Datenbanken

Einführung in die Nutzung von Datenbanken

Vorlesung SS 2012

Fachbereich: ET/IT **Version:** SS12 (BA)- 1.3.2012

Gliederungsübersicht:

Literaturhinweise:

(1) Connolly, Th.; Begg, C.:
Database Solutions
Pearson Education Limited, 2004

(2) Date, C.J.:
An Introduction to Database Systems (Vol. I)
Addison-Wesley, 1990

(3) DuBois, Paul
MySQL Cookbook
O'Reilly, 2007

(4) Elmasri, R; Navathe, B.:
Fundamentals of Database Systems,
3. Auflage, Addision Wesley, 2000

(5) Elmasri, R; Navathe, B.:
Grundlagen von Datenbanksystemen
Pearson Education Limited, 2002

(6) Kifer, M.; Bernstein, A.; Lewis, Ph.:
Database Systems
Pearson Education Limited, 2005

(7) Lockemann, P.C.; Schmidt, J.W.:
Datenbank-Handbuch
Springer-Verlag, 1993

(8) Lockemann, P.C.;Krüger, G.;Krumm, H.:
Telekommunikation und Datenhaltung
Hanser-Verlag, 1993

(9) Radermacher, K.; Dürr, M.:
Einsatz von Datenbanksystemen
Springer-Verlag, 1990

(10) Urman, S.:
ORACLE PL/SQL Programming
Osborne McGraw-Hill, 1996

1. Einführung / Überblick

1.1 Datenbanken und ihre Eigenschaften

Datenbanken und Datenbanksystem spielen heute eine wesentliche Rolle in vielen Bereichen. Man unterscheidet:

- ⇨ traditionelle Datenbanken, bei denen die meisten Informationen in Textform oder als numerische Werte gespeichert sind.
- ⇨ Multimediale Datenbanken, in denen darüber hinaus Informationen auch in Form von Bilder, Videos und Audios gespeichert sind.
- ⇨ Geographische Informationssysteme, spezielle Datenbanken, in denen Landkarten, Wetterkarten oder Satellitenbilder gespeichert sind
- ⇨ Data-Warehouse-Anwendungen, spezielle große Datenbankmanagementsysteme, um Informationen aus sehr großen traditionellen Datenbanken für Entscheidungsprozesse zu extrahieren und/oder zu analysieren
- ⇨ Echtzeit- oder aktive Datenbanksysteme, die zur Steuerung von Industrie- und Fertigungsprozessen eingesetzt werden.

Eine Datenbank ist dabei eine Sammlung von Daten, die einen Ausschnitt der Welt beschreiben. Daten sind dabei bekannte Tatsachen, die gespeichert werden können und die eine (implizite) Bedeutung in dem betrachteten Ausschnitt haben.
Diese Definition einer Datenbank ist sehr allgemein gehalten, in den verschiedenen Lehrbüchern stößt man auf unterschiedliche Definitionen für den Begriff Datenbank, Die folgenden Eigenschaften, die für eine Datenbank gefordert werden, schränken diese sehr allgemein gehaltene Definition sinnvoll ein:

- Eine Datenbank stellt Aspekte der „realen Welt“ dar. Änderungen in dem betrachteten Ausschnitt (so genannte Miniwelt) müssen sich in der Datenbank widerspiegeln.
- Eine Datenbank ist eine logisch zusammenhängende Sammlung von Daten mit einer daraus ableitbaren Bedeutung, d.h. eine rein zufällige Sammlung von Daten ist demnach keine Datenbank
- Eine Datenbank wird für einen ganz bestimmten Zweck entworfen und erstellt. Sie wird von bestimmtem Benutzergruppen in zweckbezogenen Anwendungen verwendet.

Eine Datenbank kann jede beliebe Größe und Komplexität aufweisen. Unabhängig von der Größe gibt es aber stets ein Softwaresystem (d.h. eine Menge von Programmen), die zur Erstellung, Konstruktion und Manipulation einer Datenbank eingesetzt wird. Ein solches Softwaresystem heißt Datenbankmanagementsystem (DBMS). Es ist demnach ein generelles / universelles (d.h. unabhängig von der Miniwelt) Softwaresystem für die

- Definition, d.h. für die Festlegung der Typen, Strukturen und Einschränkungen für die zu speichernden Daten
- Konstruktion, d.h. für die physischen Speicherung auf dem vom DBMS kontrollierten Speichermedium
- Manipulation, d.h. für das Einfügen, Ändern und Löschen von Daten in der Datenbank bzw. das Zusammentragen von Informationen aus den in der Datenbank gespeicherten Daten

Datenbank und Datenbankmanagementsystem zusammen bilden das Datenbanksystem

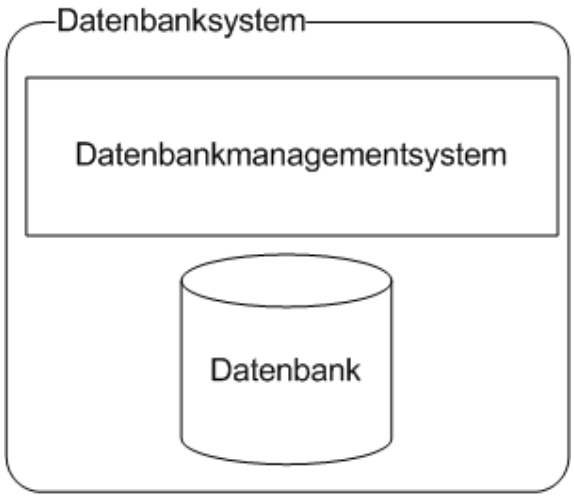
1.2 Datenbanksysteme und Datenbank-Management-Systeme

Ein **Datenbanksystem (DBS)** ist ein System zur elektronischen Datenverwaltung. Die wesentliche Aufgabe eines DBS ist es, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern und benötigte Teilmengen in unterschiedlichen, bedarfsgerechten Darstellungsformen für Benutzer und Anwendungsprogramme bereitzustellen.

Ein DBS besteht aus zwei Teilen: der Verwaltungssoftware, genannt **Datenbank-managementsystem** (**DBMS**) und der Menge der zu verwaltenden Daten, der eigentlichen **Datenbank**.

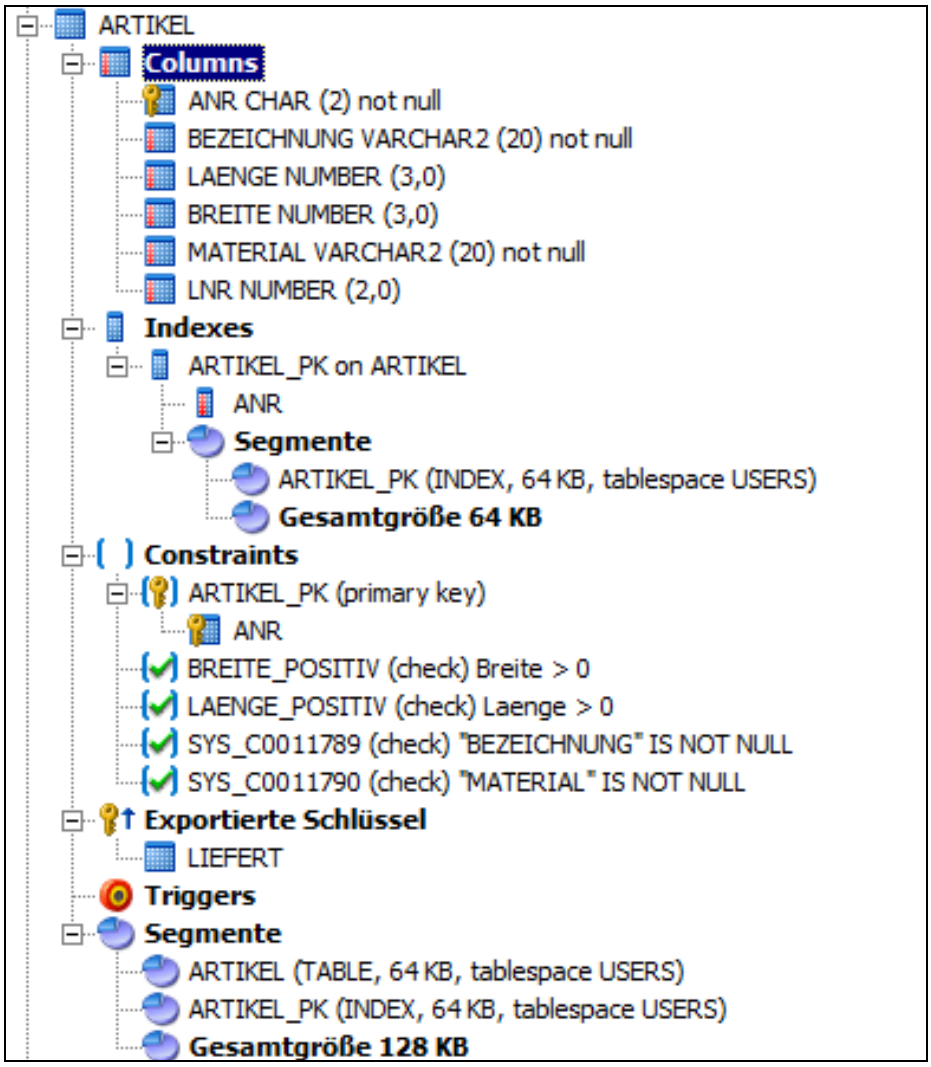
Die Verwaltungssoftware organisiert intern die strukturierte Speicherung der Daten gemäß eines vorgegebenen Datenbankmodells und kontrolliert alle lesenden und schreibenden Zugriffe auf die Datenbank. Als externe Schnittstelle stellt sie eine Datenbanksprache zur Formulierung von Abfragen, zum Einfügen und Ändern von Daten und für administrative Befehle zur Verfügung.

Die Datenbank enthält zusätzlich zu den eigentlichen Daten noch die Beschreibung der Daten (Struktur, Restriktionen, Zugriffsrechte, …), den so genannten Datenkatalog.



Die Datenbank enthält dabei die in der Datenbank gespeicherten Daten über den Ausschnitt der realen Welt, der der Datenbank zugrunde liegt und darüber hinaus aber auch im so genannten Datenkatalog (Metadaten) die Daten über Aufbau, Typen, Struktur und Einschränkungen der Daten.
Das Datenbankmanagementsystem operiert dabei auf beiden Teilen der Datenbank, d.h. ein grundlegendes Merkmal des Datenbankansatzes ist, dass ein Datenbanksystem nicht nur die eigentlichen Daten, sondern auch eine vollständige Definition deren Struktur und der Einschränkungen bezüglich der Daten enthält.
Eine Datenbank hat normalerweise viele Benutzer, die jeweils eigene Sichten auf die in der Datenbank gespeicherten Daten haben. Auch diese Sichten (so genannte Views) sind wieder im Datenbanksystem (im Datenkatalog) gespeichert.
Eine wesentliche Forderung an ein Datenbankmanagementsystem ist die Nebenläufigkeits-kontrolle (Concurrency Control), d.h. die Fähigkeit, Zugriffe (sehr) vieler Benutzer, die quasi gleichzeitig erfolgen, in „vernünftiger Weise“ zu koordinieren.

Datenkatalog (Struktur und Einschränkungen für die Tabelle Artikel)



Daten (Inhalt für die Tabelle Artikel)

	ANR	BEZEICHNUNG	LAENGE	BREITE	MATERIAL	LNR
1	R1	Regal	80	40	Metall	8
2	R2	Regal	80	60	Metall	8
3	S1	Schrank	100	60	Holz	6
4	S2	Schrank	80	40	Holz	6
5	T1	Tisch	80	60	Kunststoff	6
6	T2	Tisch	100	80	Kunststoff	8

Benutzergruppen

⇨ **Datenbankverwalter (Datenbankadministrator, kurz DBA)**
Dem DBA obliegt die Verwaltung einer oder mehrerer Datenbanksysteme. Jedes DBS ist dabei als Ressource in dem betreffenden Unternehmen oder der Organisation zu verstehen. Der DBA ist demnach der Ansprechpartner, wenn es um Fragen der Verfügbarkeit, der Sicherheit, der Performanz dieser Ressource bzw. der zugrunde liegenden Hard- und Software geht. Er richtet i. A. neue Benutzer ein, vergrößert oder verkleinert den von der Datenbank belegten physischen Speicher, regelt und überwacht die Abläufe zur Datensicherung und das Hoch- und Runterfahren des DBS und überwacht durch ein entsprechendes Monitoring die Arbeit des DBMS und versucht so, mögliche Störungen bereits früh zu erkennen und entsprechende Gegenmaßnahmen zu veranlassen.

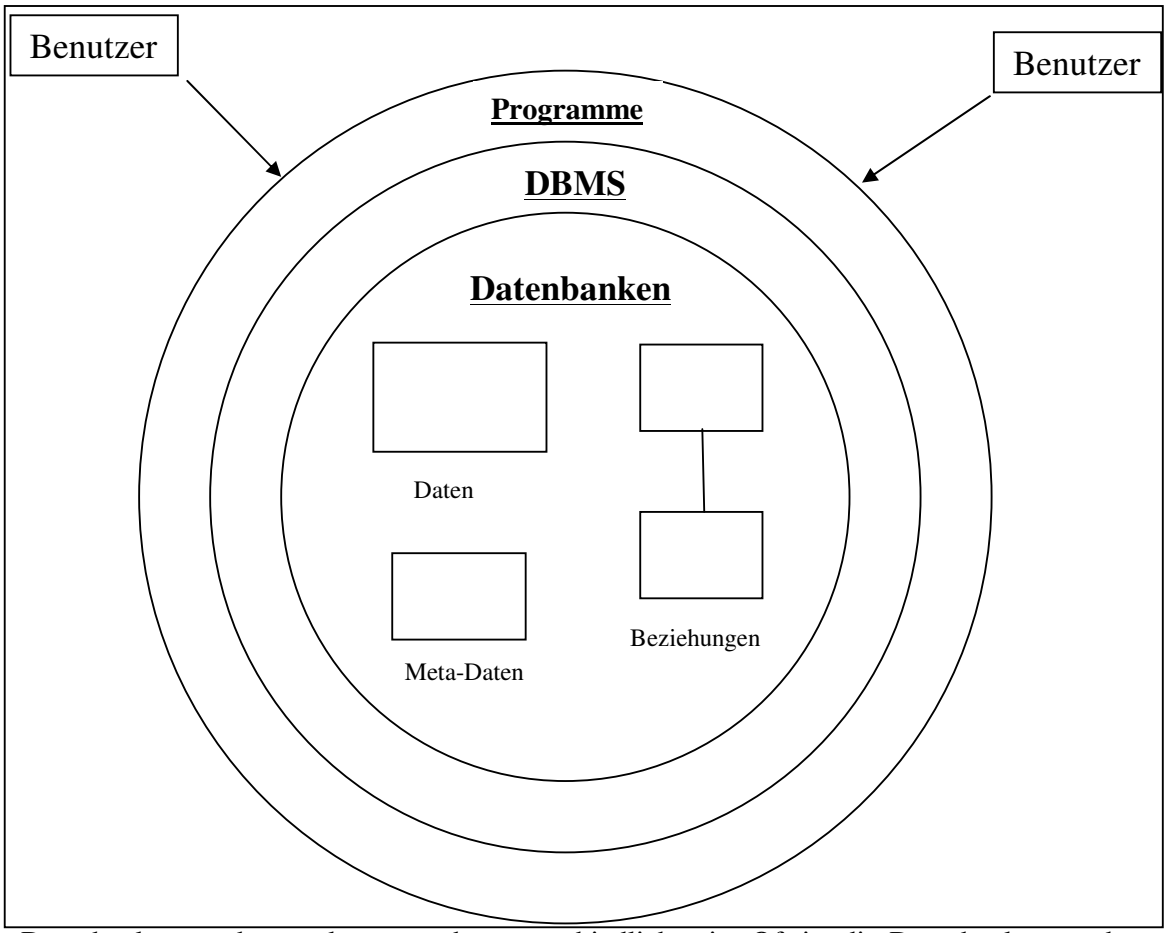
⇨ **Datenbankdesigner**
Der Datenbankdesigner ist für die Festlegung der in der Datenbank zu speichernden Daten und der Auswahl entsprechender Strukturen zu ihrer Darstellung und Speicherung zuständig. Dazu steht er im engen Kontakt mit allen potentiellen Datenbanknutzern, versucht deren Anforderungen und Wünsche in den Entwurf der Datenbank einfließen zu lassen. Der Datenbankdesigner arbeitet damit im Gegensatz zum DBA vorwiegend auf inhaltlicher Ebene, d.h. er verwaltet nicht eine Ressource wie der DBA, sondern entwickelt und realisiert den Entwurf für die Ressource DBS. Dabei ergibt sich natürlich häufig die Notwendigkeit zur engen Zusammenarbeit mit dem DBA (und umgekehrt) insbesondere wenn es um die physische Implementierung der Datenbank geht.

⇨ **Endbenutzer**
Dieser Personenkreis beschreibt die eigentlichen Nutzer des DBS. Die Endbenutzer greifen auf die in der Datenbank gespeicherten Daten zu, fügen neue Daten hinzu, verändern vorhandene Daten oder löschen Daten. Dabei lassen sich hier wieder mehrere Gruppen unterscheiden, je nach Art und zeitlicher Verteilung der Datenbanknutzung.

- Es gibt z.B. Endbenutzer die über vorgefertigte, standardisierte häufig wiederkehrende Abläufe auf die Datenbank zugreifen (ohne das ihnen Art und Umfang dieser Zugriffe im Detail bekannt sind, naive Endbenutzer), z.B. Mitarbeiter, die über ein Terminal Wareneingänge verbuchen oder auch Internetnutzer, die über Web-Anwendungen auf eine Datenbank zugreifen.
- Daneben gibt es aber auch gelegentliche Endbenutzer, die für eine einmalige Aufgabenstellung mit speziellen Werkzeugen Daten aus der Datenbank extrahieren, z.B. als Entscheidungsgrundlage oder zur Abschätzung von Investitionen oder zur Unterstützung der Fachabteilungen.
- Weiter sind Einzelbenutzer denkbar, die spezielle Datenbanksysteme (als persönliche) Datenbanksysteme führen. Sie sind dann oft auch gleichzeitig DBA und Datenbankdesigner für diese Systeme.
- Als professionelle Endbenutzer kann man vielleicht eine weitere Gruppe bezeichnen, die detaillierte Kenntnisse der vorhandenen Datenbanksysteme haben, gepaart mit komplexen, fachspezifischen Anforderungen, für die es (noch) keine standardisierten Lösungen gibt. Diese Endbenutzergruppe übernimmt dabei oft auch Aufgaben des Datenbankdesigners mit.

Definition: Datenbank-Anwendung

Eine Anwendungssoftware, die über ein DBMS auf Datenbanken zugreift und die darin gespeicherten Informationen für den Benutzer verfügbar macht.



Datenbankanwendungen können sehr unterschiedlich sein. Oft ist die Datenbankanwendung Teil einer Internet-Anwendung, d.h. als Benutzerschnittstelle wird ein Browser eingesetzt oder der Benutzer kommuniziert mit dem DBMS

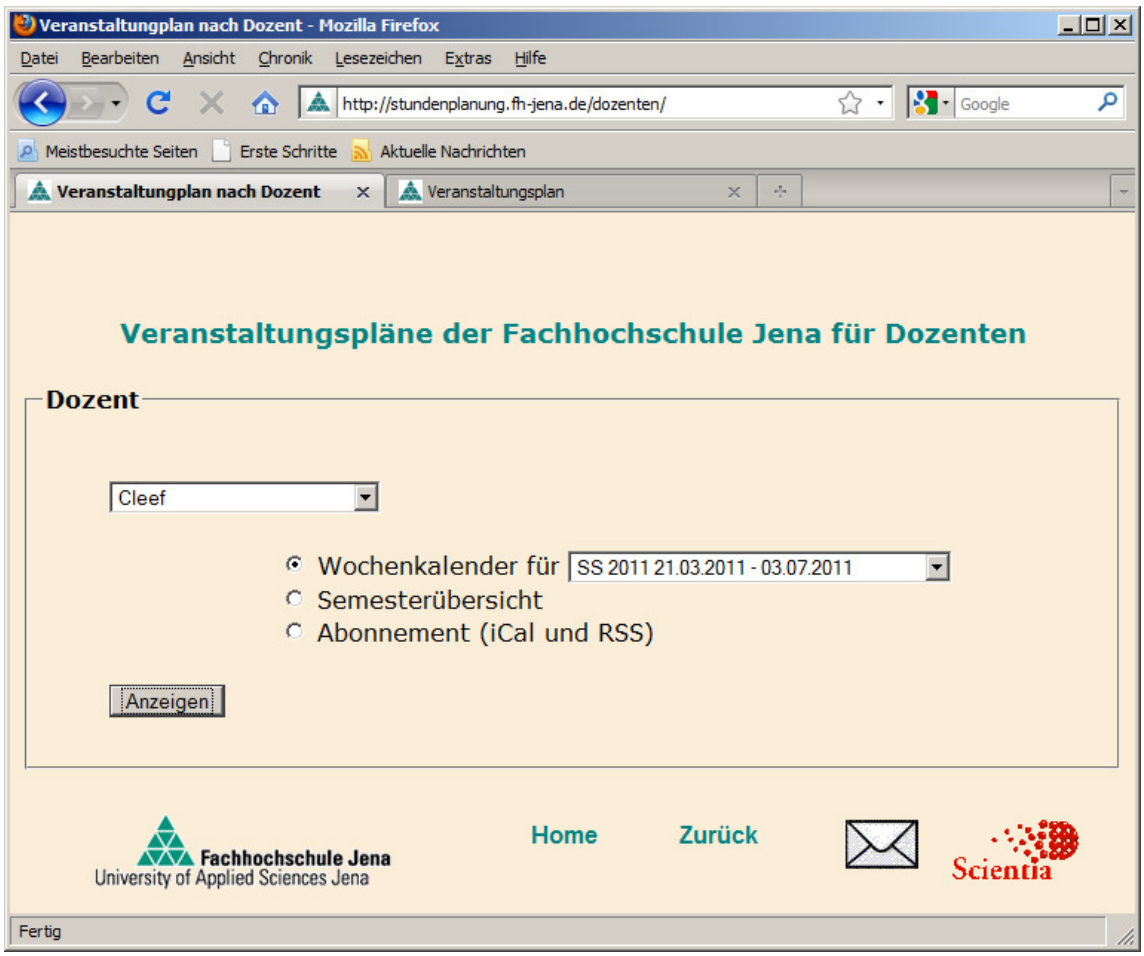
Übersicht über die im FB GW vorhandenen Datenbanksysteme:

- ⇨ Oracle Datenbank, Version 11g (DB-Server)
- ⇨ MySQL Datenbank, Version 5.1 (DB-Server)
- ⇨ MS Access (Enduser-DB)

Übersicht über die vorhandenen DB-Anwendungen

- ⇨ sqlplus (Shell-Progamm Oracle-DB)
- ⇨ Oracle Enterprise Manager (Oracle-DB)
- ⇨ Oracle SQL Developer
- ⇨ mysql (Shell-Programm MySQL-DB)
- ⇨ MySQL Administrator (MySQL-DB)
- ⇨ MySQL Query Browser (MySQL-DB)
- ⇨ SQL Developer 2.3 (Vielzahl von DB)
- ⇨ MS Access (Microsoft Enduser-DB)

Beispiel: Stundenplanung (Hochschulanwendung)

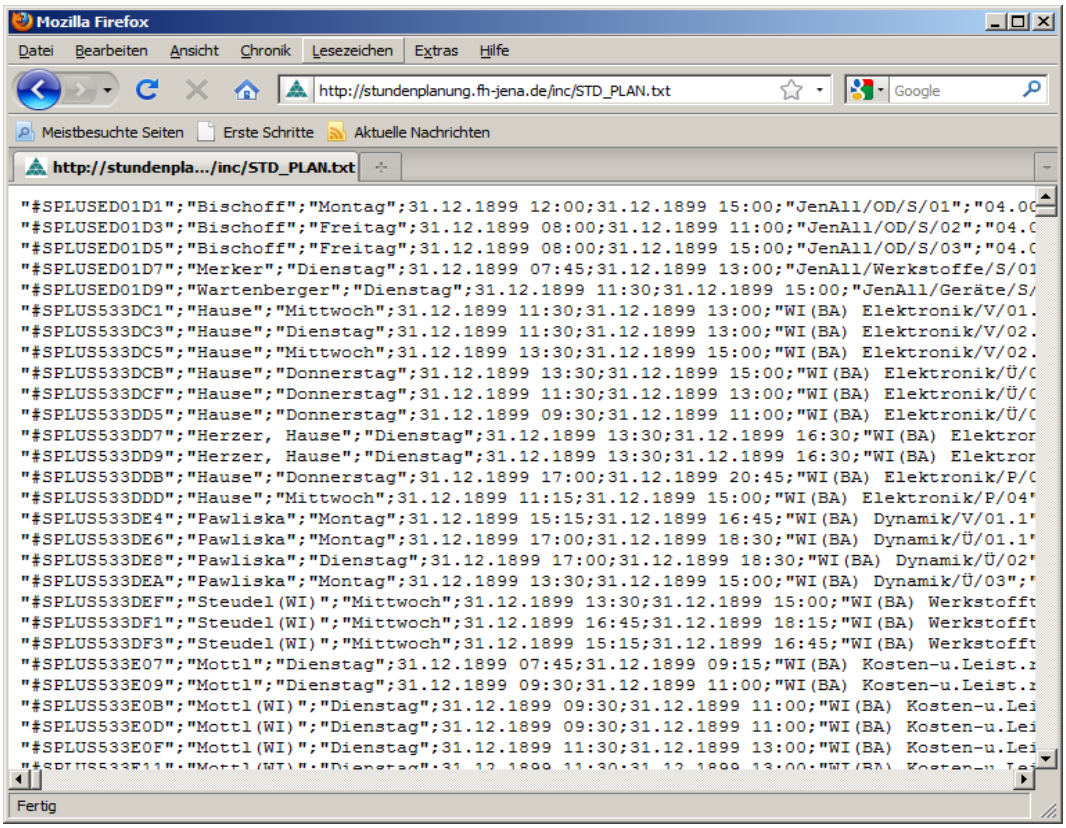


Dies ist eine vorgefertigte Anwendung, die von Endbenutzern ohne weitere Kenntnisse der zugrunde liegenden Datenbank benutzt werden kann, um Stundenplan-Informationen auszuwählen und anzuzeigen. Dabei können die Abfragen in gewissen Grenzen variiert werden. Es gibt jedoch keine Möglichkeit, abweichend von der gegebenen Abfrage(struktur), eigene individuelle Abfragen zu formulieren.

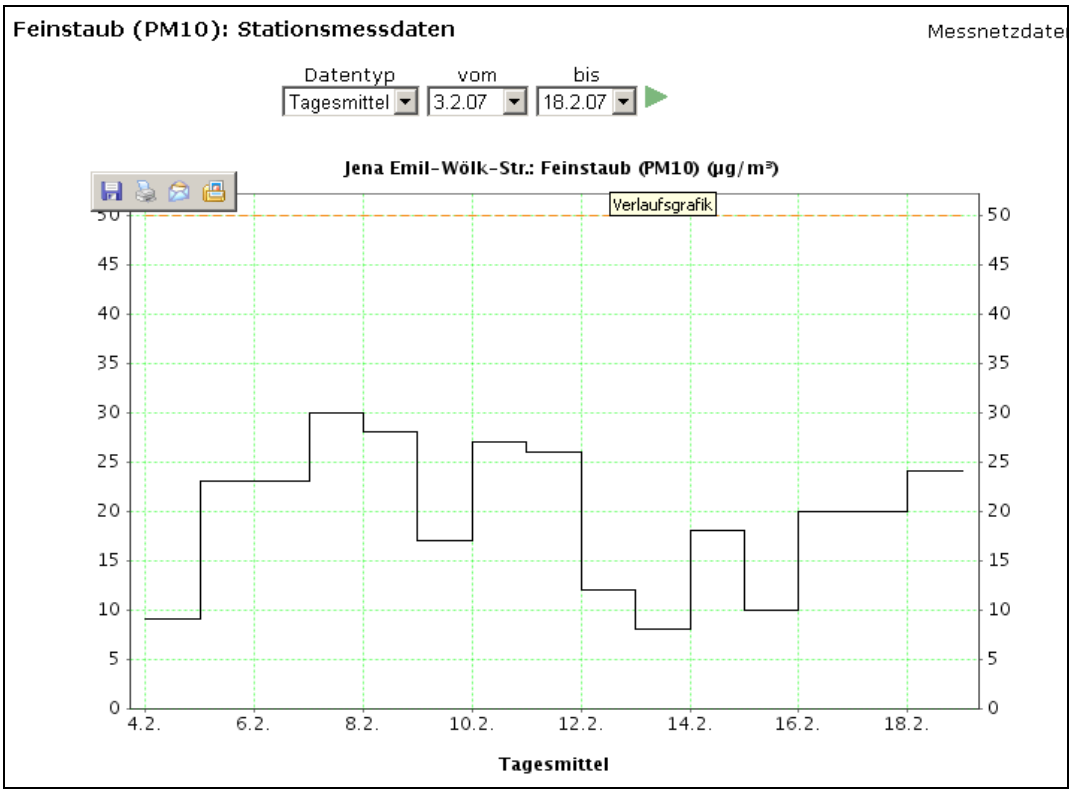
Darüber hinaus werden aber auch die gesamten Daten zur Stundenplanung in einer komprimierten (Tabellen)Form bereitgestellt. Diese kann herunter geladen werden. Damit sind dann beliebige individuelle Abfragen möglich. Jedoch veralten die Daten mit jeder relevanten Stundenplan Änderung bzw. müssen erneut herunter geladen werden, da kein mittelbarer Zugriff auf die Datenbank erfolgt.

Eine weitere Variante, die aber nur eingeschränkt für den mit der Stundenplanung betrauten Personenkreis angeboten wird, ist der direkte Zugang zum eingesetzten DBS.

Beispiel: Stundenplanung (Datei mit komprimierten Stundenplandaten)



Beispiel: Messwerte Feinstaubbelastung (Internet-Anwendung)



Hier liegt eine ähnliche Situation wie bei der Stundenplanung vor. Auch hier gibt es öffentlich zugängliche Zugriffsmöglichkeiten auf die gesammelten und in einer Datenbank gespeicherten Daten zur Feinstaubbelastung für die verschiedenen Messstationen.

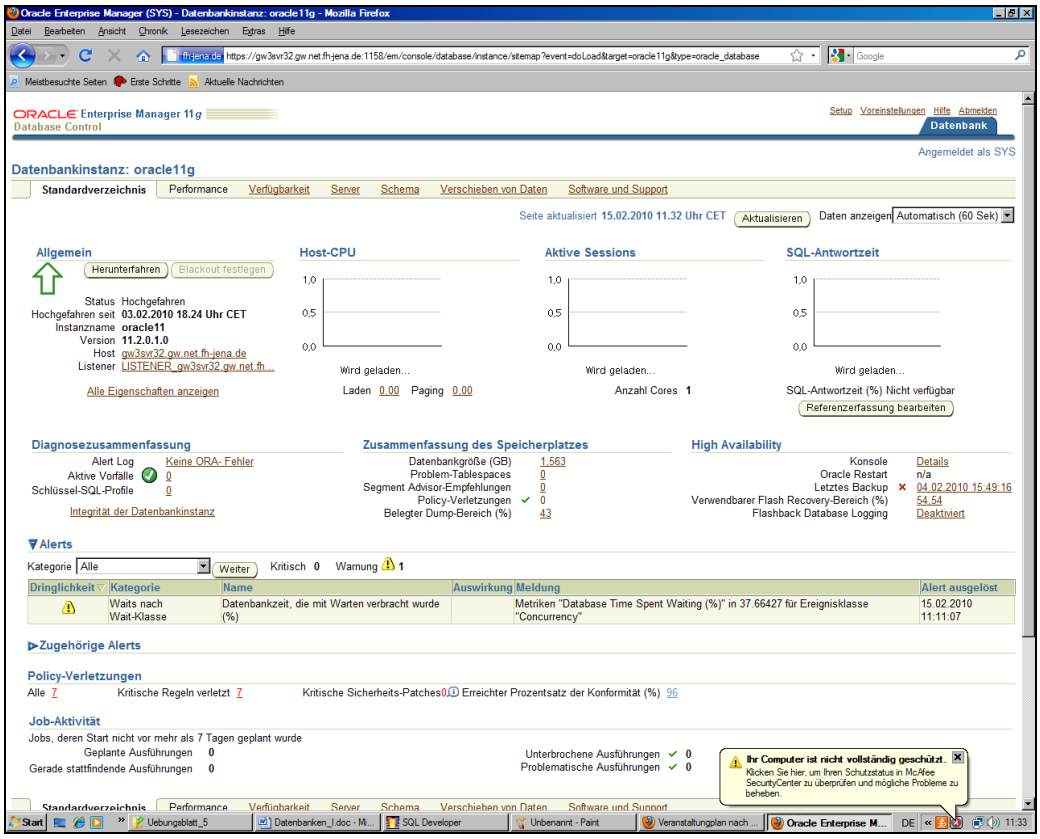
Auch hier gibt es jedoch die Möglichkeit, alle verfügbaren Messdaten als Datei herunter zuladen und dann individuell auszuwerten. Die so bereitgestellten Daten stellen jedoch wieder nur eine „Momentaufnahme“ des Datenbankzustandes dar, die möglicherweise schnell veralten kann, da auch hier wieder nicht direkt auf die zugrunde liegende Datenbank zugegriffen werden kann.

