomares Informationsteilchen enthält. D.h. Spalten führen die Namen der einzelnen Felder auf und Zeilen sind die Datensätze, die die zusammengehörigen Informationen beinhalten. Jeder Datenbankspalte kann ein anderer Datentyp zugeordnet sein.

  Eine Tabelle ist ein Datenbankobjekt, sie ist Teil der Metadaten der Datenbank.

Tupel: Der Begriff Tupel[9] wird in der theoretischen Informatik für geordnete Werteansammlungen und - insbesondere in der Relationenalgebra - als Synonym für Datensatz verwendet. Seine Werte werden Attribute genannt.

Tabellen in relationalen Datenbanken enthalten Tupel. Diese Aussage stimmt jedoch nicht immer, denn sie würde bedeuten, dass für einen Datensatz mit z.B. 3 Feldern immer genau bestimmt ist, welches Feld die Komponente 1, die Komponente 2 und die Komponente 3 der geordneten Menge (= Tupel) wäre. Tatsächlich kennen manche relationale Datenbanksysteme solch eine Anordnung ihrer Datensätze nicht. Der Zugriff auf ein Feld erfolgt bei ihnen durch einen Feldnamen. Komponenten von Tupeln haben aber keine Namen, sondern sind durch ihre Position im Tupel bestimmt. Einem Datensatz, geordnet nach Feldnamen in der Datenbank entspricht daher eher einem Namensraum (Namespace) wie er in objektorientierten Programmiersprachen Anwendung findet. Die Teile des theoretischen, relationalen Modells, die den Datensätzen relationaler Datenbanken entsprechen, sind tatsächlich Tupel. Dabei ist ein Tupel eine genau bestimmte Art von Abbildung. Nur werden diese in vielen relationalen Datenbanken eben durch Namensräume ersetzt.

Attribut: Ein Attribut (von lateinisch attribuere = zuteilen, zuordnen) ist ein im Wert einer Variablen gespeichertes Merkmal eines konkreten Objekts. In Programmiersprachen werden Objekte durch die Gesamtheit ihrer Attributwerte beschrieben (Wert, Namen, Typ, Gültigkeit, ...)

Sichten (Views)

- Ein View ist ein gespeicherter SELECT einer oder mehrerer Tabellen. Die wiedergegebenen Zeilen werden über den SELECT-Befehl definiert, der die Spalten der Quelltabellen aufführt. Nur die View-Definition wird in der Datenbank gespeichert, er repräsentiert nicht direkt physikalisch gespeicherte Daten. Der WHERE-Befehl kann ebenfalls verwendet werden. Ein View hat keine Eingabeparameter.

  Der View kann mit einer virtuellen Tabelle verglichen werden. Er kann, in fast allen Belangen, wie eine Tabelle behandelt werden, da er als Basis für Abfragen und sogar in manchen Fällen für Updates verwendet wird. Es ist ebenfalls möglich, SELECT, PROJECT, JOIN und UNION Operationen auf Views vorzunehmen, als wären sie Tabellen.
  Views geben dem Endanwender eine individuelle Sicht einer unterliegenden Tabelle in einer Datenbank und sie vereinfachen den Datenzugriff, indem sie den User von Details über die Art der Informationsverteilung auf diverse Tabellen abschirmen. Sie bieten außerdem Sicherheit, indem bestimmte Spalten in den Tabellen vor Endanwendern verborgen werden.

  Views erlauben dem Entwickler, seine Daten scheinbar zu denormalisieren und Informationen von zwei oder mehreren Tabellen in einer einzigen (virtuellen) Tabelle zu kombinieren.

[9] Das Wort „Tupel“ wurde vermutlich als Kunstwort aus der Endung von Wörtern wie Quintupel, Sixtupel u.s.w. gebildet. Das Wort Quintupel ist vermutlich mit dem Adjektiv quintupel (fünffach) verwandt, das vom spätlateinischen quintuplex (fünffältig) stammt. Damit ist ein „n-Tupel“ in etwa ein „n-Fältiges“.

Prozeduren

- Eine gespeicherte Prozedur (Stored Procedure) ist eine Reihe von SQL-Befehlen, gespeichert in einem eigenständigen Programm in der Datenbank, als Teil der Metadaten, oft auch als Routinen bezeichnet.

  Eine Prozedur kann spezielle Prozesse in den Metadaten und den Daten innerhalb der Datenbank vornehmen. Die Programmausführung geschieht auf dem Server. Gestartet werden können sie durch den EXECUTE PROCEDURE - Befehl unter Angabe des Prozedurnamens und einer Auflistung der Parameter.
  Jede Stored Procedure ist ein eigenständiges Codemodul, das interaktiv oder als Teil eines SELECT-Befehls von einer anderen Stored Procedure oder einer anderen Anwendungsumgebung ausgeführt werden kann. Sie können direkt von den Anwendungen aufgerufen werden oder an Stelle von einer Tabelle oder eines Views in einem SELECT-Befehl eingesetzt werden. Stored Procedure können Eingabeparameter erhalten und Werte an Anwendungen zurückgeben.

Ereignisse (Trigger**)**

- Ein Trigger ist eine unabhängige Reihe von Befehlen, gespeichert in einem eigenständigen Programm (SQL-Skript) in einer Datenbank. Trigger werden automatisch in einer Datenbank ausgeführt, wenn bestimmte Ereignisse eintreten. Beispielsweise ist es möglich, vor einer Eingabe zu prüfen, ob ein Primärschlüssel schon vorhanden ist oder nicht und - wenn nötig - einen Wert durch einen Generator zuzuordnen. Diese Ereignisse sind tabellen- und zeilenabhängig. (Z.B. BEFORE INSERT, BEFORE UPDATE, AFTER DELETE)

  Ein Trigger ist ein Datenbankobjekt, das hauptsächlich der Integrität und Sicherheit dient. Er kann eine oder mehrere Ausführungsanweisungen enthalten. Trigger nehmen keine Eingabeparameter an und geben keine Ausgabeparameter zurück. Trigger sind fast identisch mit Stored Procedures, mit Ausnahme der Art und Weise, wie sie aufgerufen werden. Trigger werden automatisch aufgerufen, wenn eine Änderung in einer Tabellenzeile vorgenommen wird.

Generatoren

- Generatoren sind automatische, die ganze Datenbank umfassenden, fortlaufende Zähler. Ein Generator ist ein Datenbankobjekt und Teil der Metadaten der Datenbank. Er ist ein fortlaufender Zähler der automatisch in eine Zeile eingegeben werden kann. Generell wird ein Generator verwendet, um eindeutige Identifikationsnummern für Primärschlüssel zu bestimmen. Dies geschieht meist im Zusammenspiel mit einem Trigger (z.B. BEFORE INSERT).

  Beispiel:

```
TRIG_TBLADRESSEN BEFORE INSERT

BEGIN

  IF(NEW.ID IS NULL) THEN
    NEW.ID = GEN_ID(GEN_TBLADRESSEN_ID,1);

END
```

Exceptions

- Exceptions sind benutzerdefinierte Fehlermeldungen. Sie werden speziell für die Datenbank geschrieben und zur Verwendung in Stored Procedures und Triggern in der Datenbank gespeichert.

  Wenn beispielsweise in einem Trigger festgestellt wird, dass ein Wert in einer Tabelle falsch ist, wird die Exception ausgelöst. Das führt zu einem Roll Back der gesamten Transaktion, die die Client-Anwendung versucht zu committen. Exceptions können verschachtelt werden. Sie können von mehreren Modulen in einer Anwendung genutzt werden und sogar von mehreren Anwendungen einer Datenbank. Sie bieten einen einfachen Weg, um den Umgang mit vorprogrammierten Eingabefehlern zu standardisieren.

  Exceptions werden typischerweise verwendet, um Programmlogik zu implementieren. Zum Beispiel möchte man nicht, dass ein Benutzer einen Lagerartikel, der schon von einem anderen Benutzer für dessen Kunden reserviert wurde, zwischenzeitlich verkauft.

Benutzerdefinierte Funktion

- Eine benutzerdefinierte Funktion (User-Defined-Function UDF) wird verwendet, um Aufgaben zu bewältigen, die die Datenbank nicht standardmäßig anbietet oder nicht bewältigen kann.

  Man kann sie als externe Datenbankfunktion beschreiben, die komplett in einer anderen Sprache, beispielsweise in C++, VB oder Pascal geschrieben sein kann, um Datenmanipulationsaufgaben zu übernehmen, die nicht direkt von dem Datenbankmanagement unterstützt werden.

  UDFs können auf dem Server aufgerufen und ausgeführt werden. Diese Funktionen können für sich stehen oder in Bibliotheken gesammelt sein. UDFs bieten die Möglichkeit, eigene Funktionen (wie SUBSTR) zu erzeugen und in die Datenbank selbst zu integrieren. (DECLARE EXTERNAL FUNCTION).

Role

- Eine Role ist eine definierte Gruppe von Privilegien. Sie vereinfacht das Zuweisen von Benutzerrechten, da mehreren Benutzern dieselbe Role zugewiesen werden kann. Zum Beispiel können in einer großen Verkaufsabteilung alle Mitarbeiter, die in die Abwicklung eingehender Aufträge involviert sind, der Role "Auftragsabwicklung" zugeordnet sein. Sollte eine Änderung der Rechte dieser Benutzer notwendig werden, muss lediglich die Role geändert werden.

Index

- Ein Datenbankindex, oder kurz Index (im Plural "Indexe" oder "Indizes"), ist eine von der Datenstruktur getrennte Datenstruktur in einer Datenbank, die die Suche und das Sortieren nach bestimmten Feldern beschleunigt.

  Ein Index besteht aus einer Ansammlung von Zeigern (Verweisen), die eine Ordnungsrelation auf eine oder mehrere Spalten in einer Tabelle definieren. Wird bei einer Abfrage eine indizierte Spalte als Suchkriterium herangezogen, sucht das Datenbankmanagement-system (DBMS) die gewünschten Datensätze anhand dieser Zeiger. In der Regel finden hier B+-Bäume Anwendung. Ohne Index müsste die Spalte sequentiell durchsucht werden, was selbst mit modernster Hardware und Software viel Zeit in Anspruch nehmen kann.

Relationale Datenbanken

Das relationale Datenmodell wurde von Edgar F. Codd entwickelt und 1970 in seiner Arbeit „A relational model of data for large shared data bases“[10] vorgestellt. Diese Arbeit gilt als eine der wichtigsten Arbeiten der Informatik.

Im relationalen Datenmodell werden alle Entitäten[11] der realen Welt (Beispiel: Kunde, Rechnung, Auftrag) durch Relationen dargestellt. Relationen sind mehrspaltige Tabellen, deren Spalten man als Attribute bezeichnet.

Relationale Datenbanken bestehen aus beliebig vielen Einzeltabellen (Relationen), die in beliebiger Art und Weise miteinander verknüpft werden können (Beziehungen).

In diesen Tabellen werden die Informationen in Spalten und Reihen gespeichert, wobei die Reihen die zusammenhängenden Datensätze, z.B. eine Adresse eines Kunden aufnehmen und die Spalten die Attribute z.B. Name, Ort und Strasse beinhalten. Die gespeicherten Daten einer Spalte sind immer vom gleichen Attributstyp (long, integer, double, string, blob..).

Ein wichtiger Schritt des Modellierungsprozesses nach Codd ist die Normalisierung der Daten und Datenstrukturen in einer Datenbank. Diese soll Redundanzen verringern und Anomalien verhindern, um so die Wartung einer Datenbank zu vereinfachen, sowie die Konsistenz der Daten zu gewährleisten.

Die vier Normalformen, die seitdem bei dem relationalen Datenbankentwurf zum Einsatz kommen, stammen ebenfalls von Codd. Mittlerweile wurden diese ergänzt und weitere hinzugefügt.

Für jedes Attribut einer Relation wird ein zulässiger Wertebereich festgelegt. Die Relation ist dann eine Teilmenge des kartesischen Produkts [12] der Attribut-Wertebereiche.

- ***Angestellter (Name, <u>Angestellter-Nr</u>, Funktion)***

bezeichnet eine Relation (Tabelle), in der die Daten von Angestellten abgelegt werden. In der Klammer sind die Attribute (Spalten) der Tabelle aufgeführt. Wichtig ist hierbei die Unterscheidung zwischen der obigen Strukturbeschreibung (Benennung der Relation und ihrer Attribute) und den einzelnen Datensätzen (Instanziierungen) des festgelegten Typs.
(Ismir Schnuppe, 4711, ‘Verkauf-Süd‘).

Es lassen sich durch Relationen aber nicht nur Entitäten im Sinne von existierenden Personen oder (realen oder geistigen) Objekten erfassen, sondern auch Beziehungen zwischen diesen.

Betrachtet man beispielsweise die Entitäten der Relation *Angestellter* die Entitäten der Relation Abteilung: ***Abteilung (<u>Abteilung-Nr</u>, Abteilung-Name)*** so lässt sich die Beziehung Angestellter "*gehört zu Abteilung*" durch eine Relation mit den Attributen *Abteilung-Nr und Angestellter-Nr* darstellen.

- ***Angestellter-Abteilung(Abteilung-Nr#, Angestellter-Nr#)***

[10] Die Grundlagen der Theorie der relationalen Datenbank wurden von Edgar F. Codd in den 60ern und 70ern gelegt und in seiner Arbeit A Relational Model of Data for Large Shared Data Banks beschrieben.
Theoretisch basieren alle Operationen auf der relationalen Algebra.

[11] Eine Entität (Entity) ist ein Objekt der Realität. Siehe auch Kapitel: Entity-Relationship-Modell

[12] Es seien A und B Mengen. Das kartesische Produkt A x B (gelesen: A kreuz B) ist die Menge aller geordneten Paare (a,b), wobei a ein Element der Menge A und b ein Element der Menge b ist.

Eine solche Relation kann durchaus noch zusätzliche Attribute enthalten, die Eigenschaften der Beziehung beschreiben. Im Beispiel etwa das Eintrittsdatum oder Anschrift und Telefonnummer.

Schlüssel (Primär-, Fremdschlüssel)

Um Doppeldeutigkeiten zu vermeiden, sind die Spalten und die Reihen einer Relation eindeutig zu identifizieren. Es dürfen weder die Spalten noch die Reihen mehrdeutig vorkommen. Bei Spalten geschieht dies über die Namensgebung, bei den Datensätzen über einen oder mehrere Schlüssel (Key).

Die bereits erwähnte Normalisierung (Normalformen nach Codd) fordert, dass jede Entität, im aktuellen Beispiel „jeder einzelne Angestellter“ innerhalb der Relation eindeutig identifiziert werden kann. Hierzu wird in eine Relation ein „Schlüssel“ beispielsweise eine Angestellten-Nr oder ein Zähler eingefügt. Dieses Schlüsselattribut nennt man Primärschlüssel. Primärschlüssel kann auch eine Kombination von mehreren Attributen sein.

In dem Fall, dass ein Angestellter nur zu einer einzigen Abteilung gehört, kann man die Beziehung alternativ durch eine zusätzliche Spalte in der Angestellten-Tabelle darstellen. In diese Spalte trägt man die dann Abteilungsnummer (Primärschlüssel der Relation Abteilung) ein. Eine solche Spalte, die Schlüsselwerte aus einer fremden Relation enthält, nennt man auch Fremdschlüssel.

In der Literatur findet man für den Primärschlüssel (Primary-Key) häufig die Abkürzung PK, für den Fremdschlüssel (Foreign-Key) die Abkürzung FK. In der Chen-Notation[13] werden Primärschlüssel unterstrichen und Fremdschlüssel mit dem Zeichen ‚#‘ gekennzeichnet.

Für Schlüsselattribute gelten zusätzliche Regeln und Einschränkungen (Constraints). Beim Primärschlüssel gilt:

- der Skalar „Primärschlüssel“ muss einzigartig sein
- der Primärschlüssel kann nicht NULL sein.
- der Primärschlüssel ist nicht auf ein Attribut beschränkt, aber er darf aber auch in Kombination keine Duplikate aufweisen.

Beim Fremdschlüssel sind diese Anforderungen eher selten aktiv gesetzt. Beim Fremdschlüssel gelten andere Regeln, zum Beispiel solche für die Erhaltung der referenziellen Integrität[14] der Datenbank. Deshalb muss:

- ein Fremdschlüssel auf Referentielle Integrität geprüft werden.
- ein Fremdschlüssel darf nur existierende Werte der Mastertabelle annehmen.

Leistungsfähige relationale Datenbankmanagementsysteme (DBMS) bieten interne Funktionen (CASCADE, TRIGGER) mit deren Hilfe die Integritätsbedingungen der Relationen überwacht und nach Datenänderung wieder hergestellt werden können.

Attributbezeichnung

Für die Namensgebung der Spalten gelten in verschiedenen Datenbanksystemen unterschiedliche Restriktionen. Während man beispielsweise unter MS-Access nahezu jede beliebige Zeichenkombination, inklusive Umlaute, Leerzeichen, Groß- und Kleinschreibung im Namen verwenden kann, stellen viele Datenbankserver hier deutlich gesteigerte Anforderungen an die Bezeichnung der Tabellenfelder.

[13] Chen, Peter: „The entity relationship model towards a unified view of data“ ACM Transactions on Database Systems, 1976
[14] Siehe auch Kapitel: Begriffe

Bei einer geplanten Spaltenbezeichnung mit "Name", "Vorname", "Alter" und "Typ" wäre nur die Bezeichnung "Vorname" zulässig. Das Wort "Name" macht hierbei sogar bei MS-Access Probleme, da z.B. im Programmcode der Befehl Me.Name (Me = Selbstreferenz) den Klassennamen und nicht den Inhalt des Feldes "Name" liefert. Die Bezeichnungen "Alter" und "Typ" sind Schlüsselwörter im SQL-Syntax (z.B. ALTER TABLE) und führen dadurch oft zu Missverständnissen.

Es gibt in der verschiedenen SQL-Dialekten Korrekturmöglichkeiten um diese syntaktische Restriktion zu umgehen. Bei Microsoft kann man beispielsweise Spaltenbezeichnungen in eckige Klammen ([Meine Spalte]) fassen. Bei vielen Datenbanken findet man bei der Installation oder im Parameterbereich Einstellmöglichkeiten wie beispielsweise ‚quoted identifiers". Wird diese Option gewählt, dann können Bezeichnungen künftig auch mit Sonderzeichen oder Leerschritte benannt werden, müssen aber fortan immer in Gänsefüßchen angegeben sein. Ähnliche Funktion hat der Parameter „sensitive identifiers" der sich auf die Unterscheidung von Groß- und Kleinschreibung bezieht (case sensitiv).

Aus eigener Erfahrung rate ich von dieser Nomenklatur dringend ab. Zum einen sieht der Anwender bei professioneller Programmierung die interne Bezeichnung der Relation eh nicht in seiner Benutzermaske und zum anderen muss diese Bezeichnung an den unterschiedlichsten Stellen immer wieder eingegeben oder korrigiert werden. Der Programmieraufwand wächst deutlich.

Transaktion

Ein Datenbanksystem muss die parallele Verarbeitung verschiedener Benutzeraufträge (so genannte Transaktionen) ermöglichen. Wesentlich dafür ist die Einhaltung der folgenden vier Eigenschaften, die unter dem **ACID-Prinzip** bekannt sind:

- **Atomarität** (**A**tomicity)

  Eine Transaktion ist die kleinste, nicht mehr zerlegbare Einheit von Verarbeitungsschritten und wird nur vollständig oder gar nicht ausgeführt (COMMIT). Sollte es bei der Ausführung zu einem Fehler bzw. Abbruch kommen, werden alle innerhalb der Transaktion bereits getätigten Änderungen an der Datenbank wieder zurückgenommen (ROLLBACK).

- **Konsistenz** (**C**onsistency)

  Eine Transaktion überführt eine Datenbank immer von einem konsistenten Zustand in einen anderen konsistenten Zustand, d.h. alle Integritätsbedingungen der Datenbank werden eingehalten.

- **Isolation** (**I**solation)

  Jede Transaktion läuft isoliert und in jeder Hinsicht unabhängig von anderen Transaktionen ab. Dazu gehört auch, dass jeder Transaktion nur diejenigen Daten aus der Datenbank zur Verfügung gestellt werden, die Teil eines konsistenten Zustands sind. Sollten parallele Transaktionen um Ressourcen konkurrieren, so müssen die Transaktionen serialisiert werden.

- **Persistenz** (**D**urability)

  Das Ergebnis einer erfolgreichen Transaktion bleiben in der Datenbank persistent[15].

[15] Persistenz: lat. persistere „verharren", allgemein: etwas mit dauerhafter Beschaffenheit oder Beharrlichkeit, das langfristige Fortbestehen einer Sache

Vorlesung: Datenbanken

Kapitel (B2): Grundlagen relationaler Datenbanken

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

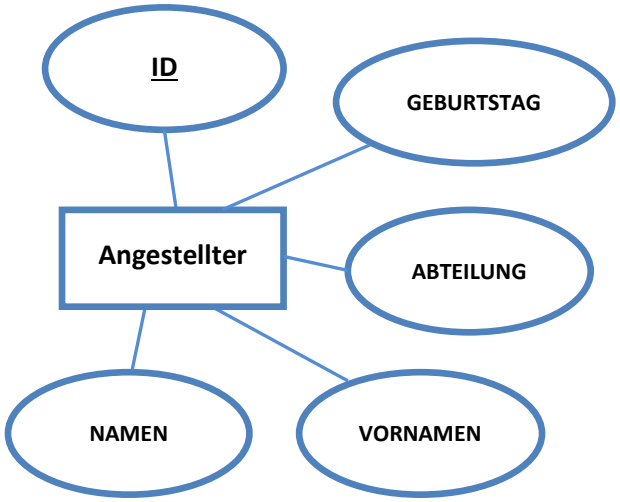
Beispiel: Anlage einer Relation

Gegeben ist:

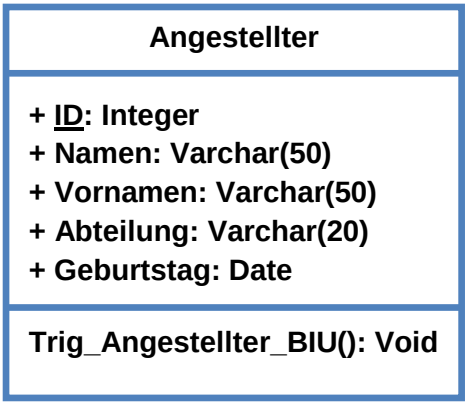
Angestellter (ID, VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)

alternative Darstellung als

Entity-Relationship-Modell (ERM)



und als **Klassendiagramm**.



Beispiel der Relation

Angestellter (ID, NAMEN, VORNAMEN, ABTEILUNG , GEBURTSTAG)

Anlegen der Tabelle

```
CREATE TABLE ANGESTELLTER (
    ID          INTEGER NOT NULL,
    NAMEN       VARCHAR(50),
    VORNAMEN    VARCHAR(50),
    ABTEILUNG   VARCHAR(20),
    GEBURTSTAG  DATE );
```

Bestimmen des Primärschlüssels

```
ALTER TABLE ANGESTELLTER
      ADD CONSTRAINT PK_ANGESTELLTER PRIMARY KEY (ID);
```

Erzeugen eines (Auto-)Zählers für den Primärschlüssel - **MySQL**

```
ALTER TABLE `ANGESTELLTER` MODIFY COLUMN `ID` INTEGER(11)
NOT NULL AUTO_INCREMENT UNIQUE;
```

Erzeugen eines (Auto-)Zählers + Triggers für den Primärschlüssel - **Firebird**

```
CREATE GENERATOR GEN_ANGESTELLTER_ID;

CREATE OR ALTER TRIGGER TRIG_ANGESTELLTER_BIU FOR ANGESTELLTER
ACTIVE BEFORE INSERT OR UPDATE POSITION 0
AS
BEGIN
  IF(NEW.ID IS NULL) THEN NEW.ID = GEN_ID(GEN_ANGESTELLTER_ID,1);
END
```

Erzeugen weiterer Indizes[1] für schnelles Auffinden von Datensätzen

```
CREATE INDEX ANGESTELLTER_IDX1 ON ANGESTELLTER (NAMEN);
CREATE INDEX ANGESTELLTER_IDX2 ON ANGESTELLTER (VORNAMEN);
```

Erzeugen von Musterdaten
Im Beispiel als INSERT – Anweisung mit festen Werten (VALUES)

```
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Pitt', 'Bull', 'Verkauf', '1954-06-05');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kai', 'Nesau', 'Produktion', '1955-07-14');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Halter', 'Transport', '1956-09-26');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Fänger', 'Verkauf', '1956-09-26');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Klara', 'Korn', 'Produktion', '1957-03-20');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ken', 'Guhru', 'Verkauf', '1957-04-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Lee', 'Gewiese', 'Produktion', '1957-11-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Martin', 'Shorn', 'Transport', '1958-12-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mark', 'Kiese', 'Verkauf', '1964-08-15');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Franz', 'Ösisch', 'Produktion', '1964-09-16');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kristian', 'Harten', 'Verkauf', '1965-02-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Rosa', 'Schlüpfer', 'Produktion', '1966-01-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Karla', 'Schnikof', 'Transport', '1966-11-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Matt', 'Brötchen', 'Verkauf', '1966-12-24');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Geber', 'Transport', '1966-12-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Hans', 'Estadt', 'Transport', '1967-05-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Lin', 'Seneintopf', 'Verkauf', '1967-06-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Max', 'Imum', 'Produktion', '1969-10-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Pfiff', 'Transport', '1970-06-10');
```

[1] - Suchkriterien sind in den meisten Datenbanken ‚case-sensitive'.
- Ein ‚aufsteigender' und ein ‚absteigender' Index muss separat definiert werden.

```
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Axel', 'Flecken', 'Verkauf', '1970-07-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bo', 'Densee', 'Produktion', '1970-08-12');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Norma', 'Lerweise', 'Produktion', '1970-09-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Max', 'Imal', 'Produktion', '1971-06-05');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Axel', 'Haar', 'Produktion', '1971-07-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Manni', 'Fest', 'Produktion', '1971-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kalle', 'Brieren', 'Produktion', '1972-04-03');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Petro', 'Leum', 'Verkauf', '1972-05-04');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Eddi', 'Kett', 'Produktion', '1972-06-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Arno', 'Nym', 'Produktion', '1973-10-17');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Leo', 'Pard', 'Produktion', '1975-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Karl', 'Kulation', 'Buchaltung', '1987-11-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mark', 'Klöschen', 'Verkauf', '1975-10-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Uri', 'Nieren', 'Produktion', '1976-02-19');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Hauchen', 'Produktion', '1976-11-28');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Karl', 'Rasiert', 'Transport', '1977-07-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kurt', 'Zeitgedächtnis', 'Verkauf', '1977-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Karl', 'Sruhe', 'Produktion', '1954-06-05');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Scheinend', 'Produktion', '1955-07-14');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ann', 'Alphabet', 'Transport', '1956-09-26');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Donna', 'Wetter', 'Produktion', '1956-09-26');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ben', 'Efitz', 'Verkauf', '1957-03-20');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ted', 'Owierung', 'Buchaltung', '1987-02-24');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Nena', 'Del', 'Verkauf', '1957-04-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Armin', 'Osäure', 'Produktion', '1957-11-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Peer', 'Son', 'Produktion', '1958-12-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ninah', 'Dran', 'Buchaltung', '1966-12-24');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Marga', 'Rine', 'Produktion', '1964-08-15');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kurt', 'Zundklein', 'Verkauf', '1957-03-20');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kai', 'Lerei', 'Verkauf', '1964-09-16');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ken', 'Ziffer', 'Produktion', '1965-02-25');
```

```
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bert', 'Igermann', 'Verkauf', '1957-04-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Jean', 'Darmerie', 'Verkauf', '1966-01-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bea', 'Beitung', 'Produktion', '1967-05-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Rick', 'Schah', 'Verkauf', '1966-11-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Niko', 'Tin', 'Produktion', '1967-06-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Lou', 'Ping', 'Produktion', '1966-12-24');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Elle', 'Mente', 'Produktion', '1966-12-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Niko', 'Laus', 'Produktion', '1967-05-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bill', 'Derrahmen', 'Produktion', '1977-07-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Frank', 'Furtammain', 'Verkauf', '1967-06-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Katha', 'Maran', 'Produktion', '1969-10-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Sam', 'Urai', 'Produktion', '1970-06-10');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Lou', 'Latsch', 'Produktion', '1970-07-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Thorge', 'Schossen', 'Produktion', '1970-08-12');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Sam', 'Enbank', 'Buchaltung', '1966-01-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Stan', 'Desamt', 'Verkauf', '1970-09-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Andre', 'Seite', 'Produktion', '1971-06-05');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Anne', 'Theke', 'Produktion', '1971-07-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Armin', 'Gips', 'Produktion', '1971-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Sam', 'Stage', 'Produktion', '1977-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Atze', 'Tylsalicylsäure', 'Verkauf', '1972-04-03');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Jack', 'Enknopf', 'Buchaltung', '1965-02-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mira', 'Belle', 'Produktion', '1972-05-04');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Marie', 'Enkäfer', 'Verkauf', '1972-06-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kai', 'Nelust', 'Buchaltung', '1956-09-26');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Knut', 'Schen', 'Verkauf', '1973-10-17');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Klaus', 'Uhr', 'Produktion', '1975-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Axel', 'Nässe', 'Produktion', '1975-10-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Rosi', 'Ne', 'Verkauf', '1976-02-19');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Wim', 'Pernschlag', 'Verkauf', '1976-11-28');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ernst', 'Haft', 'Produktion', '1977-07-22');
```

```
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Volkar', 'Putt', 'Buchaltung', '1956-09-26');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mag', 'Nicht', 'Verkauf', '1977-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Clair', 'Werk', 'Produktion', '1990-05-09');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Hella', 'Wahnsinn', 'Buchaltung', '1975-10-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Eros', 'Ion', 'Transport', '1969-10-25');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ted', 'Dybär', 'Verkauf', '1988-09-24');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ernst', 'Gemeint', 'Produktion', '1975-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Olli', 'Bergott', 'Transport', '1971-08-23');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Till', 'Siter', 'Produktion', '1971-07-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mona', 'Tsende', 'Produktion', '1972-06-21');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Theo', 'Retisch', 'Buchaltung', '1957-11-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Nick', 'Erchen', 'Produktion', '1976-02-19');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ben', 'Utzmich', 'Transport', '1978-03-31');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Peter', 'Sburg', 'Produktion', '1966-12-27');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Rhea', 'Lität', 'Produktion', '1976-11-28');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mäh', 'Drescher', 'Produktion', '1986-07-29');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Karl', 'Lauer', 'Produktion', '1987-08-30');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Frika', 'Delle', 'Buchaltung', '1958-12-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Eric', 'Tion', 'Buchaltung', '1966-11-22');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Karl', 'Ender', 'Produktion', '1990-12-09');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bill', 'Iger', 'Produktion', '1970-06-10');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mario', 'Nette', 'Produktion', '1970-07-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bea', 'Tmung', 'Produktion', '1970-08-12');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Lilly', 'Putaner', 'Produktion', '1970-09-11');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Kenn', 'Stdumich', 'Produktion', '1954-06-05');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Knut', 'Schfleck', 'Produktion', '1971-06-05');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Finn', 'Derlohn', 'Produktion', '1972-05-04');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Erkan', 'Nalles', 'Produktion', '1972-04-03');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Gitta', 'Rensolo', 'Produktion', '1973-10-17');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ismir', 'Schnuppe', 'Produktion', '1964-09-16');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ben', 'Ebelt', 'Produktion', '1964-08-15');
```

```
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Mark', 'Aber', 'Produktion', '1955-07-14');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Bill', 'Dschirm', 'Produktion', '1986-06-13');
INSERT INTO ANGESTELLTER ( VORNAMEN, NAMEN, ABTEILUNG, GEBURTSTAG)
        VALUES ( 'Ismir', 'Uebel', 'Produktion', '1987-05-12');
COMMIT WORK;
```

Beispiel der angelegten Relation ANGESTELLTER (Ausschnitt)

ID	VORNAMEN	NAMEN	ABTEILUNG	GEBURTSTAG
1	Pitt	Bull	Verkauf	05.06.1954
2	Kai	Nesau	Produktion	14.07.1955
3	Ann	Halter	Transport	26.09.1956
4	Ann	Fänger	Verkauf	26.09.1956
5	Klara	Korn	Produktion	20.03.1957
6	Ken	Guhru	Verkauf	21.04.1957
7	Lee	Gewiese	Produktion	27.11.1957
8	Martin	Shorn	Transport	11.12.1958
9	Mark	Kiese	Verkauf	15.08.1964
10	Franz	Ösisch	Produktion	16.09.1964
11	Kristian	Harten	Verkauf	25.02.1965
12	Rosa	Schlüpfer	Produktion	25.01.1966
13	Karla	Schnikof	Transport	22.11.1966
14	Matt	Brötchen	Verkauf	24.12.1966
15	Ann	Geber	Transport	27.12.1966
16	Hans	Estadt	Transport	21.05.1967
17	Lin	Seneintopf	Verkauf	22.06.1967
18	Max	Imum	Produktion	25.10.1969
19	Ann	Pfiff	Transport	10.06.1970
20	Axel	Flecken	Verkauf	11.07.1970
21	Bo	Densee	Produktion	12.08.1970
22	Norma	Lerweise	Produktion	11.09.1970
23	Max	Imal	Produktion	05.06.1971
24	Axel	Haar	Produktion	22.07.1971
25	Manni	Fest	Produktion	23.08.1971
26	Kalle	Brieren	Produktion	03.04.1972
27	Petro	Leum	Verkauf	04.05.1972
28	Eddi	Kett	Produktion	21.06.1972
29	Arno	Nym	Produktion	17.10.1973
30	Leo	Pard	Produktion	23.08.1975
31	Karl	Kulation	Buchaltung	23.11.1987
32	Mark	Klöschen	Verkauf	27.10.1975
33	Uri	Nieren	Produktion	19.02.1976
34	Ann	Hauchen	Produktion	28.11.1976
35	Karl	Rasiert	Transport	22.07.1977
36	Kurt	Zeitgedächtnis	Verkauf	23.08.1977
37	Karl	Sruhe	Produktion	05.06.1954
38	Ann	Scheinend	Produktion	14.07.1955
39	Ann	Alphabet	Transport	26.09.1956
40	Donna	Wetter	Produktion	26.09.1956

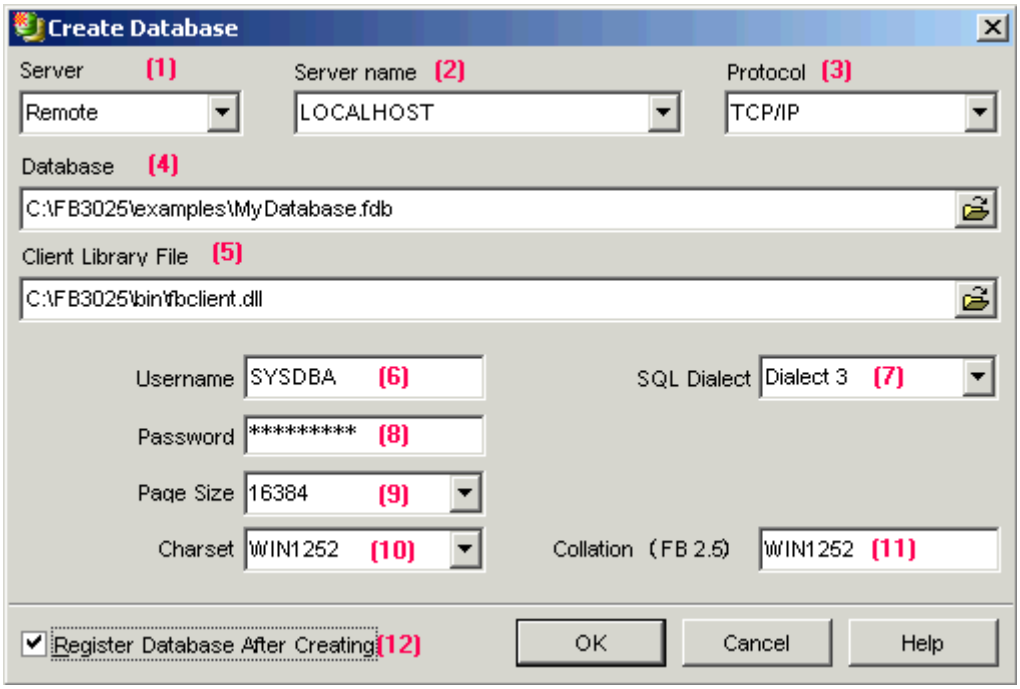
Erzeugen einer Datenbank mit Interbase / Firebird

```
CREATE DATABASE C:\DATABASEFILES\DHBW_FB.fdb
DEFAULT CHARACTER SET ISO8859_1 PAGE_SIZE 16384;
```

im Skript ausführen oder
oder mittels Datenbank-Management-Software.

Beispiel: IBExpert / Online-Hilfe

A new database can be created by simply using the IBExpert menu item *Database / Create Database...* or using the respective icon in the Database toolbar.
The *Create Database* dialog appears:



(1) Server: first the server which is to store the database needs to be specified. This can be local or remote.

- **Remote:** the remote connection needs to be defined by specifying **(2)** Server name and **(3)** Protocol. The drop-down list shows all servers previously connected to/from this workstation/PC.
- **Local: LOCALHOST** (own Server). To create a new database on the same machine where IBExpert is in use, you do not need to enter a server name.

We recommend always referencing a server, even if your database is sitting locally on your machine. Going directly using the local specification can cause problems (refer to **(3) Protocol** below), particularly with Windows Vista, so always use the *Remote* and *LOCALHOST* options.

The DOS PING LOCAL HOST or PING SRVNAME command shows the path if unknown (it is not necessary to know which operating system is running or where this server is). By specifying a local server, fields **(2)** and **(3)** are automatically blended out, as they are in this case irrelevant.

(2) Server name: must be known when accessing remotely. The following syntax should be used:

- Windows SERVER_NAME:C:\path\database.fdb
- Linux SERVER_NAME:/path/database.fdb

The standard port for Firebird and InterBase® is **3050**. However this is sometimes altered for obvious reasons of security, or when other databases/Firebird versions are already using this port. If a different port is to be used for the Firebird/InterBase® connection, the port number needs to be included as part of the server name. For example, if port number 3055 is to be used, the server name is SERVER/3055. If you use multiple Firebird versions and have a database, db1, sitting locally on C:\ root using the Firebird version on port 3052 (which has been specified in the firebird.config), the database connection path would be:

localhost3052:C:\db1.fdb

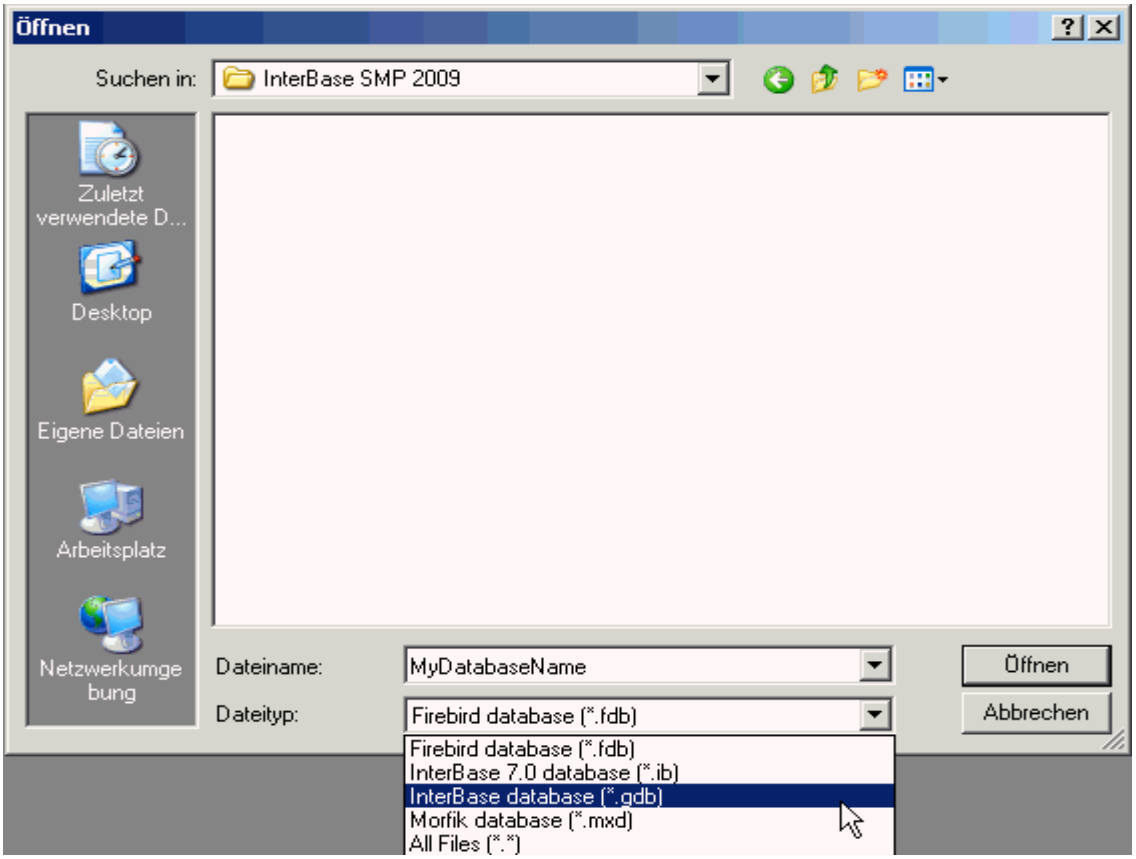
(3) Protocol: a drop-down list of three options: TCP/IP, NetBEUI or SPX. As a rule we recommend you always use TCP/IP (worldwide standard).

- SPX used to be used by Novell; now even Novell supports TCP/IP.
- NetBEUI (aka WNET) - is not really a network protocol, it simply accesses the line. It is slow as it makes everything available everywhere and anyone can access the information. This is also purely a Windows protocol. Since Firebird 2.0 this protocol no longer performs client impersonation (see *Change to WNET ("NetBEUI") Protocol*).

Note: in DOS the TRACERT command lists the protocol route. TCP/IP intelligently takes another direction, if one or part of the lines on the quickest route is blocked or down.

As the local protocol should only be used if really necessary on machines that are isolated and not part of any network, specify the database server connection if possible using *Remote* and *LOCALHOST* and selecting one of the above protocols. Although the introduction of the new local Firebird protocol, XNET, in Firebird 2.0 has solved many of the former problems of the previous local transport protocol (often referred to as IPC or IPServer) - please refer to *Local protocol--XNET* for further information.

(4) Database: by clicking on the folder icon to the right of this field, the path can easily found and specified, the database name entered, and the suffix selected from the pull-down list. The database name must always be specified with the drive and path when creating a database. Please note that the database file for a Windows server must be on a physical drive on the server, because Firebird/InterBase® does not support databases on mapped drive letters. The database suffixes do not have to adhere to the forms offered in the list.



(5) Client Library File: This displays the path and client library file name, as specified in the *Default Client Library* option, found in the IBExpert *Options menu* item, *Environment Options / Preferences*. This can, of course, be overwritten if wished.

(6) User Name: Only those names may be entered when creating a database, which already exist in the server security database ISC4.GDB, security.fdb or since Firebird 2.0 the new security2.fdb (which stores server rights; user rights for the database objects are stored in the database itself). The person creating the database becomes the database owner. Only the database owner and the SYSDBA (System Database Administrator) are allowed to perform certain operations upon the database (such as a database shutdown). Therefore if the database owner is defined as the SYSDBA, this is the only person entitled to perform these operations. *Note:* when a role with the name SYSDBA is created, no other users (not even the SYSDBA) can access the database. Therefore ensure the database is created by another user already registered in the security database and not the SYSDBA. This way there are at least two users able to perform key administrative tasks.

(8) Password: The passwords are encrypted in the ISC4.GDB, security.fdb or security2.fdb. If you insist upon using the SYSDBA name as the database owner, at least change the standard password **(masterkey)** to ensure at least some degree of security! The masterkey password should be changed as soon as possible after creating the database.

Firebird/InterBase® verifies only the first 8 characters of a password, even if a longer word is entered, i.e. in the case of the masterkey password only "masterke" is verified. All characters following the 8th are ignored.

(7) SQL Dialect: Here Dialect 1 (up to and including InterBase® 5) or 3 (InterBase® 6/Firebird) needs to be specified. For more information regarding this subject, please refer to *SQL Dialect*.

(9) Page size: Specifies the database page size in bytes. For more information regarding this subject, please refer to *Page size*.

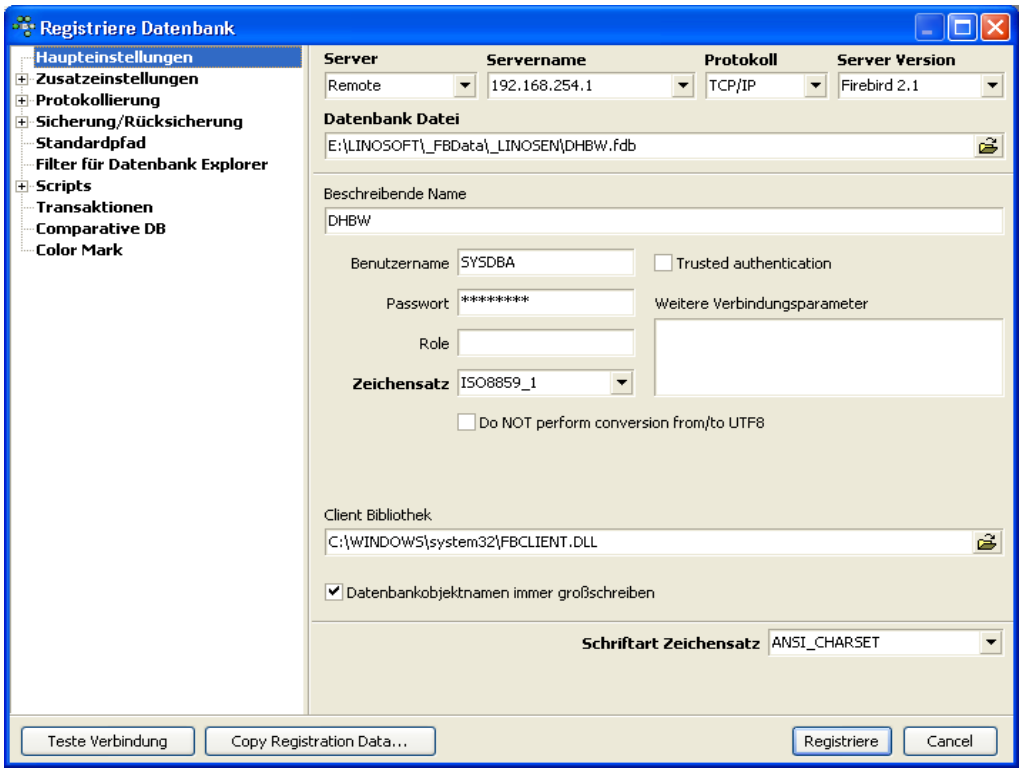
(10) Charset: Here the default character set can be defined for the database. (A default character set can be specified as default for all new databases in the IBExpert Options menu item, Environment Options, under *Default character set*.) This character set is useful, when the database created is to be used for foreign languages as it is applicable for all areas of the database unless overridden by the domain or field definition. If not specified, the parameter defaults to NONE, i.e. values are stored exactly as typed. For more information regarding this subject, please refer to *Default character set*.

(11) Collation (FB 2.5): This field is new to IBExpert version 2010.07.26 and allows you to specify a default collation for Firebird 2.5 databases.

(12) Register Database After Creating: This checkbox automatically generates the *Database Registration* dialog so that the database can be registered. Registration is necessary, so that IBExpert recognizes that a database is present. The *Register Database* dialog however offers many further options. We recommend clicking this checkbox (the default setting), so that the database is registered immediately after creation. If the database is not registered at the time of creation, it cannot be seen in the DB Explorer of the left of the IBExpert screen. This means that the user must know exactly where the new database can be found (i.e. which server, path, name etc.) when registering at a later date.