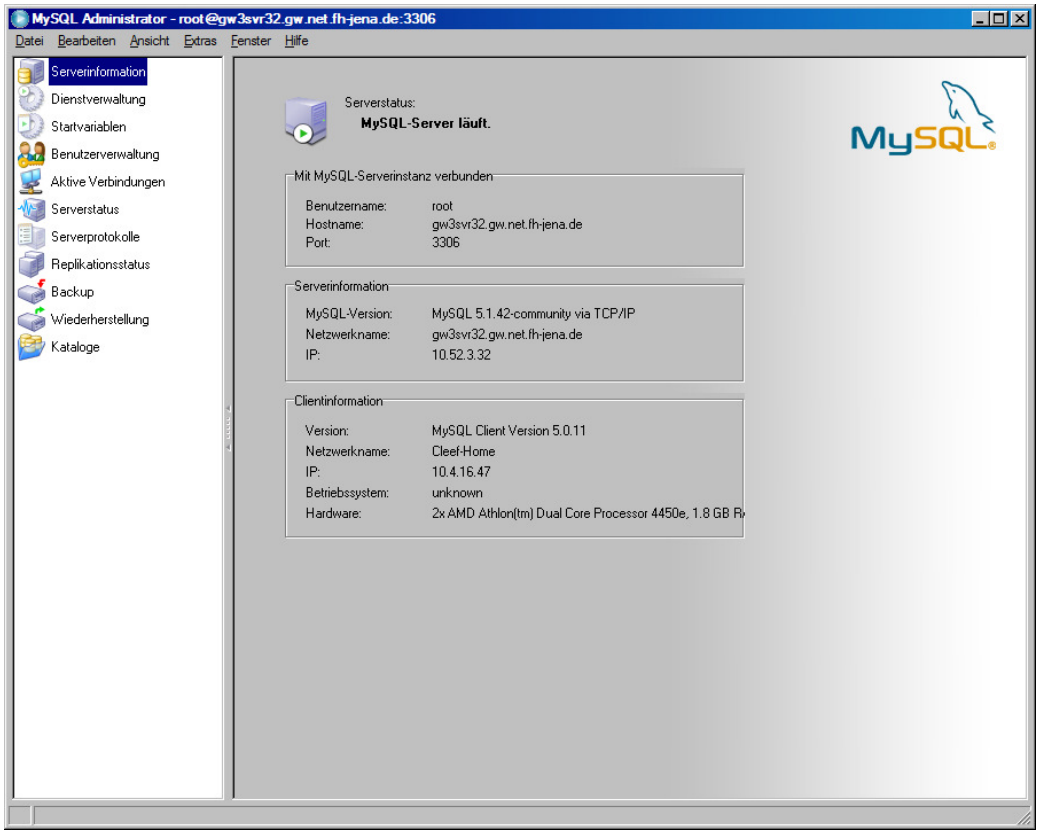
Beispiel: Klimastation FH Jena

Dies ist die Anwendung, die im Internet angeboten wird und zahlreiche vorgefertigte Auswertungen anbietet, die teilweise individuell mit speziellen Abfragewerten durch den Benutzer variiert werden können.

Für die Übungen wurden uns ähnlich wie bei den Stundenplandaten auch hier die Klimadaten in komprimierter Form als Textdateien für die Jahre 2006 bis 2010 zur Verfügung gestellt und können individuelle ausgewertet werden.

1.3 DBA-Anwendungen (Oracle Enterprise Manager, MYSQL Administrator)

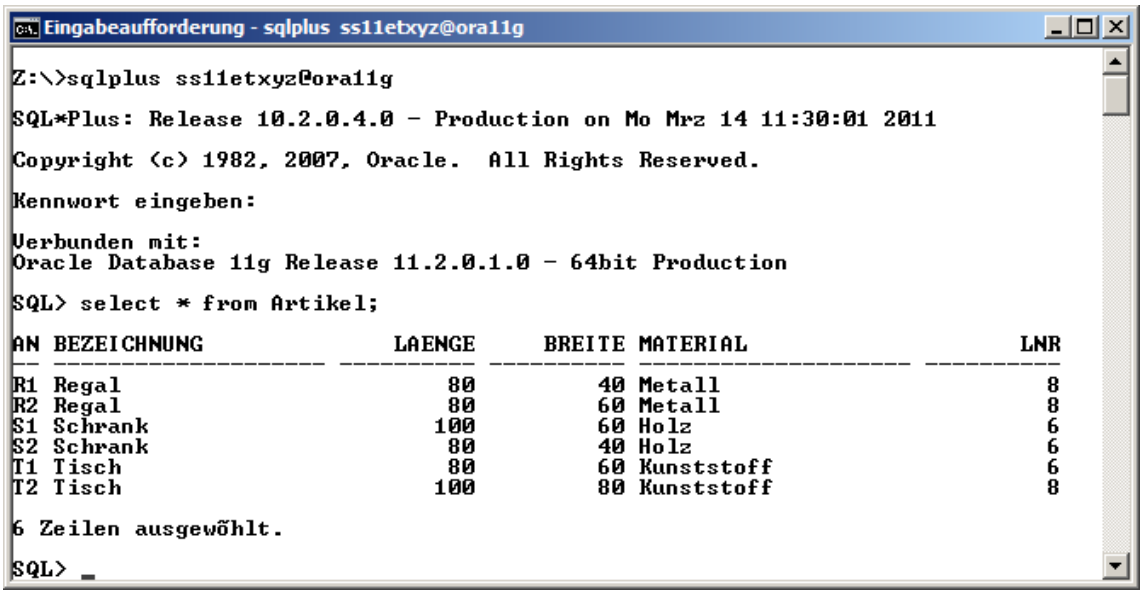




1.4 DB-Hilfsprogramme

Jedes Datenbanksystem verfügt über ein einfaches Hilfsprogramm zur Ausführung von Datenbank-Kommandos. Diese einfachen Hilfsprogramme sind Teil der DB-Software und stehen darüber hinaus auch als unabhängige Client-Anwendung zur Verfügung. Diese Anwendungen laufen in der Regel als „Konsolanwendungen“, benötigen also keine graphische Oberfläche. Über diese Schnittstelle werden i.A. auch die Kommandos zur System-Administration ausgeführt, oft aber aus einer graphischen Oberfläche heraus.

In der „Dialogvariante“ werden die Eingabe der einzelnen Kommandos und die Anzeige der vom Datenbankserver gelieferten Ergebnisse nicht in einem eigenen besonderen Fenster, sondern innerhalb der Eingabeaufforderung/Shell bearbeitet:



Daneben gibt es noch die „Batchvariante“, bei der alle auszuführenden Kommandos in einer SQL-Skriptdatei zusammengestellt wurden und diese dann im Hintergrund ausgeführt werden können. Eine solche SQL-Skriptdatei kann auch innerhalb der Dialogvariante ausgeführt werden.

Beispiele:
```
sqlplus ss11etxyz/geheim@ora11g                (Dialoganwendung)
sqlplus ss11etxyz/geheim@ora11g  @datei.sql    (Batchanwendung)
```

Wird eine Skriptdatei bereits beim Start angegeben (und nicht erst im Dialogfenster), muss diese am Ende als letzte Anweisung eine exit-Anweisung enthalten, fehlt die exit-Anweisung, so bleibt die Anwendung nach Ausführung der letzten Anweisung stehen und wartet auf weitere Anweisungen, die aber nicht eingegeben werden können.

Wird keine SQL-Skriptdatei angegeben, so startet das Programm im Dialogmodus und meldet sich mit dem SQL-Prompt im Zeilenmodus der Eingabeaufforderung. Danach können einzelne Anweisungen direkt eingegeben, ausgeführt und das Ergebnis angezeigt werden.

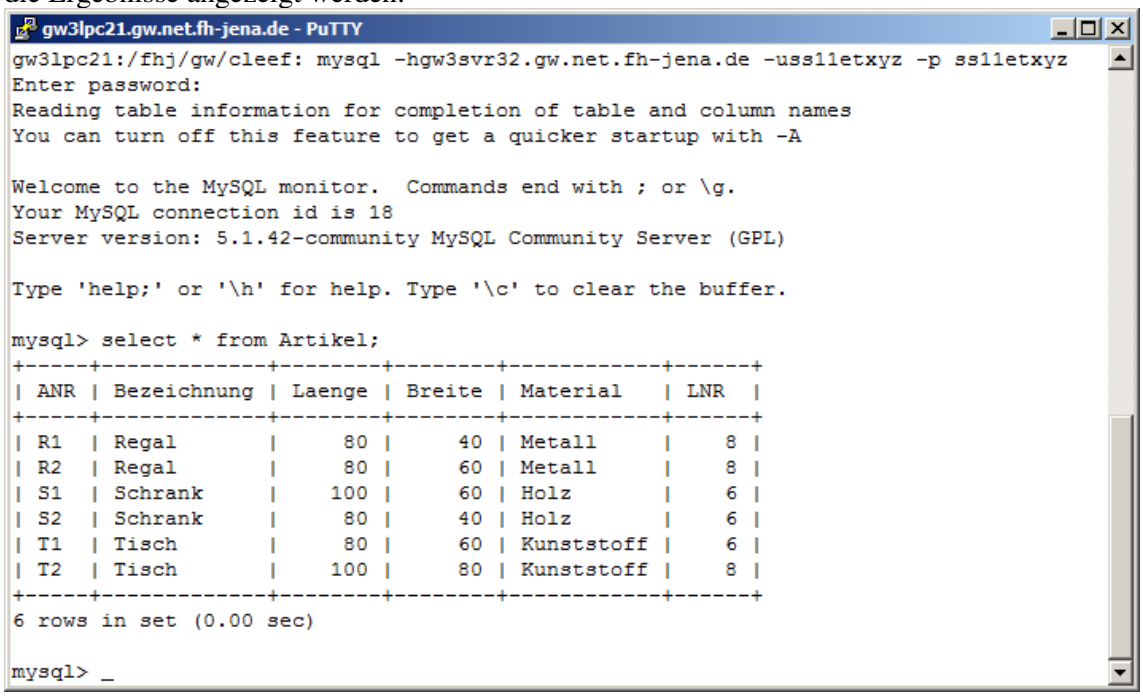
Neben den Datenbanken ORACLE und ACCESS steht als weitere Datenbank MYSQL zur Verfügung, für die anders als für Oracle und Access keine Lizenzgebühren zu zahlen sind. MYSQL ist für zahlreiche Plattformen erhältlich. Für die praktischen Übungen steht dabei im Hochschulnetz ein MySQL-Datenbankserver zur Verfügung:

Auch für die Datenbank MYSQL gibt es ein zu sqlplus analoges Programm mit Namen mysql und es unterstützt in analoger Weise den interaktiven (Dialogvariante) und nicht interaktiven (Batchvariante) Gebrauch für die Eingabe der Anweisungen und die Ausgabe der Ergebnisse. In der Aufrufsyntax unterscheiden sich die beiden Programme sqlplus und myql deutlich.

Beispiele:

mysql -h gw3svr32.gw.net.fh-jena.de -u ss11etxyz -p -D ss11etxyz (MYSQL-DB)
sqlplus ss11etxyz@ora11g (ORACLE-DB)

Hier wird jetzt auf dem MySQL-Server **gw3svr32.gw.net.fh-jena.de** als Benutzer **ss11etxyz** eine Verbindung zur Datenbank **ss11etxyz** aufgebaut. Der Aufruf im Beispiel erfolgt ohne die Umlenkung der Standardeingabe, es wird also die interaktive Programmversion gestartet, der mysql-Prompt angezeigt und es können im Dialog Anweisungen eingegeben, ausgeführt und die Ergebnisse angezeigt werden:



Beide Programme werden mit dem Kommando **exit** (oder **quit**) beendet. Beide Programme bieten die Möglichkeit, die Ausgabe in eine Datei umzulenken. Dies ist insbesondere für die Batchvariante von Interesse.

1.5 Merkmale, Vor- und Nachteile des Datenbankansatzes

⇨ **Redundanzkontrolle**

Beim Einsatz traditioneller Dateiverarbeitung pflegt jede Benutzergruppe ihre eigenen Dateien, auf die dann mit entsprechenden Anwendungen zugegriffen wird. Dabei werden häufig die gleichen Daten mehrfach geführt. Diese redundant geführten Daten müssen manuell gepflegt und konsistent gehalten werden. Dies ist zeit- und kostenaufwendig und insbesondere fehleranfällig. Beim Datenbankansatz werden diese Redundanzen entweder ganz vermieden oder (wenn z.B. aus Performanzgründen erforderlich) kontrolliert und maschinell durch ein DBMS gepflegt.

⇨ **Zugriffsbeschränkungen**

Beim Datenbankansatz wird ein aufwendiges Sicherheits- und Autorisierungssystem bereitgestellt, mit dem die Zugriffsrechte und mögliche Nutzungsbeschränkungen definiert festgelegt werden können. Der DBA verfügt dabei über besondere Zugriffsprivilegien, da er für die Vergabe und den Entzug von Zugriffsrechten zuständig ist. Beim Einsatz eines entsprechend ausgestatteten DBS kann sehr differenziert festgelegt werden, welche Benutzer welche Daten sehen darf, welche er ändern und welche er löschen darf. Eine solche zentral verwaltete Zugriffskontrolle trägt erheblich dazu bei, die Daten (z.B. eines Unternehmens oder einer öffentlichen Einrichtung) gegen Angriffe von innen und außen wirksam zu schützen. Die Benutzerkonten (Benutzername, Kennwort, Zugriffsrechte, ...) sind ebenfalls als Metadaten im DBS enthalten.

⇨ **Mehrbenutzerbetrieb**

Der Datenbankansatz unterstützt nicht nur unterschiedliche Benutzergruppen (mit gänzlich verschiedenen Kenntnissen und Fähigkeiten), sondern er ermöglicht für den konkurrierenden Zugriff von (sehr) vielen Benutzer, den Rückgriff auf robuste, erprobte und zentral im DBMS implementierte Algorithmen zur Nebenläufigkeitskontrolle. Diese Fähigkeit zur Kontrolle konkurrierender Zugriffe bietet der Datenbankansatz sowohl im Fall zentraler Datenhaltung als aber auch im Fall verteilter Datenhaltung.

⇨ **Integritätsbedingungen**

Die Konsistenz der Daten in der Datenbank wird vom DBMS überwacht und kontrolliert. Damit lassen sich beim Datenbankansatz Einschränkungen (so genannte Integritätsbedingungen) formulieren und wirkungsvoll vom DBMS überwachen, dies entlastet die Anwendungen von einer Vielzahl von Prüfungen und verhindert so auch wirkungsvoll, dass die Daten (z.B. durch fehlerhaft programmierte Anwendungen) nicht mehr konsistent sind. Dabei verweigert das DBMS die Ausführung jeder Transaktion, die zu inkonsistenten Daten führen würde. Durch die Möglichkeit, solche Integritäts-bedingungen durch den DBA zeitweilig ausschalten zu lassen, bleibt die Flexibilität aber erhalten.

⇨ **Backup und Recovery**

Der Datenbankansatz unterstützt wirkungsvoll die Transaktionskontrolle und hält durch entsprechende Mechanismen (Aufzeichnung der Änderung zur Wiederherstellung, langfristige Archivierung aller Änderungen) die Daten auch dann konsistent, wenn es aufgrund von Hardware- oder Softwarefehlern zum Absturz (d.h. zum unbeabsichtigten und unkontrolliertem Ende) von Anwendungsprogrammen oder des DBMS kommt Für das Verhalten beim Restart nach einem solchen Absturz werden unterschiedliche Möglichkeiten angeboten. Aus den Anforderungen der permanenten Verfügbarkeit von DBMS ergeben sich Forderungen, alle Maßnahmen zur Datensicherung (volle oder inkrementelle Sicherung der gespeicherten Daten) parallel zum normalen Benutzerbetrieb durchführen zu können.

⇨ **Nutzung von Standards**

Der Datenbankansatz fördert und unterstützt die Durchsetzung einheitlicher unternehmensweiter Standards bzgl. Namen, Ein- / Ausgabeformaten, Aufbau und Struktur von Daten, Fachbegriffe usw.

⇨ **Kürzere Entwicklungszeiten**

Der Datenbankansatz führt mittel- und langfristig zu einer Vereinfachung und insbesondere zu einer Verkürzung der Entwicklungszeit (zwischen dem Formulieren einer Anforderung und der Einführung der softwaretechnischen Umsetzung). Die Entwicklung von Datenbankanwendungen wird wirkungsvoll durch sehr leistungsfähige Werkzeuge unterstützt, die teilweise direkt vom Hersteller des DBS (mit den nötigen Detailkenntnissen des DBMS) oder aber auch von spezialisierten Herstellern solcher Werkzeuge kommen.

⇨ **Hohe Flexibilität**

Der Datenbankansatz gewährleistet eine hohe Flexibilität bei der Änderung von Aufbau und Struktur der gespeicherten Daten (Metadaten), aber auch des Umfeldes (Benutzer, Rollen, Sicherungsstrategien, Vergrößerung oder Verlagerung des physischen Datenbestandes, usw.). Alle diese Änderungen werden durch die Funktionalität des DBMS abgedeckt, es sind keine Änderungen in den Anwendungen erforderlich, sofern sie nicht auch von den Änderungen mittelbar betroffen sind. Diese Änderungen sind i. A. parallel zum laufenden Betrieb möglich, d.h. der Satzaufbau (z.B. für den Kundenstammsatz) kann erweitert werden, während quasi „gleichzeitig“ Anwendungen auf diesen Kundenstammsatz zugreifen.

⇨ **Hohe Verfügbarkeit**

Heutige Datenbankmanagementsysteme sind für den Dauerbetrieb konzipiert. Dies ergibt sich oft auch aus dem globalen Einsatz von Datenbankanwendungen im Internet. Während in Deutschland die Arbeitszeit zu Ende geht, beginnt sie irgendwo anders auf der Welt und umgekehrt, so dass man die Datenbank eigentlich zu keinem Zeitpunkt einfach herunterfahren kann. Durch entsprechende Investitionen (Replizierung, Spiegelung) in die Infrastruktur lässt sich heute der aus der Sicht des Benutzers störungsfreie Betrieb eines DBS mit sehr großer Wahrscheinlichkeit permanent sicherstellen.

Es ergeben sich aber auch möglicherweise Nachteile beim Datenbankansatz, auf die geachtet werden sollte:

⇨ **Hohe Anfangsinvestitionen**

Bevor man in den Nutzen der Vorteile eines DBS kommt, sind zunächst oft in erheblichem Umfang nicht nur in das DBS selbst, sondern auch in die Anwendungen (bereits vorhanden oder neu zu erstellen) zu investieren. Oft ist eine Lösung ohne DBS schneller und kostengünstiger verfügbar, da man erst nach dieser Anfangsinvestition in den Genuss kürzerer Entwicklungszeiten kommt. Hier ist sehr sorgfältig zu prüfen, ob sich diese Anfangsinvestition unter Kostengesichtspunkten, aber auch unter dem Aspekt der Verfügbarkeit „rechnet".

⇨ **Universalität des DBMS für die Definition und Verarbeitung der Daten**

Der sehr allgemeine (universelle) Datenbankansatz, der weitgehend unabhängig von der konkreten Aufgabenstellung ist, für die ein DBS eingesetzt werden soll, ergibt möglicherweise bei sehr spezialisierten Anwendungen, die vielleicht auch noch hohe Anforderungen an die Performanz und Effizienz stellen oder systembedingt nur sehr begrenzte Hard- /Software-Ressourcen haben, ein nicht zu verkraftenden Overhead beim Einsatz einer Datenbank. Hier ist ein Verzicht auf die Universalität zugunsten einer ganz speziell zugeschnittenen Lösung angebracht.

⇨ **Mehraufwand für Sicherheit, Nebenläufigkeit, Konsistenzsicherung und Backup / Recovery**

Der Datenbankansatz hat hier seine Stärken. Werden diese Anforderungen in einem konkreten Fall nicht gestellt (a priori kein Mehrbenutzerbetrieb, keine schützenswerten Inhalte oder sonstige (z.B. bauliche) Zugangssicherungen ohne Vernetzung), so entfallen damit ganz wesentliche Leistungsmerkmale, die beim Einsatz eines DBMS immer auch mitgeliefert werden. Auch hier stellt sich dann die Frage, ob der Overhead beim Einsatz einer Datenbanklösung gerechtfertigt ist, oder ob man nicht mit einer einfachen Dateilösung wesentlich besser fährt.

2. Relationale Datenbanken

2.1. Grundlagen der Relationen-Algebra

Definition: Relation

Eine Relation ist eine benannte Tabelle mit Spalten und Zeilen.

- ⇨ Die Begriffe Relation und Tabelle werden synonym benutzt.
- ⇨ Die Tabellenstruktur ist die „Außenansicht“ und muss nicht mit der tatsächlich gespeicherten Struktur übereinstimmen.
- ⇨ Die Relation wird durch einen Namen identifiziert. Innerhalb einer Datenbank bzw. innerhalb eines Datenbankschemas muss der Name einer Relation eindeutig sein.

Definition: Attribut

Ein Attribut ist eine benannte Spalte einer Relation.

- ⇨ Die Begriffe Attribute und Spalte werden synonym benutzt.
- ⇨ Der Name einer Spalte muss innerhalb einer Relation eindeutig sein.

Name	Ort	Status	PLZ	Telefon	Branche

Definition: Wertebereich (engl. domain)

Der Wertebereich beschreibt die Menge der möglichen / zulässigen Werte für ein Attribut.

- ⇨ Nicht alle Werte eines Wertesbereiches müssen auch tatsächlich vorkommen.
- ⇨ Wertebereiche werden häufig durch so genannte Datentypen beschrieben, typische Beispiele sind dabei:
  - numerische Datentypen (ganze Zahlen, Gleitkommazahlen unterschiedlicher Genauigkeit)
  - alphanumerische Datentypen (Zeichenketten fester bzw. variabler Länge, begrenzte Länge)
  - Datentypen für Datum/Zeit (mit entsprechender Konvertierung in die gebräuchlichen Formate, z.B.: 15.03.2011)
  - Datentypen zur Speicherung von Binärdaten (Audio, Video, Picture)

Definition: Grad

Die Anzahl der Attribute einer Relation, d.h. die Anzahl der Spalten einer Tabelle wird als Grad der Relation/Tabelle bezeichnet.

Definition: Tupel

Eine einzelne Zeile einer Relation (d.h. die Menge der Werte aller Attribute für einen Datensatz) wird als Tupel bezeichnet. Jedes Tupel stellt damit einen Datensatz der Relation dar.

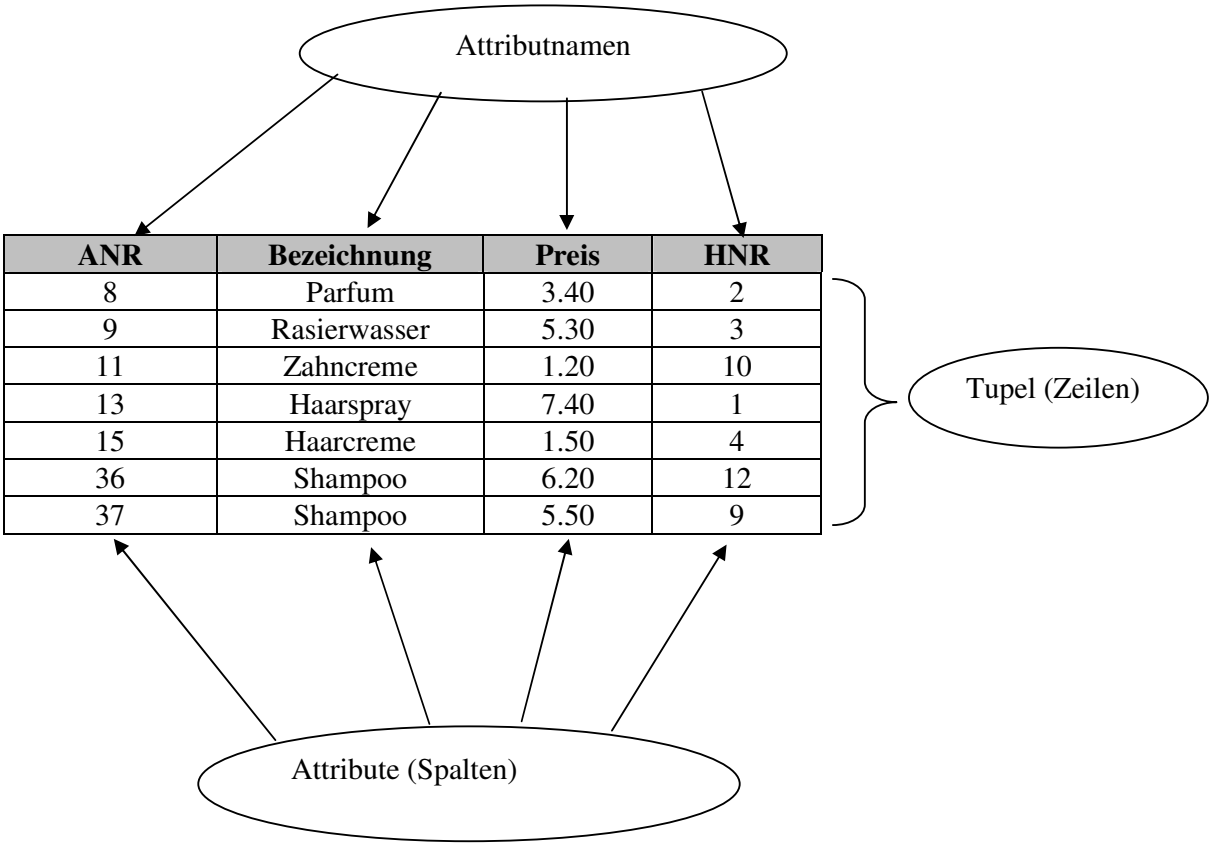
- ⇨ Die Begriffe Tupel, Zeile oder Datensatz (engl. record) werden synonym benutzt.
- ⇨ Tupel werden häufig auch wie Vektoren in der Mathematik geschrieben.
- ⇨ Die Reihenfolge der Spalten und Zeilen kann beliebig verändert werden, ohne dass sich dadurch die Relation verändert!

Definition: Kardinalität

Die Anzahl der Tupel einer Relation, d.h. die aktuelle Anzahl der Zeilen einer Tabelle wird als Kardinalität bezeichnet.

Beispiel:

Relation: **Erzeugnis**



Grad = 4 und **Kardinalität** = 7

Gegenüberstellung der Bezeichnungen:

Relationenalgebra	Umgangssprache
Relation	Tabelle
Attribut	Spalte
Attributname	Spaltenüberschrift
Wertebereich	Datentyp
Tupel	Zeile
Grad	Spalten(an)zahl
Kardinalität	Zeilen(an)zahl
leere Menge	NULL-Wert

Als Wertebereiche könnten z.B. folgende Datentypen verwendet werden:

ANR DECIMAL(2)

Damit wird das Attribut **ANR** als zweistellige Dezimalzahl (+ / -) festgelegt, der Wertebereich ist also die Menge {-99, -98, ….., -1, 0, 1, 2,……, 98, 99}.

Bezeichnung VARCHAR(30)

Damit wird das Attribut **Bezeichnung** als Text mit variabler Länge bis maximal 30 Zeichen (Buchstaben, Ziffern und Sonderzeichen) festgelegt.

Preis DECIMAL(5,2)

Damit wird das Attribut **Preis** als Gleitkommazahl mit maximal 3 Vor- und 2 Nachkommastellen festgelegt.

HNR DECIMAL(2)

Damit wird das Attribut **ANR** als zweistellige Dezimalzahl (+ / -) festgelegt, der Wertebereich ist also die Menge {-99, -98, ….., -1, 0, 1, 2,……, 98, 99}, d.h. die Attribute **ANR** und **HNR** haben identische Wertebereiche.

Die Wertebereiche ergeben sich aus dem in der Datenbank abgebildeten Gegenstandsbereich! Die Hersteller von Datenbank-Management-Servern bieten in der Regel neben den im Standard vorgesehenen Datentypen hinaus oft noch weitere herstellerspezifische Datentypen.

Definition: Relationale Datenbank (Relationales Datenbankschema)

Eine relationale Datenbank (ein relationales Datenbankschema) ist eine Sammlung von unterscheidbaren (einfach normalisierten) Relationen.

Eine Relation ist dabei einfach normalisiert, wenn die Attribute nur einzelne Werte, aber keine Listen, Aufzählungen oder Strukturen enthalten können.

Die Bezeichnungsweise ist hier leider nicht allgemein gültig. Beim Oracle Datenbankserver definiert jeder Benutzer sein eigenes Datenbankschema, während beim MySQL Datenbankserver jeder Benutzer seine eigene Datenbank definiert. In beiden Fällen kann das Schema bzw. die Datenbank über den Benutzernamen identifiziert und angesprochen werden.

Beispiel einer nicht einfach normalisierten Relation (Stundenplan-Tabelle STD_PLAN.txt)

ID	Dozenten	Wochentag	Beginn	Ende	Bezeichnung
#SPLUSA61D03	Bresack, Schulz,R.(ET)	Donnerstag	12:00	15:00	……
#SPLUSA61D01	Bresack, Schulz,R.(ET)	Donnerstag	12:00	15:00	……
#SPLUSA61CFF	Bresack, Dittrich	Freitag	08:00	11:00	……
#SPLUSA61CFD	Bresack, Dittrich	Dienstag	12:00	15:00	……
…….	……	……..	……	……..	……
…….	……	……..	……	……..	……
…….	……	……..	……	……..	……
…….	……	……..	……	……..	……

Definition: Datenmodell

Ein zusammenfassende Sammlung von Konzepten zur Beschreibung der Daten einer Organisation, ihrer Beziehungen untereinander und der für sie gültigen Einschränkungen.

Eigenschaften von Tabellen im relationalen Modell

- Innerhalb einer relationalen Datenbank hat jede Tabelle einen eindeutigen Namen.
- Jede Zelle (d.h. das Element am Schnittpunkt einer Zeile und Spalte) einer (einfach normalisierten) Tabelle enthält genau einen (atomaren)Wert.
- Die Spalten einer Tabelle haben unterschiedliche Namen.
- Die Werte in einer Spalte sind alle aus einem gegebenen Wertebereich oder NULL.
- Die Reihenfolge der Spalten ist ohne Bedeutung, d.h. eine Vertauschung von Spalten einschließlich ihrer Namen lässt die Tabelle unverändert.
- Die Zeilen (Tupel) einer Tabelle sind alle verschieden, d.h. zwei beliebige Zeilen haben in mindestens einer Spalte verschiedene Werte.
- Die Reihenfolge der Zeilen ist ohne Bedeutung, d.h. eine beliebige Anordnung der Zeilen lässt die Tabelle unverändert.

Definition: Schlüssel(kandidat)

Ein Schlüsselkandidat (oder kurz Schlüssel) ist jede (minimale) Menge von einem oder mehreren Spalten einer Tabelle, durch deren Werte die Zeilen der Tabelle eindeutig identifiziert werden können.

Eigenschaften:

- eindeutig identifizierend, d.h. es gibt höchstens eine Zeile in der Tabelle, die in den Spalten des Schlüssels mit den Schlüsselwerten übereinstimmt.
- minimal, d.h. wird eine beliebige Spalte des Schlüssels weggelassen, ist die verbleibende Menge der Spalten nicht mehr eindeutig identifizierend.
- Es kann mehrere Schlüsselkandidaten für eine Tabelle geben.

Definition: Schlüsselattribut

Die Spalten eines Schlüsselkandidaten/eines Schlüssels heißen Schlüsselattribute

Definition: Primärschlüssel

Ein Schlüsselkandidat einer Tabelle wird ausgewählt, um die Zeilen der Tabelle eindeutig zu identifizieren. Dieser Schlüsselkandidat ist damit besonders ausgezeichnet und heißt Primärschlüssel. Eventuell noch vorhandene weitere Schlüsselkandidaten heißen Sekundärschlüssel.

Definition: Fremdschlüssel

Eine oder mehrere Spalten einer Tabelle, die in einer anderen Tabelle Schlüsselkandidat sind, heißt Fremdschlüssel.

Eigenschaften:

- Beziehungen zwischen den Zeilen zweier Tabellen in einer Datenbank können durch einen Fremdschlüssel ausgedrückt werden.
- Ein Fremdschlüssel stellt eine Referenz auf einen Schlüsselkandidaten einer anderen (referenzierten) Tabelle der Datenbank her.
- Die Werte der Fremdschlüsselattribute bestimmen genau eine Zeile in der referenzierten Tabelle.
- Fremdschlüssel im relationalen Modell werden eingesetzt, um die redundante Speicherung beschreibender Attribute zu vermeiden.

Tabelle: **Hersteller (Primärschlüssel: HNR)**

HNR	Name	Ort	Umsatz
1	Badendorf	Hamburg	hoch
2	Gondi	Jena	mittel
3	Hankel	Weimar	mittel
4	KMEX	Mannhein	niedrig
9	Baff	Leverkusen	hoch
10	Plendax	Hamburg	mittel
12	Boyer	Jena	hoch

Beispiel für die Verwendung eines Fremdschlüssels:

Tabelle: **Erzeugnis (Primärschlüssel: ANR)**

ANR	Bezeichnung	Preis	HNR
8	Parfum	3.40	2
9	Rasierwasser	5.30	3
11	Zahncreme	1.20	9
13	Haarspray	7.40	1
15	Haarcreme	1.50	4
36	Shampoo	6.20	2
37	Shampoo	5.50	9

Tabelle: **Hersteller (Primärschlüssel: HNR)**

HNR	Name	Ort	Umsatz
1	Badendorf	Hamburg	hoch
2	Gondi	Jena	mittel
3	Hankel	Weimar	mittel
4	KMEX	Mannhein	niedrig
9	Baff	Leverkusen	hoch
10	Plendax	Hamburg	mittel
12	Boyer	Jena	hoch

Integritätsbedingungen im relationalen Modell

Definition: null

Der symbolisch Wert null (englische Bezeichnung für „nichts“) für eine Spalte einer Tabelle wird verwendet um auszudrücken, dass der betreffende Attributwert unbekannt oder nicht anwendbar ist.