Registrieren einer Datenbank

Beispiel mit Firebird / IBExpert :



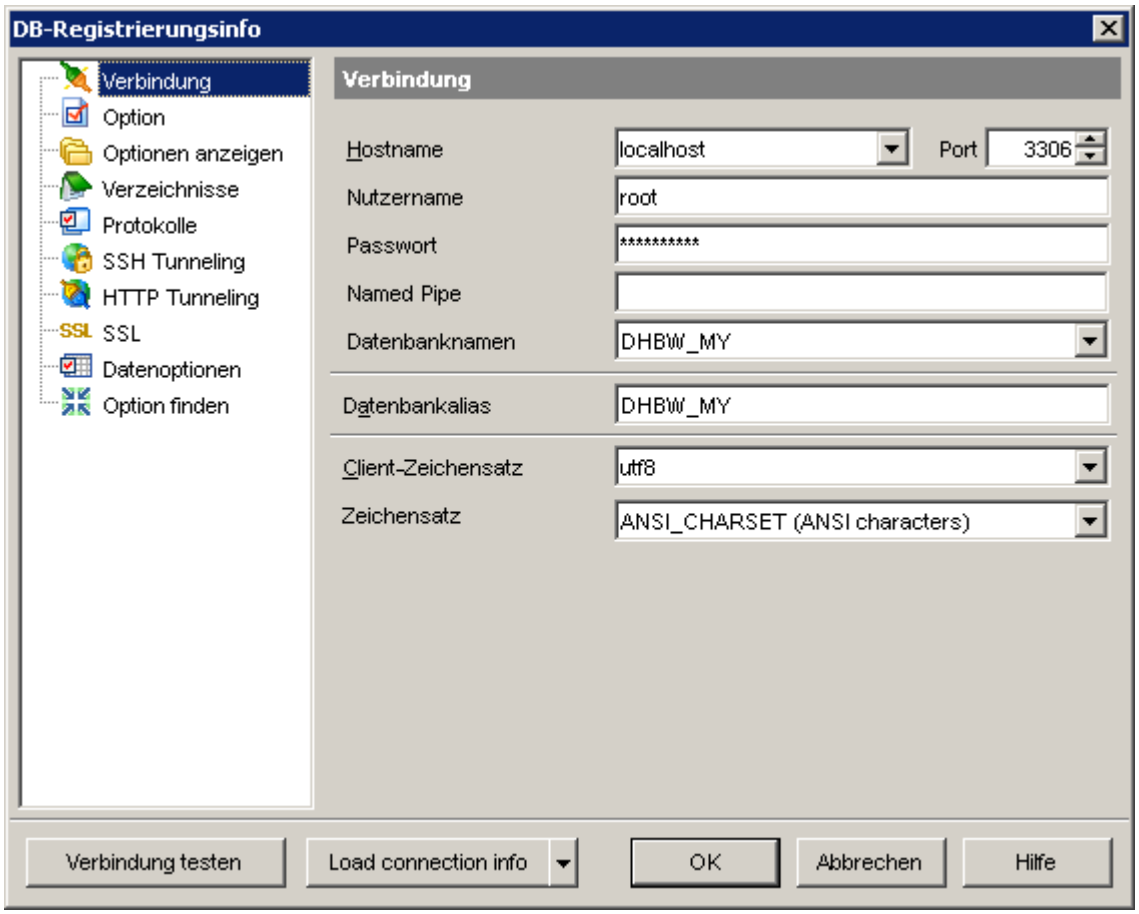
Erzeugen einer Datenbank mit MySQL

```
CREATE DATABASE `DHBW_MY`;
```

im Skript ausführen
oder mittels Datenbank-Management-Software.

Beispiel: MySQL / SQL-Manager

Registrieren einer Datenbank



Vorlesung: Datenbanken

Kapitel (C): Inkonsistenz / Integrität / Redundanz

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Inkonsistenz / Integrität

(Inkonsistenz: latein. *in* nicht, *con* zusammen, *sistere* halten)
(Integrität: latein. *integritas* Unversehrtheit / latein. *integer* unversehrt)

Inkonsistenz

bezeichnet einen Zustand, in dem zwei Dinge, die beide jeweils als gültig angesehen werden, nicht miteinander vereinbar sind. Die Konsistenz ist der gegensätzliche Begriff dazu.

Als inkonsistent gilt / gelten

- in der Soziologie z.B. Schüler, wenn sie nach dem einem Merkmal (ihrer Ausbildung) hoch, nach dem anderen Merkmal (ihrem Vermögen) niedrig einzuschätzen sind.

- in der Informatik bedeutet die Inkonsistenz von Daten die Widersprüchlichkeit zwischen den Informationen und deren Beziehungen.

So können in einer Datenbank beispielsweise Verknüpfungen zwischen Tabelleneinträgen nicht mehr eindeutig sein, weil der Verbindungsschlüssel auf keinen oder mehrere Einträge in einer anderen Tabelle verweist (*).

(*) Diese Aussage gilt nur unter Vorbehalt, denn hier pfuscht der Hinweis in der Literatur ein wenig. Ohne die Formulierung weiterer Bedingungen ist diese Aussage so nicht korrekt. Eine Adresse ist auch dann gültig, wenn sie keinen oder mehrere Ansprechpartner in ihrer Partnertabelle referenziert. Für den Partnereintrag hingegen gilt die Aussage ohne Einschränkung. Existiert ein Partnereintrag ohne entsprechenden Verbindungsschlüssel in der Adressentabelle, so ist er in jedem Fall inkonsistent. (Fehlende Referenz = Datenleiche).

Inkonsistente Daten

Eine logische Übereinstimmung von Dateninhalten über mehrere Relationen (Tabellen) hinweg ist meist schwierig zu realisieren. Bei jeder Änderung eines Dateninhalts, der auf mehrere Tabellen verteilt ist, müssten alle Tabellen synchronisiert werden. Änderungen müssen in den verschiedenen Dateien aufeinander abgestimmt erfolgen.

Hierzu bieten die meisten professionellen Datenbanken entsprechende Reaktionsmöglichkeiten und deren Properties wie beispielsweise die Änderungs- und Löschweitergabe (CASCADE) an.

Beispiel

In einem Unternehmen werden Änderungen an Konstruktionszeichnungen mit Datum und Mitarbeiter-Namenszeichen dokumentiert. Das Zeichen "RZ" steht für den Mitarbeiter Reiner Zufall. Dieser geht nun in Rente, die Mitarbeiterdaten werden samt Handzeichen nach einer angemessenen Wartefrist aus dem System entfernt.

In naher Zukunft beginnt eine neue Mitarbeiterin ihre Tätigkeit. Die Dame erhält als Namenskürzel "RZ" für Rote Zora.

Versäumt man nun eine Datenkorrektur der 'alten' Zeichnungen oder vergisst es, eine neue Gültigkeitsregel hinzuzufügen (z.B. Gültig ab/bis), so kommen bei der Abfrage der Zeichnungs-änderungen, durchgeführt von Frau Zora, auch die alten Konstruktionen von Herrn Zufall.

Ein klassisches Beispiel für Daten-Inkonsistenz.

Integrität

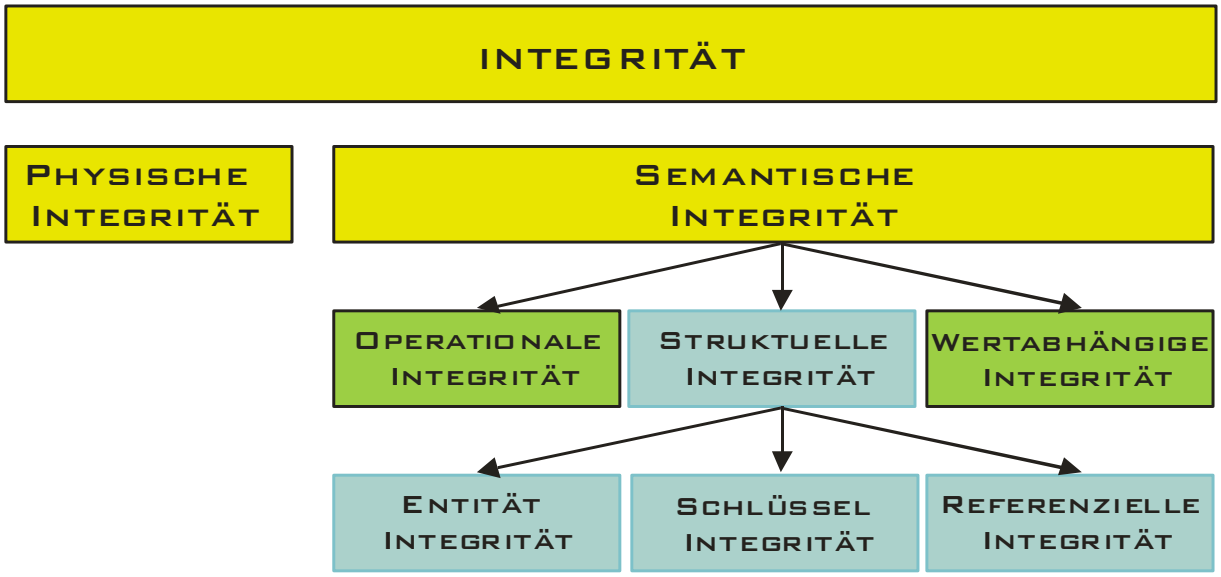
Integrität der Datenbank bedeutet die Gewährleistung einer korrekten und widerspruchsfreien Speicherung von Daten, deren Korrektheit und Vollständigkeit und die Vermeidung aller möglichen Störungen beziehungsweise eine Reaktion zur Herstellung eines integeren Zustandes auf diese Störungen.

Datenbankinterne Reaktion zur Herstellung eines integeren Zustandes sind Funktionen und Beziehungen wie TRIGGER und CASCADE, die unabhängig von externer Programmierung auf Datenänderungen reagieren.

Externe Programme stellen die Vollständigkeit beziehungsweise das Vermeiden 'Halb'-ausgefüllten Datensätzen oft mittels 'Pflichtfelder' her, die vor einer erlaubten Speicherung ausgefüllt sein müssen.

Ein anderer Weg sind Gültigkeitsregeln, Ereignisabfragen, Trigger oder Constraints in Datenbanksystemen. Vor einer Änderung (UPDATE) werden Datenfelder und deren Inhalte geprüft. So könnte beispielsweise ein Datum geprüft werden, ob es innerhalb / außerhalb eines erlaubten Zeitfensters liegt.

Die Integrität eines Datenfeldes wird auch als Wertebereichsintegrität bezeichnet.
Mit (In)konsistenz hat dies nichts zu tun.



Physische Integrität

Behandelt alle Aspekte der physikalischen Speicherung von Daten. Sie berücksichtigt Hard- und Softwarefehler. Sie behandelt die Probleme des gleichzeitigen Zugriffs auf Daten.

Semantische Integrität

Die Integrität auf Datensatzebene, oft auch als Datenintegrität bezeichnet, bestimmt die Korrektheit einer zusammengehörigen Information, eines zusammengehörigen Datensatzes oder Tabellenzeile.

Auch wenn die Datenfelder [Anrede] und [Geschlecht] jeweils für sich korrekt und innerhalb erlaubter Grenzen und Wertebereiche ausgefüllt sind, so stellen die Kombinationen [Herr + Weiblich] sowie [Frau + Männlich] fehlerhafte Dateninhalte dar.

Mit anderen Worten: die semantische Integrität (Semantik: Bedeutungslehre) bezeichnet die Regeln, die durch die Interpretation der in der Datenbank enthaltenen Informationen durch den Benutzer festgelegt werden.

Operationale Integrität

Vermeidung von Fehlern durch gleichzeitigen Zugriff mehrerer Benutzer.
Die operationale Integrität beschreibt auch die Kontrolle von (parallelen) Transaktionen.

Eine Transaktion ist eine Folge von logisch zusammengehörigen Datenmanipulationen (Anweisungen in SQL / DML). Sie fasst alle für eine aus Sicht der Anwendung elementare (atomare) Operation notwendigen Datenmanipulationen zusammen.

Die Integrität eines Datenbestands ist nur gewährleistet, wenn Transaktionen vollständig ausgeführt werden. Transaktionen, die z.B. wegen eines Systemfehlers die nur teilweise abgearbeitet wurden, müssen vollständig rückgängig gemacht werden (COMMIT / ROLLBACK).

Strukturelle Integrität

Die strukturellen Integritätsbedingungen sind vom Datenmodell abhängig. Im relationalen Datenmodelle werden Referentielle-, Schlüssel- und Entität-Integrität unterschieden:

Referentielle Integrität

(In)Konsistenz ist gleich (In)Korrektheit in der referenziellen Integrität.

Unter der referenziellen Integrität versteht man die Integrität auf Beziehungsebene. Sie befasst sich mit der Korrektheit zwischen Attributen von Relationen und der Erhaltung der Eindeutigkeit ihrer Schlüssel. Referenzen dürfen beispielsweise nicht ins Leere zeigen.

Sie ist kein Maß für die Korrektheit oder Vollständigkeit der darin enthaltenen Daten.

Die referentielle Integrität gewährleistet, dass in einer Detail-(Child)-Tabelle die nötigen Fremdschlüssel nur solche Werte annehmen können, wie sie in der betreffenden Master-(Parent)-Tabelle der Beziehung bereits existent sind.

Die referentielle Integrität gewährleistet, dass alle Änderungen eines Datenbankeintrages an alle mit dieser Information in Beziehung stehende Relationen weitergegeben werden.

In allen höheren Datenbankensystemen kann man Reaktionen (TRIGGER) auf die entsprechenden Ereignisse (wie z.B. das Einfügen, Ändern, Löschen) einstellen und aktivieren.

Wird in der Mastertabelle ein Datensatz gelöscht, so werden die in den referentiell verbundenen Tabellen hinterlegten Löschregeln automatisch aktiviert. (CASCADE).

Ob die betroffenen Detail-Datensätze dann ebenfalls gelöscht werden oder nur einen Standardwert annehmen, wird vom Datenbank-Entwickler vorher festgelegt. Diese Überlegungen sind einzeln für alle Relationen durchzuführen. Selten kann man diese Regeln blind anwenden.

Wird beispielsweise ein Lieferant gelöscht, so muss eine Löschweitergabe an die Artikelrelation vorher prüfen ob Bestände vorhanden sind. Ein Löschen aller Artikel dieses Lieferanten darf nicht erfolgen, solange noch Lagerbestände vorhanden sind.

Schlüsselintegrität

Schlüsselintegrität bedeutet, dass innerhalb einer Relation nicht zwei Datensätze mit demselben Schlüssel zugelassen sind. Wird die Schlüsselintegrität verletzt, so ist ein eindeutiger Zugriff auf einzelne Entities nicht mehr gewährleistet.

In relationalen Datenbanksystemen wird zur Wahrung einer Schlüsselintegrität das gewünschte Schlüsselfeld als "PRIMARY KEY"- Feld charakterisiert und somit mit Constraints belegt.

Entität Integrität

(Entität: Neu-latein. *entitas* von latein. *ens* Seiendes, Ding, Existierendes)

Integrität der Datenobjekte (Entity). Sie legt fest, ob die Grenzen des Attributtyps eingehalten werden Alpha- / numerisch, Zahlenbereiche, Datumformate. Ist ein Attribut auf die Größe von 30 Zeichen beschränkt, so dürfen logischerweise nicht mehr Zeichen pro Element gespeichert werden. Andere Regeln sind EMPTY und NULL. Beispielsweise darf keine Komponente des Primärschlüssels den Wert NULL haben.

Constraints

(deutsch „Zwangsbedingungen“)

Constraints definieren Bedingungen die beim Einfügen, Ändern und Löschen von Datensätzen in der Datenbank erfüllt werden müssen. Neben Bedingungen auf Datenfeld-Ebene (Entität) wie ‚NOT NULL‘ können auch Bedingungen auf Tabellen-Ebene (Relationen) wie ‚UNIQUE‘ definiert werden.

Beispiele für Constraints:

- **PRIMARY KEY** der Skalar muss einzigartig sein und kann nicht NULL sein.
  Ein Schlüsselattribut ist nicht auf ein Attribut beschränkt, es darf aber keine (auch nicht in Kombinationen) Duplikate aufweisen.
- **FOREIGN KEY** der Skalar muss auf Referentielle Integrität geprüft werden.
  Ein Fremdschlüssel darf nur existierende Werte der Mastertabelle annehmen.
- **NOT NULL** der Skalar kann nicht NULL (Leer) sein.
- **UNIQUE** der Skalar muss innerhalb des Attributes einzigartig sein.
  Beispielsweise eine laufende Nummer. Keine zwei Attribute können den gleichen Wert aufweisen

Redundanz

(latein. redundare „im Überfluss vorhanden sein“)

bezeichnet allgemein das zeitgleiche, mehrfache Vorhandensein funktions-, inhalts- oder wesensgleicher Objekte.

Als redundant gilt / gelten

- in der Geräte- und Anlagentechnik das Vorhandensein funktional gleicher oder vergleichbarer Ressourcen eines technischen Systems, wenn diese bei einem störungsfreien Betrieb im Normalfall nicht benötigt werden.

- in der Informationstheorie und Informatik ist Redundanz die Differenz zwischen Nachrichtengehalt und Informationsdichte (Entropie)

- in der Literaturwissenschaft das mehrfache Vorhandensein ein und derselben Information

Beispiel

Beispiele für Redundanzen sind ein zweiter Server mit gleichen Aufgaben im gleichen Netzwerk oder mehrere Funkgeräte im gleichen Flugzeug oder der zweite Lungenautomat des Tauchers.

In jedem besseren Server sind heute RAID-Festplatten-Systeme (RAID - *Redundant Array of independent Disks).* Während die meisten verwendeten Techniken und Datenanwendungen darauf abzielen, Redundanzen, d.h. das Vorkommen doppelter Daten zu vermeiden, werden bei RAID-Systeme redundante Informationen gezielt erzeugt, damit beim Ausfall einzelner Komponenten das RAID als Ganzes seine Funktionalität behält. d.h. auch wenn eine Festplatte aus dem RAID-Verbund komplett ausfällt, läuft der Server störungsfrei weiter.

Redundanz <> Datensicherung

Eine externe Datensicherung (Backup) auf Magnetband, CD, DVD usw. ist per Definition keine Redundanz, da sie niemals zeitgleich zur Verfügung steht. Sie kann daher den Ausfall des Systems nicht abfangen. Eine Datensicherung ist eine (veraltete) Momentaufnahme am Tag X / um Y:YY Uhr, die eine Sekunde später schon nicht mehr die Wiederherstellung der Originaldaten ermöglicht, da üblicherweise am System weitergearbeitet wird.

Den Reserve-Lungenautomat, den der Taucher zuhause im Schrank hat, erhöht die Sicherheit seines aktuellen Tauchganges nicht messbar.

Eine gespiegelte Festplatte die parallel im System läuft erfüllt dagegen alle Anforderungen an Redundanz. Die Daten sind mehrfach und zeitgleich vorhanden und können beim Ausfall der ersten Partition sofort deren Prozesse steuern. Das System würde ohne den Festplattenspiegel ebenfalls laufen, was dem Vorhandensein mehrfacher („überflüssiger“) Ressourcen entspricht.

Gespiegelte Festplatten erhöhen die Ausfallsicherheit des Systems, erfüllen alleine jedoch nicht die Anforderungen an eine Datensicherung. Wenn die Hardware abfackelt ist der Spiegel ebenfalls Schrott. Eine Datensicherung dient auch dem Wiederherstellen versehentlich gelöschter Daten. Nicht selten werden nur aus diesem Grund Sicherungen in Tages- oder Wochenintervallen durchgeführt. Auf dem Plattenspiegel wäre die betreffende Information aber im gleichen Moment wie auf der Primärpartition gelöscht und eine Wiederherstellung somit nicht möglich.

Redundanz <> Duplikate

Duplikate sind Redundanzen, wenn Datensätze wie beispielsweise Adressen oder Artikel versehentlich doppelt erfasst wurden. Diese Duplikate sind eher unkritisch. Man kann sie bei der Datenpflege leicht abfragen und entfernen.

Die systembedingten Redundanzen lassen sich dagegen nicht entfernen.

Beispiel

In einer Datenbank für Fakturierung werden Rechnungen erfasst. Pro Rechnung werden neben den einzelnen Artikeln auch die Kundendaten z.B. deren Adressen gespeichert.
Mit anderen Worten, ein Unternehmen, das 3000 Rechnungen im Jahr erfasst, hat die Adresse ebenfalls 3000 Mal redundant gespeichert.

Nun könnten man erwidern: Dann speichert man die Adresse halt in einer separaten Relation und verknüpft die Informationen per Schlüsseleintrag miteinander (Normalisierung). Wenn es dann so einfach wäre, könnte man sich die teure Ausbildung der Administratoren ersparen.

Per Gesetz sind Unternehmen dazu verpflichtet ihre Daten mindestens 10 Jahre für eine Steuerprüfung zu archivieren. In diesen 10 Jahren firmieren Kunden teilweise mehrere Male um. Ein Beispiel aus der direkten Nachbarschaft: Aus der Robert Bosch GmbH wird die Rexroth Bosch Group usw. Die Rechnungsdaten 'alter', bereits gedruckter Rechnungen dürfen sich nachträglich selbstverständlich nicht mehr ändern.

Die Datenbank-Administratoren haben nun die Wahl zwischen Pest und Cholera.

Lösung 1 - Die aktuellen Adressdaten werden in den Rechnungen gespeichert. Nun kann sich die Adresse ändern wie sie will. Auch wenn man eine 'alte' Rechnung aufruft enthält sie die ehemals korrekte Kundeninformation.

Problem 1 - Die Kundeninformation existiert redundant gleich der Anzahl der Rechnungen. Nicht selten erfassen Unternehmen mit Angeboten, Bestellungen, Lieferscheinen und Rechnungen 5000 Belege pro Monat. Das sind 60.000 Belege pro Jahr. In 10 Jahren sind dann die Adressdaten locker 1/2-Million-Mal redundant im System. Dies mit allen (Neben)-Wirkungen auf Ressourcen und Geschwindigkeit.

Lösung 2 - In der Rechnungsdatenbank wird per Schlüssel auf die Adressdaten zugegriffen.

Problem 2 - Es müssten dann für den gleichen Kunden mehrere Adressen verfügbar sein. Entweder man erweitert die bestehenden Kundendaten mit einer per Datum gesteuerten (Adressen)-Historie (z.B. bis 15.4.20xx Bosch, ab 16.4.20xx Rexroth usw.) oder man legt eine komplett neue Adresse für den gleichen Kunden an. In beiden Fällen hat man dann Adressduplikate und die Gefahr der Inkonsistenz, da einem Kunden dann mehrere Adressenschlüssel zugeordnet sind. Ein Problem beider Lösungen sind die bereits bestehenden Verknüpfungen mit dem bestehenden, 'alten' Adressenschlüssel. Existierende Abnahmeverträge müssten überarbeitet werden. Die 'neue' Adresse wird im System auch nicht als 'alte' erkannt. Bei der Abfrage nach Kundenumsatz werden getrennte Umsätze dargestellt. Mengenrabatte müssen neu berechnet werden, da die bisherigen Umsätze nicht automatisch dem 'neuen' Kunden zugeordnet werden. usw.

Problem X - Es gäbe noch mehrere Beispiele für die Kombination von Speicherung und / oder Verknüpfung. Systembedingten Redundanzen sind jedoch nicht eliminierbar.

Lösung X - Es gibt keine. Je nach Bedarf, Anzahl der Belege pro Jahr, das Vorhandensein von Abrufaufträgen und / oder Liefervereinbarungen und nicht zuletzt auch nach Kundenwunsch wird die eine oder andere Lösung als kleineres Übel angesehen.

Vorlesung: Datenbanken

Kapitel (D1): Entity Relationship Modell

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Entity-Relationship-Modell

(Entität: Neu-latein. *entitas* von latein. *ens* Seiendes, Ding, Existierendes)
(deutsch: Gegenstands- oder Objekt-Beziehungs-Modell)

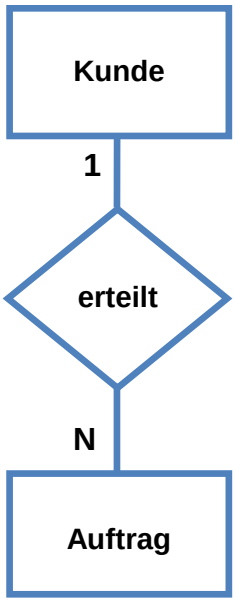
Das Entity-Relationship-Modell, kurz ER-Modell oder ERM dient dazu, im Rahmen der semantischen Datenmodellierung (Semantik: Bedeutungslehre) einen Ausschnitt der realen Welt zu beschreiben.

Um eine komplexe Applikation für ein relationales Datenbanksystem zu realisieren, muss die benötigte Datenstruktur genau analysiert werden. Da ein Anwendungsprogramm ebenfalls einen Ausschnitt aus der realen Welt simulieren soll, suchte man eine Möglichkeit, die reale Welt in einem solchen Modell abzubilden. Dazu dient das Entity-Relationship-Modell.

Der deutsche Begriff "Objekt-Beziehungs-Modell" kann unter Umständen zu Verwechslungen mit den Begriffen der objektorientierten Programmierung führen, ist aber hier nicht gemeint.

Das Verfahren nach Chen[1]

Das Entity-Relationship-Modell (ERM), war das erste Beschreibungsmittel zur Erstellung von konzeptionellen Schemata. Es ist aufgrund seine Einfachheit bewährt und bis heute im Einsatz.



(ER-Notation)

Das ER-Modell besteht aus einer Grafik und einer Beschreibung der darin verwendeten Elemente, wobei Dateninhalte, deren Bedeutung (Semantik) und Datenstrukturen dargestellt werden.

Ein ER-Modell dient sowohl in der konzeptionellen Phase der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern (dabei wird nur das Was, also die Sachlogik, und nicht das Wie, also die Technik, behandelt), als auch in der Implementierungsphase als Grundlage für das Design der überwiegend relationalen Datenbank.

Entities werden durch Rechtecke, Beziehungen (Relationships) durch Rauten dargestellt.

Da beispielsweise zu jedem Kunden eine Vielzahl von Aufträgen gehören kann, jeder Auftrag aber eindeutig einem Kunden zugeordnet wird, kann man diese Beziehung als 1:N Beziehung charakterisieren. Die Mengenverhältnisse werden auch als Kardinalität der Beziehung bezeichnet.

Natürlich kann es in der Realität auch Kunden geben, die bislang noch keinen Auftrag erteilten. Hier hilft es zusätzliche Regeln zu definieren.