- ⇨ Der Wert eines Primärschlüssel darf nicht **null** sein.
- ⇨ Der symbolische Wert **null** gehört nicht zum Wertebereich eines Attributs.
- ⇨ Für beliebige Spalten einer Tabelle kann die Verwendung von **null** zugelassen oder ausgeschlossen werden.

Beispiele:

- In einer Spalte sind die Telefonnummern der Dozenten gespeichert, ist die Telefonnummer für einen Dozenten unbekannt, kann dies durch Verwendung des Wertes **null** ausgedrückt werden.
- In einer Spalte ist die Raumnummer des Büros der Dozenten gespeichert, für externe Dozenten, die an der FH kein Büro haben, kann dieser Sachverhalt durch Verwendung des Wertes **null** ausgedrückt werden.

Definition: Referentielle Integrität

Referentielle Integrität bezeichnet eine Eigenschaft einer Datenbasis im relationalen Modell, die sicherstellt, dass Fremdschlüsselreferenzen zu keinen Widersprüchen führen, d.h. es gibt zu keiner Zeit für einen Fremdschlüssel einen Wert, der nicht auch als Wert des referenzierten Schlüssels vorhanden ist.

Definition: Allgemeine Bedingungen

Die zulässige Wertemenge für ein Attribut bzw. für eine Menge von Attributen kann über die durch den Datentyp festgelegte Wertemenge hinaus durch Bedingungen (Constraints) weiter eingeschränkt werden.

Beispiele:

- Der Wert der Spalte Preis in der Tabelle Erzeugnis muss zwischen 0,00 und 10,00 liegen
- Der Wert der Spalte Bezeichnung in der Tabelle Erzeugnis darf nicht **Null** sein.
- Als Wert der Spalte Umsatz in der Tabelle Hersteller sind nur die Texte „hoch“, „mittel“ und „niedrig“ zugelassen.

Übersicht Integritätsbedingungen

⇨	Primärschlüsselbedingung	(primary key constraint)
⇨	Schlüsselbedingung	(unique constraint)
⇨	Fremdschlüsselbedingung	(foreign key constraint)
⇨	Pflichtwert	(not null constraint)
⇨	Optionaler Wert	(null constraint)
⇨	Allgemeine Bedingung	(check constraint)

Hinweise:

- Jede Integritätsbedingung sollte einen Namen haben, um sie bei einem Verstoß benennen zu können.
- Alle Integritätsbedingungen werden normalerweise ausschließlich durch das DBMS und nicht durch die Anwendungen überwacht.
- Die Überwachung einer Integritätsbedingung ist normalerweise aktiv (enable), sie kann aber auch bei Bedarf deaktiviert werden (disable).

2.2. Beispiele für relationale Datenbanken

Es werden vier Beispiele für relationale Datenbanken beschrieben, die alle drei auf mindestens einem Datenbankserver gespeichert sind und in einzelnen Übungen benutzt werden und darüber hinaus auch beim individuellen Arbeiten als Beispieldatenbanken verwendet werden können.

2.2.1 Messdaten der Klimastation der FH Jena

In der Klimastation, die vom Fachbereich Maschinebau betrieben wird und auf deren Messdaten über eine eigene Internetanwendung zugegriffen werden kann, werden permanent eine Vielzahl von Klimadaten (Temperatur, Luftdruck, Regenmenge, Luftfeuchtigkeit, Globalstrahlung,…..) erfasst und alle 10 Minuten in einer Datenbank gespeichert. Der die Klimastation betreuende Mitarbeiter hat uns freundlicherweise alle Messdaten für die Jahre 2006 und 2010 zur Verfügung gestellt.
Alle diese Messdaten für die Jahre 2006 und 2010 sind in einer Tabelle abgelegt (jeweils eine Zeile für die Messwerte einer Messung). Die Tabelle enthält also 6 * 24 * 365 = 52560 Zeilen pro Jahr. Jede Zeile enthält Datum und Uhrzeit der Messung und für jeden Klimawert die beiden Extremwerte (Minimum und Maximum) und den Durchschnittwert, den die Sensoren während der **verstrichenen 10 Minuten** registriert haben. Die so bereitgestellte Datenbank besteht aus einer einzigen Tabelle

Tabelle: **Messwerte**

- ⇨ Datum der Messung
- ⇨ Uhrzeit der Messung
- ⇨ Mittel-, Minimal- und Maximalwert der Temperatur in °C
- ⇨ Mittel-, Minimal- und Maximalwert der relativen Feuchte in %
- ⇨ Mittel-, Minimal- und Maximalwert der Niederschlagsmenge in mm
- ⇨ Mittel-, Minimal- und Maximalwert der Globalstrahlung in W/m^2
- ⇨ Mittel-, Minimal- und Maximalwert des Luftdrucks in hPA
- ⇨ Mittel-, Minimal- und Maximalwert der Windgeschwindigkeit in m/s
- ⇨ Mittel-, Minimal- und Maximalwert der Windrichtung als Winkel 0..360°

Die Tabelle enthält also insgesamt für die fünf Jahre 262800 einzelne Temperatur-Messwerte!

Es sind nicht immer alle Attributwerte für einen Datensatz vorhanden, d.h. es fehlen manchmal einzelne Werte für bestimmte Messgrößen. In diesen Fällen ist der entsprechende Attributwert null.

2.2.2 Jenaer Verkehrsbetriebe

Die Jenaer Verkehrsbetriebe haben freundlicherweise die Streckennetzinformationen der öffentlichen Verkehrsmittel in Jena zur Verfügung gestellt. Diese wurden (vereinfacht) und in einer Datenbank zusammengestellt, in der zu allen Linien neben dem Streckenverlauf (also der Abfolge der einzelnen Haltestellen) auch noch die Entfernung und die Fahrzeit von der Anfangshaltestelle der Linie bis zur betreffenden Haltstelle gespeichert sind.
Die ursprünglichen Rohdaten (Stand 1999, heute vereinzelt nicht mehr aktuell) waren gegeben als

Tabelle: **JENAH_HAL**

Diese Tabelle enthält alle **Haltepunkte** der Bus und Bahnlinien. Ein Haltepunkt wird dabei beschrieben durch:

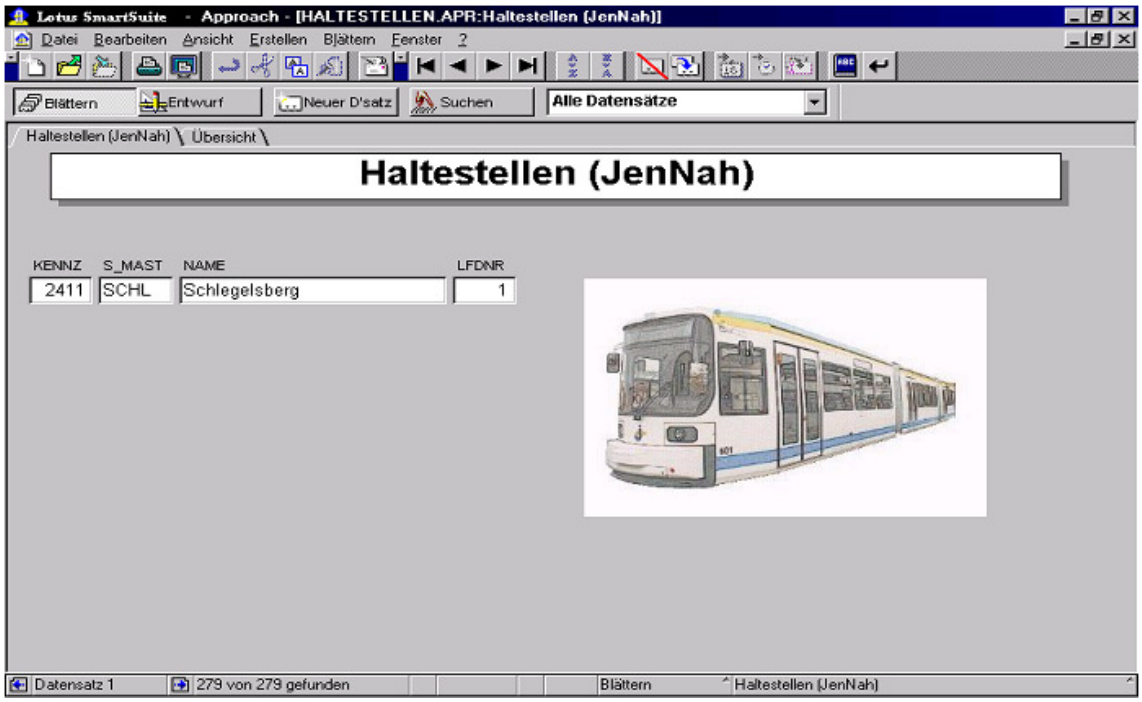
- ⇨ Kennzeichen des Haltepunktes, hier sind Angaben zum Stadtteil, zur Hauptlinie und zur Richtung (stadteinwärts / stadtauswärts) verschlüsselt. Diese Angaben werden hier nicht weiter benötigt und nur angeboten für spätere Erweiterungen.
- ⇨ örtliche Differenzierung des Haltepunkte (Mast-Angabe), hier wird im Wesentlichen nach den verschiedenen Haltepunkten einer Haltestelle unterschieden, z.B. bzgl. der verschiedenen Richtungen der Linien oder bzgl. Ankunft / Abfahrt an Endhaltestellen oder für Zugangspunkte zu einem Betriebshof.
- ⇨ Name / Bezeichnung der Haltestelle, so wie er/sie verwendet wird, dabei können sich für die einzelnen Haltepunkte verschiedene Namen (Bezeichnungen) für die Haltestelle ergeben. Über einen der Namen der Haltestelle wird in der Umgangssprache der Haltepunkt beschrieben, obwohl die Angabe eines Namens alleine oft nicht ausreicht und es einer (impliziten) Ergänzung zum genauen Haltepunkt bedarf, um den gewünschten Bahn- / Bus-Haltepunkt zu identifizieren.
- ⇨ fortlaufende Nummer der Haltepunkte.

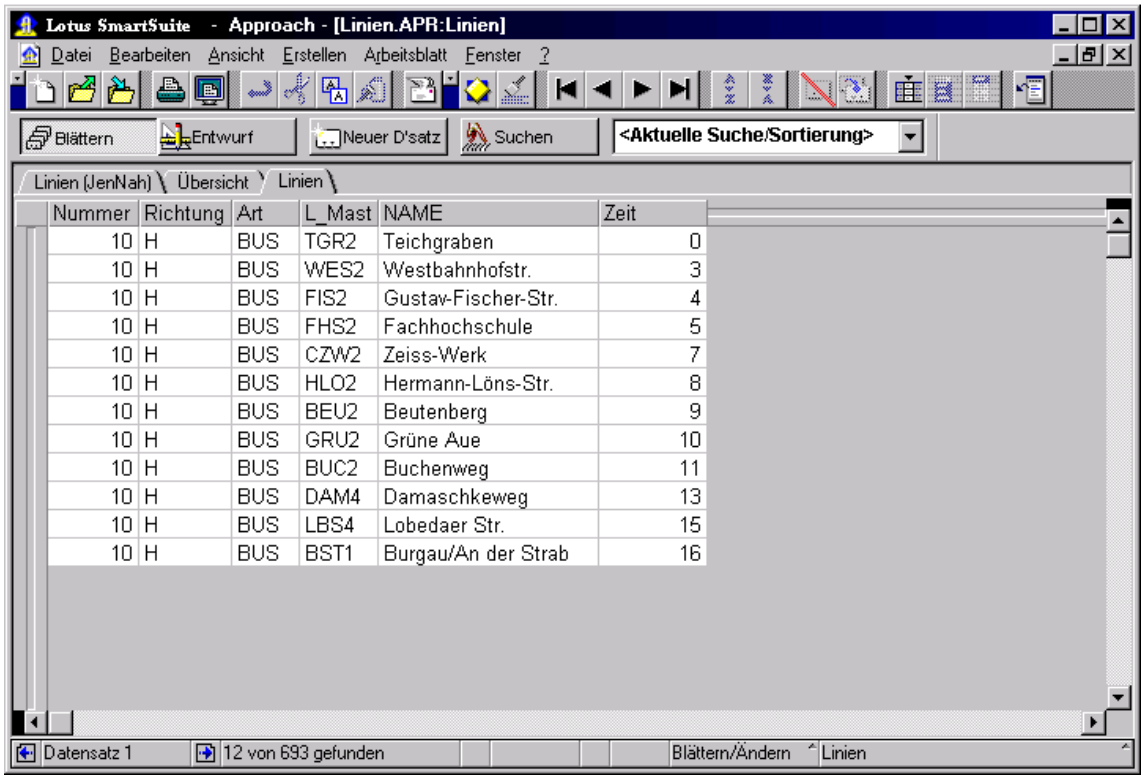
Tabelle: **JENAH_LIN**

Diese Tabelle enthält die genaue Linienführung aller Bus- und Bahnlinien einschließlich Angaben zur Fahrstrecke (in Metern), zur Fahrzeit (in Minuten) und zur Art des Verkehrsmittels (BUS/BAHN). Für die Streckenführung der einzelnen Linien wird in Hin- und Rückfahrt unterschieden. Die einzelnen Haltestellen werden durch Mast-Angabe des Haltepunktes festgelegt:

- ⇨ Nummer der Bus- oder Bahn-Linie
- ⇨ Angaben zur Fahrtrichtung (H = Hinfahrt, R = Rückfahrt)
- ⇨ örtliche Differenzierung des Haltepunktes (Mast-Angabe) zur Festlegung der Haltestelle.
- ⇨ kumulierte Fahrstrecke in Meter ab der Anfangshaltestelle
- ⇨ kumulierte Fahrzeit in Minuten ab der Anfangshaltestelle
- ⇨ Art des Verkehrsmittels: BUS oder BAHN

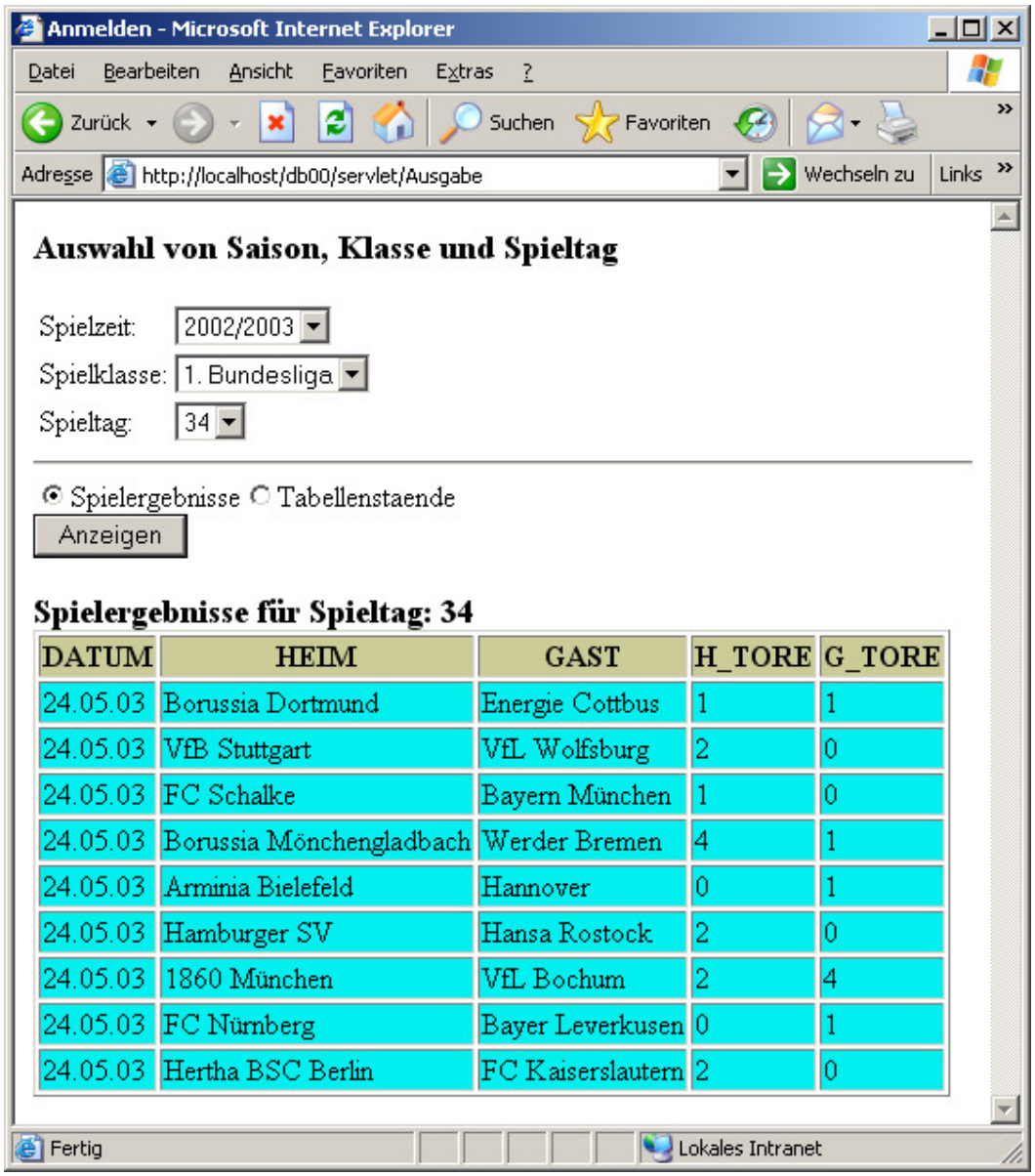
Diese Rohdaten wurden in ein Datenbankmodell umgesetzt, dass die Informationen auf mehrere Tabellen verteilt und außerdem weitere Tabellen hinzufügt, in der Informationen zur Erreichbarkeit von Einrichtungen mit öffentlichen Verkehrsmitteln hinterlegt sind.


Lotus SmartSuite - Approach - [HALTESTELLEN.APR:Haltestellen (JenNah)]
Datei Bearbeiten Ansicht Erstellen Blättern Fenster ?
Blättern
Entwurf
Neuer D'satz
Suchen
Alle Datensätze
Haltestellen (JenNah) \ Übersicht \
Haltestellen (JenNah)
KENNZ
S_MAST
NAME
LFDNR
2411
SCHL
Schlegelsberg
1
Datensatz 1
279 von 279 gefunden
Blättern
Haltestellen (JenNah)


Lotus SmartSuite - Approach - [Linien.APR:Linien]
Datei Bearbeiten Ansicht Erstellen Arbeitsblatt Fenster ?
Blättern
Entwurf
Neuer D'satz
Suchen
<Aktuelle Suche/Sortierung>
Linien (JenNah) \ Übersicht \ Linien \
Nummer Richtung Art L_Mast NAME Zeit
10 H BUS TGR2 Teichgraben 0
10 H BUS WES2 Westbahnhofstr. 3
10 H BUS FIS2 Gustav-Fischer-Str. 4
10 H BUS FHS2 Fachhochschule 5
10 H BUS CZW2 Zeiss-Werk 7
10 H BUS HLO2 Hermann-Löns-Str. 8
10 H BUS BEU2 Beutenberg 9
10 H BUS GRU2 Grüne Aue 10
10 H BUS BUC2 Buchenweg 11
10 H BUS DAM4 Damaschkeweg 13
10 H BUS LBS4 Lobedaer Str. 15
10 H BUS BST1 Burgau/An der Strab 16
Datensatz 1
12 von 693 gefunden
Blättern/Ändern
Linien

2.2.3 Ergebnisse der drei höchsten deutschen Spielklassen (Fußball)

Die Zeitungsgruppe Thüringen hat freundlicherweise die Rohdaten für ihre Ergebnisdienste (z.B. unter ww.tlz.de/Sporttabellen) der deutschen Fußball-Ligen zur Verfügung gestellt. Aus diesen Daten wurden die Daten (vor Einführung der dritten Bundesliga) für die drei höchsten deutschen Spielklassen (1. und 2. Bundesliga und Regionalliga, Staffel Nord und Süd) für mehrere Saisons extrahiert und in einer Datenbank auf dem Oracle Datenbankserver gespeichert. In einer anderen spezialisierten Datenbankveranstaltung wurde für diese Datenbank eine Internetanwendung realisiert, mit der im Browser auf diese Datenbank zugegriffen werden kann:



Bei Auswahl **Spielergebnisse** wird auf die Tabelle **Spiel**, bei Auswahl **Tabellenstände** auf die Tabelle **Ergebnis** zugegriffen. Die Daten werden dabei durch die vorgegebene Saison (hier 2002/2003), die Spielklasse (hier 1. Bundesliga) und den Spieltag (hier 34. und damit letzter Spieltag) selektiert.

In der Datenbank sind mehrere miteinander verknüpfte Datenbanktabellen in einem so genannten Datenbankmodell zusammengestellt:

Tabelle; **Spielklasse**
repräsentiert eine Spielklasse im Fußball, z.B. 1. Bundesliga

Tabelle: **Mannschaft**
repräsentiert eine Mannschaft eines Vereins (unabhängig von einer Spielklasse) z.B. Bayern München

Tabelle: **Spielzeit**
repräsentiert eine Saison ohne Bezug zu einer Spielklasse, z.B. 2003/2004

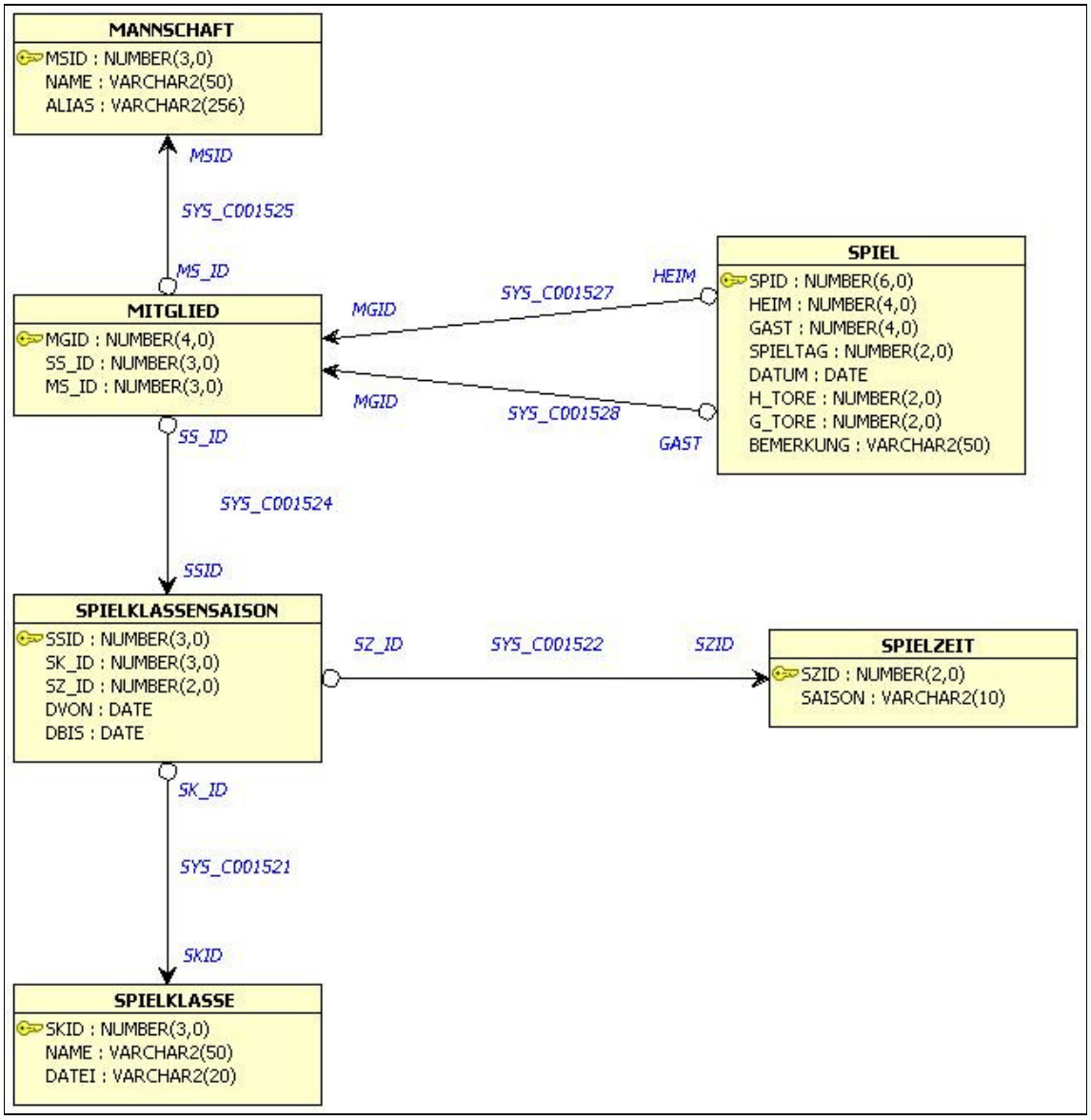
Tabelle: **Spielklassensaison**
repräsentiert eine Spielzeit für eine Spielklasse, enthält Anfangs- und Enddatum, z.B. „1. Bundesliga“, „2003/2004“ vom: „22.06.03“ bis: „22.05.04“

Tabelle: **Mitglied**
repräsentiert eine Mannschaft in einer Spielklassensaison, in den Spielen wird auf den entsprechenden Index in Mitglied und nicht in Mannschaft referenziert z.B. „Energie Cottbus“ spielte „2002/2003“ in der „1. Bundesliga“

Tabelle: **Spiel**
repräsentiert ein konkretes Spielergebnis, es werden zwei Mitglieder (HEIM und GAST“) referenziert, eine fortlaufende Nummer beschreibt den Spieltag, das Datum des Spiels ist neben dem Ergebnis (H_TORE und G_TORE) und einer Bemerkung (Spielbelegung, Abweichende Spielwertung, usw.) aufgeführt. z.B. „Energie Cottbus“ (HEIM) gegen „Bayern München“ (GAST) am „1.3.2003“ Spielzeit: „2002/2003“ Spielklasse: „1. Bundesliga“, Ergebnis: 0 : 2

Tabelle: **Ergebnisse**
repräsentiert den Tabellenstand in den verschiedenen Spielklassen nach den einzelnen Spieltagen, dabei wird der Punktestand und die Tordifferenz ausgewiesen, die dann den Tabellenstand bestimmen.

Übersicht Datenmodell: Tabellen einschließlich Verknüpfungen



Die Tabelle **Ergebnis** ist hier nicht aufgeführt, da sie keine so genannte Basistabelle ist, da ihre Zeilen aus den Zeilen der Tabelle Spiel abgeleitet werden können. Dies ist aber recht aufwendig und verändert sich auch nur während der laufenden Saison aber nicht danach, deshalb wurde die Ableitung einmal ausgeführt und in einer eigenen Tabelle abgespeichert, auf deren Zeilen jetzt sofort zugegriffen werden kann.

2.2.4 Aktuelle Stundenplandaten der FH Jena für das Sommersemester

Die Stundenplandaten werden als Textdatei zur Verfügung gestellt. Der Stundenplan ist eigentlich in einer (Oracle-)Datenbank gespeichert und die Internet-Anwendungen greifen direkt auf diese Datenbank zu. In regelmäßigen Abständen werden aus dieser Datenbank die Stundenplandaten extrahiert und in kompakter Form als Textdatei im csv-Format bereitgestellt. In dieser Textdatei enthält jede Zeile eine einzelnen Veranstaltung mit Angaben zu:

ID	interner Schlüssel
Dozent	Liste aller Dozenten dieser Veranstaltung
Wochentag	Montag, Dienstag, …..
Begin	Uhrzeit im Format Stunde:Minute
Ende	Uhrzeit im Format Stunde:Minute
Veranstaltung	Titel der Veranstaltung, der Titel enthält verschlüsselt, um welche Art von Veranstaltung (Vorlesung, Übung, Praktikum,….) es sich handelt.
Raum	Liste aller Räume dieser Veranstaltung
Studentensets	Liste aller Studentensets, die dieser Veranstaltung zugeordnet sind
Turnus	direkte Angabe wie wöchentlich, 1. Hälfte, usw.
Wochenmuster	kodierte Angabe der Wochen, in denen diese Veranstaltung stattfindet,
Anfang	Datum des Tages, an dem die Veranstaltung zum ersten Mal stattfindet

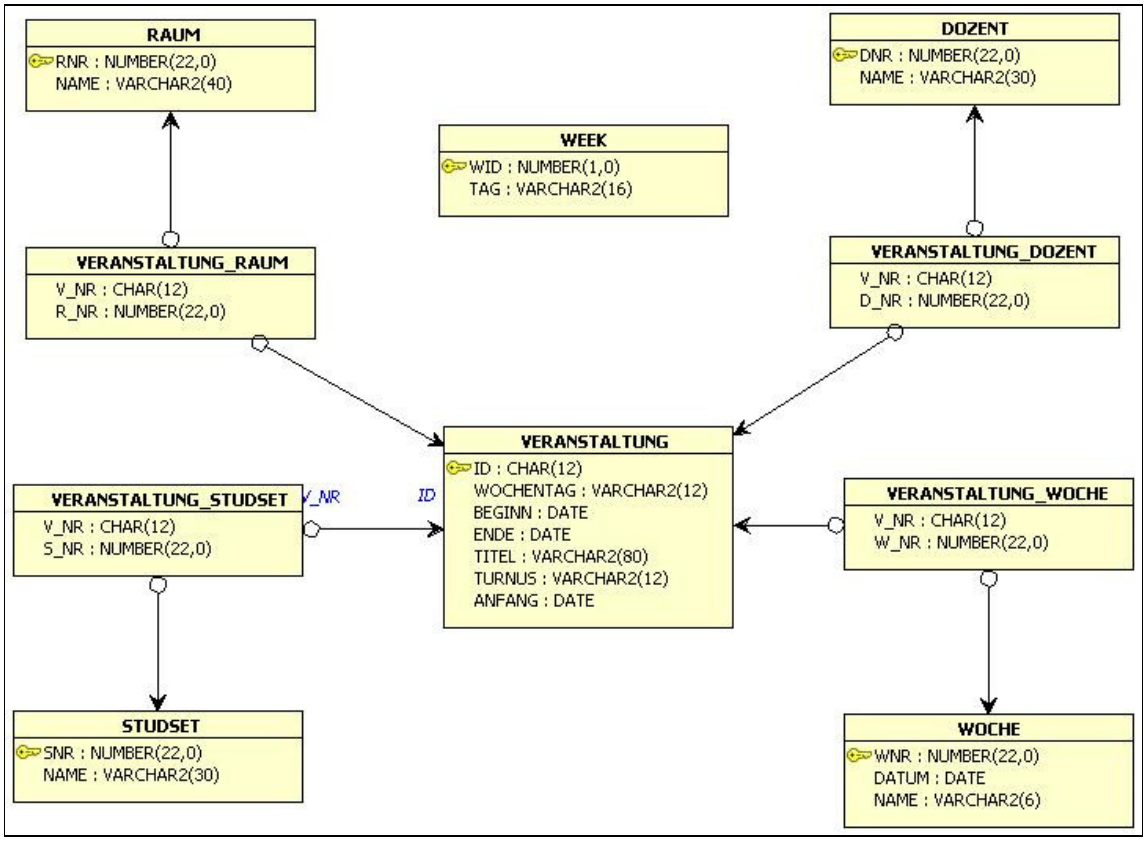
Als Trennzeichen innerhalb einer Zeile wird das Semikolon verwendet. Der Inhalt von Textfeldern ist in Anführungszeichen eingeschlossen.

Beispiel: (Daten aus dem SS 2008, ohne Anführungszeichen)

ID	#SPLUS068FB2
Dozent	Beez, Bliedtner, Lochmann1, Zweinert
Wochentag	Montag
Begin	31.12.1899 15:15
Ende	31.12.1899 18:15
Veranstaltung	MB(BA) Spanende Fertigung I/P/03.2
Raum	70.03.760**
Studentensets	MB(BA)2.11, MB(BA)2.12, MB(BA)2.13, MB(BA)2.14, MB(BA)2.15
Turnus	
Wochenmuster	000000010000000000000000000000000000000000000000
Anfang	05.05.2008 00:00:00

Diese „Rohdaten“ der Textdatei wurden in eine entsprechende Datenbank (dies war eine Aufgabe im Rahmen der Veranstaltung Datenbanken II (Diplomstudiengänge) mit normalisierten Relationen übernommen. Die einzelnen Tabellen sollten bis auf die Tabelle WEEK klar sein. Die Tabelle WEEK wird benutzt, um die Wochentage entsprechend ihrer natürlichen Reihenfolge aufsteigend zu sortieren.

Datenbank: Tabellen einschließlich Verknüpfungen



2.3 Realisierung externer Sichten

Das konzeptuelle Modell (Darstellung des als Daten abzubildenden Ausschnitts) wird in ein entsprechendes Datenbankmodell umgesetzt und auf der internen (physischen) Ebene als Dateien abgespeichert.