- Es kann Kunden ohne Auftrag geben.
- Es gibt aber keine Aufträge ohne Kunden.

Als Lösung und mögliche Darstellung bietet sich hier die sogenannte (min/max) Notation an. Anstelle der **[1]** schreibt man **(0,N) = min**destens **0** Aufträge / **max**imal beliebig viele Aufträge. In unserem Beispiel wäre dann noch das **[N]** durch **(1,1)** zu ersetzen, was bedeutet, dass ein Auftrag exakt zu einem Kunden gehört = mindestens zu einem und höchstens zu einem.

[1] Chen, Peter: „The entity relationship model towards a unified view of data“ ACM Transactions on Database Systems, 1976

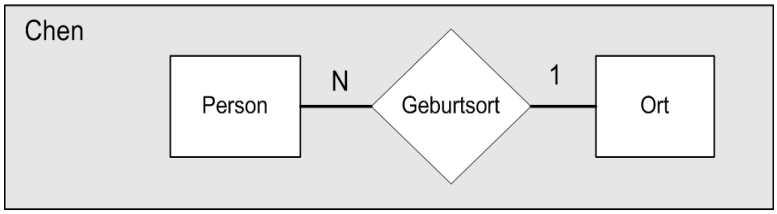
Der Einsatz von ER-Modellen ist der De-facto-Standard für die Datenmodellierung, auch wenn es unterschiedliche grafische Darstellungsformen gibt. Das erste ER-Modell wurde 1976 von Peter Chen in seiner Veröffentlichung: „The Entity-Relationship Model“ vorgestellt. Danach gab es mehrere Weiterentwicklungen, so Ende der 1980er Jahre durch Wong und Katz.

Überblick über ERM-Notationen:

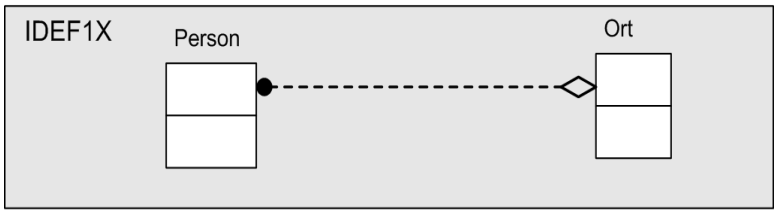
Alle nebenstehenden Notationen drücken auf ihre Art den folgenden Sachverhalt aus:

- Eine Person ist in maximal einem Ort geboren.
- Ein Ort ist Geburtsort von beliebig vielen Personen.
- Ein Ort kann ein Geburtsort sein, muss es aber nicht sein.

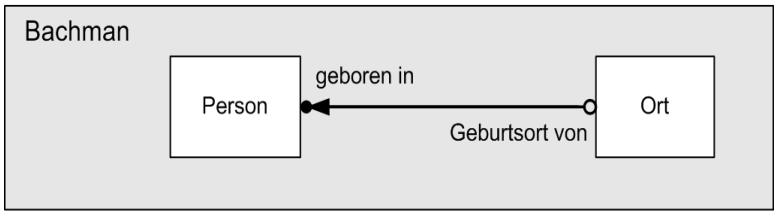
- Chen-Notation von Peter Chen, dem Entwickler der ER-Diagramme, 1976.



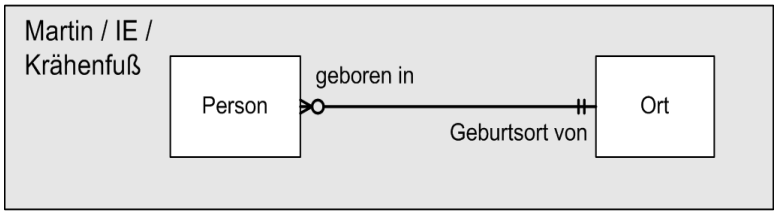
- IDEF1X als langjähriger De-Facto-Standard bei U.S. amerikanischen Behörden.



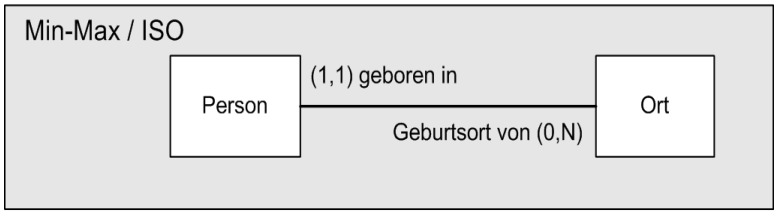
- Bachman-Notation von Charles Bachman als weit verbreitete Diagramm-Sprache.



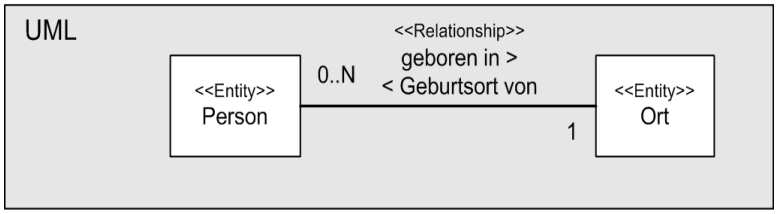
- Martin-Notation (Krähenfuß-Notation) als weit verbreitete Diagramm-Sprache (Information Engineering).



- (min, max)-Notation von Jean-Raymond Abrial.



- UML als Standard, den selbst ISO in eigenen Normen als Ersatz für ER-Diagramme verwendet.



Entitätsmenge / Entitätstyp

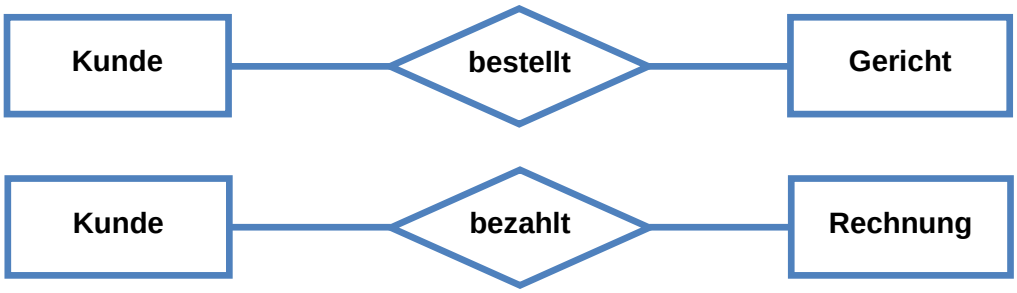
Eine Entität (Entity) ist ein Objekt der Realität. Beispielsweise ein konkreter Kunde (*Herr Reiner Zufall)*, eine erbrachte Leistung (*Schnitzel mit Bratkartoffeln*) und die Gegenleistung (*Rechnung / Bon R2331*).

Entitäten werden zu Entitätsmengen oder Entitätstypen (Entitytyp) zusammengefasst und als Rechtecke dargestellt.

Beziehungen / Beziehungstyp

Zwischen den Entitäten einer Entitätsmenge besteht eine Beziehung. Im Beispiel: Reiner Zufall bestellt Schnitzel mit Bratkartoffeln und Reiner Zufall bezahlt die Rechnung Nummer R2331.

Diese Beziehungen (Relationships) fasst man unter Beziehungstypen zusammen und stellt sie als Rauten dar.



Kardinalität

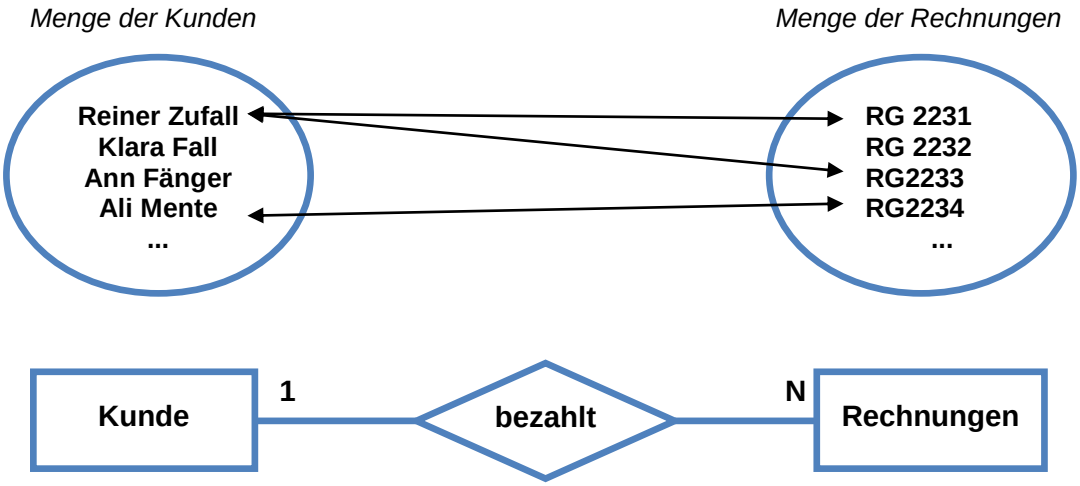
Die Kardinalität gibt an, wie viele Entitäten aus jeder Entitätsmenge an einer Beziehung beteiligt sein können. Sie wird an der Verbindungslinie je nach Notation durch eine Zahl oder ein Buchstaben oder eine Kombination aus Zeichen und/oder Buchstaben dargestellt.

Bei der **1:1 - Beziehung** gibt es zu jeder Entität der einen Menge eine Entsprechung in der anderen Entitätsmenge

Man liest die Kardinalitäten dieses ER-Diagramms wie folgt:

- Ein Staatsbürger hat **genau eine** Steuernummer.
- Eine Steuernummer gehört **genau einem** Staatsbürger.

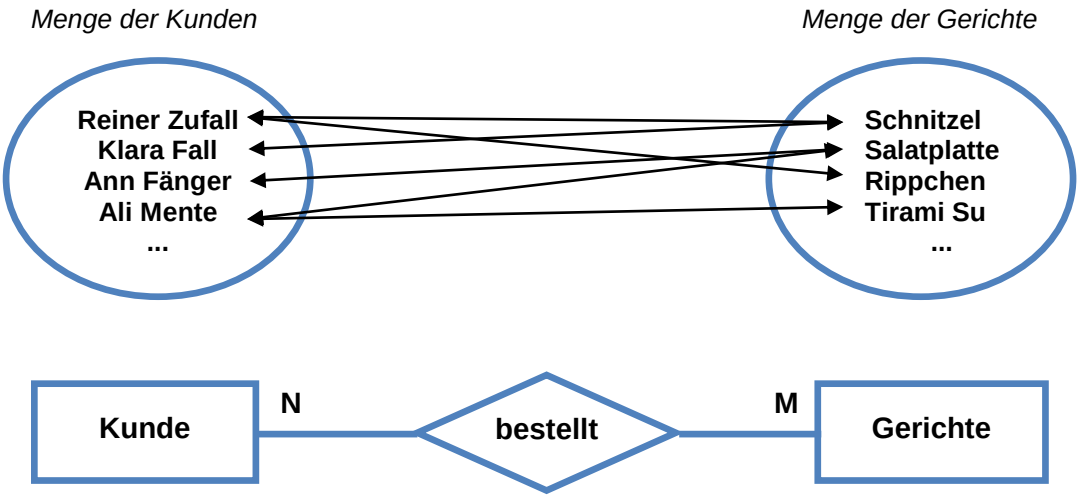
Bei der **1:N - Beziehung** gibt es zu jeder Entität der einen Menge mehrere Entsprechungen in der anderen Entitätsmenge.



Man liest die Kardinalitäten dieses ER-Diagramms wie folgt:

- Ein Kunde kann **mehrere** Rechnungen bezahlen.
- Eine Rechnung wird von **genau** einem Kunden bezahlt.

Bei der **N:M - Beziehung** gibt es zu jeder Entität der einen Menge mehrere Entsprechungen in der anderen Entitätsmenge und umgekehrt.

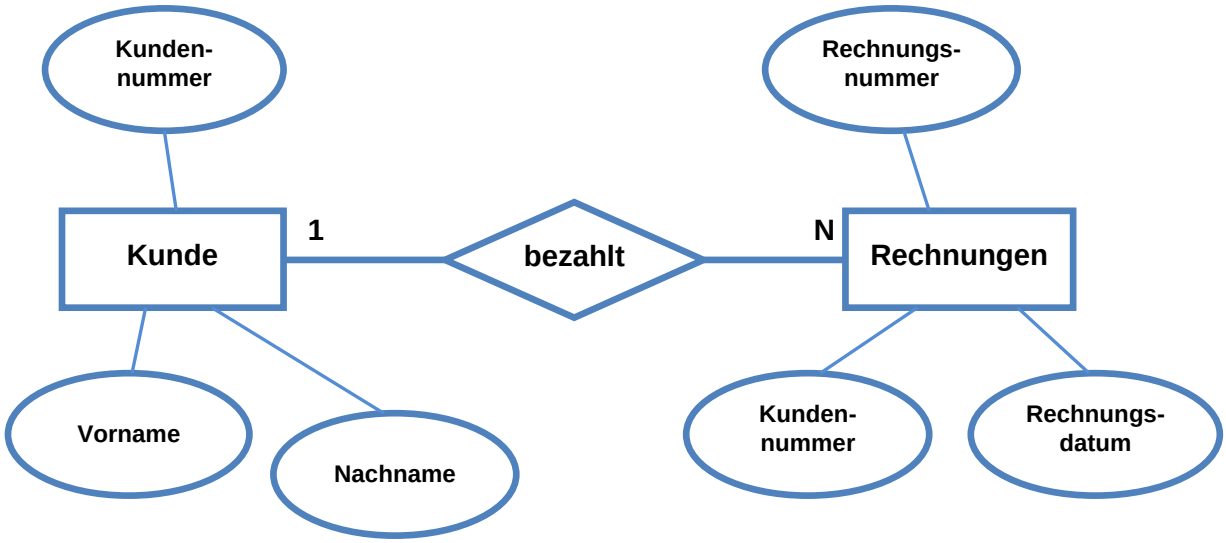


Man liest die Kardinalitäten dieses ER-Diagramms wie folgt:

- Ein Kunde bestellt **mehrere** Gerichte.
- Ein Gericht wird von **mehreren** Kunden bestellt.

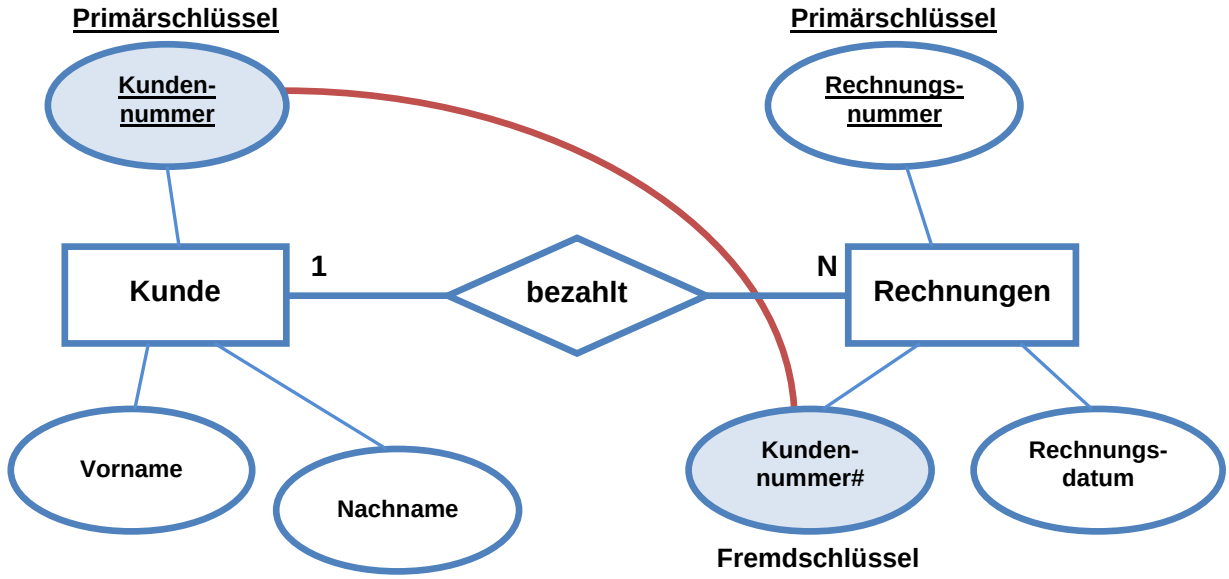
Attribut

Als Attribute werden die Eigenschaften eines Entitäts- oder Beziehungstyps bezeichnet, z.B. sind *Nachname, Vorname* und *Kundennummer* Attribute der Entitätsmenge *Kunde.*
Attribute werden mit Ellipsen dargestellt.



Schlüsselattribut

Das Schlüsselattribut ist das Attribut (oder die Attributkombination) einer Entitätsmenge dessen Wert jedes Element dieser Menge (also jede Entität) eindeutig identifiziert.



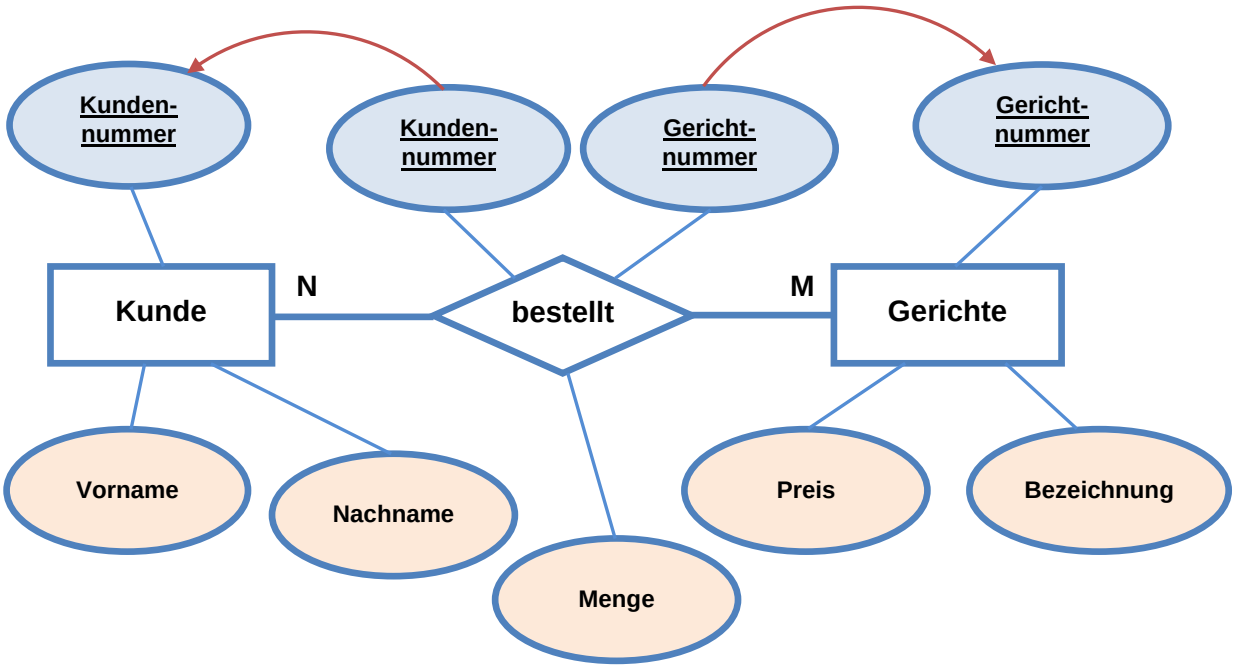
Die identifizierenden Attribute *Kundennummer* bei der Entitätsmenge *Kunde* und *Rechnungsnummer* bei der Entitätsmenge *Rechnung* sind unterstrichen. Man bezeichnet sie auch als **Primärschlüssel.**

Das Attribut *Kundennummer* bei der Entitätsmenge *Rechnung* verweist auf den Primärschlüssel *Kundennummer* in der Relation *Kunde.* Das Attribut *Kundennummer* ist bei der Entitätsmenge *Rechnung* ein **Fremdschlüssel.** Fremdschlüssel werden oft mit einem nachgestellten „#“ - Zeichen gekennzeichnet.

Bei N:M - Beziehungen kann man in der Entitätsmengen keinen Fremdschlüssel unterbringen, weil zu jeder Entität der einen Menge mehrere Entitäten der anderen Menge gehören können. Gleichzeitig darf ein Attribut nur einen Wert beinhalten[2].

Deshalb erhält in diesem Fall der Beziehungstyp auch Attribute.

Zu den Attributen, die dem Beziehungstyp zugeordnet werden, gehören die Primärschlüssel der verknüpften Entitätsmengen. Dazu kommen noch andere Attribute, die die zwei verknüpften Entitäten näher beschreiben.



Kombinationsschlüssel

In der oben gezeigten Notation gehören zu dem Beziehungstyp *bestellt* die Attribute *Kunden-nummer, Gerichtnummer* und *Menge.*

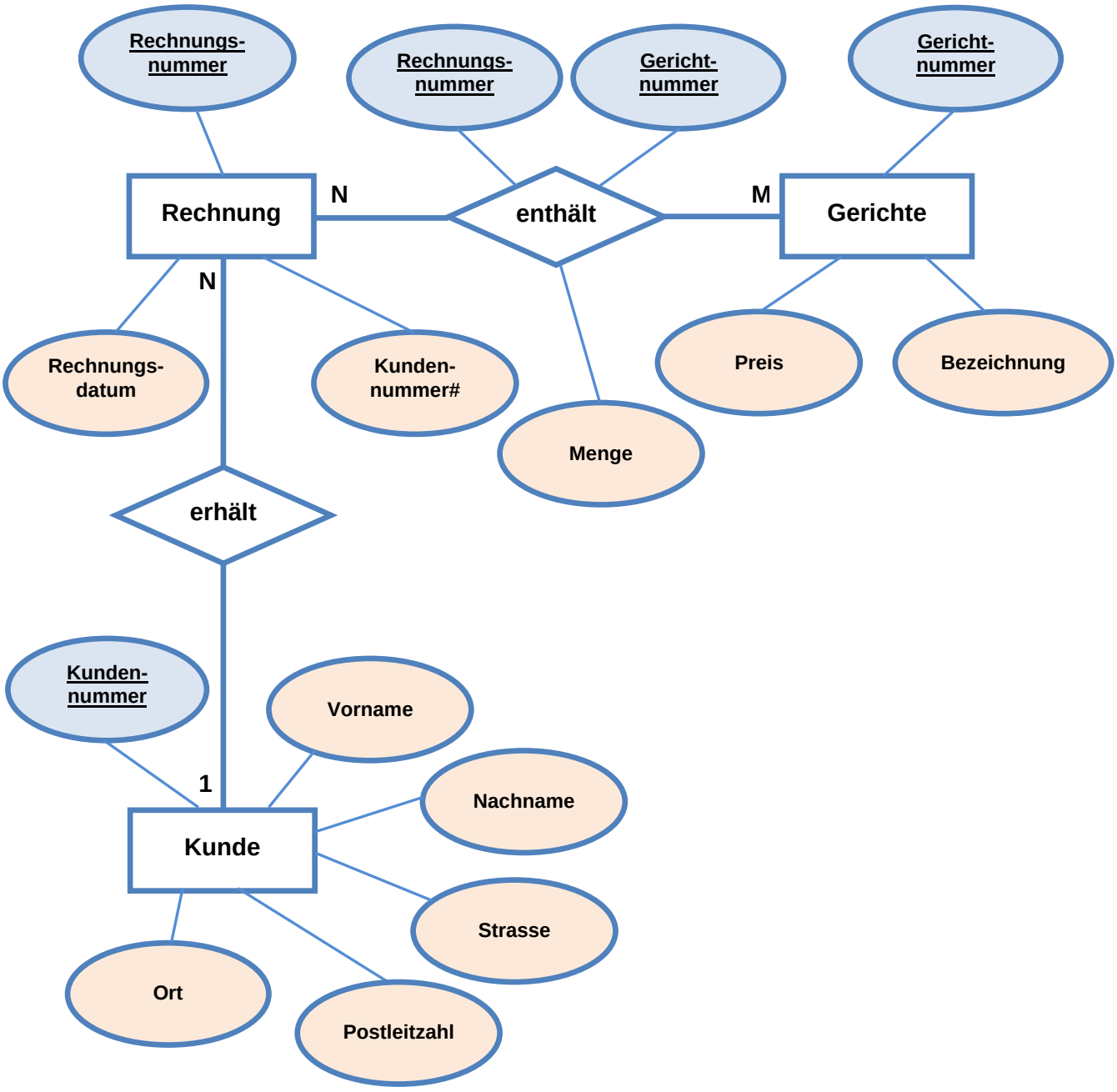
Hierbei identifizieren die Attribute *Kundennummer* und *Gerichtnummer* gemeinsam jede einzelne Beziehung. Man spricht in diesem Fall von einem Kombinationsschlüssel**.**

Die einzelnen Teile eines Kombinationsschlüssels sind gleichzeitig auch Fremdschlüssel, weil sie in einer anderen Entitätsmenge Primärschlüssel sind.