Im Fall einer relationalen Datenbank führt dies zu einer Reihe von Tabellen, die als **Basistabellen** bezeichnet werden, da sie als Umsetzung des konzeptuellen Modells die Grundlage für die diversen externen Sichten bilden, d.h. alle externen Sichten werden direkt oder indirekt aus den Basistabellen abgeleitet.

Jede externe Sicht ist selbst wieder eine Relation, kann also wie eine Tabelle gesehen werden. Die Spalten und die Zeilen einer solchen externen Sicht sind jetzt aber nicht mehr (wie bei den Basistabellen) statisch abgespeichert, sondern werden jeweils dynamisch erzeugt. Für die dynamische Bereitstellung von Daten in Form von externen Sichten, gibt es im relationalen Datenbankmodell drei grundlegende Standardoperationen:

- Projektion (Auswahl und Kombination von Spalten)
- Selektion (Auswahl und Kombination von Zeilen)
- Verbund (Auswahl und Kombination von Tabellen)

Das Ergebnis ist dabei stets wieder eine Tabelle. Grundsätzlich gibt es für die Spalten einer externen Sicht die folgende Möglichkeiten (Projektion):

- Es können Spalten vorhandener Tabellen oder Sichten mit einem anderen Namen aufgenommen werden.
- Es können Spalten vorhandener Tabellen oder Sichten in einer anderen Reihenfolge aufgenommen werden.
- Es kann eine Teilmenge der Spalten vorhandener Tabellen oder Sichten aufgenommen werden
- Es können einzelne Spalten vorhandener Tabellen oder Sichten zu neuen Spalten zusammengesetzt werden (arithmetische Verknüpfung, Zeichenkettenverknüpfung,...). Die Inhalte der so hinzugefügten Spalten ergeben sich dabei zeilenweise (z.B. aus den Spalten Netto und MwSt wird eine neue Spalte Brutto zusammengesetzt, die Werte der neuen Spalte ergeben sich dabei zeilenweise nach der Formel Netto*MwSt).
- Es können neue Spalten hinzugefügt werden, die für alle Zeilen der Tabelle einen konstanten Wert enthalten, z.B. kann in einer Sicht das aktuelle Tagesdatum hinzugefügt werden, das natürlich für alle Zeilen identisch ist.
- Es können neue Spalten hinzugefügt werden, in denen berechnete Inhalte angezeigt werden, die sich als Funktionswert aus den Zeilenwerten einer Spalte einer Basistabelle oder einer Sicht ergeben. Die dabei benutzten Funktionen heißen Aggregatfunktionen und typische Vertreter sind **Anzahl** (berechnet die von NULL verschiedenen Werte in einer Spalte), **Maximum/Minimum**, (berechnet den größten/kleinsten Wert in einer Spalte), **Summe/Durchschnitt** (berechnet die Summe / den Durchschnitt aller Werte in einer Spalte). Aggregatfunktionen verdichten die Informationen indem sie die (vielen) Einzelwerte in einer Spalte durch einen zusammenfassenden Wert ersetzen.

Für die Zeilen einer externen Sicht ergeben sich weiter folgende Möglichkeiten (Selektion):

- Es können die Zeilen in einer externen Sicht in eine bestimmte Reihenfolge gebracht werden (Die Ausgabe kann bzgl. einer Spalte in auf- oder absteigender Reihenfolge erfolgen)
- Es können Bedingungen gestellt werden, d.h. es werden nur die Zeilen in die Sicht übernommen, die die Bedingung erfüllen (oder nicht erfüllen). Eine Bedingung ist dabei ein Ausdruck, der für jede Zeile ausgewertet werden kann und der für jede Zeile entweder den Wert wahr oder falsch ergibt.
- Es können Zeilen nach einem bestimmten Kriterium (Bedingung) zunächst gruppiert. d.h. nach gleichen Inhalten eines Feldes gebündelt werden. Die Anwendung der Aggregat-Funktionen erfolgt jetzt nicht mehr auf alle Zeilen der Spalte, sondern jeweils getrennt für die einzelnen Gruppen, z.B. wenn man nicht die Anzahl aller Studenten an der Fachhochschule Jena haben möchte, sondern die Anzahl der Studenten jeweils getrennt pro Fachbereich.

Um die Möglichkeiten zur Bildung von externen Sichten zu veranschaulichen, betrachten wir folgende einfache Datenbank, mit drei Basistabellen:

Beispiel:

Tabelle: **Lieferant**

LNR	Name	Ort	Status
1	Karcher	Jena	10
4	Schmidt	Weimar	20
6	Mey	Gera	10
8	Runge	Apolda	30
9	Todd	Jena	30

Primärschlüssel: **LNR**

Tabelle: **Artikel**

ANR	Bezeichnung	Laenge	Breite	Material	LNR
R1	Regal	80	40	Metall	8
R2	Regal	80	60	Metall	8
S1	Schrank	100	60	Holz	6
S2	Schrank	80	40	Holz	6
T1	Tisch	80	60	Kunststoff	6
T2	Tisch	100	80	Kunststoff	8

Primärschlüssel: **ANR**
Fremdschlüssel: **LNR (auf den Primärschlüssel von Lieferant)**

Tabelle: **liefert**

LNR	ANR	Mindestmenge	Lieferzeit	Preis
8	R1	3	5	200
4	R1	1	1	220
8	R2	3	5	250
4	R2	1	1	300
6	S1	2	10	50
9	S1	5	5	50
8	S1	5	5	60
8	S2	1	10	40
6	S2	5	5	30
1	T1	10	5	120
4	T1	5	8	150
6	T1	5	5	100
8	T1	10	5	100
9	T2	2	6	90
1	T2	5	5	100
8	T2	2	6	90

Primärschlüssel: **LNR, ANR**
Fremdschlüssel: **LNR (auf den Primärschlüssel von Lieferant)**
ANR (auf den Primärschlüssel von Artikel)

Projektionen:

LNR	ANR	Kleinste_Bestellmenge	Lieferzeit_in _Tagen	Preis_in_Euro
8	R1	3	5	200
4	R1	1	1	220
8	R2	3	5	250
4	R2	1	1	300
6	S1	2	10	50
9	S1	5	5	50
8	S1	5	5	60
8	S2	1	10	40
6	S2	5	5	30
1	T1	10	5	120
4	T1	5	8	150
6	T1	5	5	100
8	T1	10	5	100
9	T2	2	6	90
1	T2	5	5	100
8	T2	2	6	90

Ort	Name	LNR	Status
Jena	Karcher	1	10
Weimar	Schmidt	4	20
Gera	Mey	6	10
Apolda	Runge	8	30
Jena	Todd	9	30

Status	Name
10	Karcher
20	Schmidt
10	Mey
30	Runge
30	Todd

ANR	Bezeichnung	Flaeche
R1	Regal	3200
R2	Regal	4800
S1	Schrank	6000
S2	Schrank	3200
T1	Tisch	4800
T2	Tisch	8000

LNR	Name	Ausland
1	Karcher	nein
4	Schmidt	nein
6	Mey	nein
8	Runge	nein
9	Todd	nein

Anzahl_Lieferanten	Maximaler_Status
5	30

Selektionen:

(Aufsteigend nach LNR sortierte Ausgabe)

LNR	ANR	Mindestmenge	Lieferzeit	Preis
1	T1	10	5	120
1	T2	5	5	100
4	R1	1	1	220
4	R2	1	1	300
4	T1	5	8	150
6	S1	2	10	50
6	S2	5	5	30
6	T1	5	5	100
8	R1	3	5	200
8	R2	3	5	250
8	S1	5	5	60
8	S2	1	10	40
8	T1	10	5	100
8	T2	2	6	90
9	S1	5	5	50
9	T2	2	6	90

(Bedingung: Status = 10)

LNR	Name	Ort	Status
1	Karcher	Jena	10
6	Mey	Gera	10

(gruppiert nach gleichen Werten in der Spalte ANR)

ANR	Anzahl_Lieferanten	Minmaler_Preis
R1	2	200
R2	2	250
S1	3	50
S2	2	30
T1	4	100
T2	3	90

(gruppiert nach gleichen Werten in der Spalte Ort)

Ort	Anzahl_Lieferanten
Apolda	1
Gera	1
Jena	2
Weimar	1

(gruppiert nach gleichen Werten in der Spalte Material)

Material	Minimale_Breite
Holz	40
Kunststoff	60
Metall	40

Tabellenverbund:

Für die betrachteten Tabellen in einer externen Sicht ergeben sich dabei folgende Möglichkeiten (Verbund):

- Die externe Sicht basiert auf einer einzelnen Tabelle (Basistabelle oder externe Sicht)
- Die externe Sicht basiert auf dem Verbund (englisch: join) von zwei Tabellen (Basistabellen oder externe Sichten).
  Dabei ergibt sich als Verbundergebnis eine große neue Tabelle, die sich aus den Spalten der beiden verbunden Tabellen zusammensetzt. Es gibt hierbei verschiedene Varianten, wie sich die Spalten und die Zeilen der Verbundtabelle bilden.
- Die externe Sicht basiert auf dem Verbund von mehr als zwei Tabellen (Basistabellen oder externe Sichten).
  Das Verbundergebnis ergibt sich jetzt schrittweise als Verbund der ersten zwei Tabellen, dann dem Verbund des Ergebnisses dieses ersten Verbunds mit der dritten Tabelle, dann dem Verbund dieses Ergebnisses mit der vierten Tabelle usw. Auch dabei gibt es wieder die verschiedenen Varianten.

Variante 1: Natürlicher Verbund (Natural join) zweier Tabellen

Dabei werden alle *gleich(namig)en Spalten* nur einmal in der Verbundtabelle aufgeführt, d.h. das Ergebnis eines natürlichen Verbunds enthält alle Spalten der ersten Tabelle und alle Spalten der zweiten Tabelle, sofern nicht schon namensgleich in der ersten Tabelle enthalten.

Die Zeilen werden gebildet, indem die Zeilen der ersten Tabelle mit allen Zeilen der zweiten Tabelle kombiniert werden, die in den *gleichnamigen Spalten* beider Tabellen auch *gleiche Werte* haben.

LNR	Name	Ort	Status
1	Karcher	Jena	10
4	Schmidt	Weimar	20
6	Mey	Gera	10
8	Runge	Apolda	30
9	Todd	Jena	30

Artikel.LNR = Lieferant.LNR

ANR	Bezeichnung	Laenge	Breite	Material	LNR
R1	Regal	80	40	Metall	8
R2	Regal	80	60	Metall	8
S1	Schrank	100	60	Holz	6
S2	Schrank	80	40	Holz	6
T1	Tisch	80	60	Kunststoff	6
T2	Tisch	100	80	Kunststoff	8

Die Gleichheitsbedingung für die gleichnamigen Spalten wird dabei als Verbund-Bedingung bezeichnet. Gibt es mehrere gleichnamige Spalten werden die durch den logischen AND-Operator verknüpften Gleichheitsbedingungen als Verbund-Bedingung genommen.

Ergebnis des natürlichen Verbunds Artikel und Lieferant:

LNR	ANR	Bezeichnung	Laenge	Breite	Material	Name	Ort	Status
8	R1	Regal	80	40	Metall	Runge	Apolda	30
8	R2	Regal	80	60	Metall	Runge	Apolda	30
6	S1	Schrank	100	60	Holz	Mey	Gera	10
6	S2	Schrank	80	40	Holz	Mey	Gera	10
6	T1	Tisch	80	60	Kunststoff	Mey	Gera	10
8	T2	Tisch	100	80	Kunststoff	Runge	Apolda	30

Die gleichnamigen Spalten der Verbund-Bedingung stehen dabei bei der unspezifischen Angabe * traditionell am Anfang. Ein natürlicher Verbund ist symmetrisch, d.h. die Reihenfolge, in der die zu verbindenden Tabellen aufgeführt werden, hat keinen Einfluss auf das Ergebnis (abgesehen vielleicht von der „natürlichen“ Reihenfolge der Spalten).

Bildet man jetzt den natürlichen Verbund der Tabellen Artikel und liefert, so bekommt man zu jedem Artikel die Lieferbedingungen des „Hauptlieferanten“ angezeigt, da jetzt die zwei Tabellen die beiden gleichnamigen Spalten ANR und LNR haben, also als Verbund-Bedingung

Artikel.ANR = liefert.ANR AND Artikel.LNR = liefert.LNR

verwendet wird.

Ergebnis des natürlichen Verbunds Artikel und liefert:

LNR	ANR	Bezeichnung	Laenge	Breite	Material	Mindest-menge	Liefer-zeit	Preis
8	R1	Regal	80	40	Metall	3	5	200
8	R2	Regal	80	60	Metall	3	5	250
6	S1	Schrank	100	60	Holz	2	10	50
6	S2	Schrank	80	40	Holz	5	5	30
6	T1	Tisch	80	60	Kunststoff	5	5	100
8	T2	Tisch	100	80	Kunststoff	2	6	90

Was aber, wenn man diese Angaben nicht nur für den Hauptlieferanten, sondern für alle Lieferanten haben möchte ?

Dann kann man nicht den natürlichen Verbund verwenden, da dabei auch stets nur der Hauptlieferant „verbunden“ wird (Artikel.LNR = liefert.LNR).

Ein natürlicher Verbund ist ein spezieller innerer Verbund. Beim inneren Verbund können im einfachsten Fall in einer Using-Klausel alle gleichnamigen Spalten aufgeführt, die in der Verbund-Bedingung berücksichtigt werden sollen:

Ergebnis des inneren Verbunds Artikel und liefert mit using(ANR):

ANR	Bezeichnung	Laenge	Breite	Material	LNR	LN	Mindest-menge	Liefer-zeit	Preis
R1	Regal	80	40	Metall	8	8	3	5	200
R1	Regal	80	40	Metall	8	4	1	1	220
R2	Regal	80	60	Metall	8	8	3	5	250
R2	Regal	80	60	Metall	8	4	1	1	300
S1	Schrank	100	60	Holz	6	6	2	10	50
S1	Schrank	100	60	Holz	6	9	5	5	50
S1	Schrank	100	60	Holz	6	8	5	5	60
S2	Schrank	80	40	Holz	6	8	1	10	40
S2	Schrank	80	40	Holz	6	6	5	5	30
T1	Tisch	80	60	Kunststoff	6	1	10	5	120
T1	Tisch	80	60	Kunststoff	6	4	5	8	150
T1	Tisch	80	60	Kunststoff	6	6	5	5	100
T1	Tisch	80	60	Kunststoff	6	8	10	5	100
T2	Tisch	100	80	Kunststoff	8	9	2	6	90
T2	Tisch	100	80	Kunststoff	8	1	5	5	100
T2	Tisch	100	80	Kunststoff	8	8	2	6	90

Da die Spalte LNR aus der Join-Bedingung heraus genommen wurde, gibt es bei diesem Verbund jetzt zwei Spalten LNR (Spalte LNR aus Artikel) und LN (Spalte LNR aus liefert, umbenannt in LN). In der Spalte LNR steht die Lieferanten-Nummer des Hauptlieferanten und in der Spalte LN steht die Lieferanten-Nummer des Lieferanten der den betreffenden Artikel zu den Bedingungen liefert.

Der natürliche Verbund und auch der innere Verbund mit der using-Klausel können dabei nur angewandt werden, wenn die betreffenden Spalten gleichnamig sind. Die allgemeinste Form des inneren Verbunds lässt deshalb die Möglichkeit zu, die Verbund-Bedingung explizit anzugeben. Diese Form muss dann angewandt werden, wenn die betreffenden Spalten nicht gleichnamig sind.

Variante 2: innerer Verbund (inner join) zweier Tabellen

Allgemein wird der innere Verbund gebildet, in dem alle Spalten der ersten Tabelle und alle Spalten der zweiten Tabelle nacheinander aufgeführt werden. Alle gleichnamigen Spalten der beiden Tabellen werden im allgemeinen Fall also mehrfach aufgeführt und müssen entsprechend mit dem Tabellennamen qualifiziert und/oder umbenannt werden, d.h. der Grad des inneren Verbunds ist gerade die Summe der Grade der am Verbund beteiligten Tabellen.

Die Zeilen des inneren Verbund werden im allgemeinen Fall durch die Kombination aller Zeilen der ersten Tabelle mit allen Zeilen der zweiten Tabelle gebildet, d.h. der Kardinalität des inneren Verbundes ist zunächst einmal das Produkt der Kardinalitäten der beteiligten Tabellen.

Ergebnis des inneren Verbunds der Tabellen Artikel und Lieferant (ohne Einschränkungen)

ANR	Bezeichnung	Laenge	Breite	Material	LNR	LNR	Name	Ort	Status
R1	Regal	80	40	Metall	1	1	Karcher	Jena	10
R2	Regal	80	60	Metall	1	1	Karcher	Jena	10
S1	Schrank	100	60	Holz	1	1	Karcher	Jena	10
S2	Schrank	80	40	Holz	1	1	Karcher	Jena	10
T1	Tisch	80	60	Kunststoff	1	1	Karcher	Jena	10
T2	Tisch	100	80	Kunststoff	1	1	Karcher	Jena	10
R1	Regal	80	40	Metall	4	4	Schmidt	Weimar	20
R2	Regal	80	60	Metall	4	4	Schmidt	Weimar	20
S1	Schrank	100	60	Holz	4	4	Schmidt	Weimar	20
S2	Schrank	80	40	Holz	4	4	Schmidt	Weimar	20
T1	Tisch	80	60	Kunststoff	4	4	Schmidt	Weimar	20
T2	Tisch	100	80	Kunststoff	4	4	Schmidt	Weimar	20
R1	Regal	80	40	Metall	6	6	Mey	Gera	10
R2	Regal	80	60	Metall	6	6	Mey	Gera	10
S1	Schrank	100	60	Holz	6	6	Mey	Gera	10
S2	Schrank	80	40	Holz	6	6	Mey	Gera	10
T1	Tisch	80	60	Kunststoff	6	6	Mey	Gera	10
T2	Tisch	100	80	Kunststoff	6	6	Mey	Gera	10
R1	Regal	80	40	Metall	8	8	Runge	Apolda	30
R2	Regal	80	60	Metall	8	8	Runge	Apolda	30
S1	Schrank	100	60	Holz	8	8	Runge	Apolda	30
S2	Schrank	80	40	Holz	8	8	Runge	Apolda	30
T1	Tisch	80	60	Kunststoff	8	8	Runge	Apolda	30
T2	Tisch	100	80	Kunststoff	8	8	Runge	Apolda	30
R1	Regal	80	40	Metall	9	9	Todd	Jena	30
R2	Regal	80	60	Metall	9	9	Todd	Jena	30
S1	Schrank	100	60	Holz	9	9	Todd	Jena	30
S2	Schrank	80	40	Holz	9	9	Todd	Jena	30
T1	Tisch	80	60	Kunststoff	9	9	Todd	Jena	30
T2	Tisch	100	80	Kunststoff	9	9	Todd	Jena	30

Grad : Anzahl Spalten Artikel + Anzahl Spalten Lieferant = 6 + 4 = 10
Kardinalität: Anzahl Zeilen Artikel * Anzahl Zeilen Lieferant = 6 * 5 = 30

Auf diese Ergebnistabelle des inneren Verbunds können dann die Standardoperationen Projektion und Selektion angewandt werden. Insbesondere kann eine Verbundbedingung hinzugefügt werden, so dass sich mit der Bedingung:

Artikel.LNR = Lieferant.LNR

gerade der natürliche Verbund ergibt (und der Grad sich um eins verkleinert, da die gleichnamige Spalte nur einmal aufgeführt wird).

Variante 3: rechter/linker äußerer Verbunds (left / right outer join) zweier Tabellen

Betrachten wir jetzt den Fall, dass wir eine Tabelle erzeugen wollen, die **alle Lieferanten** enthält zusammen mit den Artikeln, für die der betreffende Lieferant der Hauptlieferant ist. Ist ein Lieferant für keinen Artikel Hauptlieferant, so sollen in der betreffenden Tabelle die Spalten aus der Tabelle Artikel leer bleiben:
In der Using-Klausel müssen auch hier wieder die gleichnamigen Spalte(n) aufgeführt werden, über die die beiden Tabellen verbunden werden sollen.

Ergebnis des äußeren Verbunds (Lieferant, Artikel) using(LNR):

LNR	Name	Ort	Status	ANR	Bezeichnung	Laenge	Breite	Material
1	Karcher	Jena	10					
4	Schmidt	Weima	20					
6	Mey	Gera	10	S2	Schrank	80	40	Holz
6	Mey	Gera	10	S1	Schrank	100	60	Holz
6	Mey	Gera	10	T1	Tisch	80	60	Kunststoff
8	Runge	Apolda	30	T2	Tisch	100	80	Kunststoff
8	Runge	Apolda	30	R2	Regal	80	60	Metall
8	Runge	Apolda	30	R1	Regal	80	40	Metall
9	Todd	Jena	30					

Da hier die Tabelle Lieferant im Verbund als erste Tabelle steht, bildet sie beim äußeren Verbund die Ausgangsbasis, d.h. zu jedem Wert in der Spalte LNR von Lieferant gibt es mindesten eine Zeile im Ergebnis, nämlich

- mit leeren Artikel-Spalten, wenn der Wert nicht in Artikel.LNR vorkommt
- mit den Zeilen aus dem inneren Verbund Lieferant,Artikel using(LNR) für den entsprechenden Wert in LNR

Demnach ist im Gegensatz zum inneren Verbund der äußere Verbund nicht symmetrisch, eine Tabelle bildet stets die Ausgangsbasis, wird die Reihenfolge vertauscht, ergibt sich i.A. ein anderes Ergebnis:

Ergebnis des äußeren Verbunds (Artikel, Lieferant) using(LNR):

LNR	ANR	Bezeichnung	Laenge	Breite	Material	Name	Ort	Status
6	S2	Schrank	80	40	Holz	Mey	Gera	10
6	S1	Schrank	100	60	Holz	Mey	Gera	10
6	T1	Tisch	80	60	Kunststoff	Mey	Gera	10
8	T2	Tisch	100	80	Kunststoff	Runge	Apolda	30
8	R2	Regal	80	60	Metall	Runge	Apolda	30
8	R1	Regal	80	40	Metall	Runge	Apolda	30

Da in der Tabelle Artikel (jetzt die Ausgangsbasis) in der Spalte LNR nur die Werte { 6, 8 } enthält, tauchen auch nur diese im äußeren Verbund auf, da es weiter zu jedem dieser Werte auch (genau) eine Zeile in Lieferant gibt, sind in allen Zeilen die Lieferanten-Spalten gefüllt.

3. SQL – Standard für Relationale Datenbanken

Die Syntax-Diagramme sind im Anhang A1 ausführlich beschrieben. Ein Rückgriff auf diese Syntax-Diagramme ist wohl immer dann notwendig, wenn ein SQL-Kommando aufgrund eines Syntaxfehlers nicht ausgeführt werden konnte. Die dabei erzeugte Fehlermeldung gibt zwar einen Hinweis auf die Ursache, aber in vielen Fällen bedarf es zur Fehlerkorrektur des Rückgriffs auf die formale Syntaxdefinition.
Hier an dieser Stelle beschränken wir uns darauf, die Syntax allgemein zu beschreiben und zu erläutern und versuchen so einen Einstieg in die deklarative Denkweise bei der Anwendung von SQL zu finden.

Mit SQL (Structured Query Language) steht eine genormte Sprache für die Kommunikation mit relationalen DBMS zur Verfügung.

Charakteristische Eigenschaften von SQL:

- ⇨ SQL ist einfach zu erlernen.
- ⇨ SQL kommt ohne Programmiersprachenkenntnisse aus.
- ⇨ SQL ist deklarativ, d.h. es wird beschreiben, welche Informationen ausgegeben werden sollen, nicht aber, wie dies geschehen soll.
- ⇨ SQL ist formatfreier Text, kann also mit jedem einfachen Texteditor geschrieben werden.
- ⇨ SQL stellt Funktionen für alle Benutzergruppen zur Verfügung.
- ⇨ SQL besitzt eine Befehlsstruktur, mit (englischen) Schlüsselworten für die einzelnen Anweisungen.
- ⇨ SQL ist ein internationaler Standard (z.Z. SQL3 von 1999), der von allen Anbietern von Datenbanksystemen unterstützt wird.

Übersicht SQL-Anweisungen:

- ⇨ zur Abfrage der aktuellen Datenbankinhalte (QL)
- ⇨ zur Verwaltung der Datenbankinhalte (DML)
- ⇨ zur Definition von Datenbankobjekten (DDL)
- ⇨ zur Spezifikation der Details der physischen Speicherung (DSL)
- ⇨ zur Verwaltung des Database Manangement Systems (DBMS)
- ⇨ zur Sicherung der Benutzer- und Zugriffskontrolle

3.1 Überblick Befehle zum Anlegen, Löschen und Ändern von Tabellen

3.1.1 Neuanlegen (CREATE-TABLE)

```
CREATE TABLE <Tabelle>
(
    <Spaltendefinition>,
    ……..,
    ……..,
    <Tabellenbedingung>,
    ……..,
    ……..
);
```

<Tabelle> steht dabei für den Tabellennamen, entweder in der Form <Schema>.<Tab-Name> oder einfach nur <Tab-Name>.

<Schema> steht dabei für den Benutzernamen (z.B. studetpm), in dessen Schema die Tabelle angelegt werden soll. <Tab-Name> steht dabei für den Namen der Tabelle (z.B. liefert), die angelegt werden soll.

Eine <Spaltendefinition> besteht dabei aus:

<Spalten> <Datentyp> DEFAULT <Ausdruck> <Spaltenbedingung>,

<Spalte> steht dabei für den Namen der Spalte (z.B. Preis)

<Datentyp> steht dabei für den Datentyp (z.B. CHAR(2), VARCHAR2(60), NUMBER(5,2))

<Ausdruck> steht dabei für einen gültigen Ausdruck, der den Wert festlegt, mit der die betreffende Spalte standardmäßig vorbelegt wird, wenn kein spezielle Wert (z.B. beim insert-Befehl) angegeben wird. Der Teil DEFUALT <Ausdruck> kann auch entfallen.

<Spaltenbedingung> steht dabei für eine der folgenden Bedingungen, die an die betreffende Spalte gestellt wird und die vom DBMS überwacht wird. Jeder Bedingung kann durch

CONSTRAINT <Name der Bedingung>

ein schemaweit eindeutiger Name gegeben werden, der bei entsprechenden Systemmeldungen benutzt wird. Mögliche Bedingungen sind:

NULL	die Spalte kann auch leer sein
NOT NULL	die Spalte darf nicht leer sein
PRIMARY KEY	die Spalte ist als Primärschlüssel für die Tabelle definiert
UNIQUE	die Spalte ist als (weiterer) Schlüssel definiert
CHECK (<Bedingung>)	für die Spalte wird ein Bedingung (z.B. Preis > 0.0) definiert
REFERENCES <Tabelle> (<Spalte>)	die Spalte ist als Fremdschlüssel auf <Spalte> in <Tabelle> definiert

<Tabellenbedingung> steht dabei für eine der folgenden Bedingungen, die an mehrere Spalten der betreffenden Tabelle gestellt wird und die vom DBMS überwacht wird, d.h. ist von einer Bedingung nicht nur eine einzelne, sondern mehrere Spalten der Tabelle betroffen, so muss die entsprechende Bedingung als <Tabellenbedingung> geschrieben werden.

Auch hierbei kann wieder jede Bedingung durch

CONSTRAINT <Name der Bedingung>

ein schemaweit eindeutiger Name gegeben werden, der bei entsprechenden Systemmeldungen benutzt wird. Mögliche Bedingungen sind:

PRIMARY KEY (<Spalte1>, <Spalte2>,…….)

UNIQUE(<Spalte1>, <Spalte2>,…….)

CHECK (<Bedingung>)

FOREIGN KEY (<Spalte1>, <Spalte2>,…) REFERENCES <Tabelle> (<Spalte1>, <Spalte2>,…..)

Beispiel:

Tabelle: **Hersteller (Primärschlüssel: HNR)**

HNR	Name
1	Badendorf
2	Gondi
3	Hankel
4	KMEX
9	Baff
10	Plendax
12	Boyer

Tabelle: **Erzeugnis (Primärschlüssel: ANR, Fremdschlüssel HNR, Referenz auf Hersteller)**

ANR	Bezeichnung	Preis	HNR
8	Parfüm	3.40	2
9	Rasierwasser	5.30	3
11	Zahncreme	1.20	10
13	Haarspray	7.40	1
15	Haarcreme	1.50	4
36	Shampoo	6.20	12
37	Shampoo	5.50	9

```
CREATE TABLE Hersteller
(
    HNR  NUMBER(2)  CONSTRAINT  Hersteller_PK  PRIMARY KEY,
    Name  VARCHAR2(20)
);

CREATE TABLE Erzeugnis
(
    ANR             NUMBER(2),
    Bezeichnung     VARCHAR2(20),
    Preis           NUMBER(5,2),
    HNR             NUMBER(2),
    CONSTRAINT  Erzeugnis_PK  PRIMARY KEY (ANR),
    CONSTRAINT  Erzeugnis_Hersteller_FK FOREIGN KEY ( HNR )
                REFERENCES Hersteller
);
```

3.1.2 Entfernen (DROP-TABLE)

DROP TABLE <Tabelle>

<Tabelle> steht dabei für den Tabellennamen, entweder in der Form <Schema>.<Tab-Name> oder einfach nur <Tab-Name>. <Schema> steht dabei für den Benutzernamen (z.B. studetpm), in dessen Schema die Tabelle entfernt werden soll. <Tab-Name> steht dabei für den Namen der Tabelle (z.B. liefert), die entfernt werden soll.

3.1.3 Ändern (ALTER-TABLE)

Dieser SQL-Befehl bietet zahlreiche Möglichkeiten, eine vorhandene Tabelle strukturell zu verändern, d.h. es können Spalten und Bedingungen gelöscht, hinzugefügt oder geändert werden. Einzelne Bedingungen können gezielt aktiviert (ENABLE) bzw. deaktiviert (DISABLE) werden. Darüber hinaus kann mit diesem SQL-Befehl der Name einer Tabelle geändert werden. Wir beschränken uns an dieser Stelle darauf, einige typische Anwendungen dieses Befehl beispielhaft vorzustellen. Eine ausführlichere Darstellung findet man im Anhang A1.

Beispiele: (Tabellen aus Beispiel in 3.1.1)

```
ALTER TABLE  Erzeugnis  MODIFY
(
    Preis       NUMBER(6,2)         DEFAULT  1.00
                        CONSTRAINT   Preis_positiv  CHECK (Preis > 0.00)
);

ALTER  TABLE  Hersteller  MODIFY   Erzeugnis_Hersteller_FK   DISABLE;

ALTER  TABLE  Hersteller  MODIFY   Erzeugnis_Hersteller_FK   ENABLE;

ALTER  TABLE  Hersteller  RENAME TO  Produzent;
```

3.2 Überblick Befehle zum Einfügen, Löschen und Ändern von Tabellenzeilen

3.2.1 Zeilen einfügen (INSERT)

Hier wird zunächst nur der einfache Fall betrachtet, dass die einzugebenden Zeilen als Werteliste gegeben sind:

INSERT INTO <Tabelle> (<Liste der Spalten>) VALUES (<Liste der Werte>);

Jeder einzelne Befehl fügt genau eine Zeile in die betreffende Tabelle <Tabelle> ein.

<Liste der Spalten>
ist dabei eine durch Komma getrennte Aufzählung der Spalten(namen)

<Liste der Werte>
ist dabei eine durch Komma getrennt Aufzählung konstanter Werte. Die Werteliste muss bezgl. Anzahl, Reihenfolge und Datentyp zur <Liste der Spalten> passen, d.h. in der einzufügenden Zeile wird der erste Wert der < Liste der Werte> in die erste Spalte der <Liste der Spalten> übernommen, der zweite Wert in die zweite Spalte usw.
Entsprechend dem Datentyp der Spalte muss sich an der entsprechenden Position in der <Liste der Werte> ein numerischer Wert (z.B. eine Konstante 5.2) oder eine Zeichenkette (z.B. 'Müller') befinden, die Zeichenkette muss dabei in einfache Hochkomma eingeschlossen sein

Die <Liste der Spalten> kann weggelassen werden, dann werden implizit alle Spalten in der Reihenfolge genommen, in die sie beim CREATE-TABLE-Befehl erstellt worden sind.
Ist für eine Spalte ein DEFAULT-Wert angegeben oder ist NULL ein zulässiger Wert, kann beim INSERT-Befehl auf diese Spalte (und ihren Wert) verzichtet werden, für sie wird dann automatisch der DEFAULT-Wert oder NULL übernommen.

Beispiele: (Tabellen aus Beispiel in 2.3)

```
INSERT INTO Lieferant (LNR, Status, Ort, Name)
            VALUES    (10 , 10 , 'Apolda' , 'Schulze');

INSERT INTO Lieferant VALUES (20 , 'Schneider', 'Weimar' , 10 );
```

Darüber hinaus gibt es noch die Möglichkeit, die einzufügenden Zeilen durch eine Abfrage bereitzustellen, d.h. es wird zunächst mit einem Select-Befehl eine temporäre Tabelle erstellt, deren Zeilen dann anschließend in die Tabelle eingefügt wird.

INSERT INTO <Tabelle> (<Liste der Spalten>) (<Abfrage>);

Die erste Varinate wird durch das Schlüsselwort VALUES festgelegt, bei der zweiten Variante fehlt dieses Schlüsselwort und an Stelle der Werteliste für eine einzelne Zeile ist eine Abfrage (Select-Befehl) angegeben.