[2] Siehe auch 1. Normalform „Attribute sind Atomar“

Entity-Relationship-Modell am Beispiel einer einfachen Fakturierung.

Relation: **Kunde**: Kundennummer, Vorname, Nachname, Strasse, Postleitzahl, Ort
Relation: **Gerichte**: Gerichtnummer, Bezeichnung, Preis
Relation: **Rechnung**: Rechnungsnummer, Rechnungsdatum, Kundennummer#
Relation: **Enthält**: Rechnungsnummer, Gerichtnummer, Menge



Vorlesung: Datenbanken

Kapitel (D2): Entity Relationship Modell

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Entity-Relationship-Modell / ÜBUNG

Erläuterungen

Die Notation für Entity-Relationship-Diagramme hat sich in den zurückliegenden Jahren stark verändert. Neben der ursprünglichen Darstellungsform, die auf P.P.Chen (1976) zurückgeht, unterstützen Softwaretools vielfältige Formen, bis hin zur Krähenfuß-Notation.

Aus didaktischen Gründen soll hier die einfache Notation nach Chen eingesetzt werden. Diese ist übersichtlich und einfach auch von Hand zu erstellen.

Die Entitätsmengen werden durch Substantive, die Beziehungen durch Verben beschrieben. Die Kardinalitäten sind in Min-Max-Schreibweise anzugeben. Attribute werden nicht im ER-Diagramm eingetragen.

Aufgabe - Verwaltung eines Fahrradhandels

Ein Fahrradhändler will für seinen Handel eine Datenbank entwerfen lassen. Bei einem Gespräch mit Ihnen als Datenbankspezialist erhalten Sie folgende Informationen:

Der Fahrradhändler hat üblicherweise einen gewissen Bestand an Fahrrädern im Laden stehen, damit die Kunden sich informieren können. Diese Fahrräder stehen natürlich zum Verkauf.

Von manchen sich gut verkaufenden Modellen hat der Händler sogar mehrere Einzelstücke am Lager. Der Fahrradhändler bezieht seine Modelle von verschiedenen Herstellern. Üblicherweise verkauft er alle Modelle eines Herstellers. Zur Kontaktaufnahme mit dem Hersteller, ist dessen Postanschrift und Telefonnummer in die Datenbank zu übernehmen.

Mit den Herstellern hat er vereinbart, dass er eine Datei erhält, in der für die einzelnen Modelle sein jeweiliger Einkaufspreis enthalten ist. Dieser EK-Preis ist auch in der Datenbank zu speichern. Der Fahrradhändler hat bei den Herstellern jeweils eine eigene Kundennummer.

Bei dem Gespräch ergeben sich drei klassische Geschäftsfälle:

1. Ein Kunde kauft ein am Lager befindliches Fahrrad. Dieses Fahrrad hat eine eindeutige Rahmennummer und einen Verkaufspreis. Sofern der Kunde noch nicht in der Datenbank enthalten ist, wird dieser vom Fahrradhändler eingetragen. Typischerweise enthält die bisherige Kundenkartei des Fahrradhändlers auch Kunden, die noch nie ein Fahrrad bei ihm erworben haben. Anschließend wird der Verkauf des Fahrrades mit dem VK-Preis und Datum in die Datenbank aufgenommen.

2. Ein Kunde kauft ein Fahrrad anhand einer Modellbezeichnung. Dieses Modell muss der Fahrradhändler dann beim Hersteller bestellen. Dazu erstellt der Fahrradhändler eine Vorbestellung des gewünschten Modells eines Herstellers mit dem ausgehandelten VK-Preis und einem voraussichtlichen Liefertermin. Wenn das Fahrrad angeliefert wird, wird es in den Lagerbestand überführt. Der Kunde kann dann das Fahrrad abholen. Insofern sind die Daten wie unter 1. einzutragen.

3. Ein Kunde bringt ein Fahrrad zur Reparatur. Dieses Fahrrad muss nicht bei dem Händler gekauft worden sein. Allerdings repariert der Fahrradhändler lediglich Fahrräder von Herstellern, mit denen er eine Geschäftsbeziehung hat. Bei solch einem Auftrag sind die Art der Reparatur, das Datum und der Preis zu speichern. An einem Fahrrad können mehrere Reparaturen an einem Termin erforderlich werden. Auch kann ein Fahrrad mehrfach in Reparatur gewesen sein.

Aufgabe:

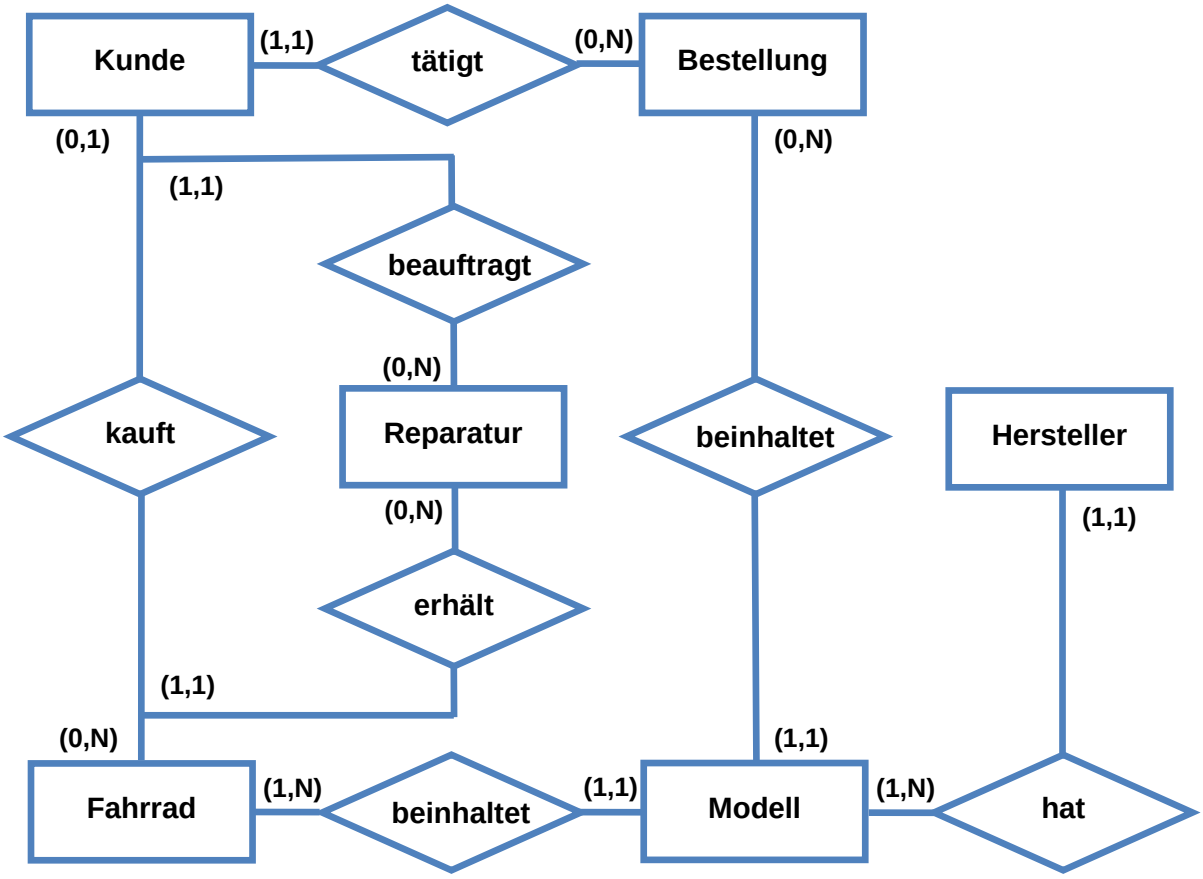
Verschaffen Sie sich anhand der obigen Beschreibung einen Überblick über die zu verwaltenden Daten.

Bilden Sie aus den Angaben geeignete Entitätsklassen.

Erstellen Sie für die Problembeschreibung das Entity-Relationship-Modell.

Zeichnen Sie das Diagramm (ohne Attribute), bestehend aus den Entitätsklassen mit ihren Beziehungen und Kardinalitäten in der Form [Min,Max]. Der Beziehungsname soll kurz und präzise die Beziehung zwischen den Entitätsklassen wiedergeben.

Lösungsvorschlag:



Lösungsvorschlag Relationen

```
Kunde      (ID,Postanschrift)
Fahrrad    (RahmenNr,ModellID#)
Verkauf    (ID,KundeID#,RahmenNr#,VkPreis,Verkaufsdatum)
Modell     (ID,Bezeichnung,EKPreis,HerstellerID#)
Hersteller (ID,Postanschrift,Telefonnr,KundenNr)
Bestellung (ID,KundeID#,ModellID#,VKPreis,Liefertermin)
Reparatur  (ID,KundeID#,RahmenNr#,Reparaturart,Datum,Preis)
```

Überprüfen des Entwurfes durch Normalisieren

Die erste Normalform (1NF)
„Alle Attribute sind atomar“

Das Attribut Postanschrift ist, sofern noch nicht erfolgt, in seine Bestandteile aufzulösen. Es ergibt sich nachfolgende Änderung der Tabellenstruktur:

```
Kunde      (ID,Name,Vorname,Strasse,Plz,Ort)
Fahrrad    (RahmenNr,ModellID#)
Verkauf    (ID,KundeID#,RahmenNr#,VkPreis,Verkaufsdatum)
Modell     (ID,Bezeichnung,EKPreis,HerstellerID#)
Hersteller (ID,Name,Vorname,Strasse,Plz,Ort,Telefonnr,KundenNr)
Bestellung (ID,KundeID#,ModellID#,VKPreis,Liefertermin)
Reparatur  (ID,KundeID#,RahmenNr#,Reparaturart,Datum,Preis)
```

Die zweite Normalform (2NF)
„Jedes Nicht-Schlüssel-Attribut ist voll funktional vom Primärschlüssel abhängig“

Dies hier bereits gewährleistet. Jede Relation besitzt schon einen Primärschlüssel (ID, RahmenNr). Ein zusammengesetzter Schlüssel wird im Beispiel nicht verwendet.

Die dritte Normalform (3NF)
„Keine transitive Abhängigkeit eines Nicht-Schlüssel-Attributes vom Primärschlüssel“

Dies ist hier bei den Tabellen Kunde und Hersteller noch nicht erfüllt, da der Ort transitiv[1] über die Plz vom Primärschlüssel abhängt. Die theoretische Lösung ist das Auslagern der Postleitzahlen und Orte in eine eigene Tabelle und das Hinzufügen des dort erforderlichen Primärschlüssels als Fremdschlüssel in die Tabellen Kunde und Hersteller.
Somit ergibt sich folgende Tabellenstruktur

```
Kunde      (ID,Name,Vorname,Strasse,PlzID#)
Fahrrad    (RahmenNr,ModellID#)
Verkauf    (ID,KundeID#,RahmenNr#,VkPreis,Verkaufsdatum)
Modell     (ID,Bezeichnung,EkPreis,HerstellerID#)
Hersteller (ID,Name,Vorname,Strasse,PlzID#,Telefonnr,KundenNr)
Bestellung (ID,KundeID#,ModellID#,VkPreis,Liefertermin)
Reparatur  (ID,KundeID#,RahmenNr#,Reparaturart,Datum,Preis)
Plz        (ID,Plz,Ort)
```

[1] Transitivität, adj. transitiv (latein: transitus ‚Übergang‘)
Ein Nichtschlüsselattribut darf nicht von einer Menge aus Nichtschlüsselattributen abhängig sein.
Ein Nichtschlüsselattribut darf nur direkt von einem Schlüssel (PK) abhängen.

Festlegung der Datentypen und Schlüssel

TABELLE	ATTRIBUT	DATENTYP	SCHLÜSSELTYP	REFERENZTABELLE
KUNDE	ID	Counter	**PK**	
	Name	Text(50)		
	Vorname	Text(50)		
	Strasse	Text(50)		
	PlzID	Integer	**FK**	**PLZ.ID**
FAHRRAD	RahmenNr	Integer	**PK**	
	ModellID	Integer	**FK**	**MODELL.ID**
VERKAUF	ID	Counter	**PK**	
	KundeID	Integer	**FK**	**KUNDE.ID**
	FahrradID	Integer	**FK**	**FAHRRAD.RahmenNr**
	VkPreis	Float		
	Verkaufsdatum	Date		
MODELL	ID	Counter	**PK**	
	Bezeichnung	Text(50)		
	EkPreis	Float		
	HerstellerID	Integer	**FK**	**HERSTELLER.ID**
HERSTELLER	ID	Counter	**PK**	
	Name	Text(50)		
	Vorname	Text(50)		
	Strasse	Text(50)		
	PlzID	Integer	**FK**	**PLZ.ID**
	Telefonnr	Text(30)		
	Kundennr	Text(30)		
BESTELLUNG	ID	Counter	**PK**	
	KundeID	Integer	**FK**	**KUNDE.ID**
	ModellID	Integer	**FK**	**MODELL.ID**
	VkPreis	Float		
	Liefertermin	Date		
REPARATUR	ID	Counter	**PK**	
	KundeID	Integer	**FK**	**KUNDE.ID**
	RahmenNr	Integer	**FK**	**FAHRRAD.RahmenNr**
	Reparaturart	Text(100)		
	Datum	Date		
	Preis	Float		
PLZ	ID	Counter	**PK**	
	Plz	Text(10)		
	Ort	Text(50)		

Vorlesung: Datenbanken

Kapitel (D3): Entity Relationship Modell

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Entity-Relationship-Modell / ÜBUNG II

Aufgabe - Verwaltung von Geschäftsprozessen und Projekten

Im Auftrag eines Unternehmens der IT-Branche soll zur Vereinfachung interner Geschäftsprozesse eine Firmendatenbank entwickelt werden.
Das Unternehmen gliedert sich in einzelne Abteilungen (Beratung, Entwicklung, Marketing etc.). Ein(e) MitarbeiterIn ist einer Abteilung zugehörig. Die Arbeit im Unternehmen ist in Projekten organisiert. Die Mitarbeiter des Unternehmens sind mit einer bestimmten Stundenzahl den einzelnen Projekten zugeordnet.

Im Einzelnen sind folgende Anforderungen zu beachten:

1 Jede Abteilung besitzt eine Bezeichnung.

2 Jede(r) MitarbeiterIn ist charakterisiert durch einen Namen und Anschrift.

3 Jeder Arbeitsvertrag legt eine Funktion (z.B. „Anwendungsentwickler“) und Gehalt eines Mitarbeiters fest.

4 Jedes Projekt besitzt einen Namen und eine eindeutige Projektnummer.

5 Für jeden Mitarbeiter ist ein Arbeitsvertrag gültig, jeder Arbeitsvertrag ist genau für einen Mitarbeiter gültig.

6 Jeder Mitarbeiter ist einer Abteilung unterstellt.

7 Mitarbeiter können gleichzeitig an mehreren Projekten teilnehmen, wobei der Stundenanzahl, den sie für das Projekt arbeiten, erfasst wird.

Erläuterungen

Die Notation für Entity-Relationship-Diagramme hat sich in den zurückliegenden Jahren stark verändert. Neben der ursprünglichen Darstellungsform, die auf P.P.Chen (1976) zurückgeht, unterstützen Softwaretools vielfältige Formen, bis hin zur Krähenfuß-Notation.

Aus didaktischen Gründen soll hier die einfache Notation nach Chen eingesetzt werden. Diese ist übersichtlich und einfach auch von Hand zu erstellen.

Die Entitätsmengen werden durch Substantive, die Beziehungen durch Verben beschrieben. Die Kardinalitäten sind in Min-Max-Schreibweise anzugeben. Attribute werden im ER-Diagramm nicht eingetragen.

Aufgabe:

Verschaffen Sie sich anhand der obigen Beschreibung einen Überblick über die zu verwaltenden Daten.

Bilden Sie aus den Angaben geeignete Entitätsklassen.

Erstellen Sie für die Problembeschreibung das Entity-Relationship-Modell.

Entwerfen Sie für die Problembeschreibung ein geeignetes Entity-Relationship-Modell. Zeichnen Sie das ERM ohne Attribute, bestehend aus den erforderlichen Entitätsklassen und Beziehungen sowie deren Kardinalitäten in der Form "[Min,Max]".

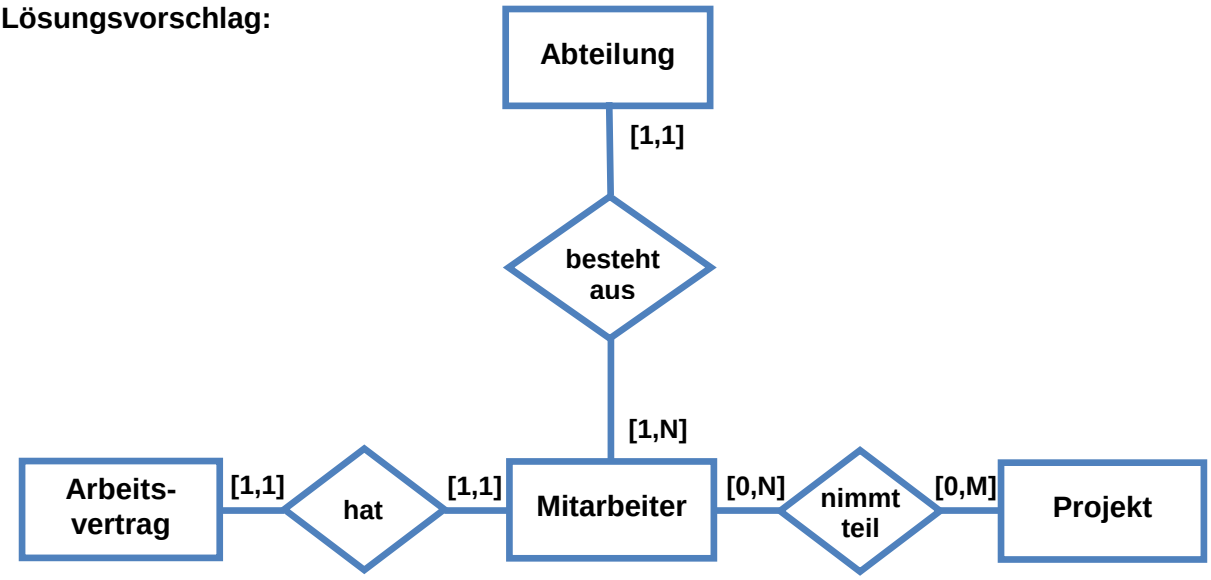
Überführen Sie das ERM in das Relationale Datenbankmodell und normalisieren Sie die entstehenden Relationen bis hin zur dritten Normalform nach Codd. Geben Sie alle entstandenen Relationen in der folgenden Schreibweise an:

Tabelle(PK,Attribute..,FK#...)

Überführen Sie die Relationen in Tabellen und beschreiben Sie diese in folgender Form:

TABELLE | ATTRIBUT | DATENTYP | SCHLÜSSELTYP | REFERENZTABELLE

Lösungsvorschlag:



obligatorische **1:N** –Beziehung zwischen Abteilung und Mitarbeiter
obligatorische **1:1** –Beziehung zwischen Mitarbeiter und Arbeitsvertrag
obligatorische **N:M**–Beziehung zwischen Mitarbeiter und Projekt

Lösungsvorschlag Relationen:

Es entstehen die folgenden Grundtabellen:

Mitarbeiter, Abteilung, Arbeitsvertrag, Projekt:

```
Mitarbeiter     (MID,Name,Anschrift,AID#)
Abteilung       (AID,Bezeichnung)
Arbeitsvertrag  (MID#,Funktion,Gehalt)
Projekt         (PID,Projektname)
```

Als Beziehungstabelle (N:M) entsteht:

```
Teilnahme       (MID#,PID#,Projektstunden)
```

Überprüfen des Entwurfes durch Normalisieren

Die erste Normalform (1NF)
„Alle Attribute sind atomar“

Das Attribut Anschrift ist, sofern noch nicht erfolgt, in seine Bestandteile aufzulösen.
Es ergibt sich nachfolgende Änderung der Tabellenstruktur:

```
Mitarbeiter     (MID,Name,Strasse,Plz,Ort,AID#)
Abteilung       (AID,Bezeichnung)
Arbeitsvertrag  (MID#,Funktion,Gehalt)
Projekt         (PID,Projektname)
Teilnahme       (MID#,PID#,Projektstunden)
```

Die zweite Normalform (2NF)
„Jedes Nicht-Schlüssel-Attribut ist voll funktional vom Primärschlüssel abhängig“

Dies hier bereits gewährleistet. Jede Relation besitzt schon einen Primärschlüssel (xID). Ein zusammengesetzter Schlüssel wird im Beispiel *Teilnahme* verwendet.

Die dritte Normalform (3NF)
„Keine transitive Abhängigkeit eines Nicht-Schlüssel-Attributes vom Primärschlüssel“

Dies ist hier bei den Tabellen Mitarbeiter noch nicht erfüllt, da der Ort transitiv[1] über die Plz vom Primärschlüssel abhängt. Die theoretische Lösung ist das Auslagern der Postleitzahlen und Orte in eine eigene Tabelle und das Hinzufügen des dort erforderlichen Primärschlüssels als Fremdschlüssel in die Tabelle Mitarbeiter.

Somit ergibt sich folgende Tabellenstruktur

```
Mitarbeiter     (MID,Name,Strasse,PLZID#,AID#)
Abteilung       (AID,Bezeichnung)
Arbeitsvertrag  (MID#,Funktion,Gehalt)
Projekt         (PID,Projektname)
Teilnahme       (MID#,PID#,Projektstunden)
Plz             (PLZID,Plz,Ort)
```

Festlegung der Datentypen und Schlüssel

TABELLE	ATTRIBUT	DATENTYP	SCHLÜSSELTYP	REFERENZTABELLE
MITARBEITER	MID	Counter	**PK**	
	Name	Text(50)		
	Strasse	Text(50)		
	PLZID	Integer	**FK**	**PLZ.PLZID**
	AID	Integer	**FK**	**ABTEILUNG.AID**
ABTEILUNG	AID	Counter	**PK**	
	Bezeichnung	Text(50)		
ARBEITSVERTRAG	MID	Integer	**PK,FK**	**MITARBEITER.MID**
	Funktion	Text(50)		
	Gehalt	Float		
PROJEKT	PID	Counter	**PK**	
	Projektname	Text(50)		
TEILNAHME	MID	Integer	**PK,FK**	**MITARBEITER.MID**
	PID	Integer	**PK,FK**	**PROJEKT.PID**
	Projekt-stunden	Integer		
PLZ	PLZID	Counter	**PK**	
	Plz	Text(10)		
	Ort	Text(50)		

[1] Transitivität, adj. transitiv (latein: transitus ‚Übergang')
Ein Nichtschlüsselattribut darf nicht von einer Menge aus Nichtschlüsselattributen abhängig sein.
Ein Nichtschlüsselattribut darf nur direkt von einem Schlüssel (PK) abhängen.

Vorlesung: Datenbanken

Kapitel (E): Structured Query Language

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Structured Query Language (SQL)

Als externe Kommunikationsschnittstelle stellen Datenbanksysteme eine Datenbanksprache (SQL[1]) zur Formulierung von Abfragen, zum Einfügen und Ändern von Daten und für administrative Befehle (DDL[2] / DML[3] / DCL[4]) zur Verfügung.

Mit dieser Sprache können sowohl die eigentlichen Anwenderdaten als auch die interne Datenbankstruktur und -organisation (Metadaten) definiert und verändert werden.

Die Syntax der SQL ist relativ einfach aufgebaut und semantisch an die englische Umgangssprache angelehnt. SQL stellt eine Reihe standardisierter Befehlen nach ANSI[5] und ISO[6] zur Definition und Manipulation von Datenstrukturen nach der relationalen Algebra zur Verfügung. Durch seine Rolle als Quasi-Standard ist SQL von großer Bedeutung, da eine weitgehende Unabhängigkeit von der benutzten Software erzielt werden kann.

Die Syntax für Datenauswahl startet mit dem Schlüsselwort **SELECT**. Für Manipulationen von Anwenderdaten dienen **INSERT**, **UPDATE** und **DELETE**.

Änderungen an den Metadaten wie das Erstellen oder Löschen einer Tabelle aktiviert man mit der Syntax **CREATE TABLE** oder **DROP TABLE**. Äderungen an Tabellen werden oft mit der Sequenz **ALTER TABLE** eingeleitet.

Die meisten heute verbreiteten Datenbanksysteme implementieren mehr oder weniger große Teile des SQL-Sprachstandards. Dadurch ist es theoretisch möglich, Anwendungsprogramme zu erstellen, die vom verwendeten Datenbanksystem unabhängig sind. Die Praxis scheitert regelmäßig an vermeidbaren Kleinigkeiten, die es dann zu umgehen gilt. So ist beispielsweise zur Behandlung von ***NULL***[7] immer Syntax vorhanden, dieser wird aber durch unterschiedliche Aufrufe realisiert.

NULL bezeichnet „das Fehlen eines Wertes“ also ein Leeres Datenfeld, Nicht zu verwechseln mit 0 oder ‘0‘. Das eine gibt den Zahlenwert, das andere das Zeichen wieder. Ein Leerschritt oder Leerzeichen ist ebenfalls nicht NULL sondern ein Zeichen (ASCII 20Hex).

Beispiel: Drei Spalten werden addiert: A = 1.234; B = 100; C = 5
Summe: **A + B + C** = 1.234 + 100 +5 --> Ergebnis = 1.339

ist eine der Spalten jedoch NULL so ist das Ergebnis nicht die Addition der beiden anderen Spalten sondern ebenfalls NULL. Um ein korrektes Ergebnis zu erzielen, muss hier das Vorkommen von NULL abgefangen werden.

Durch Befehle wie **NZ** (Microsoft, **N**ull-to-**Z**ero) oder **COALESCE** (Interbase, Firebird) oder **NVL** (Oracle, **N**ull**v**a**l**ue) wird der Inhalt einer Variablen auf ‚NULL‘ geprüft und gegebenenfalls durch den angegebenen Wert ersetzt. So liefert NZ(Feldname,0) die Ziffer 0 anstelle eines leeren Eintrages, sonst der Inhalt von Feldname.

Drei Spalten werden addiert: **NZ(A,0) + NZ(B,0) + NZ(C,0)**

Darüber hinaus gibt es noch diverse, herstellerspezifische Erweiterungen, die nicht dem Standard-Sprachumfang entsprechen.

---

[1] SQL Structured Query Language - Abfrage von Daten
[2] DDL Data-Definition Language - Änderungen an den Metadaten
[3] DML Data-Manipulation Language - Manipulationen von Anwenderdaten
[4] DCL Data-Control Language – Zugriffsrechte / Beschränkungen der Anwender
[5] ANSI American National Standards Institute
[6] ISO Internationale Organisation für Normung, das deutsche Gegenstück ist das Deutsche Institut für Normung e.V. (DIN)
[7] NULL Nullwert (kurz NULL) bezeichnet man in der Informatik einen Zustand, der das Fehlen eines Wertes anzeigen soll.

Einfache Beispiele für SQL-Abfragen: Beispieltabelle ***tblZitate***

ID	Namen	Vorname	Merkmal	Zitat
1	Goethe, von	Johann Wolfgang	dt. Schriftsteller	Nichts ist schrecklicher als ein Lehrer, der nicht mehr weiß als das, was die Schüler wissen sollen
2	Baker	Josephin	Tänzerin	Unsere Träume können wir erst dann verwirklichen, wenn wir uns entschließen, einmal daraus zu erwachen
3	Cocteau	Jan	frz. Schriftsteller	Die große Stärke der Narren ist es, dass sie keine Angst haben, Dummheiten zu sagen.
4	Curie	Marie	poln.-frz. Chemikerin	Ein Gelehrter in seinem Laboratorium ist nicht nur ein Techniker; er steht auch vor den Naturgesetzen wie ein Kind vor der Märchenwelt
5	Casanova	Giacomo	ital. Schriftsteller	Einen Dummkopf betrügen heißt, den Verstand rächen
6	Einstein	Albert	dt. Physiker	Zwei Dinge sind unendlich, das Universum und die menschliche Dummheit, aber bei dem Universum bin ich mir noch nicht ganz sicher
7	Einstein	Albert	dt. Physiker	Probleme kann man niemals mit derselben Denkweise lösen, durch die sie entstanden sind
8	Einstein	Albert	dt. Physiker	Mach' dir keine Sorgen wegen deiner Schwierigkeiten mit der Mathematik. Ich kann dir versichern, dass meine noch größer sind

DDL - CREATE

Aufgabe: Erzeuge die Tabelle tblZitate

```
CREATE TABLE TBLZITATE (
    ID        INTEGER NOT NULL,
    NAMEN     VARCHAR(50),
    VORNAME   VARCHAR(50),
    MERKMAL   VARCHAR(30),
    ZITAT     VARCHAR(1024) );
```

```
/*Erzeuge den Primary Key*/
ALTER TABLE TBLZITATE ADD CONSTRAINT PK_TBLZITATE PRIMARY KEY (ID);

/* Erzeuge Generator und Trigger (Autowert)*/
CREATE GENERATOR GEN_TBLZITATE_ID;

SET TERM ^ ;
CREATE OR ALTER TRIGGER TBLZITATE_BI FOR TBLZITATE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(GEN_TBLZITATE_ID,1);
END^
SET TERM ; ^
```

DML - INSERT INTO

Einfügen von Daten in eine Tabelle

```
INSERT INTO Tabellenname ( Spaltenname(n) )
VALUES ( Wert(e) );
```

Tabellennamen	
Spaltenname	Liste der Spaltennamen, getrennt durch Komma
Values	Liste der Werte, getrennt durch Komma. Texte, je nach SQL-Dialekt, in Hochkommata oder Gänsefüße Bei Zahlenwerte ist das Komma meist durch einen Dezimalpunkt zu ersetzen (17,55 -> 17.55)

Aufgabe: Ergänze das Zitate von Berlichingen

```
INSERT INTO tblZitate (Namen, Vorname, Merkmal, Zitat )
VALUES ('von Berlichingen', 'Gottfried',
         'Romanfigur von Goethe',
         'Wo viel Licht ist, ist auch viel Schatten');
```

Übertragen von Daten aus einer Tabelle in eine andere Tabelle

INSERT INTO *Zieltabelle* (*Spaltenname(n)*)
SELECT *Spaltenname(n)*
FROM *Quelltabelle*
WHERE *Bedingung* ***AND*** *Bedingung* ***OR*** *Bedingung*
ORDER BY *Reihenfolge* ***ASC,*** *Reihenfolge* ***DESC;***

Zieltabelle	
Spaltenname	Liste der Spaltennamen der Zieltabelle, getrennt durch Komma
Quelltabelle	
Spaltenname	Liste der Spaltennamen der Quelltabelle, getrennt durch Komma Es müssen nicht alle Spalten einer Tabelle aufgezählt werden, jedoch muss die Anzahl der Spalten übereinstimmen. Die Spaltennamen In Ziel- und Quelltabelle können unterschiedlich sein. Der Datentyp muss gleich sein oder umgewandelt werden. Typumwandlung je nach Dialekt mit CStr(), CDate() oder CAST AS LONG oder T
Bedingung	Auswahlkriterien der Quelltabelle
Reihenfolge	Meist nicht nötig, es sei denn man möchte die Daten bewusst in einer Reihenfolge z.B. mit fortlaufender ID erzeugen

Aufgabe: Ergänze die Zitate in tblZitate mit denen der Tabelle tblLexikon

```
INSERT INTO tblZitate ( Namen, Vorname, Merkmal, Zitat )
SELECT Nachname, Vorname, 'Keines', Texte
FROM tblLexikon WHERE Texte IS NOT NULL;
```

Beispiele:

```
INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Goethe, von', 'Johann Wolfgang ', 'dt.
Schriftsteller', 'Nichts ist schrecklicher als ein Lehrer, der nicht
mehr weiß als das, was die Schüler wissen sollen');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Baker', 'Josephin', 'Tänzerin', 'Unsere Träume
können wir erst dann verwirklichen, wenn wir uns entschließen, einmal
daraus zu erwachen');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Cocteau', 'Jan', 'frz.  Schriftsteller',
'Die große Stärke der Narren ist es, dass sie keine Angst haben,
Dummheiten zu sagen');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Curie', 'Marie', 'poln.-frz. Chemikerin', 'Ein
Gelehrter in seinem Laboratorium ist nicht nur ein Techniker, er steht
auch vor den Naturgesetzen wie ein Kind vor der Märchenwelt');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Casanova', 'Giacomo', 'ital. Schriftsteller',
'Einen Dummkopf betrügen heißt, den Verstand rächen');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Einstein', 'Albert', 'dt. Physiker',
'Zwei Dinge sind unendlich, das Universum und die menschliche
Dummheit, aber bei dem Universum bin ich mir noch nicht ganz sicher');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Einstein', 'Albert', 'dt. Physiker',
'Probleme kann man niemals mit derselben Denkweise lösen, durch die
sie entstanden sind');

INSERT INTO TBLZITATE (NAMEN, VORNAME, MERKMAL, ZITAT)
              VALUES ('Einstein', 'Albert', 'dt. Physiker',
'Mach dir keine Sorgen wegen deiner Schwierigkeiten mit der
Mathematik. Ich kann dir versichern, dass meine noch größer sind');
```

SQL - SELECT

Abfragen von Daten aus Tabelle(n)

SELECT	*Spaltenname(n)*
FROM	*Tabellenname(n) oder/und Sicht(en) oder/und Stored Procedure(s)*
WHERE	*Bedingung **AND** Bedingung **OR** Bedingung*
GROUP BY	*Gruppierung*
HAVING	*wie Where jedoch auf Aggregationsfunktion bezogen (Sum..)*
ORDER BY	*Reihenfolge **ASC**, Reihenfolge **DESC**;*

Spaltenname	Liste der Spaltennamen, getrennt durch Komma oder Zeichen "*" für alle Spalten der Tabelle(n)
Tabellennamen	
Bedingung	*Auswahlkriterie(n).*
Reihenfolge	Spaltennamen, nach denen die Ausgabe sortiert werden soll. Bei aufsteigender Sortierung mit dem Zusatz ASC (ascending, meist optional), bei absteigender Sortierung mit dem Zusatz DESC (descending).

Aufgabe: Zeige das 4. Zitat der Tabelle tblZitate

```
SELECT Namen, Zitat FROM tblZitate WHERE ID = 4;
```

	NAMEN	ZITAT
▶	Curie	Ein Gelehrter in seinem Laboratorium ist nicht nur ein Techniker; er steht auch vor den Naturgesetzen wie ein Kind vor der Märchenwelt

Aufgabe: Zeige alle Zitate von Albert Einstein in Reihenfolge

```
SELECT Zitat FROM tblZitate
WHERE Vorname = 'Albert' AND Namen = 'Einstein'
ORDER BY Zitat;
```

	ZITAT
	Mach' dir keine Sorgen wegen deiner Schwierigkeiten mit der Mathematik. Ich kann dir versichern, dass meine noch größer sind
	Probleme kann man niemals mit derselben Denkweise lösen, durch die sie entstanden sind
▶	Zwei Dinge sind unendlich, das Universum und die menschliche Dummheit, aber bei dem Universum bin ich mir noch nicht ganz sicher

Beim Vergleich von Textinhalten sind Trennzeichen, meist Hochkommata oder Gänsefüße, zu verwenden. Viele Datenbankserver sind Case-sensitiv, das bedeutet, dass nur exakte Übereinstimmung inklusive Groß- Kleinschreibung mit dem Textinhalt abfragbar sind. In diesen Fällen verwendet man Konvertierungen wie beispielsweise **UPPER(Vorname) = 'ALBERT'.**

Ein Hinweis an dieser Stelle: Eine mit UPPER/LOWER sortierte Tabelle kann nicht mehr den für das Originalfeld hinterlegten Index nutzen und wird dadurch deutlich langsamer. Unter Umständen ist es trotz unvermeidlicher Redundanz hier geschickter ein zusätzliches ‚Sortierfeld' in die Tabelle einzufügen.

Bei Selektion von Teiltexten kommen Muster und Sonderzeichen wie **[% * ? _]** zum Einsatz.

Hier einige Beispiele zum Mustervergleich:

'A_Z' Alle Zeichenketten die mit einem 'A' beginnen, worauf ein weiteres Zeichen folgt, und mit einem 'Z' enden. 'ABZ' und 'A2Z' würden beispielsweise diese Bedingung erfüllen, 'AKKZ' hingegen nicht. Bei Microsoft auch als Zeichenfolge 'A?Z' bekannt.

'ABC%' Alle Zeichenketten, die mit 'ABC' beginnen. Sowohl 'ABCD' als auch 'ABCABC' würden diese Bedingung erfüllen. Bei Microsoft auch als Zeichenfolge 'ABC*' möglich.

'%XYZ' Alle Zeichenketten, die auf 'XYZ' enden. So würden beispielsweise sowohl 'WXYZ' als auch 'ZZXYZ' diese Bedingung erfüllen. Bei Microsoft auch als Zeichenfolge '*ABC'.

'%AN%' Alle Zeichenketten, die an irgendeiner Stelle das Muster 'AN' enthalten. Sowohl 'LOS ANGELES' als auch 'SAN FRANCISCO' würden zum Beispiel diese Bedingung erfüllen. Bei Microsoft auch als Zeichenfolge '*AN*' möglich.

DISTINCT Der Zusatz DISTINCT gibt an, dass aus der Ergebnisrelation gleiche Ergebnismengen entfernt werden sollen. Es wird also jeder Datensatz nur einmal ausgegeben, auch wenn er mehrfach in der Tabelle vorkommt.

```
SELECT DISTINCT Zitat FROM tblZitate
WHERE Vorname = 'Albert' AND Namen = 'Einstein'
ORDER BY Zitat;
```

SQL-Funktionen / Aggregationsfunktionen

Eine Funktion ist eine besondere Art von Befehlswort in der Befehlsmenge der SQL. Funktionen / Aggregate sind aus einem Wort bestehende Befehle, die genau einen Wert zurückgeben.

Aggregation, auch als Konsolidierung oder Verdichtung bezeichnet, ist das Zusammenfassen einer Reihe von Daten zu einem einzelnen Datum. Beispielsweise lassen sich aus einer Menge von Zahlen der Mittelwert, das Minimum bzw. Maximum oder die Summe bestimmen. Solche Funktionen, die einer Menge von Zahlen einen einzelnen Wert zuordnen, nennt man Aggregationsfunktionen. Das Ergebnis wird dann stellvertretend für die Quelldaten verwendet.

Aggregate können optional durch Eingabeparameter bestimmt werden. Beispielsweise die Summe ***SUM(Attribut)***, den Min/Max-Wert ***MIN(Attribut) / MAX(Attribut)*** oder der Durchschnittswert ***AVG(Attribut)*** einer Datenmenge. Andere Funktionen benötigen keine Eingabeparameter, wie etwa die Funktion ***CURRENT_TIME***, die die aktuelle Systemzeit zurückgibt. Diese Kategorie von Funktionen bezeichnet man auch als **Skalare**.

Der SQL99-Standard definiert eine Reihe nützlicher Funktionen. Zusätzlich hat jeder Datenbankhersteller eine eigene Liste weiterer Funktionen, die sich nicht im SQL-Standard wiederfinden. Im Anschluss ist für verschiedene DatenbankImplementierungen eine Auswahl und Beschreibungen der internen Funktionen aufgelistet.

Neben den Aggregaten erlauben einige Datenbankhersteller die Verwendung eigener benutzerdefinierter Funktionen (user-defined functions, UDFs). Nähere Informationen zu UDFs sind in den jeweiligen SQL-Handbüchern der Datenbankhersteller gelistet.

Aggregate geben auf der Basis anderer Werte einen einzelnen Wert zurück. Wenn Aggregat-Funktionen unter vielen anderen Ausdrücken in der Elementliste einer SELECT-Anweisung verwendet werden, dann muss diese Anweisung eine **GROUP BY**-Klausel haben. Dies ist nicht notwendig, wenn das Aggregat der einzige von der SELECT-Anweisung gelieferte Wert ist.

Aggregat gibt es mit den unterschiedlichsten Anwendungen und Eigenschaften.
Beispielhaft eine Auflistung der internen Funktionen von Interbase / Firebird:

```
ABS(), ACOS(), ASCII_CHAR(), ASCII_VAL(), ASIN(), ATAN(), ATAN2(),
BIN_AND(), BIN_OR(), BIN_SHL(), BIN_SHR(), BIN_XOR(), BIT_LENGTH(),
CAST(), CEIL(), CEILING(), CHAR_LENGTH(), CHARACTER_LENGTH(),
COALESCE(), COS(), COSH(), COT(), DATEADD(), DATEDIFF(), DECODE(),
DIV(), EXP(), EXTRACT(), FLOOR(), GEN_ID(), GEN_UUID(), HASH(), IIF(),
LEFT(), LN(), LOG(), LOG10(), LOWER(), LPAD(), MAXVALUE(), MINVALUE(),
MOD(), NULLIF(), OCTET_LENGTH(), OVERLAY(), PI(), POSITION(), POWER(),
RAND(), REPLACE(), REVERSE(), RIGHT(), ROUND(), RPAD(), SIGN(), SIN(),
SINH(), SQRT(), SUBSTRING(), TAN(), TANH(), TRIM(), TRUNC(), UPPER()
```

Aufgabe: Ermittle die Anzahl der Zitate von Albert Einstein

```
SELECT COUNT(*) FROM tblZitate
WHERE Vorname = 'Albert' AND Namen = 'Einstein';
```

	COUNT
▶	3

Aufgabe: Ermittle die Anzahl der Zitate pro Autor

```
SELECT Namen, Vorname, COUNT(*) AS ANZAHL_ZITATE
FROM tblZitate
GROUP BY Namen, Vorname
ORDER BY Namen, Vorname;
```

	NAMEN	VORNAME	ANZAHL_ZITATE
	Baker	Josephin	1
	Casanova	Giacomo	1
	Cocteau	Jan	1
	Curie	Marie	1
	Einstein	Albert	3
▶	Goethe, von	Johann Wolfgang	1

Aufgabe: Extrahiere die ersten drei Buchstaben des Namens
und Zeige das Zitat ‚Rückwerts' für eine Rückseiten-Projektion einer Leinwand

SELECT LEFT(Namen,3**)**, **REVERSE(**Zitat**) FROM** tblZitate
ORDER BY Namen, Vorname, Zitat**;**

	LEFT	REVERSE
▶	Bak	nehcawre uz suarad lamnie ‚neßeilhcstne snu riw nnew ‚nehcilkriwrev nnad tsre riw nennök emuärT eresnU
	Cas	nehcär dnatsreV ned ‚tßieh negürteb fpokmmuD neniE
	Coc	negas uz netiehmmuD ‚nebah tsgnA eniek eis ssad ‚se tsi nerraN red ekrätS eßorg eiD
	Cur	tlewnehcräM red rov dniK nie eiw neztesegrutaN ned rov hcua thets re ;rekinhceT nie run thcin tsi muirotarobaL menies ni retrheleG niE
	Ein	dnis reßörg hcon eniem ssad ‚nrehcisrev rid nnak hcI .kitamehtaM red tim netiekgireiwhcS renied negew negroS eniek rid 'hcaM
	Ein	dnis nednatstne eis eid hcrud ‚nesöl esiewkneD neblesred tim slamein nam nnak emelborP
	Ein	rehcis znag thcin hcon rim hci nib musrevinU med ieb reba ‚tiehmmuD ehcilhcsnem eid dnu musrevinU sad ‚hcildnenu dnis egniD iewZ
	Goe	nellos nessiw relühcS eid saw ‚sad sla ßiew rhem thcin red ‚rerheL nie sla rehcilkcerhcs tsi sthciN

Aufgabe: Ermittle Anzahl, Durschnitts- sowie Minimalen- und Maximalen-Preis der Artikel.

```
SELECT COUNT(ID) AS Anzahl,
       AVG(PREIS) AS "Preis Durchschnitt",
       MIN(PREIS) AS "Preis Minimum",
       MAX(PREIS) AS "Preis Maximum" FROM ARTIKEL;
```

Aufgabe: Ermittle Anzahl und Durchschnittspreis der Lagerartikel mit Bestand von mehr als 100 St.

```
SELECT COUNT(ID) AS Anzahl,
       AVG(PREIS) AS "Preis Durchschnitt"
FROM LAGER
WHERE STUECK > 100;
```

Aufgabe: Bilde die Summe vom Betrag aller Belege die zum Netto-Termin noch nicht bezahlt sind sowie von Belegen bei denen noch Restzahlungen offen stehen.

```
SELECT SUM(R.BETRAG) AS Offen
FROM RECHNUNG R
WHERE ((R.BEZAHLTAM IS NULL) OR NOT (R.BEZAHLTREST IS NULL))
  AND  (R.DATUM + R.NETTOTAGE <= CURRENT_DATE);
```

Aufgabe: Liste aller Mitarbeiter der Nummern 11 bis 99

```
SELECT M.* FROM MITARBEITER M
WHERE M.NUMMER BETWEEN 11 AND 99;
```

Aufgabe: Liste aller aktiven Mitarbeiter die ein Handzeichen haben und deren Eintritt <= heute und deren Austritt >= heute ist, sortiert nach Namen, Vorname, Handzeichen

```
SELECT M.ZEICHEN, M.NAMEN FROM MITARBEITER M
WHERE COALESCE(M.ZEICHEN,'') <> ''
  AND IIF(M.EINTRITT IS NULL, CURRENT_DATE, M.EINTRITT)
        <= CURRENT_DATE
  AND IIF(M.AUSTRITT IS NULL, CURRENT_DATE, M.AUSTRITT)
      >= CURRENT_DATE
ORDER BY M.NACHNAME, M.VORNAME, M.ZEICHEN;
```

Aufgabe: Liste aller aktiven Mitarbeiter mit Jubiläum und der Firmenzugehörigkeit von 10 Jahren, 15 Jahren, 20 Jahren, 25 Jahren, 30 Jahren

```
SELECT M.NAMEN FROM MITARBEITER M ....
WHERE (YEAR(CURRENT_DATE) - YEAR(M.EINTRITT)) IN (30,25,20,15,10);
```

Hinweis:
Je nach Hersteller ändert sich die Syntax für YEAR und CURRENT_DATE zB in EXTRACT(YEAR FROM CURRENT_DATE) oder YEAR(NOW())

Aufgabe: Liste der Bezeichnungen der Artikel, deren ID in der Liste AUSWAHL steht

```
SELECT A.BEZEICHNUNG FROM ARTIKEL A
WHERE A.ID IN (SELECT DISTINCT X.ID FROM AUSWAHL X);
```

Liste der Funktionen / Aggregate

Funktion	*Verwendung*
AVG(Ausdruck**)**	Berechnet den Durchschnittswert einer Spalte, die durch den Ausdruck bestimmt wird
COUNT(Ausdruck**)**	Zählt die vom Ausdruck definierten Zeilen
COUNT(*)	Zählt alle Zeilen in der angegebenen Tabelle oder dem View
MIN(Ausdruck**)**	Findet den kleinsten Wert in der durch den Ausdruck angegebenen Spalte
MAX(Ausdruck**)**	Findet den größten Wert in der durch den Ausdruck angegebenen Spalte
SUM(Ausdruck**)**	Berechnet die Summe der durch den Ausdruck angegebenen Spaltenwerte

AVG / SUM

werden von Firebird, Microsoft SQL Server, MySQL, Oracle und PostgreSQL unterstützt.

-> Die folgende Abfrage berechnet durchschnittliche Verkaufszahlen
für das laufende Jahr für jeden Buchtyp:

```
SELECT BUCHTYP, AVG(VERKAEUFE) AS 'DURCHSCHNIILICHE VERKÄUFE'
FROM BUECHERBERKAUF
GROUP BY BUCHTYP;
```

-> Diese Abfrage ermittelt die Summe der Verkaufszahlen für das laufende Jahr pro Buchtyp

```
SELECT YEAR(VERKAUSDATUM) AS 'JAHR', BUCHTYP,
       AVG(VERKAEUFE) AS 'DURCHSCHNIILICHE VERKÄUFE'
FROM BUECHERVERKAUF
GROUP BY 1,2;
```

COUNT

Die Funktion COUNT gibt es in drei Varianten. **COUNT**(*) zählt alle Zeilen in der Zieltabelle, egal, ob sie Null-Werte enthalten oder nicht. **COUNT**(Ausdruck) zählt die Anzahl der Nicht-NULL-Werte in einer bestimmten Spalte oder in einem Ausdruck. **COUNT**(DISTINCT Ausdruck) schließlich zählt die Anzahl der unterschiedlichen Nicht-NULL-Werte in einer Spalte bzw. einem Ausdruck.

-> Diese Abfrage zählt alle Zeilen in einer Tabelle:

```
SELECT COUNT(*) FROM VERLAGE;
```

-> Die folgende Abfrage findet die Anzahl der Länder, in denen es Verlage gibt:

```
SELECT COUNT(DISTINCT LAND) AS 'Länderanzahl'
FROM VERLAGE;
```

MIN / MAX

MIN(Ausdruck) und **MAX**(Ausdruck) finden den kleinsten respektive größten Wert (String, Datum/Uhrzeit oder numerisch) in einer Menge von Zeilen. DISTINCT oder ALL können bei diesen Funktionen zwar verwendet werden, haben aber keinen Einfluss auf das Ergebnis.

MIN und MAX werden von Firebird, Microsoft SQL Server, MySQL, Oracle und PostgreSQL unterstützt.
MySQL unterstützt außerdem die Funktionen LEAST( ) und GREATEST( ), die die gleiche Funktionalität aufweisen.