Beispiel: (Tabellen aus Beispiel in 3.1.1)

insert into Hersteller (Select * from ss11etxyz.Hersteller);

Mit diesem insert-Befehl kann eine Tabelle (im Beispiel Hersteller) im eigenen Schema mit den Zeilen einer Tabelle (im Beispiel Hersteller) aus einen anderen Schema (im Beispiel ss11etxyz) gefüllt werden.

3.2.2 Zeilen löschen (DELETE)

Mit einem DELETE-Befehl können wahlweise alle oder ein Teil der Zeilen gelöscht werden.

DELETE FROM <Tabelle>;

DELETE FROM <Tabelle> WHERE <Bedingung>;

Die erste Variante löscht alle Zeilen der <Tabelle**>**, die zweite Variante löscht nur die Zeilen in der <Tabelle>, die die angegebene <Bedingung> erfüllen.

Beispiele: (Tabellen aus Beispiel in 2.1)

DELETE FROM Lieferant;

DELETE FROM liefert where ANR IN ('R1', 'R2');

3.2.3 Zeilen ändern (UPDATE)

Mit einem UPDATE-Befehl können eine oder mehrere Spalten für alle oder einen Teil der Zeilen einer Tabelle geändert werden. Hier wird zunächst nur der einfache Fall betrachtet, dass die die zu ändernden Spalten einzeln auf neue Werte gesetzt werden. Für jede zu ändernde Spalte wird in einer Art Zuweisung (SET-Klausel) der neue Wert ermittelt und zugewiesen, dabei kann auf den Wert vor der Änderung zurückgegriffen werden, z.B. werden mit

SET Preis = Preis * 1.05

die Werte in der Spalte Preis geändert und zwar um 5 % erhöht. Die Angabe Preis auf der rechten Seite des Gleichheitszeichens greift auf die Werte in Preis vor der Änderung zu, während die Angabe Preis auf der linken Seite des Gleichheitszeichen für den Wert nach der Änderung steht. Alle Änderungen werden in einer SET-Klausel zusammengefaßt.

SET-Klausel

SET <Spalte 1> = <Ausdruck 1>, <Spalte 2> = <Ausdruck 2>, ...

Analog zum DELETE-Befehl können auch die zu ändernden Zeilen durch eine Bedingung eingeschränkt werden:

UPDATE <Tabelle> SET-Klausel;
UPDATE <Tabelle> SET-Klausel WHERE <Bedingung>;

Die erste Variante ändert alle Zeilen in <Tabelle>, die zweite Variante ändert nur die Zeilen in <Tabelle>, die die angegebene <Bedingung> erfüllen.

Beispiele: (Tabellen aus Beispiel in 3.2)

UPDATE Artikel SET LAENGE = 90, BREITE = 90 WHERE ANR = ‘T2‘;

3.3 Befehl zur Abfrage und Auswahl von Informationen (SELECT)

Mit dem Select-Befehl können externe Sichten (englisch: views) definiert werden. Der Select-Befehl erstellt unter Verwendung der Standardoperationen (Projektion, Selektion und Verbund) eine Abfrage (englisch: Query), die als Ergebnis eine Tabelle (Relation) liefert. Diese Tabelle ist aber nicht statisch gespeichert, sondern wird bei jedem Zugriff dynamisch aus den verbundenen Tabellen erzeugt, deshalb nennt man sie eine Sicht (view).

Beispiele für Select-Befehle:

Die Beispiele beziehen sich alle auf die Datenbank mit den Tabellen Artikel, Lieferant und liefert:

Beispiel 1:

```
Select  ANR, Bezeichnung
from    Artikel
where   Material = 'Metall';
```

Schlüsselworte wie Select from
Bezeichner wie ANR Artikel
Konstanten wie 'Metall' (Textkonstante)
Operatoren wie = + - * /

Ergebnis:

ANR	Bezeichnung
R1	Regal
R2	Regal

Beispiel 2:

```
Select  LNR, Name, Status
from    Lieferant
where   Status = 10;
```

Konstanten wie 10 (Nummerische Konstante)

Ergebnis:

LNR	Name	Status
1	Karcher	10
6	Mey	10

Beispiel 3:

```
Select  LNR, Name, Status
from    Lieferant
where   Status = 10    and    Ort = 'Jena';
```

Komplexe Bedingung (UND-Verknüpfung)

Ergebnis:

LNR	Name	Status
1	Karcher	10

Beispiel 4:

```
Select  COUNT(*) AS Anzahl
from    Lieferant;
```

Aggregate-Funktionen wie COUNT SUM MAX
Alternative Bezeichner wie AS Anzahl

Ergebnis:

Anzahl
5

Beispiel 5:

```
Select  MIN(Laenge) AS Minimum,  MAX(Laenge) AS Maximum
from    Artikel;
```

Ergebnis:

Minimum	Maximum
80	100

Beispiel 6:

```
Select  ANR, Laenge * Breite AS Flaeche
from    Artikel
where   Bezeichnung > 'Stuhl';
```

Ergebnis:

ANR	Flaeche
T1	4800
T2	8000

Zur Beschreibung der Spalten einer Sicht werden Ausdrücke (englisch: expressions) verwendet, ebenso wie für die Formulierung von Bedingungen (englisch: conditions) bei der Selektion.

3.3.1 Ausdrücke und Bedingungen

Die Bildung von Ausdrücken ist schrittweise definiert. Ausgangspunkt sind die sogenannten einfachen Ausdrücken.

<einfacher Ausdruck> =
- <Schema>.<Tab-Name>.<Spaltenname>
- <Tab-Name>.<Spaltenname>
- <Spaltenname>
- Textkonstante, z.B. 'FH-JENA'
- numerische Konstante, z.B. 10 oder 12.5
- NULL
- Systemkonstante, z.B. Sysdate oder User

Diese einfachen Ausdrücke bilden die Basis für zusammengesetzte Ausdrücke :

<zusammengesetzter Ausdruck> =
- (<Ausdruck>)
- + <Ausdruck>
- - <Ausdruck>
- <Ausdruck> <op> <Ausdruck>
- F(<Ausdruck 1>,……,<Ausdruck n>)

<op> steht dabei für einen der gewöhnlichen arithmetischen Operatoren (+ , - , * , /) oder dem Operator ||, der für das Nebeneinanderschreiben (Konkatenation) von Text steht.

F steht dabei für eine vom DBMS (build-in) oder dem Anwender bereitgestellte Funktion.
Ein Ausdruck kann ausserdem noch eine Ausdrucksliste sein, d.h. eine in runden Klammern eingeschlossene Aufzählung von Ausdrücken, die einzelnen Ausdrücke sind durch ein Komma von einander getrennt.

<Ausdrucksliste> = (<Ausdruck 1>, <Ausdruck 2>,<Ausdruck 3>,……..,<Ausdruck n>)

Beispiele für numerische Funktionen (number_function):

SQRT	Quadratwurzel berechnen
ROUND	Auf- / Abrunden von Gleitkommazahlen
ABS	Betrag einer Zahl berechnen
MOD	Divisionsrest einer ganzen Zahl berechnen

Beispiele für Zeichenketten-Funktionen (character_function):

UPPER	Konvertierung in Großbuchstaben
LOWER	Konvertierung in Kleinbuchstaben
CONCAT	Verbinden von Zeichenketten (Konkatenation)
SUBSTR	Bestimmen eines Teils einer Zeichenkette
REPLACE	Ersetzen eines Teils in einer Zeichenkette
LENGTH	Bestimmen der Anzahl Zeichen einer Zeichenkette
RTRIM	Entfernen nachlaufender Leerzeichen in einer Zeichenkette
LTRIM	Entfernen vorlaufender Leerzeichen in einer Zeichenkette
TRIM	Entfernen vor- und nachlaufender Leerzeichen in einer Zeichenkette

Beispiele für Datum-/Konvertierungs-Funktionen (date_function/conversion_function):

Nicht immer sind die Namen der internen Funktionen in allen Datenbanksystemen identisch !

DATE_FORMAT	formatierte Ausgabe Datum / Zeit	**(MYSQL)**
TO_CHAR	formatierte Ausgabe Datum / Zeit	**(ORACLE)**

Darüber hinaus gibt es eine Vielzahl Datenbankspezifischer Funktionen zur Arbeit mit Datum- und Zeitangaben und weiter sind auch einige normale „Rechenoperationen“ auf Datum- und Zeitangaben übertragbar. So lässt sich etwa die Anzahl der Tage zwischen zwei Datumsangaben einfach als Differenz bestimmen und zu einer Datums- /Zeitangabe kann eine Anzahl Tage (Wochen, Monate, Stunden, Minuten, Sekunden) hinzuaddiert oder subtrahiert werden.

Beispiele für Aggregate-Funktionen (aggregate_function):

MIN	Minimum der (Spalten)Werte bestimmen
MAX	Maximum der (Spalten)Werte bestimmen
COUNT	Anzahl der (Spalten)Werte bestimmen
SUM	Summe der (Spalten)Werte bestimmen
AVG	Mittelwert der (Spalten)Werte bestimmen

Ausdrücke können nun benutzt werden, um Bedingungen zu formulieren, die in Selektionen verwendet werden können, um die Menge der ausgewählten Zeilen einzuschränken.

Eine Bedingung ist im einfachsten Fall ein einfacher Vergleich zweier Ausdrücke, also

<Ausdruck 1> <vergleichs-op> <Ausdruck 2>

dabei sind als Vergleichoperatoren die üblichen auch aus Programmiersprachen bekannten Operatoren möglich (Gleichheit: = , Ungleichheit: !=, <>, Größer bzw. kleiner: > , < , Größer gleich bzw. kleiner gleich: >=, <=)

Der zweite Ausdruck kann dabei auch durch eine Select-Anweisung bereitgestellt werden, eine solche select-Anweisung ist dann in Klammern zu setzten.

Weiter ist eine Bedingung möglich, bei der auf die Elementeigenschaft geprüft wird:
<Ausdruck> IN (<Ausdruck 1>, <Ausdruck 2>,…….., <Ausdruck n>)
bzw.
<Ausdruck> NOT IN (<Ausdruck 1>, <Ausdruck 2>,…….., <Ausdruck n>)

Die Ausdrucksliste kann dabei auch wieder durch eine Select-Anweisung bereitgestellt werden.

In einer Bedingung kann für einen Ausdruck ein Intervall angegeben werden :

<Ausdruck> BETWEEN <Ausdruck 1> AND <Ausdruck 2>

bzw.

<Ausdruck> NOT BETWEEN <Ausdruck 1> AND <Ausdruck 2>

Für die Prüfung, ob ein Ausdruck leer ist, gibt es eine Bedingung, die das spezielle Schlüsselwort NULL benutzt. Hierbei ist wichtig das eine solche Abfrage nicht mit den normalen Vergleichsoperatoren ausgeführt werden kann, sondern nur unter Verwendung des Schlüsselworts IS :

<Ausdruck> IS NULL bzw. <Ausdruck> IS NOT NULL

Für den Vergleich von alphanumerischen Daten mit einem Muster („Joker-Zeichen“) können auch nicht die normalen Vergleichoperatoren verwendet werden, sondern ein solcher Vergleich ist nur unter Verwendung des Schlüsselworts LIKE möglich:

<Ausdruck> LIKE <Muster> bzw. <Ausdruck> NOT LIKE <Muster>

Innerhalb von Muster steht dabei der Unterstrich (Zeichen: _) für ein einzelnes beliebiges Zeichen und das Prozentzeichen (Zeichen: %) für eine beliebige Anzahl μ 0 von beliebigen Zeichen. Diese Festlegung im SQL-Standard weicht von den aus dem Betriebssystemumfeld bekannten Jokerzeichen (Fragezeichen ? bzw. Stern *) ab.

Einzelne Bedingungen können durch die logischen Operatoren (NOT AND OR) zu komplexen Bedingungen zusammengestzt werden, will man dabei eine bestimmte Reihenfolge erzwingen, müssen Klammern gesetzt werden, genau so, wie man es aus der normalen Arithmetik kennt :

NOT <Bedingung>

<Bedingung 1> AND <Bedingung 2>

<Bedingung 1> OR <Bedingung 2>

dabei kann auch entsprechend geklammert werden, z.B.

NOT (<Bedingung 1> AND <Bedingung 2>)

(NOT <Bedingung 1>) AND <Bedingung 2>

(<Bedingung 1> OR <Bedingung 2>) AND <Bedingung 3>

Beispiele:

Ausdrücke:

((studetxy.erzeugnis.preis * 100) * globals.faktor)

(Laenge, ‘Laenge’, 100, NULL)

CONCAT(Vorname, Nachname) oder Vorname || Nachname

TO_CHAR(SYSDATE,’DD.MM.YYYY’)

Date_FORMAT(CURDATE() + 10, GET_FORMAT(DATETIME,'EUR'));

COUNT(HNR)

Bedingungen:

Artikel.Laenge > Artikel.Breite

Wert IN (10, 20, 30)

Status IS NULL

Wert BETWEEN (Laenge – 10) AND (Laenge + 10)

Status > 10 OR UPPER(Name) LIKE ‘%FH-JENA%’

3.3.2 Einfache Abfragen (Grundmodell ohne Gruppierung)

Eine Abfrage (Select-Anweisung) setzt sich aus mehreren Teilen zusammen:

Die Anweisung beginnt stets mit dem Schlüsselwort SELECT gefolgt von einer Liste von Ausdrücken. Die Ausdrücken legen dabei die Reihenfolge, die Datentypen und die Inhalte der Spalten der Abfrage fest. Für jeden Ausdruck (d.h. jede Spalte) kann noch eine Überschrift (Alias) angegeben werden, diese wird entweder durch Leerzeichen oder durch das Schlüsselwort AS getrennt unmittelbar hinter den betreffenden Ausdruck geschrieben:

SELECT <Ausdruck 1> <Überschrift 1>, <Ausdruck 2> <Überschrift 2>, ……..

dabei kann bei einem oder allen Ausdrücken die zusätzliche Angabe einer Überschrift entfallen.

Als nächstes folgt das Schlüsselwort FROM gefolgt von einer Liste von Tabellennamen, für die auch wiederum ein Alias angegeben werden kann, der aber jetzt durch Leerzeichen getrennt unmittelbar hinter den Tabellennamen gechrieben werden muss:

FROM <Tabelle 1> <Alias 1>, <Tabelle 2> <Alias 2>,……

die Tabellenname können dabei wieder wahlweise einschliesslich des Schemanamens oder ohne Schemanamen angegeben werden. Alle hier aufgeführten Tabellen werden als innerer Verbund zusammengeführt, d.h. es wird eine Tabelle erzeugt, die aus allen Spalten aller aufgeführten Tabellen besteht und deren Zeilen sich ergeben, in dem jede Zeile jeder Tabelle mit jeder Zeile jeder anderen Tabellen kombiniert wird. Dabei ist darauf zu achten, dass Referenzen auf Tabellenspalten immer so angeben werden, dass sie eindeutig eine Spalte der aufgeführten Tabellen bestimmen, um Mehrdeutigkeiten auszuschliessen. Die Sonderformen des Verbunds werden ebenfalls im from-Teil angegeben, z.B. für zwei Tabellen:

FROM <Tabelle 1>	NATURAL JOIN <Tabelle 2>
FROM <Tabelle 1>	INNER JOIN <Tabelle 2> USING (<Ausdruck 1>,<Ausdruck 2>…..)
FROM <Tabelle 1>	INNER JOIN <Tabelle 2> ON <Verbund-Bedingung>
FROM <Tabelle 1>	LEFT OUTER JOIN <Tabelle 2> USING (<Ausdruck 1>,<Ausdruck 2>…..)
FROM <Tabelle 1>	RIGHT OUTER JOIN <Tabelle 2> USING (<Ausdruck 1>,<Ausdruck 2>…..)

Schließlich kann im WHERE-Teil eine Bedingung angegeben werden, die die ausgewählten Zeilen bestimmt

WHERE <Bedingung>

Eine solche Abfrage nennen wir eine einfache Abfrage (eventuell mit Aggregate-Funktionen aber ohne Gruppierung) und es ergibt sich folgende Struktur für den allgemeine Fall eines inneren Verbunds:

SELECT <Ausdruck 1> <Überschrift 1>, <Ausdruck 2> <Überschrift 2>, ……..
FROM <Tabelle 1> <Alias 1>, <Tabelle 2> <Alias 2>,……
WHERE <Bedingung>

3.3.3 Einfache Abfragen (Grundmodell mit Gruppierung)

Im vorangegangen Abschnitt wurden zunächst nur solche Abfragen behandelt, die in der Ausdrucksliste entweder garkeine Aggregate-Funktionen verwendet haben oder Aggregate-funktionen immer nur auf alle Zeilen der Tabelle angewendet wurden.

Aggregate-Funktionen operieren auf den Zeilen und fassen gemäß der betreffenden Funktionn alle Zeilen zu einem Wert zusammen, so zählt z.B. die Abfrage

```
SELECT  COUNT(lnr)  AS Anzahl  FROM Lieferant;
```

die Anzahl der Zeilen der Tabelle Lieferant

Anzahl
5

während die Abfrage

```
SELECT MIN(Breite) AS Min_Breite, MAX(Breite) AS Max_Breite FROM Artikel;
```

in den Zeilen der Tabelle Artikel den kleinsten und den größten Wert in der Spalte Breite bestimmt

Min_Breite	Max_Breite
40	80

d.h. beide Abfragen liefern nur einen einzige Zeile als Ergebnis, obwohl die betrachteten Tabellen wesentlich mehr Zeilen haben.

Häufig möchte man aber eine Aggregate-Funktion nicht auf alle Zeilen einer Tabelle anwenden, sondern nur auf bestimmte Gruppen von Zeilen, so möchte man z.B. nicht die Gesamtzahl aller Lieferanten, d.h. die Anzahl der Zeilen in der Tabelle Lieferant, wissen, sondern die Anzahl der Lieferanten aus den einzelnen Orten, d.h. man möchte die Zeilen der Tabelle Lieferant zunächst in Gruppen einteilen (Kriterium: gleiche Werte in der Spalte Ort) und dann die Aggregatefunktion COUNT gruppenweise anwenden:

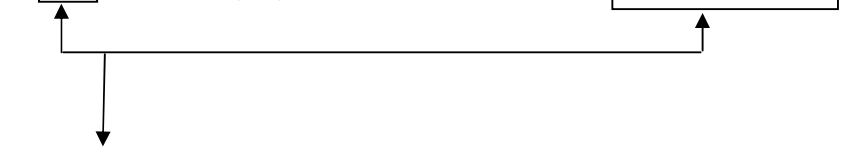
```
SELECT Ort , COUNT(lnr)  AS Anzahl  FROM Lieferant GROUP BY Ort;
```

Ort	Anzahl
Apolda	1
Gera	1
Jena	2
Weimar	1

d.h. man kann normale Ausdrücke ohne Aggreagate-Funktionen (im Beispiel **Ort**) und solche mit Aggreagate-Funktionen (im Beispiel **COUNT(lnr) AS Anzahl**) im SELECT-Teil mischen, muss dann aber mit dem Zusatz GROUP BY alle Ausdrücke ohne Aggregate-Funktionen aufführen (im Beispiel GROUP BY **Ort**).

Möchte man z.B. wissen, wieviele Artikel von den einzelnen Lieferanten geliefert werden, so kann das mit der Aggregate-Funktion COUNT und Gruppierung für die Lieferantennummer lnr ermittelt werden:

SELECT lnr , COUNT(anr) AS Anzahl FROM liefert GROUP BY lnr;



lnr	Anzahl
1	2
4	3
6	3
8	6
9	2

Analog läßt sich für jeden Artikel z.B. der kleinste Preis bestimmen:

SELECT anr , MIN(Preis) AS Min_Preis from liefert GROUP BY anr;

anr	Min_Preis
R1	200,00
R2	250,00
S1	50,00
S2	35,00
T1	100,00
T2	90,00

Häufig macht es Sinn, an die Ergebnistabelle einer solchen SELECT-Anweisung weitere einschränkende Bedingungen zu stellen. Wenn wir alle Orte wissen möchten, aus denen es z.B. mehr als einen Lieferanten gibt, so könnte man die SELECT-Anweisung

SELECT Ort, COUNT(lnr) AS Anzahl FROM Lieferant GROUP BY Ort;

Ort	Anzahl
Apolda	1
Gera	1
Jena	2
Weimar	1

als Ausgangspunkt nehmen und müßte dann für die Ergebnistabelle mit der Bedingung COUNT(lnr) > 1 die Zeilen auf die gewünschten Zeilen reduzieren. Eine solche Bedingung, die eine Aggregatefunktion enthält, kann aber nicht im WHERE-Teil eingesetzt werden, da sich die Selektion aus dem WHERE-Teil auf die Ausgangstabelle (im Beispiel Lieferant) bezieht, diese Tabelle aber keine Spalte COUNT(lnr) enthält.

Für derartige Bedingungen, also für Bedingungen, die an die Ergebnistabelle einer Aggregation gestellt werden, gibt es deshalb ein eigenes Schlüsselwort (HAVING) und dieser HAVING-Teil muss deshalb zwingend direkt im Anschluss an den Zusatz GROUP BY stehen:

```
SELECT  Ort, COUNT(lnr)  AS Anzahl
FROM Lieferant
GROUP BY Ort
HAVING COUNT(lnr) > 1;
```

Ort	Anzahl
Jena	2

in der Bedingung nach HAVING muss aber wieder der betreffende Ausdruck vollständig aufgeführt werden, es reicht hierbei nicht aus, nur den entsprechenden Spaltenalias (Überschrift) zu verwenden, d.h. die folgende SELECT-Anweisung wird nicht ausgeführt, sondern liefert eine Fehlermeldung:

```
SELECT  Ort, COUNT(lnr)  AS Anzahl
FROM Lieferant
GROUP BY Ort
HAVING Anzahl > 1;
```

Eine einfache Abfrage (mit Aggregate-Funktionen und mit/ohne Gruppierung) hat also eine der folgenden Grundstrukturen:

```
SELECT      <Ausdruck 1> <Überschrift 1>, <Ausdruck 2> <Überschrift 2>, ……..
FROM        <Tabelle 1> <Alias 1>, <Tabelle 2> <Alias 2>,……
WHERE       <Bedingung>
```

```
SELECT      <Ausdruck 1> <Überschrift 1>, <Ausdruck 2> <Überschrift 2>, ……..
FROM        <Tabelle 1> <Alias 1>, <Tabelle 2> <Alias 2>,……
WHERE       <Bedingung>
GROUP BY <Ausdruck a>, <Ausdruck b>,…..
```

```
SELECT      <Ausdruck 1> <Überschrift 1>, <Ausdruck 2> <Überschrift 2>, ……..
FROM        <Tabelle 1> <Alias 1>, <Tabelle 2> <Alias 2>,……
WHERE       <Bedingung 1>
GROUP BY    <Ausdruck a>, <Ausdruck b>,…..
HAVING      <Bedingung 2>
```

Als FROM-Klausel ist hier stets die allgemeine Form eines Tabellenverbunds angegeben, diese kann natürlich auch alternativ eine der Varianten für einen natürlichen, inneren oder äußeren Verbund sein:

3.3.4 Komplexe Abfragen (Mengentheoretische Operationen)

Einfache Abfragen können mit Hilfe der mengentheoretischen Operationen „Vereinigung“ (Schlüsselwort: UNION), „Durchschnitt“ (Schlüsselwort: INTERSECT) oder „Differenz“ (Schlüsselwort: MINUS) zu komplexen Abfragen zusammengesetzt werden:

<Abfrage 1> UNION <Abfrage 2>

<Abfrage 1> INTERSECT <Abfrage 2>

<Abfrage 1> MINUS <Abfrage 2>

(auch hier können wieder Klammern gesetzt werden, um eine bestimmte Reihenfolge der mengentheoretischen Operationen vorzugeben)

Jede einfache oder komplexe Abfrage kann schließlich als letztes durch einen einzigen ORDER BY Teil abgeschlossen werden, in dem eine Liste von Ausdrücken angegeben werden kann, nach der das Ergebnis der Abfrage sortiert (aufsteigend: ASC oder absteigend: DESC) angezeigt werden soll:

ORDER BY <Ausdruck 1> <Sortierung 1>, <Ausdruck 2> <Sortierung 2>, ……

<Sortierung n> steht dabei für ASC oder DESC. Die Sortierangabe kann auch weggelassen werden, dann gilt explizit aufsteigende Sortierung. An Stelle eines kompleten Ausdrucks kann auch ein Spaltenalias (Überschrift) aus dem SELECT-Teil angegeben werden oder einfach die laufenden Nummer einer Spalte im SELECT-Teil der Abfrage.

Beispiel:

Tabelle: **Hersteller (Primärschlüssel: HNR)**

HNR	Name
1	Badendorf
2	Gondi
3	Hankel
4	KMEX
9	Baff
10	Plendax
12	Boyer

Tabelle: **Erzeugnis (Primärschlüssel: ANR, Fremdschlüssel HNR, Referenz auf Hersteller)**

ANR	Bezeichnung	Preis	HNR
8	Parfüm	3.40	2
9	Rasierwasser	5.30	3
11	Zahncreme	1.20	10
13	Haarspray	7.40	1
15	Haarcreme	1.50	4
36	Shampoo	6.20	12
37	Shampoo	5.50	9

```
SELECT Name FROM Hersteller WHERE HNR IN
( SELECT HNR FROM Erzeugnis WHERE Bezeichnung='Shampoo' ) );
```

In diesem Beispiel wird im WHERE-Teil eine Elementbedingung formuliert, bei der die Liste durch eine SELECT-Anweisung bestimmt wird, d.h. es wird hier zunächst die SELECT-Anweisung:

```
SELECT HNR FROM Erzeugnis WHERE Bezeichnung='Shampoo';
```

ausgeführt, die für die aktuellen Daten als Ergebnis die Liste (12, 9) liefert, anschliessend wird dann die SELECT-Anweisung:

```
SELECT Name FROM Hersteller WHERE HNR IN (12, 9);
```

ausgeführt, die für die aktuellen Daten als Ergebnis

Name
Baff
Boyer

Diese Abfrage kann natürlich auch als allgemeiner Verbund der Tabellen Erzeugnis und Hersteller geschrieben werden. Da beim allgemeinen Verbund zweier Tabellen jede Zeile der einen mit jeder Zeile der zweiten kombiniert wird, muss eine Bedingung eingefügt werden, die nur solche Kombinationen zulässt, die der durch den Fremdschlüssel gegebenen Beziehung entsprechen:

```
SELECT  Name
FROM    Hersteller, Erzeugnis
WHERE   Hersteller.HNR = Erzeugnis.HNR AND Bezeichnung='Shampoo';
```

oder einfacher explizit als inneren Verbund „über LNR“:

```
SELECT  Name
FROM    Erzeugnis INNER JOIN Lieferant  USING(LNR)
WHERE   Bezeichnung='Shampoo';
```

oder noch einfacher als natürlicher Verbund (da Primär- und Fremdschlüssel gleichnamig)

```
SELECT  Name
FROM    Erzeugnis NATURAL JOIN Lieferant
WHERE   Bezeichnung='Shampoo';
```

Abfrage 1:

```
SELECT  Erzeugnis.*, Name
FROM    Erzeugnis, Hersteller
WHERE   Erzeugnis.HNR = Hersteller.HNR
   AND  Preis < 5.00;
```

Ergebnis:

ANR	Bezeichnung	Preis	HNR	Name
8	Parfüm	3.40	2	Gondi
11	Zahncreme	1.20	10	Plendax
15	Haarcreme	1.50	4	KMEX

Abfrage 2:

```
SELECT    Bezeichnung, COUNT(HNR) ANZ_HS, AVG(Preis) AVG_Preis
FROM      Erzeugnis
GROUP BY  Bezeichnung
HAVING    COUNT(HNR) > 1;
```

Ergebnis:

Bezeichnung	ANZ_HS	AVG_Preis
Shampoo	2	5.85

Abfrage 3:

```
SELECT    Produzent.Name  Hersteller
FROM      (Select * from studetxy.Hersteller)  Produzent
WHERE     Produzent.HNR > 9;
```

Ergebnis:

Hersteller
Plendax
Boyer

Abfrage 4:

```
SELECT    LNR, Name,Ort, '+' AS Bewertung
FROM      Lieferant  WHERE  Status = 10
UNION
SELECT    LNR, Name,Ort, '++' AS Bewertung
FROM      Lieferant  WHERE  Status = 20
UNION
SELECT    LNR, Name,Ort, '+++' AS Bewertung
FROM      Lieferant  WHERE  Status = 30;
```

Ergebnis:

LNR	Name	Ort	Bewertung
1	Karcher	Jena	+
4	Schmidt	Weimar	++
6	Mey	Gera	+
8	Runge	Apolda	+++
9	Todd	Jena	+++

3.3.5 Speicherung von Abfragen (CREATE VIEW)

Analog zum SQL-Befehl zum erstellen einer statischen Tabelle (CREATE TABLE ....) gibt es auch einen entsprechenden SQL-Befehl um eine dynamische Sicht zu erstellen, in diesem Fall wird aber nicht die Ergebnistabelle gespeichert, sondern die zugehörige SELECT-Anweisung, d.h. mit dem CREATE-VIEW-Befehl wird eine benannte Abfrage erstellt, diese kann danach dann analog zu (Basis)Tabellen direkt über den Namen angesprochen werden und z.B. in der from Klausel des Select-Befehls verwendet werden.

SQL-Syntax:

CREATE OR REPLACE VIEW <view> AS (<SELECT-Anweisung ohne ORDER BY>)

Die SELECT-Anweisung darf jetzt aber keine ORDER BY Angaben haben, da erst bei einem späteren Zugriff auf die gespeicherte Abfrage die Zeilen in der Ausgabe entsprechend angeordnet werden.

Beispiel:

```
CREATE OR REPLACE   VIEW Shampoo_Hersteller AS (
   SELECT   Name    FROM   Hersteller    WHERE    HNR     IN
( SELECT   HNR     FROM   Erzeugnis    WHERE    Bezeichnung='Shampoo' ) );
```

Auf diese Abfrage kann jetzt über den Namen Shampoo_Hersteller zugegriffen werden und dieser Name kann analog zu den Tabellen im FROM-Teil einer SELECT-Anweisung verwendet werden:

```
SELECT  COUNT(Name) AS Anzahl
FROM     Shampoo-Hersteller;
```

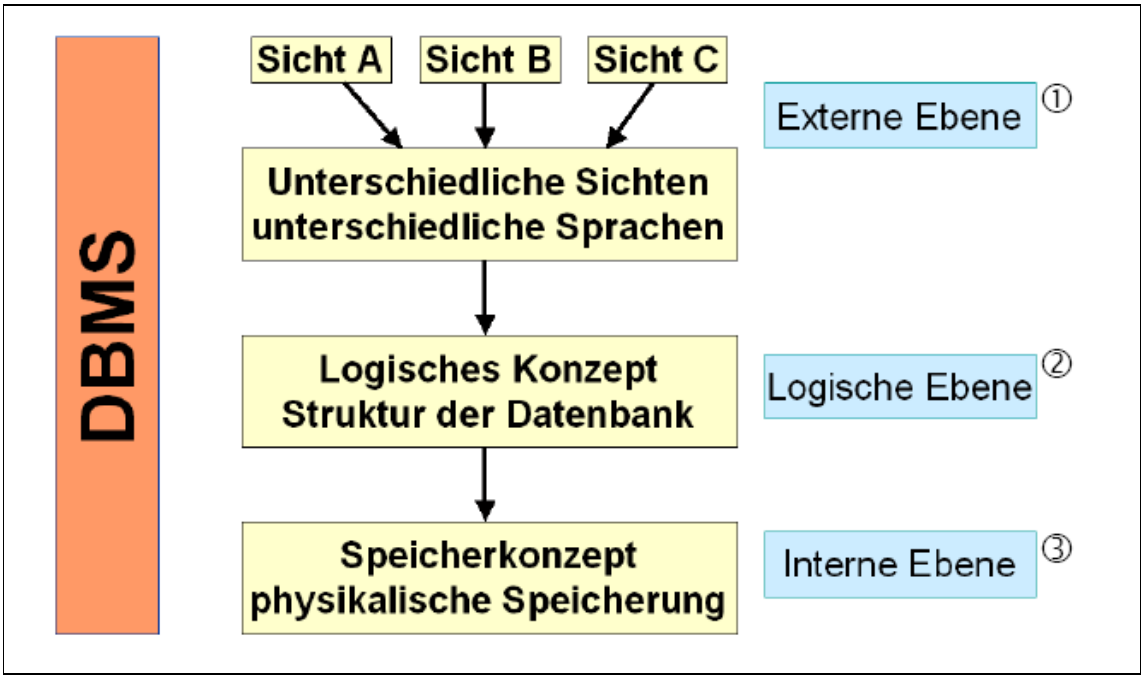
Anzahl
2

4. Datenbank-Architektur

4.1 Dreischichten-Modell

Betrachtet man die mit dem Einsatz von Datenbanksystemen verbundenen Zielsetzungen:

- Isolierung von Programmen und Daten
- Unterstützung mehrerer Benutzersichten
- Verwendung eines Katalog zur Speicherung der Datenbankbeschreibung



so hat sich daraus eine grundlegende Architektur für Datenbanksysteme ergeben, die heute als allgemeingültig angesehen werden kann, und die zumindest allen traditionellen Datenbank Systemen zugrunde liegt:

Man spricht hier von einer Drei-Schichten–Architektur, bei der für den allgemeinen Aufbau von Datenbanksystemen drei Schichten (Ebenen) unterschieden werden.

⇨ Interne Ebene (internes Schema)
Legt die Details der tatsächlichen (physischen) Speicherung der Daten und der für den Zugriff notwendigen Zugriffspfade fest.

⇨ Konzeptuelle Ebene (konzeptuelles Schema)
Beschreibt abstrahierend in einem (logischen) Modell die Gesamtheit der zu speichernden Daten, ihren Aufbau, ihre Wertebereiche, ihre Bedeutung und ihre Beziehungen zueinander und zwar in einer Form, die unabhängig von jeder tatsächlichen physischen Repräsentation ist.

⇨ Externe Ebene (externes Schema, Benutzersichten)
Beschreibt unterschiedliche Sichten auf die gespeicherten Daten, wie sie für verschiedene Benutzergruppen vorhanden sind. Jede Sicht beschreibt dabei nur den für die spezielle Benutzergruppe relevanten Ausschnitt der Datenbank. Alle gespeicherten Daten außerhalb der speziellen Sicht sind dabei für die betreffende Benutzergruppe nicht sichtbar und können nicht verarbeitet werden.

Dabei ist zu beachten, dass nur auf der internen Ebene tatsächlich Daten physisch existieren, die anderen beiden Ebenen enthalten nur Beschreibungen der Daten des DBS. Dieses Architekturmodell sichert die genannten Zielsetzungen.

Die mittlere sogenannte konzeptuelle Ebene stellt eine von der internen physischen Speicherung und der externen Nutzung der Daten unabhängige Beschreibung der Datenbank dar, diese Ebene ist die Beschreibung der „Miniwelt“ (d.h. des Ausschnitts der Welt), über die in der Datenbank Daten gespeichert sind und die den Daten in der Datenbank ihre Bedeutung geben (als Abbild der Wirklichkeit). Eine solche Beschreibung (konzeptuelles Schema genannt), sollte unabhängig von den konkreten Anwendungen existieren, die auf den Daten der Datenbank operieren.

Die Trennung von interner Speicherung und externer Verarbeitung und die Einführung einer dazwischen liegenden Schicht in Form eines konzeptuellen Schemas stellt die Grundlage für die Architektur von Datenbanksystemen dar.

Wichtig ist die Unterscheidung zwischen den Metadaten (Schemadefinition, Katalog, Data Dictionary,....) und den Anwendungsdaten. Die Metadaten ändern sich im Allgemeinen nicht sehr oft, obwohl gerade die Möglichkeit, diese Daten ohne großen Aufwand jederzeit ändern zu können, viel zur Flexibilität des Datenbankansatzes beiträgt.

Die eigentlichen Daten einer Datenbank ändern sich im Allgemeinen sehr häufig, die zu einem bestimmten Zeitpunkt in der Datenbank gespeicherten Daten werden als Datenbankzustand bezeichnet. Jedes Datenelement eines Datenbankzustandes stellt ein aktuelles Vorkommen dar (englisch orrurence). Jede Manipulation der Datenbank (Einfügen, Ändern oder Löschen von Daten) verändert den aktuellen Zustand der Datenbank.

Die Unterscheidung zwischen Datenbankschema und Datenbankzustand ist wichtig. Bei der Definition einer Datenbank (oder eines Teils davon) spezifiziert man ihr Schema, zu diesem Zeitpunkt befindet sich die Datenbank im leeren Zustand, dies ist quasi der Anfangszustand der Datenbank, Mit jeder Manipulation der Datenbank erhalten wir einen neuen Datenbankzustand, d.h. zu einem beliebigen Zeitpunkt besitzt die Datenbank einen aktuellen Zustand, das DBMS hat dabei dafür zu sorgen, dass jeder Zustand die im Schema definierte Struktur und alle auf ihr definierten Integritätsbedingungen erfüllt, man spricht dann von einem konsistenten (gültigen) Zustand.

Ausgehend vom Anfangszustand muss das DBMS sicherstellen, dass nur solche Manipulationen der Datenbank durchgeführt werden können, die von einem konsistenten Zustand wieder in eine neuen konsistenten Zustand führen. Die Überwachung der Integritätsbedingungen entlastet die Anwendungen von entsprechender Programmlogik und reduziert potentielle Fehlermöglichkeiten, da die Überwachung der Konsistenz zentral an einer Stelle (nämlich im DBMS) durchgeführt wird.

4.2 Verteilungsaspekte

Aktuelle Datenbanksysteme unterstützen im Allgemeinen den Client/Server-Ansatz, d.h. eine Datenbankanwendung teilt sich auf in ein

→ Front-End
Benutzernaher Anwendungsteil mit i. A. graphischer Benutzerschnittstelle und Anwendungslogik

→ Back-End
Datenhaltungsnaher Anwendungsteil mit Anwendungslogik und DBS

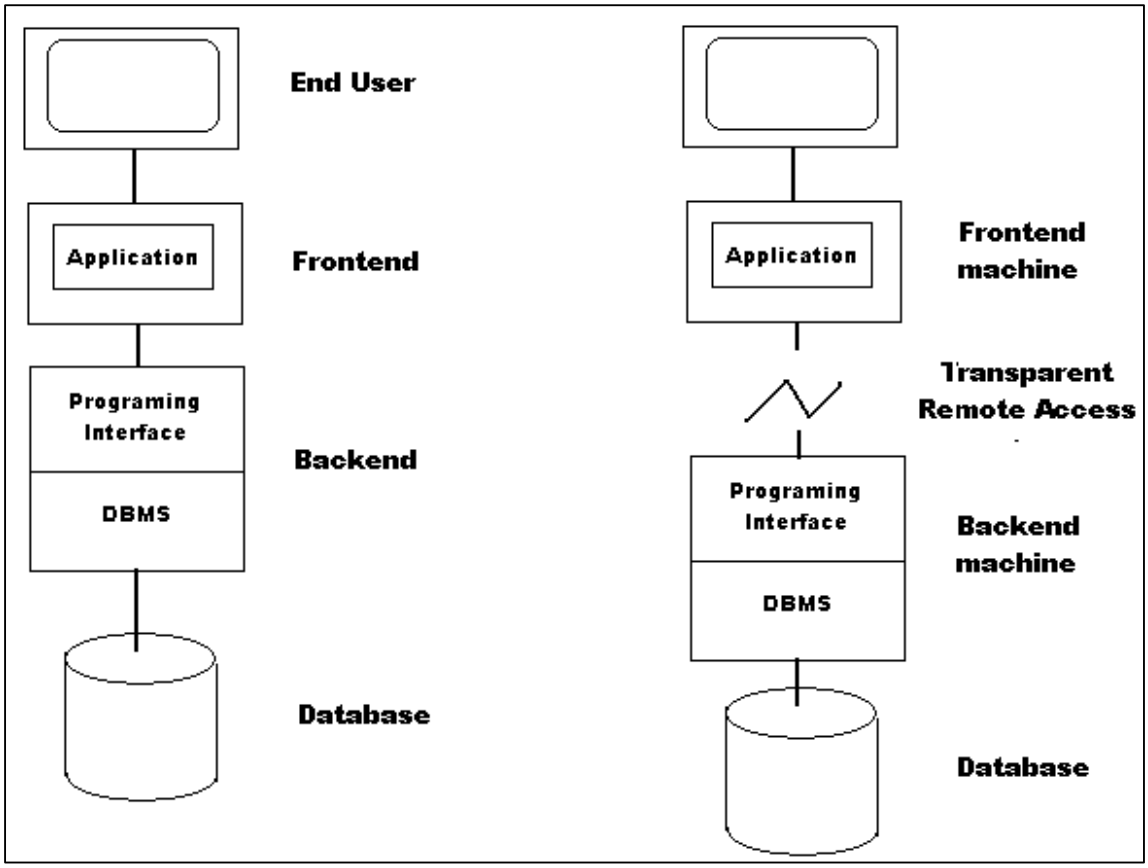
Eventuell ist das Front-End noch weiter in Graphische Benutzeroberfläche und Anwendungslogik aufgeteilt, so dass sich eine Dreiteilung:

- Graphische Benutzeroberfläche (z.B. Browser)
- Anwendungslogik (z.B. Applikationserver)
- Datenhaltung (z.B. Datenbank)

In vielen Fällen ist aber die Anwendungslogik sowohl Teil des Front- als auch des Back-End.

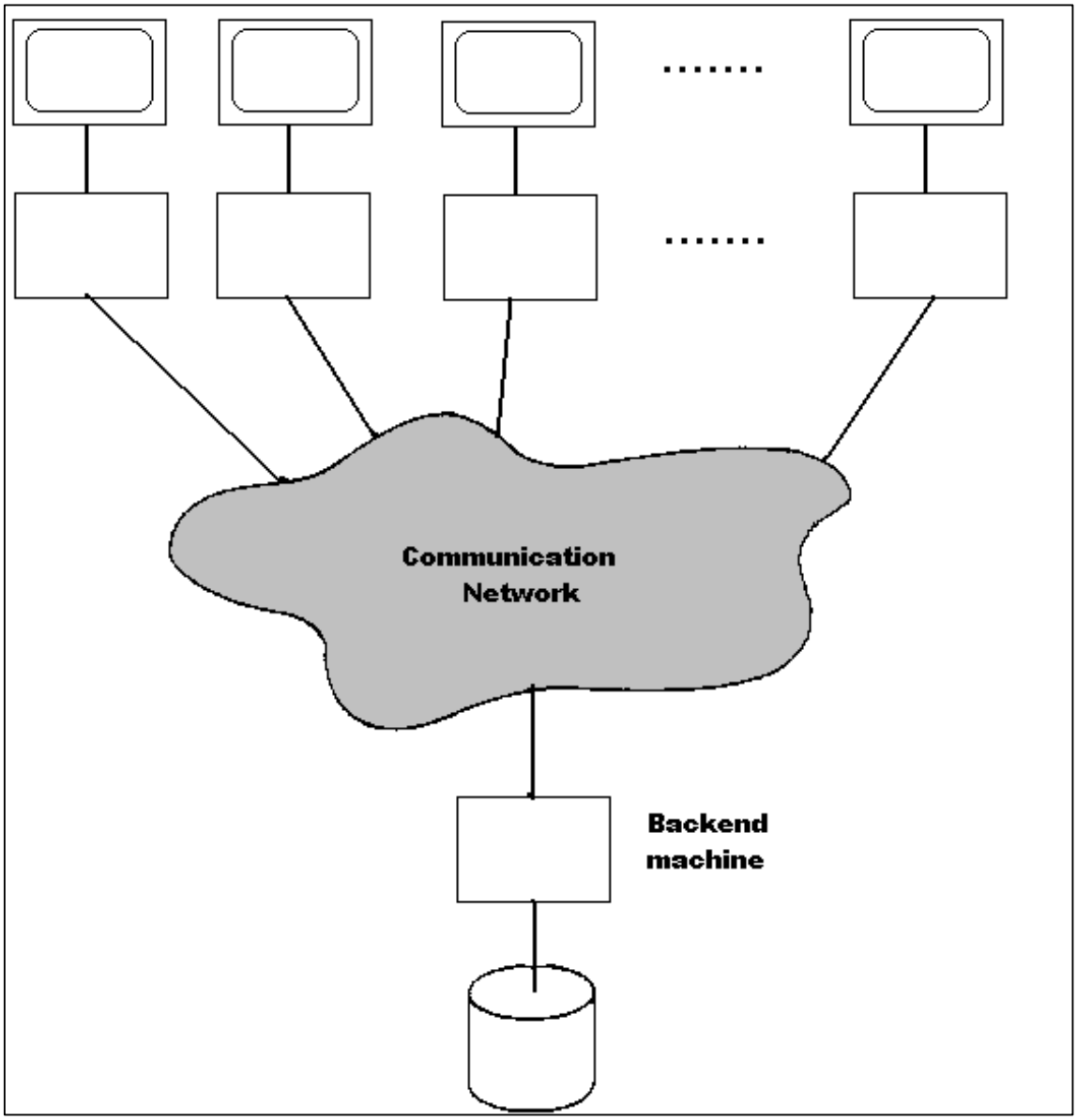
Für die Verteilung von Front-End und Back-End ergeben sich verschiedene Möglichkeiten:

Variante 1:
Anwendung und Datenbank befinden sich auf einem Rechner oder Anwendung und Datenbank befinden sich auf zwei Rechner, die für den Benutzer transparent über ein Netzwerk miteinander verbunden sind.



Variante 2:

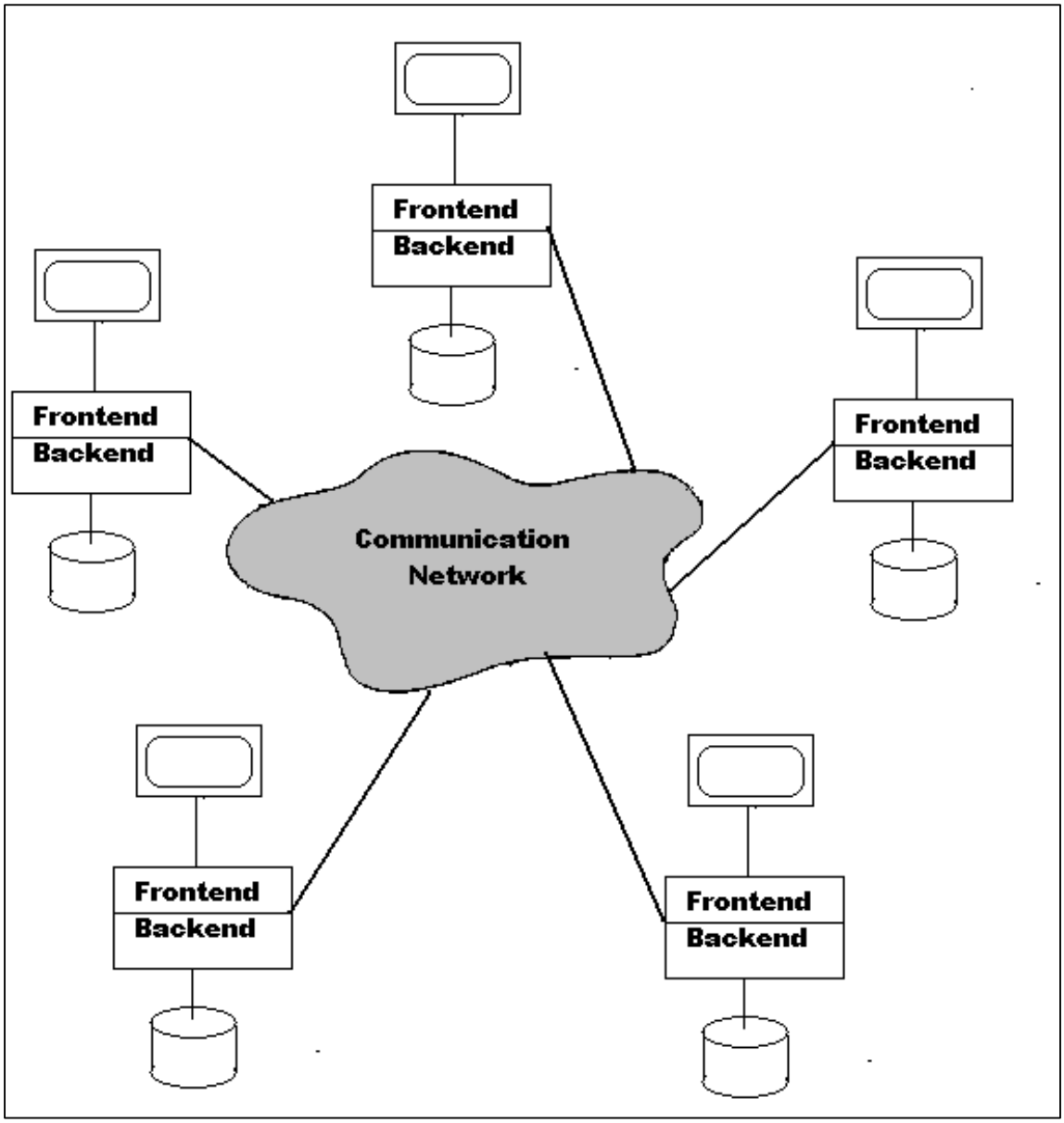
Die Datenbank befindet sich auf einem Rechner (Datenbankserver) und mehrere auf unterschiedlichen Rechnern verteilte Anwendungen kommunizieren über ein Netzwerk mit dem Datenbankmanagementsystem. Alle Daten befinden sich ausschließlich auf nur einem Rechner (Datenbankserver). Auch hierbei ist der Zugriff über das Netzwerk wieder transparent für die Benutzer.



Eventuell können dabei auch mehrere Datenbanksysteme vorhanden sein, die aber unabhängig voneinander arbeiten, d.h. in diesem Fall kommunizieren mehrere Anwendungen unabhängig voneinander mit mehreren Datenbanksystemen und benutzen dabei dasselbe Netzwerk.

Variante 3:

Im Gegensatz zur Variante 2 verfügt nun jeder beteiligte Rechner über ein eigenes (lokales) Datenbanksystem. Es gibt keinen ausgewählten Rechner mit einem zentralen Datenbanksystem. Die Daten sind auf alle Datenbanksysteme im Rechnernetz verteilt. Eine Anwendung wendet sich dabei stets an das lokale DBS, in dessen DBMS entschieden wird, ob die angeforderte Funktionalität allein lokal verfügbar ist, oder ob DBS anderer Rechner benötigt werden. Auch hier erfolgt ein Zugriff über das Netzwerk wieder transparent für den Benutzer.



Sind die verwendeten Datenbanksysteme alle von der gleichen Art (z.B. alles oracle oder alles IBM DB2) so spricht man von einem verteilten, homogenen Datenbanksystem, sonst von einem verteilten heterogenen Datenbanksystem.

4.3 Klassifikation von Datenbank-Management-Systemen

Datenbankmodell

Grundlage für die Strukturierung der Daten und ihrer Beziehungen zueinander ist das **Datenbankmodell**, das durch den DBMS-Hersteller festgelegt wird. Je nach Daten*bank*modell muss das **Datenmodell** an bestimmte Strukturierungsmöglichkeiten angepasst werden:

- hierarchisch:

Dieses historisch älteste Modell (typische Vertreter IMS der Fa IBM für den Großrechnerbereich) basiert auf einem hierarchischen Modell für die Speicherung der Daten, d.h. analog zum klassischen hierarchischen Unternehmensmodell mit einzelnen Hierarchiestufen und einem eindeutigen Zugriffspfad von einem ausgezeichneten Einstiegsknoten (Wurzel = Unternehmensleitung) über mehrere Hierarchiestufen bis zu den Datensätzen (Blätter = Mitarbeiter ohne Personalverantwortung). Eine solche Struktur wird Baum genannt, es gibt in solchen Strukturen immer genau einen Weg vom ausgezeichneten Knoten (Wurzel) zu dem einen einzelnen Blatt. Wegen der Hierarchie spricht man hier von Parent/Child-Elementen.

- netzwerkartig:

Das Netzwerkdatenmodell (typischer Vertreter IDMS der Fa. Computer Associates) unterscheidet sich vom hierarchischen Modell nur dadurch, dass nicht mehr eine einfache Baumstruktur zugrunde gelegt wird, sondern ein allgemeineres Netzmodell, bei dem es mehrere ausgezeichnete Knoten als Einstiegspunkte geben kann und dann auch mehrere Zugriffspfade von dort zu einem einzelnen Blatt. Man spricht dann auch nicht mehr von Parent/Child-, sondern von Owner/Member-Elementen.

- relational:

Den relationalen Datenbanken (typische Vertreter RDBMS der Fa. Oracle, DB2 der Fa. IBM, SQL-Server der Fa. Microsoft) liegt als Modell die mathematisch fundierte Relationen-Algebra zugrunde. Eine Relation (kann bei Verzicht auf den mathematischen Überbau) als Synonym für eine Tabelle mit einer bestimmten Anzahl (benannter) Spalten und einer Vielzahl von Zeilen angesehen werden. Alle Daten sind in solcher Tabellenform darstellbar und der Zugriff erfolgt durch drei Grundfunktionen Projektion (Auswahl von Spalten), Selektion (Auswahl von Zeilen) und Verbund (Verknüpfung von mehreren Tabellen).

- objektorientiert:

Das Objektorientierte Modell ergänzt ohne wesentliche Brüche die objektorientierten Programmiersprachen, d.h. eine objektorientierte Datenbank erlaubt die persistente Speicherung von Objekten, also von Instanzen diversen Klassen einer Anwendung über die Laufzeit der Anwendung hinaus. Die Zugriffe auf die Objekte sind im Modell der objektorientierten Anwendung festgelegt, d.h. der Zugriff auf die persistenten Daten erfolgt genau so, wie der Zugriff auf die Objekte in der Anwendung realisiert ist.

Es existiert eine Vielzahl von Misch- und Nebenformen, wie zum Beispiel das

- objektrelationale Modell.

  Das Objektrelationale Datenbankmodell ist eine Erweiterung des relationalen Modells um Ideen aus der objektorientierten Programmierung. Es ist also eine Mischform, die beide Modelle unterstützt und so das bewährte relationale Modell bereithält, ohne auf die Möglichkeiten des objektorientierten Modells insbesondere im Zusammenhang mit objektorientierten Anwendungen zu verzichten.

Die heute bekannten und auch kommerziell verfügbaren DBMS werden nach dem zugrundeliegenden Modell für die Speicherung der Daten und den Zugriff auf die Daten klassifiziert:

5. Enduser Datenbank MS Access (MS Office 2010)

5.1 Einführung

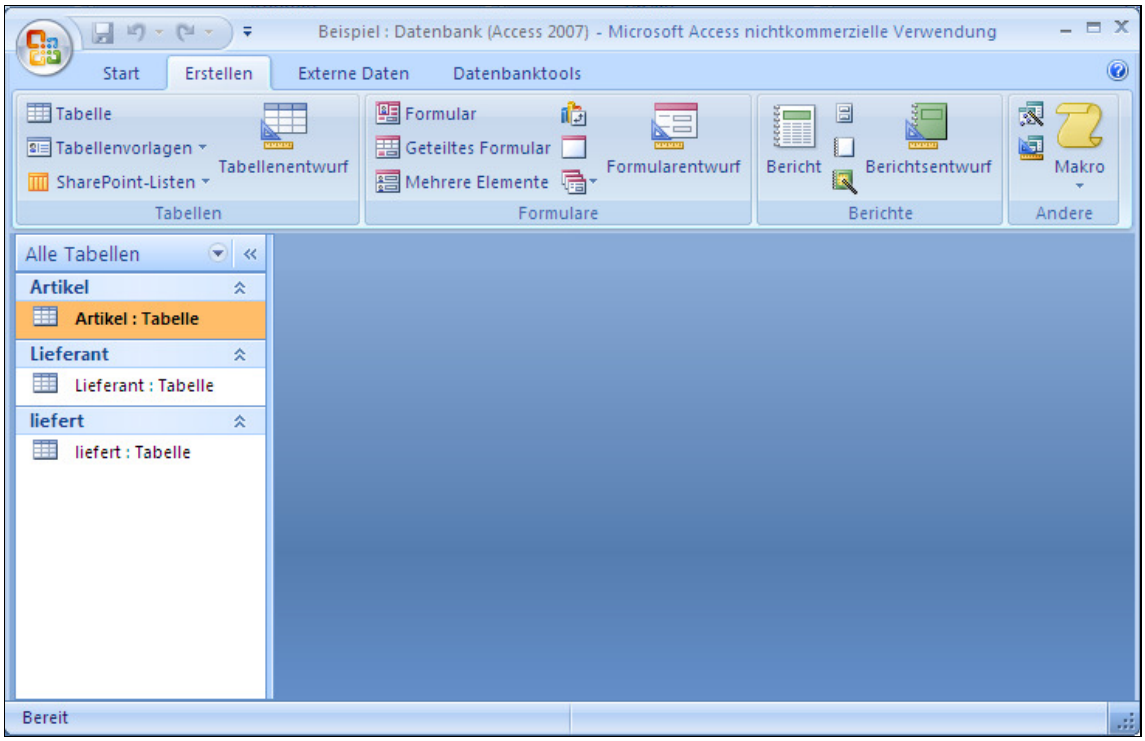
Eine Microsoft Access-Datenbank kann sechs Arten von Datenbankobjekten enthalten:

- **Tabellen** speichern Daten
- **Abfragen** sammeln die angeforderten Daten aus einer oder mehreren Tabellen. Man kann die Daten in einem Formular ansehen oder bearbeiten oder in einem Bericht ausdrucken.
- **Formulare** zeigen Daten aus Tabellen oder Abfragen an, damit man Daten ansehen, bearbeiten oder eingeben kann.
- In **Berichten** werden Daten aus Tabellen und Abfragen zusammengefasst und dargestellt, damit man sie drucken oder analysieren kann.
- **Makros** automatisieren eine Datenbank, indem sie, nach entsprechender Programmierung die einmal vom Entwickler / Benutzer festgelegten Aktionen durchführen.
- **Module** fassen Makros zusammen, um eine Datenbank zu optimieren, zu erweitern und individuell anzupassen.

Eine Microsoft Access Datenbank enthält neben der Datenbank und dem Datenbank Managementsystem im engeren Sinne (Back End) auch noch Datenbank-Anwendungen (Front End) mit graphischer Benutzeroberfläche und Anwendungslogik.

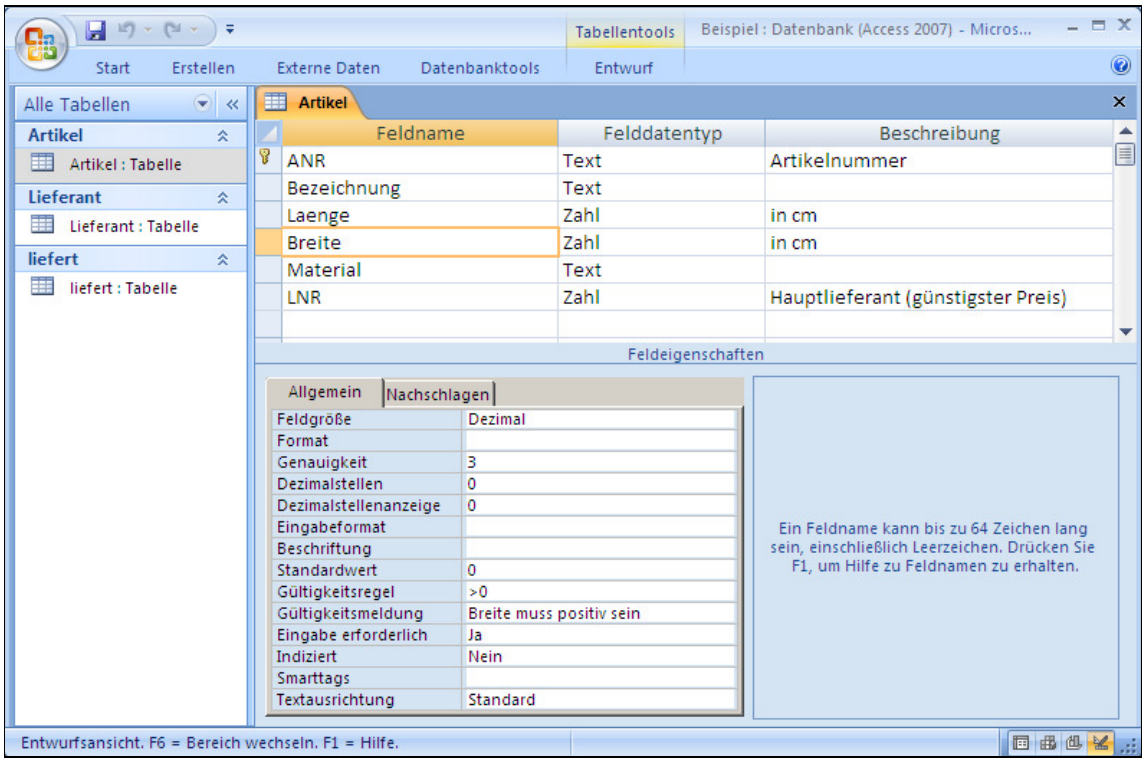
Das System bietet also neben der Datenbank (Tabelle und Abfrage) selbst, Entwicklungswerkzeuge für graphische Benutzeroberflächen (Formular), Datenbank-Auswertungen (Bericht), Anwendungslogik (Makro) und zur Strukturierung (Modul) der Anwendungen.

Gegenstand der Lehrveranstaltung wird nur der reine Datenbankteil (Tabelle und Abfrage) sein. Die Bearbeitung ist dabei eher dialogorientiert, das Anlegen, Ändern, Löschen und Anzeigen von Tabellen und Abfragen wird durch entsprechende Anwendungen mit graphischer Benutzeroberfläche unterstützt.

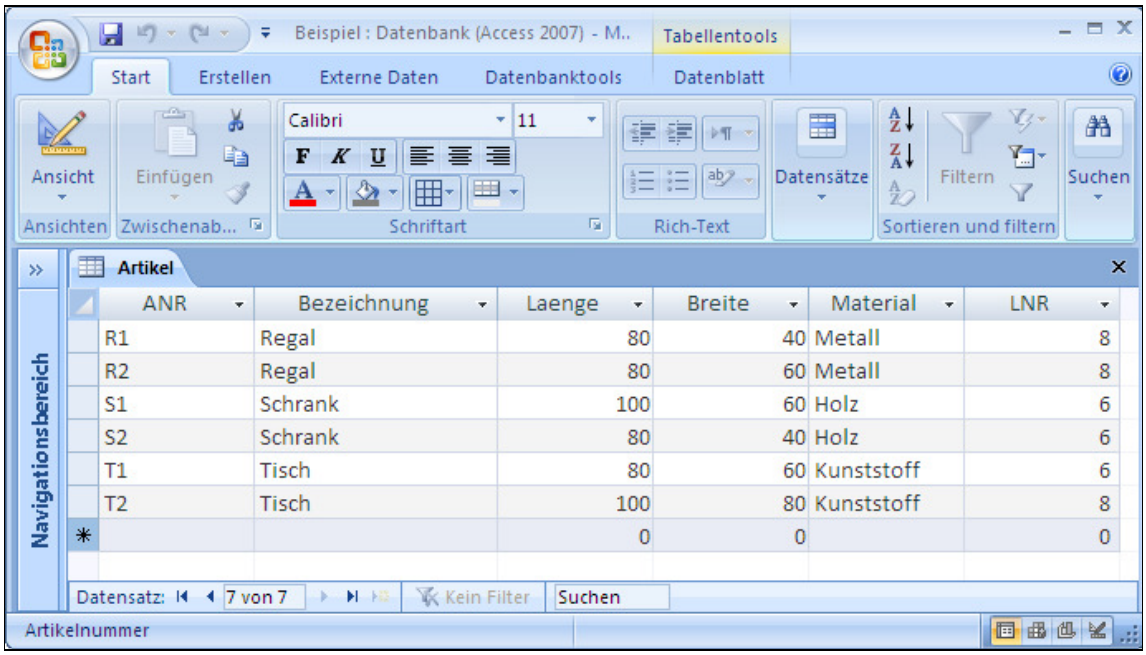


Die Übersicht listet alle vorhandenen Tabellen auf und bietet Möglichkeiten für die Neuanlage von Tabellen. Der Zugriff auf die Metadaten erfolgt hier über die Funktion **Tabellenentwurf**, der Zugriff auf den Tabelleninhalt über die Funktion **Öffnen**.

Die Funktionen Tabellenentwurf und Öffnen stehen auch im Kontextmenü einer bereits vorhandenen Tabelle zur Verfügung:



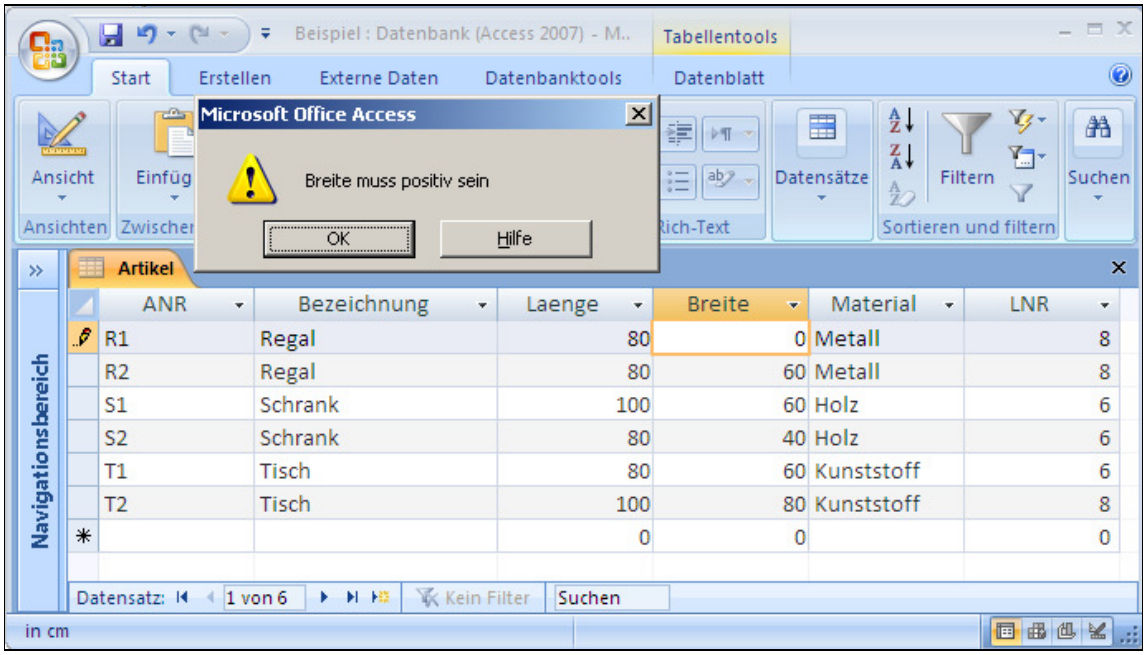
Entwurf: Struktur-Informationen (Spalten, Datentypen, Bedingungen) für die Tabelle Artikel



Öffnen: Inhalt der Tabelle Artikel

In beiden Fällen sind zusätzliche leere Zeilen vorhanden, um neue Informationen hinzufügen zu können. Beim Entwurf beziehen sich die in der unteren Hälfte angezeigten Feldeigenschaften (Feld ist hier die Bezeichnung für Spalte bzw. Attribut) stets auf das in der oberen Hälfte markierte Feld (im Beispiel Breite). In der Datenblattansicht dient die letzte mit * gekennzeichnete Zeile zum Einfügen neuer Zeilen.