-> Die Abfrage findet die besten und schlechtesten Verkaufszahlen aller gespeicherten Titel:

```
SELECT BUCHTITEL, MIN(VERKAEUFE) AS 'Penner',
       MAX(VERKAEUFE) AS 'Renner'
FROM BUECHERVERKAUF
GROUP BY BUCHTITEL;
```

Aggregate werden oft in der HAVING-Klausel von Abfragen mit GROUP BY verwendet. Die folgende Abfrage gibt alle Kategorien (Typen) von Büchern aus, bei denen der durchschnittliche Preis aller Bücher in der Kategorie über DM 30 liegt:

```
SELECT BUCHTYP AS 'Kategorie', AVG(PREIS) AS 'Durchschnittspreis'
FROM BUECHERPREISLISTE
GROUP BY BUCHTYP
HAVING AVG(PREIS) > 30;
```

Liste der Funktionen / Skalare

Funktionskategorie	*Erläuterung*
Eingebaut	Führt Operationen auf in der Datenbank eingebauten Werten oder Einstellungen durch. *Oracle verwendet den Ausdruck »built-in« für alle Oracle-spezifischen Funktionen. Dies unterscheidet sich von den hier genannten eingebauten Funktionen.*
Datum & Uhrzeit	Führt Operationen auf Datetime-Feldern durch und gibt Werte im Datetime-Format zurück.
Numerisch	Führt Operationen auf numerischen Werten durch und gibt auch solche Werte zurück.
String	Führt Operationen auf Zeichen-Werten (*char, varchar, nchar, nvarchar, clob, blob (Text))* durch und gibt einen String oder einen numerischen Wert zurück.

Firebird und Microsoft SQL Server unterstützt alle diese eingebauten skalaren Funktionen. Oracle unterstützt die oben genannten Funktionen nicht, dafür aber USER als Synonym für CURRENT_USER und SYSDATE als Synonym für CURRENT_TIMESTAMP. MySQL unterstützt alle von SQL99 definierten eingebauten skalaren Funktionen und darüber hinaus auch die beiden Oracle-Varianten. PostgreSQL unterstützt USER als Synonym für CURRENT_USER. Außerdem unterstützt MySQL NOW( ) und UNIX_TIMESTAMP( ) als Synonyme der Funktion CURRENT_TIMESTAMP. PostgreSQL unterstützt alle von SQL99 definierten eingebauten skalaren Funktionen mit Ausnahme von SESSION_USER.

-> Die folgenden Abfragen geben die Werte von eingebauten Funktionen zurück.

```
/* MySQL / Firebird */
SELECT CURRENT_TIMESTAMP;          -> '2001-12-15 23:50:26'

/* Microsoft SQL Server */
SELECT CURRENT_TIMESTAMP
GO                                 -> 'Dec 15,2001 23:50:26'

/* Oracle */
SELECT USER FROM DATABASE;         -> Admin
```

Numerische skalare Funktionen

Die Liste der offiziellen numerischen Funktionen in SQL99 ist ziemlich kurz. Die einzelnen Hersteller bieten aber eine große Anzahl weiterer mathematischer und statistischer Funktionen. MySQL unterstützt viele dieser Befehle in seinen SQL99-Inkarnationen. Die anderen Datenbankprodukte unterstützen dieselben numerischen skalaren Funktionen über ihre eigenen intern definierten Funktionen, verwenden aber nicht die gleichen Namen wie der SQL-Standard.

Funktion	***Aufgabe***
BIT_LENGTH(Ausdruck)	Gibt einen Integer-Wert zurück, der die Anzahl der Bits in einem Ausdruck repräsentiert.
CHAR_LENGTH(Ausdruck)	Gibt einen Integer-Wert zurück, der die Anzahl der -Zeichen in einem Ausdruck repräsentiert.
EXTRACT(datetime_ausdruck datumsbestandteil FROM Ausdruck)	Erlaubt das Extrahieren eines Datumsbestandteils (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR oder TIMEZONE_MINUTE) aus einem Ausdruck.
OCTET_LENGTH(Ausdruck)	Gibt einen Integer-Wert zurück, der die Anzahl der -Oktette in einem Ausdruck repräsentiert. Dieser Wert ist identisch mit BIT_LENGTH/8.
POSITION(Start_string IN Such_string)	Gibt einen Integer-Wert zurück, der die Startposition -eines Strings im Suchstring repräsentiert.

BIT_LENGTH, CHAR_LENGTH und OCTET_LENGTH

Oracle kommt von allen Herstellern der Funktion BIT_LENGTH am nächsten. Oracle verfügt über die Funktion LENGTHB, die einen Integer-Wert zurückgibt, der die Anzahl der Bytes in einem Ausdruck repräsentiert. Firebird, MySQL und PostgreSQL unterstützen CHAR_LENGTH und das SQL99-Synonym CHARACTER_LENGTH(). PostgreSQL und Firebird unterstützen darüber hinaus EXTRACT(), OCTET_LENGTH() und POSITION() gemäß dem SQL99-Standard. Die anderen beiden Hersteller haben ähnliche Funktionen mit der gleichen Funktionalität, bei SQL Server ist das die Funktion LEN, bei Oracle die Funktion LENGTH. MySQL und PostgreSQL unterstützen darüber hinaus auch die Funktion OCTET_LENGTH vollständig.

-> Das folgende Beispiel bestimmt die Länge eines Strings und eines Wertes, die aus einer Spalte abgerufen wurden:

```
/* Firebird, MySQL und PostgreSQL */
SELECT CHAR_LENGTH('Hallo');

SELECT OCTET_LENGTH(TITEL) FROM BUECHERVERKAUF;

/* Microsoft SQL Server */
SELECT DATALENGTH(TITEL)
FROM BUECHERVERKAUF
WHERE BUCHTYP = 'Populaer'
GO

/* Oracle */
SELECT LENGTH('HORATIO') "Länge in Zeichen"
FROM ...
```

EXTRACT

'Extrahiert' Teile aus gegebener Datum und Uhrzeit.

Die Funktion EXTRACT wird nur von Firebird, PostgreSQL und MySQL unterstützt.
Jeder Hersteller unterstützt aber einen anderen Befehl, der dieselbe Funktionalität bietet. Bei Oracle wird mit der Funktion TO_CHAR ein Teil aus einem Datumswert in einen Zeichenstring eingelesen. SQL Server verwendet dazu die Funktion CONVERT, Firebird die CAST-Funktion.
Die MySQL-Implementierung geht etwas über den SQL99-Standard hinaus. Dieser sieht es nicht vor, in einem Aufruf von EXTRACT() mehrere Felder zurückzugeben (wie etwa in »DAY_HOUR«). Die MySQL-Erweiterungen versuchen, das zu erreichen, was die Kombination aus DATE_TRUNC() und DATE_PART() in PostgreSQL bewirkt. MySQL unterstützt die nachfolgend genannten Datumsbestandteile, Firebird nur teilweise dafür aber zB WEEKDAY oder YESTERDAY.

Typ-Wert	***Bedeutung***	***Erwartetes Format***
SECOND	Sekunden	SECONDS
MINUTE	Minuten	MINUTES
HOUR	Stunden	HOURS
DAY	Tage	DAYS
MONTH	Monate	MONTHS
YEAR	Jahre	YEARS
MINUTE_SECOND	Minuten und Sekunden	»MINUTES:SECONDS«
HOUR_MINUTE	Stunden und Minuten	»HOURS:MINUTES«
DAY_HOUR	Tage und Stunden	»DAYS HOURS«
YEAR_MONTH	Jahre und Monate	»YEARS-MONTHS«
HOUR_SECOND	Stunden, Minuten und Sekunden	»HOURS:MINUTES:SECONDS«
DAY_MINUTE	Tage, Stunden und Minuten	»DAYS HOURS:MINUTES«
DAY_SECOND	Tage, Stunden, Minuten und Sekunden	»DAYSHOURS:MINUTES:SECONDS«

-> Dieses Beispiel extrahiert Datumsbestandteile aus diversen Datums- und Uhrzeitwerten:

```
/* Firebird / MySQL  */
SELECT EXTRACT(YEAR FROM "2013-07-02");                  -> 2013

/* MySQL  */
SELECT EXTRACT(YEAR_MONTH FROM "2013-07-02 01:02:03");  -> 201307

SELECT EXTRACT(DAY_MINUTE FROM "2013-07-02 01:02:03");  -> 20102
```

POSITION

Die Funktion POSITION gibt einen Integer-Wert zurück, der die Startposition eines Teilstrings in einem Suchstring anzeigt.

MySQL und PostgreSQL unterstützen die POSITION-Funktion ohne Abweichung vom SQL99-Standard. PostgreSQL hat eine synonyme Funktion namens TEXTPOS, MySQL die synonyme Funktion LOCATE. Oracles Äquivalent lautet INSTR. Microsoft SQL Server hat sowohl CHARINDEX als auch PATINDEX. Die beiden sind sich sehr ähnlich, PATINDEX erlaubt aber zudem die Verwendung von Wildcards in den Suchkriterien.

-> Beispiel:

```
/* Firebird */
SELECT POSITION('BE' IN 'TO BE OR NOT TO BE')          -> 4
SELECT POSITION('BE','TO BE OR NOT TO BE')             -> 4
SELECT POSITION('BE','TO BE OR NOT TO BE', 4)          -> 4
SELECT POSITION('BE','TO BE OR NOT TO BE', 8)          -> 17
SELECT POSITION('BE','TO BE OR NOT TO BE',18)          -> 0
SELECT POSITION('BE' IN 'ALAS, POOR YORICK!')          -> 0

/* MySQL */
SELECT LOCATE('bar', 'foobar');                        -> 4

/* MySQL und PostgreSQL */
SELECT POSITION('fh' IN 'snafhu');                     -> 4

/* Microsoft SQL Server */
SELECT CHARINDEX( 'cd', 'abcdefg' )
GO                                                     -> 3

SELECT PATINDEX( '%fg', 'abcdefg' )
GO                                                     -> 6
```

Stringfunktionen

Grundlegende Stringfunktionen bieten eine Reihe von Fähigkeiten und geben immer einen Stringwert als Ergebnismenge zurück. Manche Stringfunktionen sind dyadisch, arbeiten also auf zwei Strings gleichzeitig. SQL99 unterstützt nachfolgend genannten Stringfunktionen. Beispiele hierfür sprengen den Rahmen. Diese sind jedoch in ausreichender Zahl und Qualität im Internet zu finden.

Funktion	***Verwendung***
CONCATENATE (Ausdruck || Ausdruck)	Verknüpft zwei oder mehrere literale Ausdrücke, Spaltenwerte oder Variablen zu einem String.
CONVERT	Konvertiert einen String in eine andere Darstellungsweise im gleichen Zeichensatz.
LOWER	Konvertiert einen String in Kleinbuchstaben.
SUBSTRING	Extrahiert Teile des Strings.
TRANSLATE	Konvertiert einen String von einem Zeichensatz in einen anderen.
LTRIM	Entfernt führende Leer-Zeichen aus einem String.
RTRIM	Entfernt Leer-Zeichen am Ende aus einem String.
TRIM	Entfernt führende Leer-Zeichen und Leer-Zeichen am Ende aus einem String.
UPPER	Konvertiert einen String in Großbuchstaben.

SQL – JOIN / LEFT JOIN / RIGHT JOIN / OUTER JOIN

Mit der Operation **INNER-, LEFT-, RIGHT-, OUTER- JOIN** werden Tabellen miteinander verbunden. Eine Beziehung kann über Felder / Feldnamen und dem Schlüsselwort **ON** hergestellt werden. Wird keine Beziehung

- **INNER JOIN** Stellt als Ergebnismange nur Daten da, die in BEIDEN Relationen vorkommen und die gewünschte Übereinstimmung besitzen.
- **LEFT JOIN** Stellt alle Daten da, die in der erstgenannten Relation vorkommen ergänzt durch Daten, die in der zweitgenannten Relation vorkommen
- **RIGHT JOIN** Umkehr von LEFT JOIN.
  Stellt alle Daten da, die in der zweitgenannten Relation vorkommen ergänzt durch Daten, die in der erstgenannten Relation vorkommen
- **OUTER JOIN** Stellt die Gesamtheit von Informationsobjekten dar, unabhängig davon, ob durch die Join-Operation in einer anderen Tabelle eine Übereinstimmung gefunden wird oder nicht.
  OUTER JOIN ist leider nicht standardisiert.

Beispiel:

```
SELECT A.Artikelname, W.Warengruppenname
FROM Artikel A
LEFT JOIN Warengruppen W ON A.ArtikelNr = W.ArtikelNr;
```

Die Syntax ist nicht auf zwei Tabellen beschränkt:

```
SELECT A.Artikelname, W.Warengruppenname, P.Artikelpreis
FROM Artikel A
LEFT JOIN Warengruppen W ON A.ArtikelNr = W.ArtikelNr
LEFT JOIN Artikelpreise P ON A.ArtikelNr = P.ArtikelNr
WHERE A.Artikelname LIKE 'KABEL%',
```

Beispiel:

In der Relation ADRESSEN sind die Felder ADRESSENID und NAMEN, STRASSE und ORT enthalten. In der Relation PARTNER sind die Felder PARTNERID, ADRESSENID (FK), H_F und PARTNERNAME enthalten.

Wenn nun als Ergebnis eine Abfrage aller Partner einer Adresse angestrebt, so müssen die Tabellen ADRESSEN und PARTNER verbunden werden. Am effektivsten erreicht man dies über Schlüssel- und Fremdschlüssel-Attribute. Es sind aber auch andere Kombinationen möglich.

Relation ADRESSEN

	AdressenID	Namen	Strasse	Ort
▶	1447	Bosch Rexroth AG	Werkstraße	Magdeburg
	2233	Bosch Rexroth AG	Waldecker Strasse 13	Walldorf
	2234	Bosch Rexroth Electric Drives	Berliner Straße 25	Erbach
	2800	Bosch Rexroth AG	Walsroder Strasse 93	Langenhagen
	4903	Bosch Rexroth AG	Eschborner Landstrasse 130-132	Frankfurt
	7513	Bosch Rexroth AG	Siemensstraße 1	Fellbach

Relation PARTNER

PartnerID	AdressenID	H_F	Vorname	Partner
114	3638	H		Kaiser
115	3638	F		Roos
116	3638	F	Birgit	Glas
117	3638	H	Volker	Schmidt
118	3638	H		Schum
119	3638	H		Kobs
120	4903	H		Schupp
121	4903	H		Röchner
122	4903	H		Türk
123	4903	F	Sabine	Bauer

Aufgabe: Es sollen Namen und Gehälter der Mitarbeiter angezeigt werden, deren Gehalt unter dem Durchschnittsgehalt liegt.

```
SELECT A.Name, P.H_F, P.Partnername AS Ansprechpartner
FROM Adressen A
INNER JOIN Partner P ON A.AdressenID = P.AdressenID
WHERE A.Name = 'Bosch Rexroth Electric';
```

Ergebnismenge LEFT JOIN / INNER JOIN

Namen	H_F	Ansprechpartner
Bosch Rexroth Electric	F	Roos
Bosch Rexroth Electric	F	Glas
Bosch Rexroth Electric	H	Schmidt
Bosch Rexroth Electric	H	Schum
Bosch Rexroth Electric	H	Kobs
Bosch Rexroth Electric	H	Schupp
Bosch Rexroth Electric	H	Röchner
Bosch Rexroth Electric	H	Türk
Bosch Rexroth Electric	F	Bauer
Bosch Rexroth Electric	H	Jost
Bosch Rexroth Electric	H	Schaal
Bosch Rexroth Electric	H	Völker
Bosch Rexroth Electric	H	Schulung Büro
Bosch Rexroth Electric	H	Strohner
Bosch Rexroth Electric	H	Jost
Bosch Rexroth Electric	F	Spix
Bosch Rexroth Electric	H	Volk
Bosch Rexroth Electric	H	Jost

Aufgabe: Es sollen Namen und Gehälter der Mitarbeiter angezeigt werden, deren Gehalt unter dem Durchschnittsgehalt liegt.

Mitarbeiter sind in der Relation MITARBEITER gespeichert,
Gehälter sind in der Relation ARBEITSVERTRAG gespeichert.
Die Beziehung wird über den Fremdschlüssel MID hergestellt.

```
SELECT M.NAME, A.GEHALT
FROM MITARBEITER M INNER JOIN ARBEITSVERTRAG A ON M.MID = A.MID
WHERE A.GEHALT < (SELECT AVG(X.GEHALT) FROM ARBEITSVERTRAG X)
```

DML - DELETE

Löschen von Daten aus einer Tabelle

DELETE
FROM *Tabellenname*
WHERE *Bedingung **AND** Bedingung **OR** Bedingung**;***

Delete	Löscht den kompletten Datensatzes (Bei Microsoft auch DELETE *)
Tabellenname	
Bedingung	Auswahlkriterien. Ohne Bedingung wird die Tabelle komplett geleert.

Aufgabe: Lösche das Zitat von Casanova

```
DELETE FROM tblZitate WHERE ID = 5;
```

oder

```
DELETE FROM tblZitate WHERE Namen = 'Casanova';
```

DML - UPDATE

Ändern von Daten in einer Tabelle

UPDATE	*Tabellenname*
SET	*Spaltenname1 = Wert1, Spaltenname2 = Wert2*
WHERE	*Bedingung **AND** Bedingung **OR** Bedingung **;***

Update	Ändern von Inhalten einzelner Spalten einer Tabelle. Auch das Löschen von Spalteninhalten wird mittels Update durchgeführt. Beispielsweise durch SET Spaltenname = NULL. Der Befehl DELETE löscht die komplette Spalte, nicht einzelne Inhalte.
SET	SET leitet die Liste der Spaltennamen ein, die geändert werden sollen. Set steht nur bei der ersten Spalte, weitere Spalten werden durch Komma getrennt.
Bedingung	Auswahlkriterien. Ohne Bedingung werden die genannten Spalten aller Datensätze geändert.

Aufgabe: Ändere das Merkmal von Josephin Baker in „Tänzerin in Paris“

```
UPDATE tblZitate SET Merkmal = 'Tänzerin in Paris'
WHERE Namen = 'Baker' AND Vorname = 'Josephin';
```

oder

```
UPDATE tblZitate SET Merkmal = 'Tänzerin in Paris' WHERE ID = 2;
```

SQL – UNION

Union vereinigt die Ergebnismengen mehrerer SELECT-Statements. Die einzelnen Select müssen die gleiche Anzahl und Reihenfolge an Ergebnisfeldern liefern.

- **UNION** vereinigt die Ergebnismengen mehrerer Relation.
  In beiden Tabellen doppel vorkommende Datensätze werden nicht gezeigt

- **UNION ALL** vereinigt die Ergebnismengen.
  Alle, auch mehrfach vorkommende Datensätze werden ausgegeben.

- **EXCEPT** vereinigt die Ergebnismengen.
  Liefert Datensätze die in der ersten, jedoch nicht in der zweiten Tabelle vorkommen.

Leider sind auch hier viele ‚individuelle' SQL-Dialekte im Umlauf. Manche SQL verstehen den Zusatz ALL nicht, andere wollen für die gleiche Ergebnismenge von UNION den Syntax UNION DISTINCT.

Alternativ zu EXEPT wird teilweise auch die Syntax MINUS verwendet.

Tabelle Mitarbeiter_Norway

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Tabelle Mitarbeiter_USA

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Abfrage:

```
SELECT E_Name FROM Mitarbeiter_Norway
UNION ALL
SELECT E_Name FROM Mitarbeiter_USA;
```

Ergebnis:

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

Vorlesung: Datenbanken

Kapitel (G): Stored Procedures

Dipl.-Ing. Peter Lutz

LINOTEC GmbH
Geschäftsführender Gesellschafter

CLS Energy Consultants DMCC
Dubai / United Arab Emirates

Duale Hochschule Baden Württemberg
Dozent / Gutachter

Stored Procedures

Der Begriff Gespeicherte Prozedur (eng. Stored Procedure) bezeichnet eine in vielen professionellen Datenbankmanagementsystemen gegebene Möglichkeit eine logische Abfolge von Anweisungen datenbankintern auszuführen. Wie der Name vermuten lässt, können diese Abfolgen unter einem Namen im Datenbankserver gespeichert werden. Sie stehen dann auf dem Datenbankserver und in der Client-Anwendung zur Verfügung und können von beiden aufgerufen und ausgeführt werden.

Mittels gespeicherter Prozeduren lassen sich häufig verwendete Abläufe, die sonst durch viele einzelne SQL-Befehle vom Client auszuführen wären, auf das Datenbanksystem auslagern. Die Abläufe sind dann durch einen einzigen Aufruf (SELECT, CALL oder EXECUTE) ausführbar.

Neben der Syntax der ‚normalen' Abfragesprache (SQL) stellen gespeicherten Prozeduren in einem erweiterten Syntax (PSQL[1]) viele zusätzliche Befehle zur Ablaufsteuerung und/oder Auswertung von Bedingungen zur Verfügung. Man kann sie im Prinzip mit Makrosprachen bestimmter Anwendungsprogramme vergleichen. Oft wird das verwendete SQL um herstellerspezifische Funktionen erweitert. Auch der Einsatz von anderen Programmiersprachen wie etwa Java oder C# ist inzwischen teilweise möglich.

Die Anwendungsgebiete und die Vorteile von Stored Procedures sind:

Modularisierung

Keine redundante Programmierung. Der Programmcode wird nur einmal und zwar auf dem Server und nicht in x Duplikaten auf jeden Client ausgeführt. Der erweiterte Befehlssyntax aus PSQL steht auf dem Client nicht zur Verfügung.

Administration

Durch die Modularisierung und der zentralen Datenhaltung machen Änderungen einer Stored Procedure auf dem Server nicht zwingend eine Administration der Clients erforderlich.

Performance

Aus physikalischen Gründen ist der Datentransfer über die Netztopologie immer langsamer als der direkte Plattenzugriff des Servers auf seine eigene Datenbank. Durch ‚interne' Datenzugriffsoperationen fällt somit die durch den Datenaustauscht zwischen Server und Client bedingte Verzögerung komplett weg. Bei aufwendigeren Datenmanipulationen bemerkt man den Geschwindigkeitsunterschied deutlich.

Sicherheit

Gespeicherte Prozeduren tragen dazu bei, die Sicherheit einer Anwendung stark zu erhöhen. Da der Client in der Regel für die in der Procedure ausgeführten Operationen keine DELETE-, INSERT- oder SELECT-Zugriffsrechte mehr benötigt. Auch ist es Angreifern nicht möglich, selbst SQL-Befehle durchzuführen (z.B. durch SQL-Injection). Bei entsprechender Rollenverteilung (EXECUTE privilege) hat der Client ausschließlich die Möglichkeit, bereits vorgefertigte Prozeduren aufzurufen.

---

[1]
Firebird	PSQL Procedure-SQL, teilweise auch Oracles PL/SQL
Oracle	PL/SQL (Procedural Language/SQL)
Informix	SPL (Stored Procedure Language)
IBM-DB2	SQL-PL, teilweise auch Oracles PL/SQL
MySQL	SQL 2003 / erst ab Version 5.0 Stored Procedures
PostgreSQL	PL/pgSQL und diverse eigenen Konstrukte

Proceduren werden oft auch PSM (Persistent Stored Modules) bezeichnet.

Aufbau von Stored Procedures

Der DDL-Syntax zum Anlegen oder Ändern von Procedures ist.

```
CREATE [ OR ALTER ] PROCEDURE <proc_name>
   [ ( <input_params> ) ]
   [ RETURNS ( <output_params> ) ]
   AS
   BEGIN
    <psql_body>
   END
```

wobei

proc_name

> der Name der Procedure ist. Da eine Procedure über ihren Namen aufgerufen wird, muss der Name in der Datenbank eindeutig sein

input_params

> In professionelleren Systemen lassen sich die Procedure mit Parametern aufrufen. Sie sind somit in ihrer Flexibilität den Funktionen in den Clients gleichgestellt.

output_params

> Ausgabeparameter. Die Procedure kann keinen Datensatz, einen Datensatz oder beliebig viele Datensätze liefern. Sie kann für den Anwender wie eine ‚normale' Tabelle aussehen.

psql_body

> Programmanweisungen in SQL und zusätzlichem Sprachumfang (PSQL). Viele Datenbanksysteme erlauben auch die Verwendung internen Variablen, beispielsweise für Zwischenergebnisse.

Beispiel

SP_TOOL_ALTER:

Berechne das Alter einer Person zum Stichtag ausgehend vom Geburtsdatum.
Ist kein Stichtag angegeben, dann berechne das Alter zum heutigen Tag.

Mögliche Aufrufe währen

```
SELECT SP.ALTER FROM SP_TOOL_ALTER(GEBURTSTAG,STICHTAG) SP;
```

oder

```
EXECUTE PRECEDURE SP_TOOL_ALTER(GEBURTSTAG,STICHTAG)
RETURNING_VALUES(ALTER);
```

Beispiel

SP_TOOL_ALTER:

Berechnet wird die Jahresdifferenz zwischen Geburtstag und Stichtag. Wird der Stichtag nicht übergeben (NULL), dann wird das aktuelle Datum ermittelt (CURRENT_DATE). Liegt kein Geburtsdatum vor, dann wir die Procedure abgebrochen (EXIT).