Bei der Eingabe von allgemeinen Gültigkeitsregel kann neben der Bedingung auch der Meldungstext (Gültigkeitsmeldung) angegeben werden. Wird gegen eine Gültigkeitsregel verstoßen, wird der zugehörige Meldungstext ausgegeben.



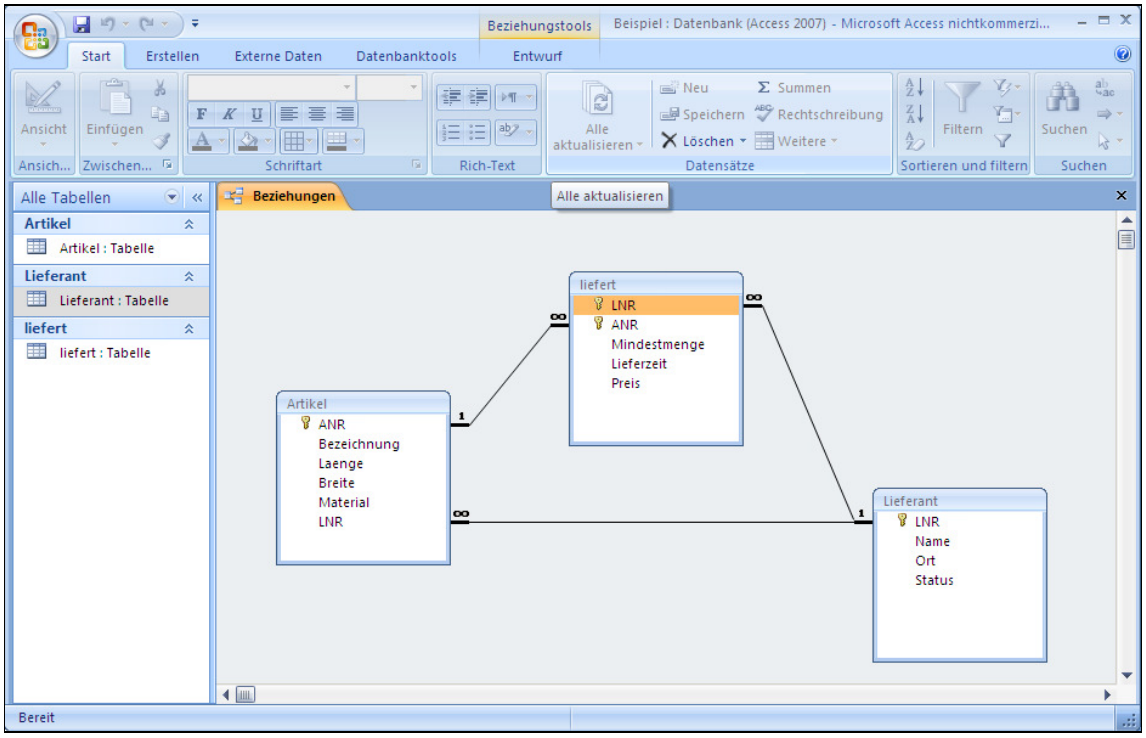
Eine Eingabe der Tabellenstruktur als SQL-Anweisungen ist zwar auch möglich, aber nur recht umständlich und im Grunde hier nicht vorgesehen. Es gibt allgemeine unabhängige Schnittstellen und Anwendungen, die diese Funktionalitäten außerhalb von MS Access bereitstellen und auf die im Kapitel 6. näher eingegangen wird.

Das Erstellen und Verwalten von Fremdschlüsseln geschieht in MS Access nicht im Kontext der betreffenden Tabelle, sondern innerhalb eines eigenen Anwendungsteils, in dem die Beziehungen zwischen den Tabellen verwaltet werden können.

5.2 Beziehungen (Fremdschlüssel) und referentielle Integrität

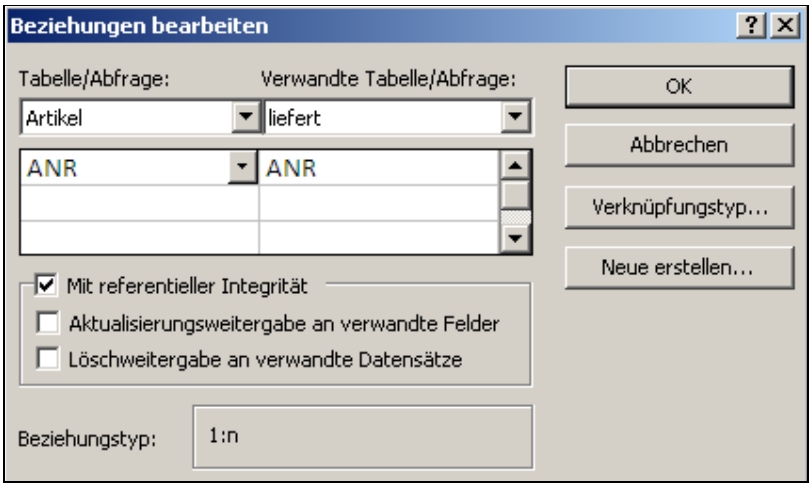
Nachdem die Tabellen erstellt wurden, aus denen sich die Datenbank zusammensetzen soll, müssen die Fremdschlüssel als Beziehungen zwischen den Tabellen definiert werden. Dadurch wird Microsoft Access mitgeteilt, in welcher Beziehung die Daten der einzelnen Tabellen zueinander stehen sollen. Dies erleichtert wiederum das Erstellen von Abfragen, Formularen und Berichten, die mehrere Tabellen umfassen.

Dialog zur Definition von Beziehungen:



Dialog zur Bearbeitung der Eigenschaften einer Beziehung (Referentielle Integrität):

Die Überwachung der referentiellen Integrität kann hier für eine Fremdschlüsselbeziehung explizit eingestellt werden. Darüber hinaus kann die referentielle Integrität auch alternativ dadurch sichergestellt werden, dass entsprechende Änderungen des Schlüsselwertes bzw. das löschen von Zeilen in der referenzierten Tabelle entsprechend weitergegeben wird.

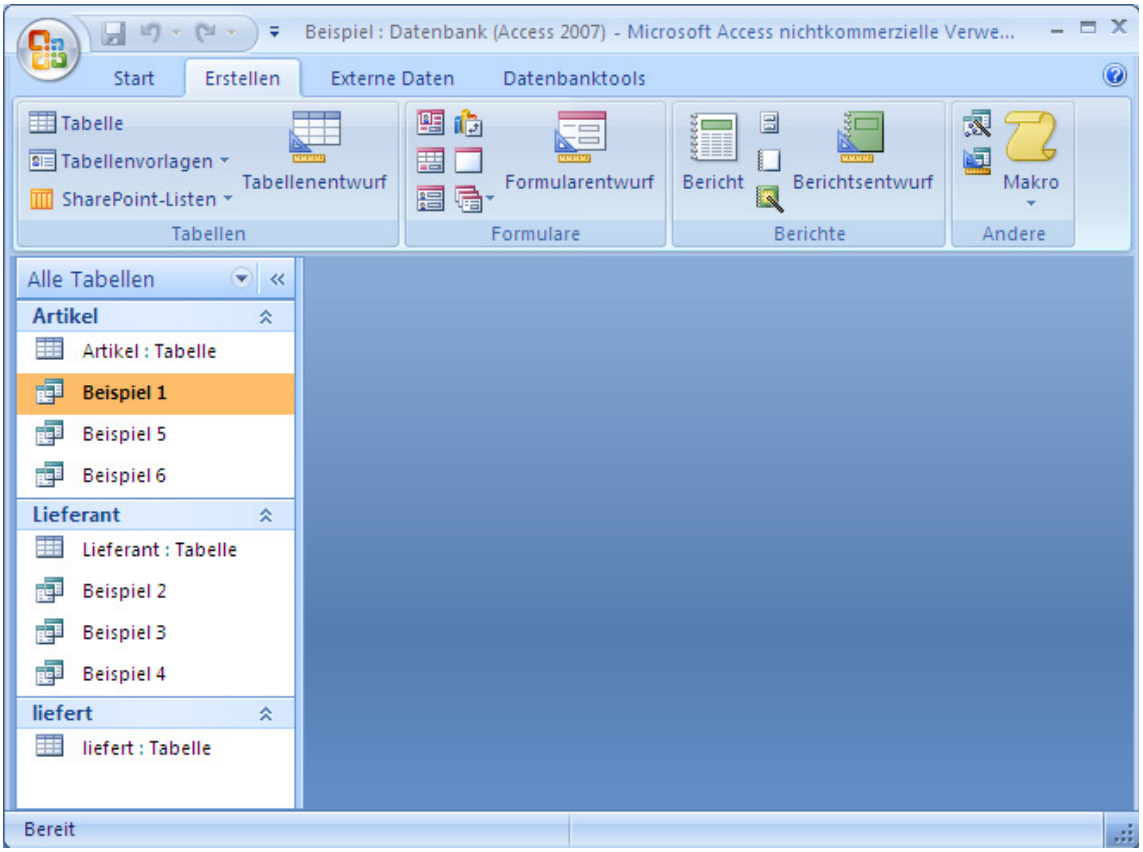


Anmerkungen

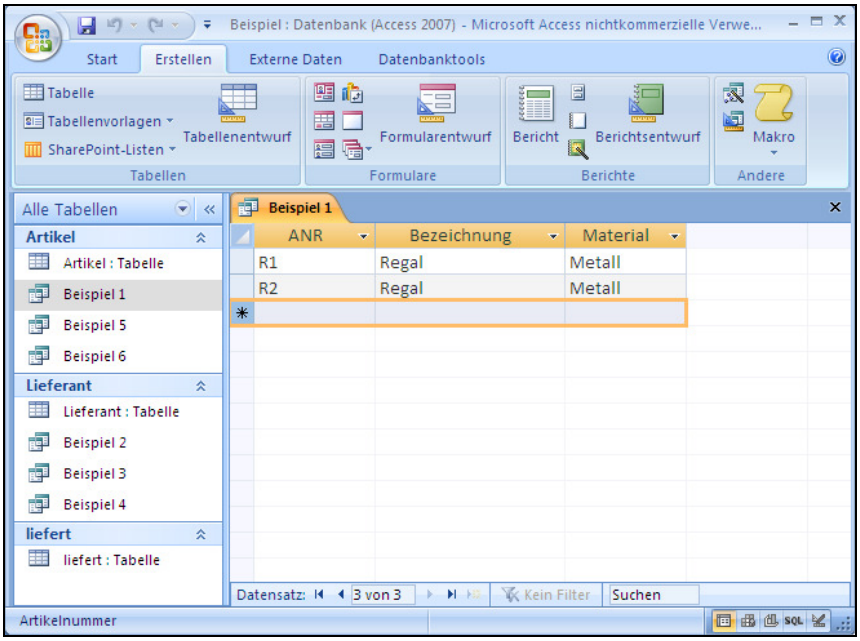
- Sie können Beziehungen erstellen, indem Sie Abfragen wie auch Tabellen verwenden. referentielle Integrität wird jedoch bei Abfragen nicht durchgesetzt.
- Wenn Sie das Fenster Beziehungen schließen, wird Microsoft Access Sie veranlassen, das Layout zu speichern. Ob Sie das Layout nun speichern oder nicht speichern, die Beziehungen, die Sie erstellten, werden in der Datenbank gespeichert.
- Die Felder, mit denen Sie zwischen zwei Tabellen eine Beziehung herstellen, brauchen nicht denselben Namen zu tragen. Die Felder müssen jedoch denselben Felddatentyp aufweisen (von einer Ausnahme abgesehen) und dieselbe Art von Informationen enthalten. Sind die übereinstimmenden Felder vom Datentyp "Zahl", müssen Sie darüber hinaus dieselbe Einstellung der Eigenschaft "Feldgröße" aufweisen. Die oben angesprochene Ausnahme besteht darin, dass Sie ein Feld vom Datentyp "Zähler" mit einem Feld vom Datentyp "Zahl" vergleichen können, dessen Eigenschaft "Feldgröße" auf "Long Integer" eingestellt ist.
- In einer Beziehung, in welcher referentielle Integrität durchgesetzt wird, können Sie weder die beiden Tabellen noch deren Felder, mit denen die Beziehung erstellt wurde, löschen, so lange nicht bis die Beziehung gelöscht wird.
- In einer Beziehung, in welcher referentielle Integrität durchgesetzt wird, können Sie im Fremdschlüsselfeld der Tabelle, mit der eine Beziehung besteht, nicht einen Wert eingeben, der nicht im Primärschlüssel der Mastertabelle besteht. Sie können jedoch im Fremdschlüssel einen Nullwert eingeben und so angeben, dass kein Bezug zwischen den Datensätzen besteht. Sie können z.B. nicht eine Bestellung haben, die einem Kunden zugeordnet ist, der nicht besteht. Sie können aber eine Bestellung haben, die niemandem zugeordnet ist, indem Sie eine "0" im Feld "Kunden-Code" eingeben.
- Sie können eine Beziehung mit einer angefügten Tabelle definieren. Microsoft Access wird jedoch zwischen den beiden Tabellen die referentielle Integrität nicht durchsetzen, es sei denn, beide Tabellen sind in derselben Datenbank und der Benutzer hat die Berechtigungen, die Beziehung in dieser bestimmten Datenbank herzustellen.
- Um eine zweite Beziehung zwischen zwei Tabellen herzustellen, fügen Sie eine der Tabellen zweimal hinzu.

5.3 Abfragen

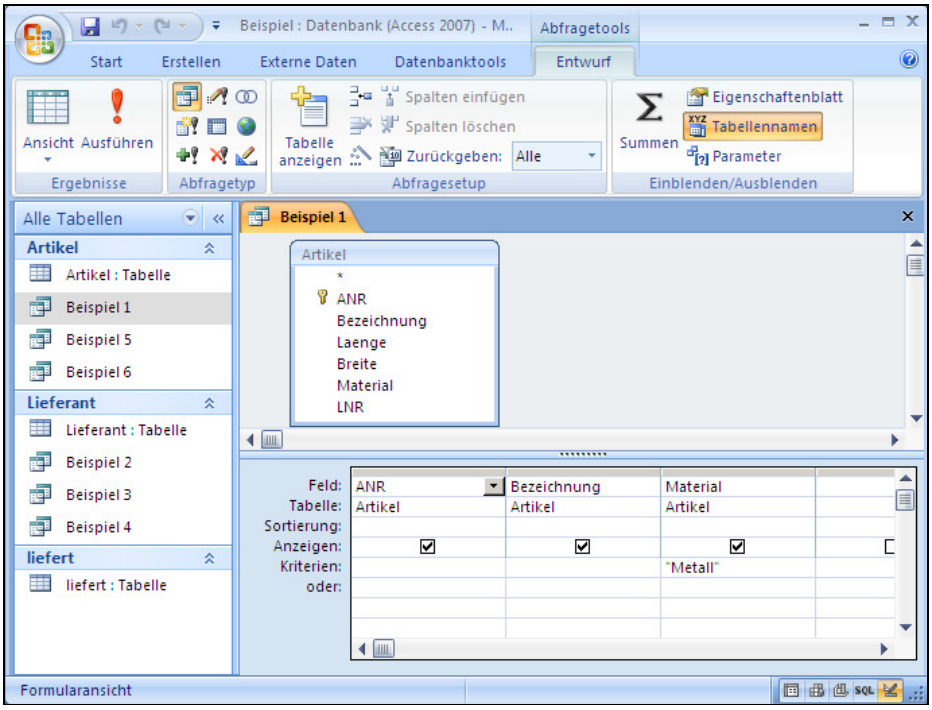
Die Übersicht der vorhandenen Tabellen wird ergänzt um die Anzeige der mit der Tabelle verbundenen Ansicht:



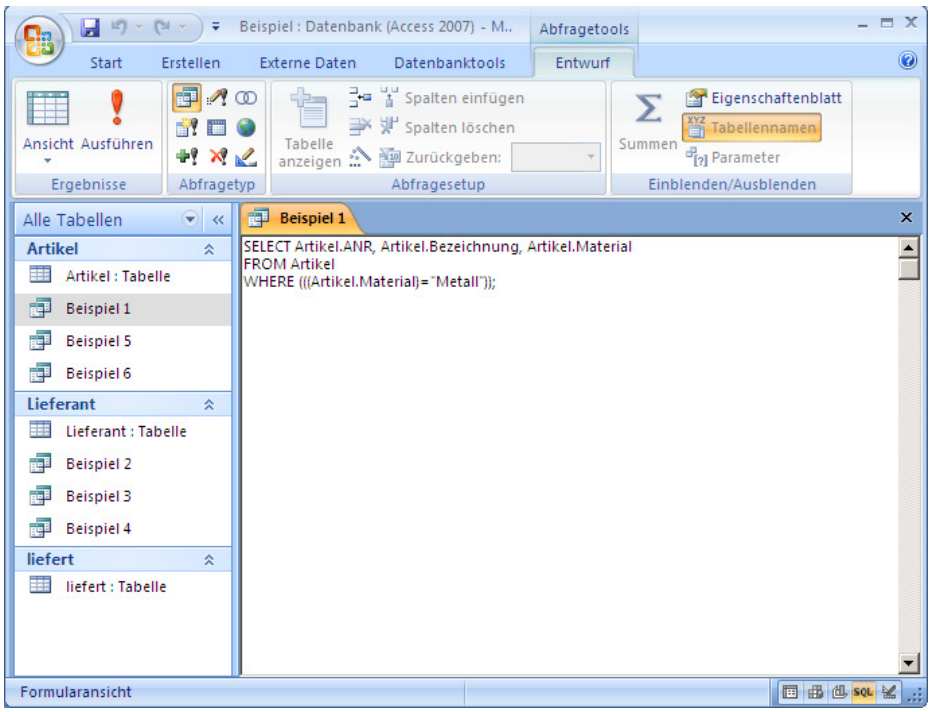
Analog zu den Tabellen, kann z.B. über das Kontext-Menü auch zu eienr Ansicht die Entwurfs- oder die Datenblatt-Ansicht angezeigt werden. Die Funktion Öffnen (Datenblatt-Ansicht) führt dabei die markierte Abfrage aus und zeigt das Ergebnis als Tabelle an:



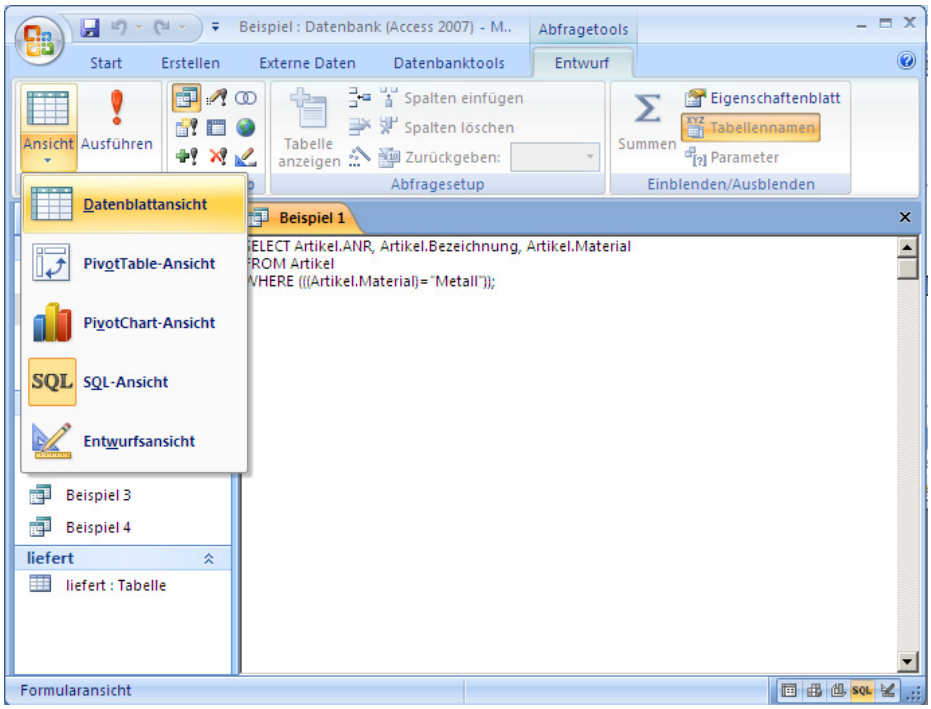
Die Funktion Entwurf startet einen dialogorientierten Anwendungsteil mit graphischer Benutzeroberfläche zur Definition der Abfrage durch Festlegung der Standardoperationen (Projektion, Selektion und Verbund).



Neben dieser sogenannten Entwurfsansicht gibt es noch eine alternative Möglichkeit, Abfragen direkt in SQL zu definieren (SQL-Ansicht).



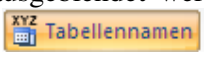
Zwischen diesen drei Ansichten (Entwurfsansicht, Datenblattansicht und SQL-Ansicht) kann über entsprechende Symbole in der Symbolleiste gewählt werden:



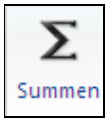
Auswahl der Ansicht in der Datenblattansicht:

SQL-Ansicht für eine Abfrage (die SQL Syntax weicht hier an einigen Stellen von der bisher vorgestellten Syntax ab)

Über das Symbol 

können die Namen der Tabellen als zusätzliche Auswahl in den angezeigten Informationen ein- und ausgeblendet werden. Sind die Tabellennamen eingeblendet, ist das Symbol farblich unterlegt 

.

Über das Symbol 

können die Aggregat- und Gruppierungsfunktionen als zusätzliche Auswahl in den angezeigten Informationen ein- und ausgeblendet werden.

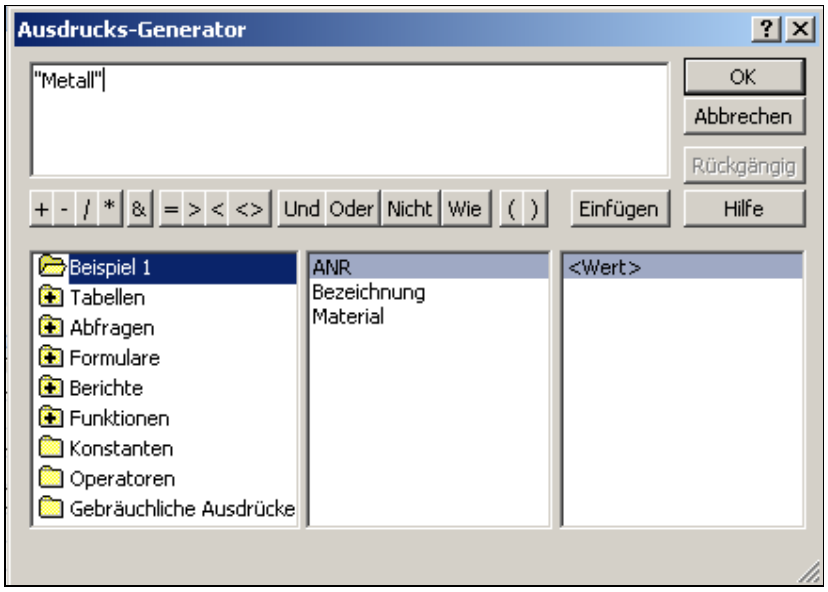
Ist die Funktionszeile eingeblendet, ist das Symbol farblich unterlegt 

.

Die Auswahl 

öffnet kontextabhängig einen Editor, mit dem eine detailliertere Eingabemöglichkeit angeboten wird. so öffnet z.B. die Taste 

einen Ausdruckseditor, wenn der Cursor in einer Kriterien-Zeile (z.B. "Metall") steht. Jetzt bestehen hier wesentlich detaillierte Möglichkeiten, das Auswahlkriterium zu editieren.



Die DDL- und DML-Elemente sind beim MS Access ein wenig verborgen. Man findet sie teilweise als eine besondere Art der Abfrage. Eine Abfrage im gewöhnlichen Sinn wird bei MS Access als Auswahlabfrage bezeichnet.

Tabellenerstellungsabfrage:	CREATE TABLE ....
Datendefinition:	CREATE TABLE.....
Anfügeabfrage:	INSERT INTO .....
Löschabfrage	DELETE FROM ....
Aktualisierungsabfrage	UPDATE .....
Union	Vereinigungsmenge von Sichten

MS Access bietet neben dem eigentlichen Datenbanksystem noch zusätzliche Entwicklungswerkzeuge bzw. Datenbankfunktionen als graphische Anwendungsprogramme, die für das Oracle bzw. MySQL-Datenbanksystem zusätzlich eingerichtet werden müssen.

Für das „experimentelle“ Entwickeln von externen Sichten erscheint der graphische Editor (Entwurfsansicht) besser geeignet als ein Arbeiten direkt mit SQL in einem einfachen Texteditor, denn die Tabellen- und Spaltennamen werden zur Auswahl angeboten, die formalen Syntaxregeln spielen dabei nur eine untergeordnete Rolle, alle in WHERE- und GROUP BY-Klauseln verwendeten Ausdrücke können ohne großen Aufwand zu Kontrollzwecken angezeigt werden. Für eine einzelnen Abfrage und ihre Spalten können in eigenen Dialogen Eigenschaften bequem festgelegt bzw. verändert werden.

Liegt eine Abfrage bereits in SQL vor, kann sie in die SQL-Ansicht übernommen werden, danach ist ein Wechsel zur Entwurfsansicht aber nur noch bedingt möglich.

6. SQL-Erweiterung: PL/SQL (Oracle RDBMS)

Mit der Erweiterung PL/SQL stehen einerseits SQL-Befehle für die Manipulation der Daten in Oracle Datenbanken und andererseits Kontrollfluß-Befehle zur Verarbeitung dieser Daten zur Verfügung. Darüber hinaus können Konstante und Variable vereinbart, Unterprogramme (Funktionen und Prozeduren) definiert und Laufzeitfehler in Behandlungsroutinen abgefangen werden. Insofern werden mit PL/SQL die Möglichkeiten und Fähigkeiten von SQL zur Datenmanipulation mit der Ausdrucksmächtigkeit einer prozeduralen Sprache kombiniert.

6.1 Block Struktur

PL/SQL ist eine blockstrukturierte Sprache, d.h. die Basiseinheiten (Prozeduren, Funktionen und unbenannte Blöcke), aus denen ein PL/SQL-Programm aufgebaut ist, sind logische Blöcke, die selber wieder eine beliebige Anzahl von geschachtelten Teilblöcken enthalten können. Typischerweise korrespondiert jeder logische Block mit einem Problem oder Teilproblem, das er löst. PL/SQL unterstützt also das Prinzip der Schrittweisen Verfeinerung. Ein Block (oder Teilblock) fasst eine Gruppe logisch zusammengehörender Vereinbarungen und Befehle zusammen. Dadurch können Vereinbarungen dort getroffen werden, wo sie gebraucht werden. Vereinbarungen gelten nur lokal innerhalb des Block (und allen tiefer geschachtelten Teilblöcken) und sind außerhalb des Blocks (d.h. in übergeordneten Blöcken) unbekannt.

```
DECLARE

   ..... Vereinbarungen (declarations)

BEGIN

   ..... Befehle (statements)

EXCEPTION

   ..... Behandlungsroutinen (Handlers)

END
```

Ein PL/SQL-Block besteht aus drei Teilen: einem Vereinbarungsteil DECLARE.....BEGIN, einem ausführbaren Teil BEGIN.....EXCEPTION oder BEGIN.....END und aus einem Teil zur Ausnahmebehandlung EXCEPTION.....END (in PL/SQL heißen Warnungen und Fehlermeldungen Ausnahmen). Nur der ausführbare Teil ist zwingend notwendig. Die anderen beiden Teile können in einem Block auch fehlen. Die angegebene Reihenfolge erscheint logisch: zunächst werden Variablen und Konstanten vereinbart, bevor sie im ausführbaren Teil manipuliert werden. Die dabei auftretenden Ausnahmebedingungen werden im dritten Teil gesondert behandelt.
Aus einer Behandlungsroutine heraus kann nicht wieder in den regulären Ausführungsteil desselben Blocks zurückgekehrt werden, der Aufruf einer Behandlungsroutine führt immer zu Blockende und zum Rücksprung in den übergeordneten Block.

6.2 Variablen und Konstante

Vereinbarte Variablen und Konstanten können in SQL-Befehlen und prozeduralen Befehlen überall dort verwendet werden, wo Ausdrücke zugelassen sind. Jedoch sind keine Referenzen auf erst später vereinbarte Variablen oder Konstanten möglich, d.h. eine Variable oder Konstante muss erst vereinbart werden, bevor sie referenziert werden kann.

Variablen können mit einem SQL-Datentyp wie CHAR, DATE, NUMBER oder mit weiteren zusätzlichen PL-Datentypen wie BOOLEAN, BINARY_INTEGER usw. vereinbart werden. Darüber hinaus können auch Sätze (Records) und Arrays (Tables) mit den zusammengesetzten Datentypen RECORD und TABLE gebildet werden. Mit der %TYPE Angabe im Verbund mit einer Variablen bzw. einer Tabellenspalte an Stelle eines expliziten Datentyps kann auf den Datentyp eine bereits definierten Variablen oder einer Tabellenspalte zurückgegriffen werden. Mit der Angabe %ROWTYPE kann analog eine Satzstruktur vereinbart werden, die sich aus den Datentypen der Spalten einer Tabelle zusammensetzt, also z.B.

```
Teile_Nummer NUMBER(4);
verfügbar    BOOLEAN;
z_Name       VARCHAR2(30);
z_Breite     Artikel.Breite%TYPE;

z_Daten      Artikel%ROWTYPE;
```

Wertzuweisungen können durch den Zuweisungsoperator (:=) erfolgen. Für lokal vereinbarte Sätze und für die Zeilen einer Tabelle kann auf einzelne Elemente mit der Punkt-Notation zugegriffen werden. also z.B.

```
Steuer          := Netto * Steuersatz;
Bonus           := aktuelles_Gehalt * 0.10;
Wert            := TO_NUMBER(SUBSTR('750 raise', 1 , 3));
gefunden        := FALSE;

z_Daten.Breite  := z_Breite / 100;
```

Eine weitere Art der Wertzuweisung an eine Variable ist die Zuweisung eines Wertes aus einer Datenbanktabelle durch einen Select-Befehl, z.B.

```
SELECT Breite * 0.01 INTO z_Breite FROM Artikel WHERE Anr = 'R1';
SELECT * INTO z_Daten FROM Artikel WHERE Anr='T1';
```

Dabei kann jedoch immer nur auf eine einzelne Zeile in der Tabelle Bezug genommen werden, d.h. der SELECT-Befehl muss genau eine Zeile liefern, sonst kommt es zu einer Ausnahme (Fehler-Situation).