Alle die im aktuellen Jahr vor dem Stichtag Geburtstag hatten, werden mit Ihrem aktuellen Alter dargestellt. Alle die erst nach dem Stichtag Geburtstag haben natürlich mit dem noch um eins verminderte Altersangabe. Bei Geburtstagen oder Stichtagen am 29. Februar im Schaltjahr geboren sind, wird außerhalb von Schaltjahren der 28. Februar zu Grunde gelegt.

```
------------------------------------------------------------------

CREATE OR ALTER PROCEDURE SP_TOOL_ALTER (
  IN_DATUM_G DATE,
  IN_DATUM_S DATE)

RETURNS (
  RET_ALTER INTEGER)

AS
------------------------------------------------------------------
BEGIN

RET_ALTER = 0;

IF(IN_DATUM_G IS NULL) THEN EXIT;
IF(IN_DATUM_S IS NULL) THEN IN_DATUM_S = CURRENT_DATE;

IF(   EXTRACT(MONTH FROM IN_DATUM_S) < EXTRACT(MONTH FROM IN_DATUM_G)
  OR (EXTRACT(MONTH FROM IN_DATUM_S) = EXTRACT(MONTH FROM IN_DATUM_G)
  AND EXTRACT(DAY FROM IN_DATUM_S)   < EXTRACT(DAY FROM IN_DATUM_G)) )

THEN BEGIN
  RET_ALTER = EXTRACT(YEAR FROM IN_DATUM_S)
            - EXTRACT(YEAR FROM IN_DATUM_G) -1;

  IF   (EXTRACT(MONTH FROM IN_DATUM_G) = 2
    AND EXTRACT(DAY FROM IN_DATUM_G)   = 29
    AND EXTRACT(MONTH FROM IN_DATUM_S) = 2
    AND EXTRACT(DAY FROM IN_DATUM_S)   = 28)
    THEN RET_ALTER = RET_ALTER +1;

END
ELSE
  RET_ALTER = EXTRACT(YEAR FROM IN_DATUM_S)
            - EXTRACT(YEAR FROM IN_DATUM_G);


------------------------------------------------------------------
SUSPEND;

END
```

Beispiel einer Stored Procedure zur Berechnung der bundesdeutschen Feiertage in Abhängigkeit vom Bundesland. Übergeben wird der gewünschte Zeitraum ‚von' Jahr (**IN_JAHR)** ‚bis' Jahr **(IN_JAHRBIS)**.

Feiertage sind entweder zu einem bestimmten, jährlich wiederkehrenden Datum wie dem 3. Oktober oder 24. Dezember festgelegt oder werden nach der Osterformel für jedes Jahr neu berechnet. Für jeden Eintrag liegt dessen Bezeichnung (Pfingstsonntag) und die Berechnungsformel (Osterdifferenz, Jährlich wiederkehrend, Mittwoch zwischen 2 Daten) und eine Farbwert (realisiert durch eine Bitmap mit einem Pixel) in einer Tabelle (tblKalender). In Brandenburg gibt es sogenannte Hochfeiertage die bei der Berechnung von Feiertagszuschlägen herangezogen werden. Fällt beispielsweise ein beweglicher Feiertag auf einen Sonntag so ist es dort kein ‚normaler' Sonn- oder Feiertag (Warum auch immer??)

Mit /**/ gekennzeichneten Texte sind nicht Teil der Procedure sondern dienen dem Verständnis.

```
CREATE PROCEDURE SP_TOOL_DATUM_FEIERTAGE (
  IN_JAHR    SMALLINT,
  IN_JAHRBIS SMALLINT)
RETURNS (
  KALENDEREINTRAG      VARCHAR(50),
  KALENDERFARBEBMP     BLOB SUB_TYPE 0 SEGMENT SIZE 80,
  KALENDERFARBEWERT    INTEGER,
  KALENDERART          SMALLINT,
  KALENDERARTTAG       SMALLINT,
  KALENDERARTMONAT     SMALLINT,
  KALENDERARTJAHR      SMALLINT,
  KALENDERDATUM        DATE,
  KALENDERHOCHFEIERTAG SMALLINT)
AS
DECLARE VARIABLE IJAHR SMALLINT;
DECLARE VARIABLE SSQL VARCHAR(1024);
DECLARE VARIABLE INC SMALLINT;
DECLARE VARIABLE DOSTERN DATE;
DECLARE VARIABLE DOFFSET DATE;
DECLARE VARIABLE IKALENDEROSTERDIFF SMALLINT;
DECLARE VARIABLE IKALENDERSONDER SMALLINT;
DECLARE VARIABLE IKALENDERJAEHRLICH SMALLINT;
DECLARE VARIABLE SKALENDERBUNDESLAND VARCHAR(10);
DECLARE VARIABLE ISTAMMID INTEGER;
DECLARE VARIABLE DGEBURTSTAG DATE;
DECLARE VARIABLE DAUSTRITT DATE;
DECLARE VARIABLE SVORNAME VARCHAR(50);
DECLARE VARIABLE SNACHNAME VARCHAR(50);
DECLARE VARIABLE SSUCHCODE VARCHAR(100);

BEGIN

IF(COALESCE(IN_JAHR,0) = 0) THEN EXIT;
IF(COALESCE(IN_JAHRBIS,0) = 0) THEN IN_JAHRBIS = IN_JAHR;

/* Lese das eingestellte Bundesland aus der Tabelle in Variable:
   SKALENDERBUNDESLAND */
SELECT K.KALENDERTAG FROM TBLKALENDER K
WHERE K.KALENDEREINTRAG = 'BXX' INTO :SKALENDERBUNDESLAND;
IF(SKALENDERBUNDESLAND IS NULL) THEN SKALENDERBUNDESLAND = 'B07';
```

```
-------------------------------------------------------------------------
IJAHR = IN_JAHR;

/* Schleife über den gewünschten Jahreszeitraum mittels Zählvariable IJAHR */
WHILE(IJAHR <= IN_JAHRBIS) DO BEGIN

-------------------------------------------------------------------------

/* Rufe die Procedure Osterberechnung für das aktuelle Jahr auf */
  EXECUTE PROCEDURE SP_TOOL_DATUM_OSTERN(:IJAHR)
  RETURNING_VALUES :DOSTERN;

/* SSQL = Stringvariable zur Aufnahme eines dynamischen SQL  */
  SSQL = 'SELECT K.KALENDEREINTRAG, K.KALENDERFARBEBMP,';
  SSQL = SSQL || ' K.KALENDERFARBEWERT,K.KALENDERHOCHFEIERTAG,';
  SSQL = SSQL || ' K.KALENDERARTTAG, K.KALENDERARTMONAT,';
  SSQL = SSQL || ' K.KALENDERARTJAHR,';
  SSQL = SSQL || ' COALESCE(K.KALENDERJAEHRLICH,0),';
  SSQL = SSQL || ' COALESCE(K.KALENDERART,0),';
  SSQL = SSQL || ' COALESCE(K.KALENDERSONDER,0),';
  SSQL = SSQL || ' COALESCE(K.KALENDEROSTERDIFF,0)';

  SSQL = SSQL || ' FROM TBLKALENDER K';
  SSQL = SSQL || ' WHERE K.KALENDERART > 0';

  SSQL = SSQL || ' AND (COALESCE(K.' ||
                        :SKALENDERBUNDESLAND || ',0)';
  SSQL = SSQL || ' = -1);';

-------------------------------------------------------------------------
/* FOR -> Schleife aller Datensätze,
   EXECUTE  STATEMENT -> ausführen des SQL,
   INTO -> Ergebnisse der Abfrage stehen in den angegebenen Variablen */
  FOR
  EXECUTE STATEMENT :SSQL
  INTO :KALENDEREINTRAG, :KALENDERFARBEBMP, :KALENDERFARBEWERT,
       :KALENDERHOCHFEIERTAG, :KALENDERARTTAG, :KALENDERARTMONAT,
       :KALENDERARTJAHR, :IKALENDERJAEHRLICH, :KALENDERART,
       :iKALENDERSONDER, :iKALENDEROSTERDIFF

-------------------------------------------------------------------------
  DO BEGIN
-------------------------------------------------------------------------
/*-Jährliche Termine-*/
      IF   (IKALENDERJAEHRLICH = -1
        AND KALENDERART       <=  1
        AND iKALENDERSONDER    =  0) THEN BEGIN

      KALENDERDATUM = CAST(KALENDERARTTAG || '.'
                        || KALENDERARTMONAT || '.'
                        || IJAHR AS DATE);
      KALENDERARTJAHR = IJAHR;
      SUSPEND;

    END
-------------------------------------------------------------------------
```

```
/*-Kundenspezifische Termine-*/
    ELSE IF   (IKALENDERJAEHRLICH = 0
           AND KALENDERART        = 0
           AND iKALENDERSONDER    = 0) THEN BEGIN

      KALENDERDATUM = CAST(KALENDERARTTAG || '.'
                       || KALENDERARTMONAT || '.'
                       || IJAHR AS DATE);
      KALENDERARTJAHR = IJAHR;
      IF(KALENDERARTJAHR = IJAHR) THEN SUSPEND;

    END
-------------------------------------------------------------------------
/* Termine mit Osterdifferenz */
    ELSE IF(KALENDERART = 2 AND IKALENDERSONDER = 0) THEN BEGIN

      DOFFSET = DOSTERN + IKALENDEROSTERDIFF;

      KALENDERARTTAG   = EXTRACT(DAY FROM DOFFSET);
      KALENDERARTMONAT = EXTRACT(MONTH FROM DOFFSET);
      KALENDERARTJAHR  = IJAHR;
      KALENDERDATUM    = DOFFSET;

      SUSPEND;
    END
-------------------------------------------------------------------------
/*-Buß- und Bettag = Mittwoch zwischen 16. und 22. November */
    ELSE IF(KALENDERART = 3 AND iKALENDERSONDER = 0) THEN BEGIN

      INC = 16;
      WHILE(INC <= 22) DO BEGIN
        IF(EXTRACT(WEEKDAY FROM CAST(INC || '.11.'
                                  || IJAHR AS DATE)) = 3) THEN BEGIN
          DOFFSET = CAST(INC || '.11.' || IJAHR AS DATE);
          LEAVE;
        END
      END

      KALENDERARTTAG   = EXTRACT(DAY FROM DOFFSET);
      KALENDERARTMONAT = EXTRACT(MONTH FROM DOFFSET);
      KALENDERARTJAHR  = IJAHR;
      KALENDERDATUM    = DOFFSET;

      SUSPEND;
    END
  END
-------------------------------------------------------------------------
  IJAHR = IJAHR +1;
  END
END
```

Beispiel einer Stored Procedure am Beispiel einer Feiertagsberechnung

Programmieren = Abbilden der Realität

Um ein ‚Programm' vernünftig zu entwerfen ist ein Verständnis der Grundlagen und der Zusammenhänge nötig. Daher hier ein Beispiel, welche unterschiedlichen Informationen einer einfachen Formel zu Grunde liegen können. Durch die Einbeziehung von Informationen ergeben sich automatisch die Rahmenbedingungen.

Feiertagsberechnung bezogen auf das Osterfest.

Osterberechnung[2] nach Meton[3] und Gauß[4]

Das christliche Osterfest wird (fast überall) am ersten Sonntag nach dem Frühlingsvollmond, an einem Sonntag, gefeiert.

Berechnung des Osterfestes

Durch das Konzil von Nicäa (Konstantinopel, dem heutigen Istanbul) im Jahre 325 nach Christi wurde der Termin des Osterfestes auf den ersten Sonntag nach dem ersten Vollmond im Frühling festgelegt. Hierdurch entstand der Zusammenhang zwischen dem Osterfest und den Mondphasen. Da der erste Vollmond im Frühling jedes Jahr auf einem anderen Tag fällt, fällt auch Ostern jedes Jahr an einen anderen Sonntag.

Frühlingsanfang ist der 21. März. Ein am 21. März stattfindender Vollmond gilt bereits als frühestmöglicher Frühlings-Vollmond. Der 22. März ist deshalb der früheste Kalendertag, auf den Ostern fallen kann. Im Julianischen Kalender fällt der letzte mögliche Ostersonntag auf den 25. April. Diese Begrenzung wurde im Gregorianischen Kalender in einer Zusatzbestimmung beibehalten. Somit gibt es in beiden Kalendern insgesamt 35 verschiedene Ostertermine.

Ostern hat den Charakter eines beweglichen Feiertages. Das Osterfest spielt eine zentrale Rolle im Kirchenjahr, da von ihm fast alle beweglichen christlichen Feiertage wie Aschermittwoch, Christi Himmelfahrt oder Pfingsten abhängen.

Metonischer Zyklus

Aufgrund verschiedener Umlaufgeschwindigkeiten sind die Stellungen von Mond und Erde nicht jedes Jahr genau gleich. Der Grieche Meton erkannte bereits im 5. Jahrhundert v.Ch., dass alle 19 Jahre die gleiche Konstellation auftritt, weswegen man von einem 19-jährigen *Metonischen Zyklus* spricht (auch als *Mondzyklus* bekannt). Mit anderen Worten: Alle 19 Jahre wiederholen sich die Mondphasen an einem bestimmten Datum.

Die Nummer eines Jahres im Zyklus nennt man *Goldene Zahl*, welche sich wie folgt errechnet:

GZ = (Jahr Mod 19) +1

Die Zahl 1 muss hinzuaddiert werden, da die Goldene Zahl von 1 bis 19 definiert ist und durch die Modulo Rechnung (Restzahldivision) auch die 0 als Ergebnis vorhanden sein kann.

[2] Osterberechnung nach Konzils von Nicäa 325 n. Chr. in der Nähe von Konstantinopel, dem heutigen Istanbul.

[3] Der Grieche Meton (5. Jahrhundert v.Ch.) erkannte, dass alle 19 Jahre die gleiche Sonne, Erde, Mond Konstellation auftritt, weswegen man von einem 19-jährigen *Metonischen Zyklus* spricht - auch als *Mondzyklus* bekannt.

[4] Johann Carl Friedrich Gauß 1777 – 1855, deutscher Mathematiker, Astronom, Geodät und Physiker.

Sonnenzirkel

Im Julianischen Kalender hat das Jahr 365 Tage. Die jährlichen Abweichung müssen durch Schalttage (alle 4 Jahre ein Schaltjahr) ausgeglichen werden. Bei 365 Tagen kann man vereinfacht davon ausgehen, dass jedes Jahr mit dem gleichen Wochentag endet wie es anfängt, denn

365 Mod 7 = 1

Die Restzahldivision mit der Anzahl der Wochentage hat das Ergebnis 1. Daraus folgt, dass jedes Jahr mit dem gleichem Tag anfängt und endet.

Da aber Restzahldivision alle 4 Jahre wegen des Schaltjahres eine 2 ergibt, liegt kein 7-jähriger Zyklus (sieben Wochentage) sondern ein 4 x 7-jähriger = 28-jähriger Zyklus vor, welchen man als *Sonnenzirkel* bezeichnet. So fallen erst nach 28 Jahren die gleichen Wochentage wieder auf die gleichen Kalenderdaten.

Die Kalenderreform

Der Julianische Kalender wurde nach dem 4.10.1582 auf den Gregorianischen Kalender umgestellt. Bei dieser Umstellung wurden die aktuellen Abweichungen von den astronomischen Daten ausgeglichen. Gleichzeitig wurde versucht, eine erneute Abweichung so gut wie möglich zu unterbinden. Da der Julianische Kalender zu dieser Zeit bereits 10 Tage den astronomischen Daten hinterher hinkte, wurden bei der Reform einfach 10 Tage übersprungen. So folgte unmittelbar auf den Donnerstag 04.10.1582 der Freitag 15.10.1582.

Der entstandene Fehler war damit ausgeglichen, und es musste nur noch dafür gesorgt werden, dass kein erneuter Fehler entsteht. Daher muss in 400 Jahren dreimal ein Schalttag entfallen. Dieser Schalttag entfällt immer dann, wenn das Säkularjahr nicht restlos durch 400 teilbar ist. Ein *Säkularjahr* (lat. *saeculum*, Jahrhundert) ist ein Jahr, das ein Jahrhundert beschließt, wie beispielsweise das Jahr 1900.

PS: Allen die das neue Millennium 2000 gefeiert hatten sei hier gesagt:
Das neue Jahrhundert begann erst 2001.

In der Umstellung im Jahre 1582 liegt auch der Grund, dass viele Kalenderberechnungen nicht für die Berechnung von Daten vor 1582 geeignet sind.

Ostersonntag alle 532 Jahre

Die Ostergrenzen wiederholten sich also infolgedessen alle 19 Jahre, während die Wochentage im Julianischen Kalender nach 28 Jahren wieder auf die gleichen Daten fielen.

Somit wiederholte sich die Abfolge der Daten des Ostersonntages alle 19 x 28 = 532 Jahre. Dieser Zyklus wurde der Dionysische (nach Dionysius Exiguus) oder Victorianische (nach Victorius) genannt. Auch findet man, dass das frühste mögliche Osterdatum der 22. März ist, das späteste hingegen der 25. April.

Mit aufwendig erstellten Tabellen, die die Goldene Zahl und die Wochentage anzeigten wurden nun die Mondzyklen berechnet. Übrigens: ein Mondzyklus beträgt 29,53059 Tage.

Formeln zur Berechnung des Ostersonntags

Die kompliziert anmutenden Regeln zur Osterberechnung in einfach handhabbare Formeln umzusetzen ist eine Aufgabe, der sich seit der Einführung der zyklischen Osterberechnung unzählige Gelehrte gestellt haben. Am bekanntesten sind wohl die von Carl Friedrich Gauß aufgestellten Formeln

Zitat[5]:
„Die Absicht dieses Aufsatzes ist nicht, das gewöhnliche Verfahren zur Bestimmung des Osterfestes zu erörtern, das man in jeder Anweisung zur mathematischen Chronologie findet, und das auch an sich leicht genug ist, wenn man einmahl die Bedeutung und den Gebrauch der dabey üblichen Kunstwörter, güldne Zahl, Epakte, Ostergränze, Sonnenzirkel und Sonntagsbuchstaben weiß, und die nöthigen Hülfstafeln vor sich hat: Sondern vor dieser Aufgabe eine von jenen Hülfsbegriffen unabhängige und bloß auf den einfachsten Rechnungs-operationen beruhende rein analytische Auflösung zu geben.“

Aus den oben genannten Parametern ergeben sich nachfolgende Rechenregeln:

```
a = Jahr Mod 19
b = Jahr Mod 4
c = Jahr Mod 7
d = (19 * a + M) Mod 30
e = (2 * b + 4 * c + 6 * d + N) Mod 7
```

Die Variable e ist die Anzahl an Tagen, die nach der Ostergrenze vergehen müssen, bis wieder Sonntag ist. Gauß wählte den 21.03.1700 und folgerte, dass der Ostersonntag der

(22 + d + e).te März sein müsse.

Ostersonntag ist der (22 + *d* + *e*). März bzw. der (*d* + *e* − 9). April.

Zusätzlich gelten für den Gregorianischen Kalender zwei Ausnahmen:

1. Ist das Ergebnis der 26. April, so fällt der Ostersonntag auf den 19. April.

2. Ist *d* > 26, *e* = 6 und ((11 x M + 11) Mod 30) < 19, oder (d = 28), (a > 10) so ist Ostern nicht der sich nach den Formeln ergebende 25. April, sondern stattdessen der 18. April.

Von der ersten Ausnahme sind z. B. die Jahre 1609, 1981, 2076 und 2133 betroffen, von der zweiten die Jahre 1954, 2049 und 2106

Vom Datum des Ostersonntages hängen einige weitere Feiertage ab. Faschingsdienstag und Aschermittwoch liegen 47 bzw. 46 Tage vor Ostern, Christi Himmelfahrt 39 Tage nach Ostern, Pfingstmontag 50 Tage danach, und Fronleichnam wird 60 Tage nach Ostern gefeiert.

[5] Dr. Carl Friedrich Gauß (1777-1855) - Berechnung des Osterfestes. Veröffentlicht in: Monatliche Correspondenz zur Beförderung der Erd- und Himmels-Kunde, August 1800

M und N sind Konstanten für den Julianischen Kalender, während sie für den Gregorianischen Kalender Hilfszahlen sind, die für bestimmte Zeiträume gelten. M beinhaltet die Anzahl an Tagen in einem bestimmten Jahrhundert die nach dem 21.3. bis zur Ostergrenze verstreichen müssen (nächster Vollmond).

Für den Julianischen Kalender ist M = 15 und N = 6.
Für den Gregorianischen Kalender gibt Gauß die Hilfszahlen M und N bis 2499 explizit an.

Zeitraum	*M*	*N*
1582-1699	22	2
1700-1799	23	3
1800-1899	23	3
1900-1999	24	5
2000-2099	24	5
2100-2199	24	6
2200-2299	25	0
2300-2399	26	1
2400-2499	25	1

Die Werte können auch rechnerisch bestimmt werden. Formeln zur Berechnung von M und N:

Es sei 'J' die Zahl der vollen Jahrhunderte, also für 1993 ist J =19,
dann ergeben sich M und N wie folgt:

M = 10 + J - int[J/4] - int[(J-14 - int[(J+8)/25)] /3]
N = Mod[(4 + J - int[J/4]) /7]

Ostern (fast) überall

Als in Deutschland die zyklische Osterberechnung auch von den Protestanten übernommen wurde, behielt Schweden eine "astronomische" Rechnung bei. Erst seit 1845 wird auch in Schweden der Ostertag zyklisch berechnet. Die Osterdaten in Schweden stimmen seit 1753 mit den gregorianischen überein, außer in den Jahren 1802, 1805 und 1818, in denen Ostern in Schweden eine Woche später als der zyklische Termin gefeiert wurde.

Auch für Finnland gibt es einige Besonderheiten zu beachten. Bis 1809 war Finnland Teil Schwedens und feierte Ostern mit diesem. Ebenso wurde der Gregorianische Kalender mit Schweden 1753 eingeführt. 1809 fiel Finnland an Russland, in dem noch der Julianische Kalender galt. Trotzdem blieb in Finnland der neue Kalender gültig, wobei man Ostern 1825, 1829 und 1845 eine Woche später als nach dem Gregorianischen Kalender feierte.

```
CREATE PROCEDURE SP_TOOL_DATUM_OSTERN (
  IN_JAHR INTEGER)
RETURNS (
  RET_DATUM DATE)
AS
DECLARE VARIABLE I_MONAT INTEGER;
DECLARE VARIABLE I_TAG INTEGER;
DECLARE VARIABLE I_MOND INTEGER;
DECLARE VARIABLE I_PHASE INTEGER;
DECLARE VARIABLE I_SONNTAG INTEGER;
DECLARE VARIABLE I_GOLD INTEGER;
DECLARE VARIABLE I_DECADE INTEGER;
DECLARE VARIABLE I_KALX INTEGER;
DECLARE VARIABLE I_KALZ INTEGER;
DECLARE VARIABLE I_TMP INTEGER;
BEGIN
  /* Die GOLDENE ZAHL alle 19 Jahre (Meton) */
  I_GOLD = MOD(IN_JAHR, 19);
  I_GOLD = I_GOLD + 1;
  /* Jahundert */
  I_DECADE = CAST(IN_JAHR/100-0.49 AS INTEGER) + 1;
  /* Anzahl Jahre mit Schaltjahr */
  I_KALX = (3 * I_DECADE);
  I_KALX = CAST(I_KALX/4-0.49 AS INTEGER) -12;
  /* Korrektur der Mondstellung an Ostern */
  I_KALZ = (8 * I_DECADE + 5);
  I_KALZ = CAST(I_KALZ/25-0.49 AS INTEGER) -5;
  /* Suche Sonntag */
  I_SONNTAG = (5 * IN_JAHR);
  I_SONNTAG = CAST(I_SONNTAG/4-0.49 AS INTEGER) - I_KALX -10;
  /* Vollmond Beachten */
  I_PHASE = (11 * I_GOLD +20 + I_KALZ - I_KALX);
  I_PHASE = MOD(I_PHASE, 30);
  IF (I_PHASE < 0) THEN I_PHASE = I_PHASE + 30;

  IF (((I_PHASE = 25) AND (I_GOLD > 11)) OR (I_PHASE = 24)) THEN
    I_PHASE = I_PHASE + 1;
  /* Suche Vollmond */
  I_MOND = 44 - I_PHASE;
  IF (I_MOND < 21) THEN I_MOND = I_MOND + 30;

  /* Naechster Sonntag */
  I_TMP = I_SONNTAG + I_MOND;

  I_TMP = MOD(I_TMP, 7);
  I_MOND = I_MOND + 7 - I_TMP;

  IF (I_MOND > 31) THEN BEGIN
    I_MONAT = 4;
    I_TAG = I_MOND - 31;
  END ELSE BEGIN
    I_MONAT = 3;
    I_TAG = I_MOND;
  END

  RET_DATUM = CAST(I_TAG || '.' || I_MONAT || '.' || IN_JAHR AS DATE);
  SUSPEND;
END
```