Eine Konstante wird ähnlich einer Variablen vereinbart, jedoch muss hier das Schlüsselwort CONSTANT verwendet werden und noch bei der Vereinbarung muss eine Wertzuweisung erfolgen. Weitere Wertzuweisungen sind nicht zulässig, z.B.

```
Minimum_status  CONSTANT  NUMBER := 10;
```

6.3 Cursors

Oracle benutzt Arbeitsbereiche, sogenannte private SQL-Bereiche, um SQL Befehle auszuführen und dort Ergebnisinformationen abzulegen. Ein PL/SQL Sprachkonstrukt, ein sogenannter Cursor erlaubt es, diese privaten SQL-Bereiche zu benennen und auf die dort gespeicherten Informationen zu zugreifen. Ein Cursor wird benutzt, um auf die einzelnen Zeilen, die eine Select-Anweisung in Form einer Tabelle als Ergebnis liefert, individuell, d.h. einzeln nacheinander zugreifen zu können.

```
DECLARE

        CURSOR c1  IS
        SELECT  Lieferzeit, Mindestmenge, Preis
        FROM    liefert, Artikel
        WHERE   Artikel.ANR =  liefert.ANR
            AND  Material       = 'Kunststoff';                 .....
```

Die Menge der Zeilen, die bei einer solchen Abfrage als Ergebnis geliefert werden, heißt die aktive Menge. Diese aktive Menge umfasst alle die Zeilen, die die angegebene Bedingung in der WHERE-Klausel erfüllen. Mit Hilfe des Cursors können die einzelnen Zeilen einer aktiven Menge getrennt und eine nach der anderen bearbeitet werden. Mit dem Open -Befehl wird die entsprechende Abfrage ausgeführt, der Fetch-Befehl stellt nacheinander jede einzelne Zeile der aktiven Menge zur Bearbeitung zur Verfügung und der Close-Befehl schließt die Bearbeitung ab.

```
DECLARE

        L_ZEIT          liefert.Lieferzeit%TYPE;
        M_MENGE         liefert.Mindestmenge%TYPE;
        A_PREIS         liefert.Preis%TYPE;

        CURSOR c1  IS
        SELECT  Lieferzeit, Mindestmenge, Preis
        FROM    liefert, Artikel
        WHERE   Artikel.ANR =  liefert.ANR
            AND  Material       = 'Kunststoff';

BEGIN
        OPEN c1;

        FETCH c1  INTO  L_ZEIT, M_MENGE, A_PREIS;
        .............
        .............
        FETCH c1  INTO  L_ZEIT, M_MENGE, A_PREIS;
        .............
        .............

        CLOSE c1;
END;
```

Auf die Spalten, die bei einem Cursor definiert wurden, kann als Struktur direkt über cursor%ROWTYPE zu gegriffen werden, d.h. für

```
CURSOR c1  IS
SELECT  Lieferzeit, Mindestmenge, Preis
FROM     liefert, Artikel
WHERE   Artikel.ANR =  liefert.ANR
    AND  Material        = 'Kunststoff'';
```

definiert

```
z_Werte     c1%ROWTYPE;
```

eine Datenstruktur mit den Feldern Lieferzeit, Mindestmenge und Preis, die danach in

```
FETCH c1  INTO  z_Werte;
```

benutzt werden kann. Auf die einzelnen Felder kann dann durch Punktnotation zugegriffen werden, also z.B.

```
z_Werte.Preis
```

Darüberhinaus gibt es noch eine spezielle Schleife für CURSOR, z.B.

```
FOR   zeile   IN   c1   LOOP
```

bei der die Datenstruktur implizit vereinbart wird, d.h. es darf hier keine explizite Deklaration der Variablen **zeile** geben..
Weiter wird **OPEN c1** bzw. **CLOSE c1** automatisch zu Beginn bzw. am Ende der FOR-Schleife durchgeführt und bei jedem Schleifendurchlauf wird automatisch ein **FETCH c1 INTO zeile** ausgeführt. Die FOR-Schleife wird beendet, wenn dieses automatische FETCH keine weitere Zeile liefert.

6.4 Kontroll-Strukturen

```
IF   Bedingungen
THEN
     Befehle
     .............
ELSE
     Befehle
     .............
END IF;

LOOP
     Befehle
     ..............
END LOOP;

FOR   ind   IN   1..Gesamtzahl        LOOP
     Befehle
     ..............
END LOOP;
```

```
WHILE   Bedingung
LOOP
    Befehle
    ..................
END LOOP;

FOR  Liefer_Bedingungen  IN  c1  LOOP
    Befehle
    ..................
END LOOP;
```

Die folgenden Attribute können zur Formulierung von Bedingungen in IF-Anweisungen bzw. Programmschleifen verwendet werden:

Cursor-Attribute (OPEN, FETCH, CLOSE cursorname)

cursorname %ISOPEN	BOOLEAN
cursorname%FOUND	BOOLEAN
cursorname%NOTFOUND	BOOLEAN
cursorname%ROWCOUNT	NUMBER

SQL-Attribute (INSERT, UPDATE, DELETE)

SQL%FOUND	BOOLEAN
SQL%NOTFOUND	BOOLEAN
SQL%ROWCOUNT	NUMBER

6.5 Ausnahme Behandlung

PL/SQL macht es einfach, vorgegebene und benutzerdefinierte Fehlersituationen, sogenannte Ausnahmen zu erkennen und zu behandeln. Tritt eine solche Fehlersituation auf, wird die normale Programmausführung unterbrochen und die entsprechende Behandlungsroutine für diese Ausnahmebedingung ausgeführt, sofern eine solche im Ausnahmebehandlungsteil des Blocks definiert ist. Nach der Behandlung einer Ausnahme, wird der Block, in dem die Behandlung aufgetreten ist bzw. behandelt wurde, direkt verlassen. Ist in dem Block, in dem die Ausnahme aufgetreten ist, keine Ausnahmebehandlung im EXCEPTION-Teil definiert, wird in den EXCEPTION-Teil des übergeordneten Block verzweigt und dort nach einer entsprechenden Ausnahmebehandlung gesucht. Ist auch dort keine vorhanden, wird die Suche im jeweils übergeordneten Block solange fortgesetzt, bis entweder ein Block erreicht ist, in dem die entsprechende Ausnahmebehandlung im EXCEPTION-Teil definiert ist oder die oberste Ebene erreicht wird. In diesem Fall wird die Ausnahmebehandlung von der Laufzeitumgebung behandelt und damit wird dann natürlich die Bearbeitung des PL/SQL-Blocks beendet.

Fest vorgegebene Ausnahmebedingungen werden vom Laufzeitsystem ausgelöst, z. B. führt eine Division durch den Wert 0 zur Ausnahmebedingung ZERO_DIVIDE.
Benutzerdefinierte Ausnahmenbedingungen werden explizit durch einen entsprechenden

Raise-Befehl ausgelöst:

```
DECLARE
        akt-stat                Lieferant.Status%TYPE;
        Lieferant_unzulaessig   EXCEPTION;      -- Definition der Ausnahme-Bedingung
BEGIN
        SELECT  Status  INTO  akt_stat
        FROM    Lieferant
        WhERE   Lnr = 5;
        IF  akt_stat = 10
        THEN
            RAISE  Lieferant_unzulaessig;   --  Ausnahmebehandlung auslösen
        ELSE
            .................
            Befehle
            .................
        END IF;
        .................
        Befehle
        .................
EXCEPTION  -- Ausnahmebehandlungs-Routinen
        WHEN  Lieferant_unzulaessig  THEN
        .................
            Befehle zur Behandlung des Ausnahme
            .................
END;
```

Ausnahmen, die bei der Ausführung von SQL-Befehlen (SELECT, INSERT, ...) auftreten können (Auswahl):

ORA-0001	DUP_VAL_ON_INDEX
ORA-1403	NO_DATA_FOUND
ORA-1422	TOO_MANY_ROWS
ORA-1476	ZERO_DIVIDE
ORA-1722	INVALID_NUMBER
ORA-6502	VALUE_ERROR
ORA-6504	ROWTYPE_MISMATCH
ORA-1017	NOT_LOGGED_ON
ORA-1001	INVALID_CURSOR

Beispiele für Ausnahmebehandlung (SELECT-Befehl)

```
DECLARE
    Z_LNR       Lieferant.LNR%TYPE := 5;
    Z_Lieferant   Lieferant%ROWTYPE;
BEGIN
    SELECT  *  INTO  Z_Lieferant
    WHERE  LNR = Z_LNR;
EXCEPTION
    WHEN  NO_DATA_FOUND THEN
       INSERT INTO Lieferant  VALUES (Z_LNR, 'Müller', 'Gera', 10);
END;

DECLARE
    Z_LNR     Lieferant.LNR%TYPE := 1;
BEGIN
    LOOP
       BEGIN
          INSERT  INTO  Lieferant  VALUES (Z_LNR, 'Müller', 'Gera', 10);
          EXIT;
       EXCEPTION
       WHEN  DUP_VAL_ON_INDEX
          Z_LNR := Z_LNR + 1;
       END;
    END LOOP;
END;
```

6.6 Gespeicherte Prozeduren und Funktionen

PL/SQL kennt zwei Typen von Unterprogrammen, nämlich Prozeduren und Funktionen. Für beide können formale Parameter/Argumente vereinbart werden, die bei der Ausführung der Unterprogramme (Aufruf der Prozeduren und Funktionen) durch aktuelle Parameterwerte ersetzt werden müssen.
Ein Unterprogramm ist wie ein kleines eigenständiges Programm, das mit einem Programmkopf beginnt und weiter aus einem optionalen Vereinbarungsteil, einem Ausführungsteil und einem optionalen Teil zur Ausnahmebehandlung besteht.

```
PROCEDURE  Prozedurname (Parameterliste)  IS
    ..............
    lokale Vereinbarungen (declarations)
    ..............

BEGIN
    ..........................
    Befehle (statements)
    ..........................

EXCEPTION

WHEN    Ausnahme_Bedigung_1
    THEN
        ...................
        Befehle  (statements)
        ...................

END Prozedurename;
```

Die Parameterliste besteht aus einer Aufzählung der einzelnen Parameter. Für jeden einzelen Parameter gilt die Syntax:

```
symbolischer_Name      [ IN | OUT | IN OUT ]  Datentyp   [ DEFAULT Wert ]
```

dabei ist jedoch zu beachten, dass der Datentyp stets ohne eine Angabe der Länge oder der Stellenzahl erfolgen muss, also einfach CHAR und nicht CHAR(20) oder NUMBER und nicht NUMBER(6).

Der Aufruf einer Prozedur in einem anderen PL/SQL-Block erfolgt durch.

```
Prozedurname (Liste der aktuellen Parameterwerte);
```

Eine Funktion ist ein Unterprogramm, das einen Wert berechnet und an das aufrufende Programm zurückgibt.

```
FUNCTION  Funktionsname (Argumentliste)  RETURN   Datentyp   IS

    .............
    lokale Vereinbarungen (declarations)
    .............

BEGIN

    .....................
    Befehle (statements)
    .....................

EXCEPTION

WHEN    Ausnahme_Bedigung_1
    THEN
        ...................
        Befehle  (statements)
        ..................

END Funktionsname;
```

Im Ausführungsteil muss mindestens einen RETURN-Befehl (RETURN Ausdruck) enthalten sein. Mit dem Returnbefehl wird der zurückzugebende Wert (=Ausdruck) festgelegt und der Funktionsaufruf beendet.

Die Argumentliste besteht aus einer Aufzählung der einzelnen Funktionsargumente. Für jedes einzelne Argument gilt die Syntax:

```
symbolischer_Name       [ IN | OUT | IN OUT ]  Datentyp   [ DEFAULT Wert ]
```

dabei ist jedoch zu beachten, dass der Datentyp stets ohne eine Angabe der Länge oder der Stellenzahl erfolgen muss, also einfach CHAR und nicht CHAR(20) oder NUMBER und nicht NUMBER(6).

Beim Aufruf in einem anderen PL/SQL-Block kann der Name der Funktion in reinen PL-Befehlen wie eine lokal vereinbarte Variable verwendet werden. Eine Verwendung in einem SQL-Befehl (z.B. INSERT) ist dagegen nicht zulässig.

Für PL/SQL gilt, dass Namen zunächst vereinbart werden müssen, bevor auf sie über den Namen Bezug genommen werden kann. Im Vereinbarungsteil kann mit einer speziellen Vereinbarung auf später definierte Prozeduren und Funktionen verwiesen werden. Eine solche “Vorwärtsvereinbarung” lautet einfach:

```
PROCEDURE  Prozedurname  (Parameterliste);
```

Es gibt eine große Anzahl von sogenannte buildin-Funktionen, die öffentlich sind und allen Benutzern so eine große Anzahl von Funktionen bereitstellen.

Liste der verfügbaren Funktionen (Auswahl):

CONCAT(string1,string2) identisch mit Operator ||
LOWER(string), **UPPER**(string)
LTRIM(string1,string2), **RTRIM**(string1,string2)
REPLACE(string1,search_string [,replace_string]))
SUBSTR((string, anfang [,länge])
LENGTH(string)
POWER(x,y)
SQRT(x)
TRUNC(x [,y])
TO_CHAR(datum [,format])
TO_DATE(string [,format])
USER
SYSDATE

6.7 Datenbank-Trigger

Ein Datenbanktrigger ist ein in der Datenbank gespeichertes Unterprogramm, das mit einer Tabelle der Datenbank verbunden ist. Dieses Unterprogramm kann automatisch ausgeführt werden, bevor oder nachdem eine Zeile der Tabelle eingefügt, geändert oder gelöscht wurde. In diesem Fall spricht man davon, dass der Trigger feuert, d.h. ereignisgesteuert ausgelöst wird.

Syntax

```
CREATE [OR REPLACE] TRIGGER    Triggername
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF Spalte [, Spalte] ...]}

[OR {DELETE | INSERT | UPDATE [OF Spalte [, Spalte] ...]}] ...

ON   Tabellenname

[ [REFERENCING { OLD [AS] Name_vorher [NEW [AS] Name_nachher]
        | NEW [AS] Name_nachher [OLD [AS] Name_vorher] } ]

FOR EACH ROW         [WHEN (condition)] ]

   pl/sql-block
```

mit:

OR REPLACE
erstellt einen Trigger unter einem bereits vorhandenen Namen und überschreibt dabei die alte Definition

Triggername
vollständiger Name, gegebenenfalls einschließlich des Schemanamens

BEFORE
gibt an , dass der Trigger feuert, bevor die auslösende Aktion ausgeführt wird

AFTER
gibt an , dass der Trigger feuert, nachdem die auslösende Aktion ausgeführt wurde

DELETE
gibt an , dass der Trigger feuert, wenn ein Löschbefehl ausgeführt wird, d.h. wenn eine Zeile aus der Tabelle entfernt wird.

INSERT
gibt an , dass der Trigger feuert, wenn ein Einfügebefehl ausgeführt wird, d.h. wenn eine Zeile in die Tabelle eingefügt wird.

UPDATE...OF

gibt an , dass der Trigger feuert, wenn ein Änderungsbefehl ausgeführt wird, d.h. wenn der Wert einer Spalte aus der OF-Klausel geändert wird. Wenn die OF-Klausel fehlt, feuert der Trigger, wann immer irgend eine Spalte verändert wird.

ON Tabellenname

gibt den vollständigen Tabellennamen an, gegebenenfalls einschließlich des Schemanamens

Ausnahme: Es ist nicht möglich, Trigger für Tabellen im SYS-Schema zu definieren.

REFERENCING

erlaubt die Vergabe von Namen für die Zeileninhalte vor (OLD) beziehungsweise nach (NEW) der Durchführung der triggerauslösenden Aktion. Innerhalb des PL/SQL-Blocks kann mit den hier vergebenen Namen auf die entsprechenden Zeileninhalte Bezug genommen werden.

FOR EACH ROW

gibt an, dass ein sogenannter Zeilentrigger definiert wird, d.h. der Trigger feuert jeweils für jede Zeile, die von der auslösenden Aktion betroffen ist, sofern die in einer eventuell vorhandenen WHEN-Klausel angegebene Bedingung erfüllt ist.

Ohne diese Angabe spricht man von einem Befehlstrigger, der nur einmal für die triggerauslösende Aktion feuert.

WHEN

enthält eine zusätzliche Bedingung für das Auslösen des Triggers. In der Bedingung müssen die vergebenen Bezugsnamen verwendet werden und es darf keine Select Anweisung benutzt werden.

Eine WHEN-Klausel kann nur für einen Zeilentrigger definiert werden.

pl/sql-block

ist der PL/SQL-Block, der automatisch von ORACLE ausgeführt wird, wenn der Trigger ausgelöst wird.

Der PL/SQL-Block darf keine Transaktion-Kontroll-Befehle (COMMIT, ROLLBACK, SAVEPOINT) enthalten.

7.1 ODBC

7.1 Grundlagen

In der heutigen Zeit gibt es eine ganze Reihe verschiedener Standards bei Datenbanken, die alle eine unterschiedliche (herstellerspezifische) Ansteuerung/Kommunikation voraussetzen. Damit es möglich wird, aus einem Anwendungsprogramm auf eine Datenbank zuzugreifen, muss i. d. R. ein **spezielles Modul** programmiert und in das Anwendungsprogramm integriert werden, das die Eingaben des Anwenders in das erforderliche Format umwandelt. Ein solches Modul hat allerdings den Nachteil, dass die Lösung auf das spezielle DB-System fixiert ist.
Bei ODBC hingegen wird ein anderer Weg beschritten. Es wird vom jeweiligen Anwendungsprogramm nicht direkt auf die entsprechende Datenbank zugegriffen, sondern auf die ODBC-Schnittstelle. Es braucht kein spezielles Modul programmiert werden. Die Komponenten von ODBC wandeln die Abfragen in das erforderliche Format um.

Definition: ODBC

ODBC (open database connectivity) ist eine von Microsoft stammende Zwischenschicht (Middleware) zur Vereinheitlichung von verschiedenen Datenbankformaten. Sie hat sich mittlerweile Plattform übergreifend als Standard durchgesetzt, der von allen wichtigen Datenbankherstellern unterstützt wird.
ODBC ist ein offenes Anwendungsprogramm als Schnittstelle zum Zugang zu Datenbanken. Verwendet man ODBC, bekommt man Zugang zu einer Vielzahl von verschiedenen Datenbanken und Anwendungen der Firma Microsoft, wie z.B. Access, dBase, Excel und Textdatenbanken, aber auch zu Datenbanken und Anwendungen vieler anderer Anbieter, wie Lotus Notes, Oracle, Sybase, DB2. ODBC basiert und ist ausgerichtet auf SQL (Standard Query Language).

Vorbereitungen

Für den Datenaustausch mittels ODBC sind spezielle Treiber erforderlich, die auf dem System, auf dem der Zugriff auf das Fremdsystem erfolgen soll, zu installieren sind. Es wird jeweils der Treiber der Datenbank oder Anwendung benötigt, auf die zugegriffen werden soll. Die Installation hängt stark von den (herstellerspezifischen) Treibern ab, deshalb sei hier auf die vom Hersteller mitgelieferten Treiberdokumentationen verwiesen. Die Installation der Treiber und die Zuordnung zu den Datenquellen und deren Konfiguration wird später beschrieben

Komponenten

Eine ODBC-Schnittstelle besteht aus folgenden Teilen:

- Der ODBC-Treiber nimmt die SQL-Anweisungen aus dem Anwendungsprogramm entgegen, leitet sie an die jeweilige Datenbank weiter, nimmt von ihr die Ergebnisse entgegen und gibt sie an die Anwendung zurück.
- Der Treiber-Manager dient der betriebssystemseitigen Verwaltung der auf einem System verfügbaren Treiber. ODBC-Treiber werden im System installiert und nicht für jede Anwendung separat. Aus diesem Grund wird eine Managementkomponente (also der Treiber-Manager) benötigt, die für die Verwaltung des Zugriffs von Anwendungen auf die zugeordneten Treiber verantwortlich ist.
- Die Datenquelle (Data Source Name, DSN) ist das Schnittstellendokumente, in dem der Treiber und die Treiber spezifischen Daten gespeichert sind. Eine Datenquellen wird von Anwendungsprogrammen für den Zugriff über ODBC benutzt.

Unter ODBC bekommt jede Datenquelle einen systemweit gültigen Namen. Die Anwendungen sprechen die Datenquellen dann über diesen Namen an.

Der ODBC-Treiber-Manager unterscheidet drei Arten von Datenquellen:

- In einer Benutzer-Datenquelle werden Informationen gespeichert, wie eine Verbindung zu einer Datenquelle hergestellt wird. Benutzer-Datenquellen sind nur für einen Benutzer sichtbar und können nur auf dem aktuellen Computer verwendet werden.
- Auf eine System-Datenquelle können alle Benutzer eines Computers und die Dienste zugreifen.
- Mit einer ODBC-Datei-Datenquelle können Verbindungen zu anderen Datenquellen hergestellt werden. Benutzer die gleiche Treiber installiert haben, können gemeinsam auf Datei-Datenquellen zugreifen.

Anwendungen setzen SQL-Anweisungen ein, um an Daten aus Datenquellen zu gelangen. Sie übergeben diese Anweisungen an den jeweiligen Treiber, den sie vom Treiber-Manager zugeordnet bekommen und nehmen von diesem die Ergebnisse der Abfrage entgegen. Die einzelnen Treiber bieten unter Umständen unterschiedliche Funktionalität ab, d.h. es kann vom Treiber nur die Funktionalität angeboten werden, die durch die betreffende Datenbank auch unterstützt wird.

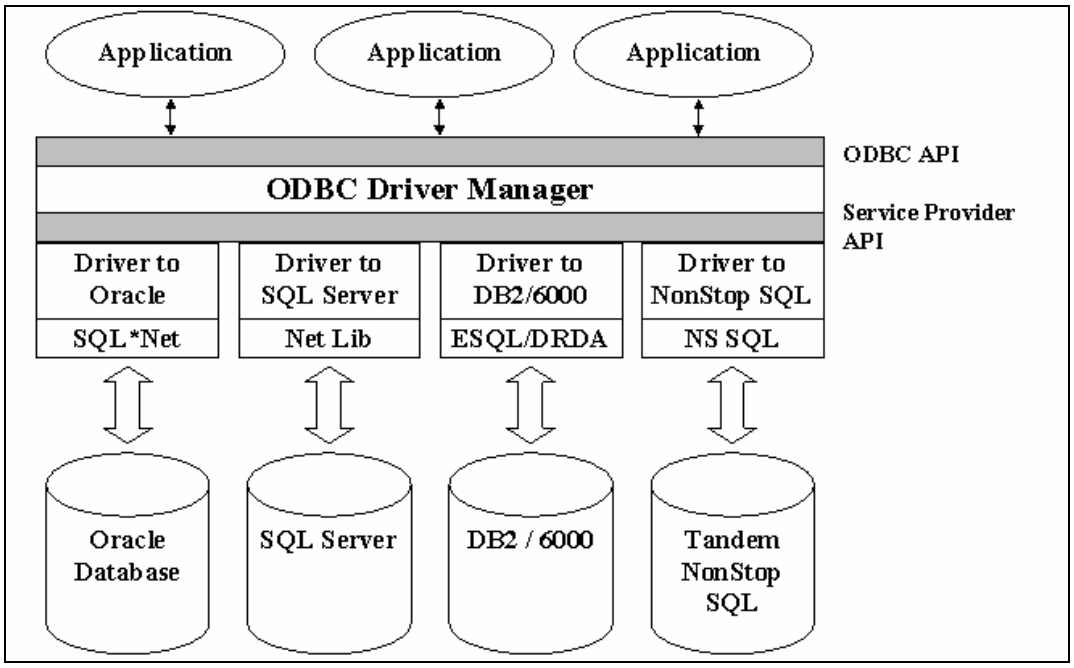
Funktionsweise

Das Zusammenspiel der einzelnen Komponenten läuft dabei folgendermaßen ab:

- Das Anwendungssystem, von dem aus ein Datenbankzugriff mittels ODBC erfolgen soll, greift über die eigene API auf den ODBC Treiber Manager zu.
- Dort ist der zuvor installierte ODBC Treiber und die konfigurierte Datenquelle registriert.
- Mit diesem Treiber wird die Verbindung zu dem Zielsystem aufgebaut und es kann auf dessen Datenbanken zugegriffen werden.
- Je nach Anfrage werden die gewünschten Daten zurückgeliefert.

Je nach Anwendungssystem stehen für den Datenimport verschiedene Möglichkeiten zur Verfügung. So bieten die Microsoft Office Anwendungen einen Abfrageassistenten. Jedoch können auch Makrosprachen oder SQL verwendet werden.

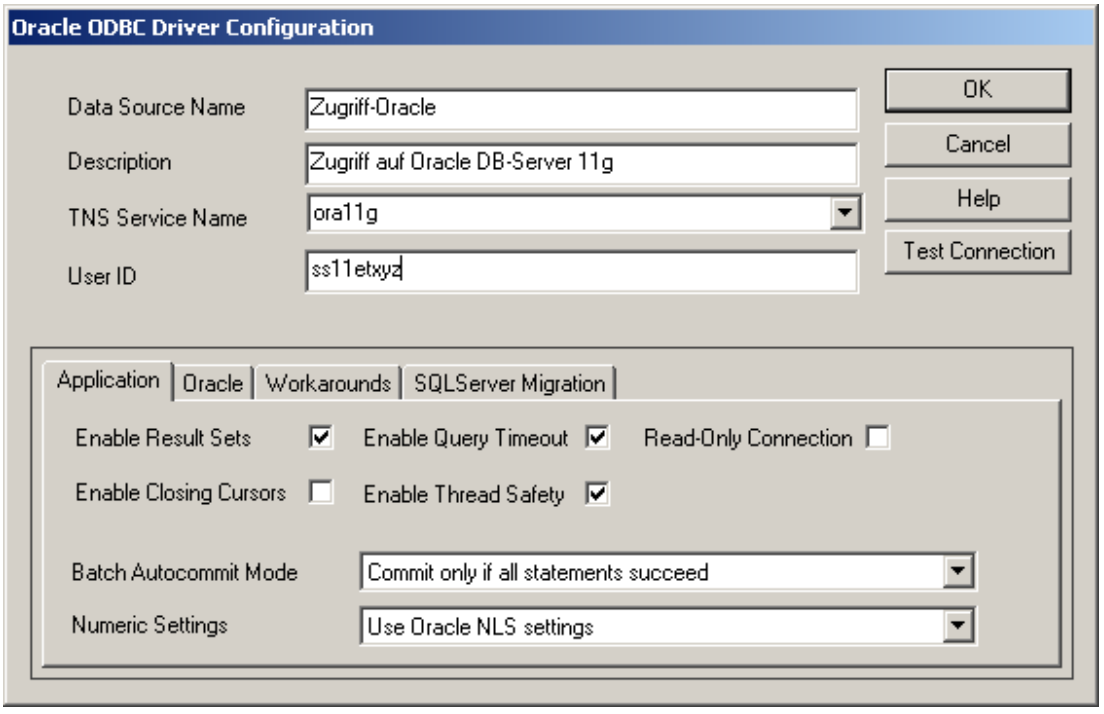
Begrenzt sind die Möglichkeiten durch einen geringeren von allen Herstellern verwendeten Standard als beispielsweise unter SQL für einzelne Datenbank-Systeme.



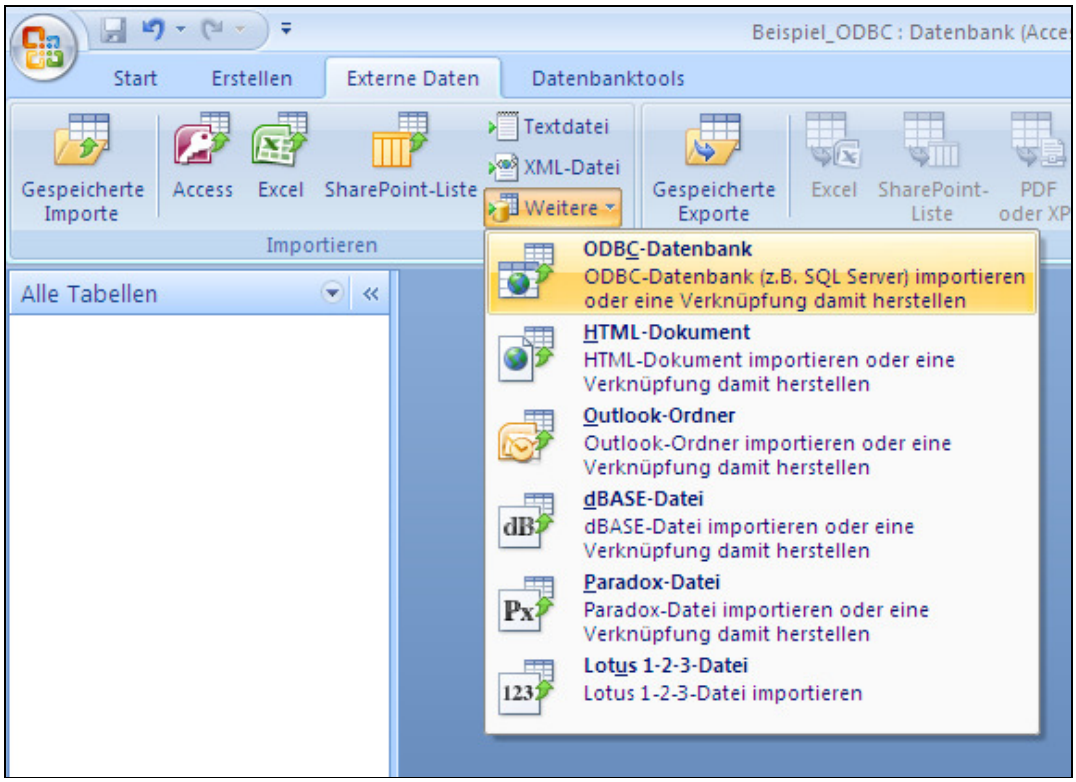
Schematische Darstellung

7.2 ODBC-Zugriffe in MS Access

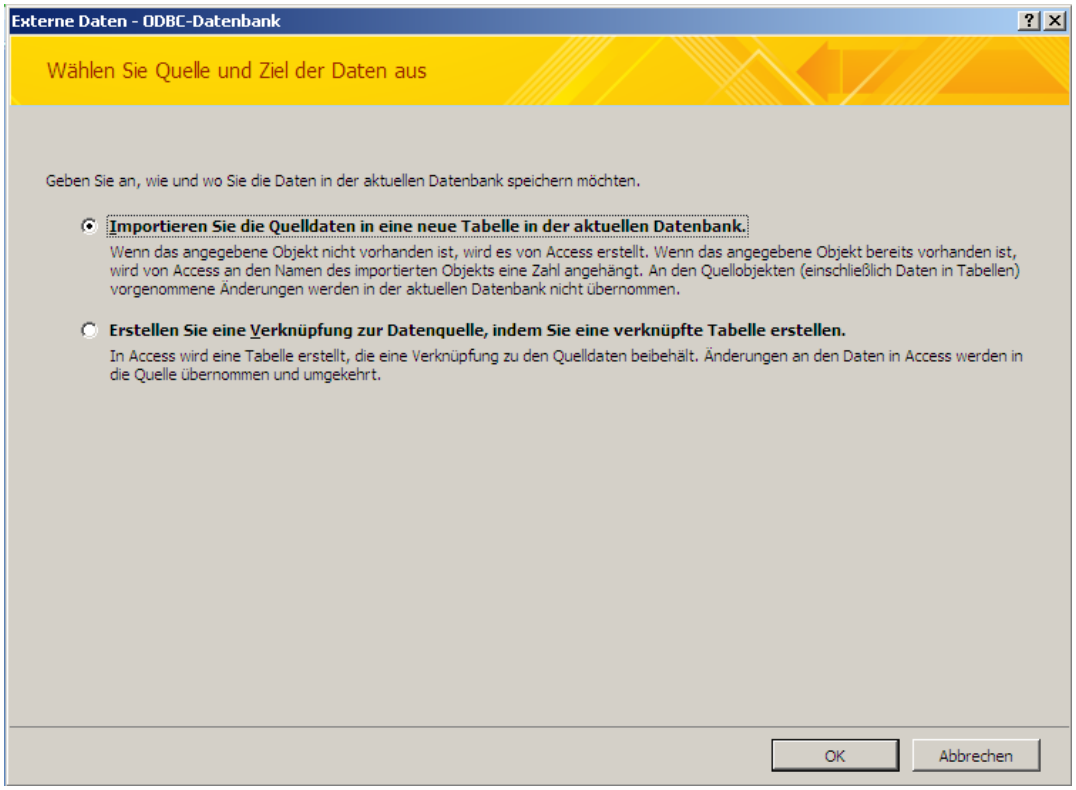
Im Microsoft ODBC Administrator wurde bereits eine Datenquelle unter Verwendung des Oracle ODBC Driver 11.01.00.07 eingerichtet. Diese Datenquelle bekommt den Namen **Oracle-Zugriff** und standardmäßig wird als Datenbank-Benutzer **ss11etxyz** auf die **Oracle 11g** Datenbank zugegriffen. Der Dialog wird dabei vom verwendeten Treiber bestimmt. Die weiteren Datenbank-, Anwendungs- und Übersetzungs-Optionen wurden unverändert übernommen.



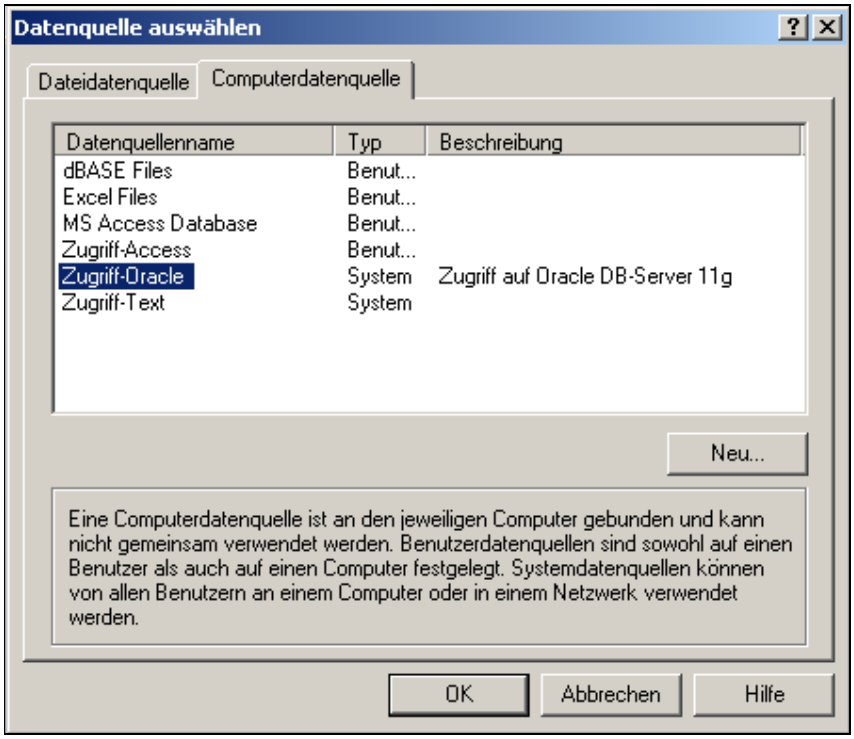
Anschliessend kann in einer MS Access Datenbank, z.B. in der zunächst leeren Datenbank Beispiel_ODBC der Import-Dialog gestarte werden:



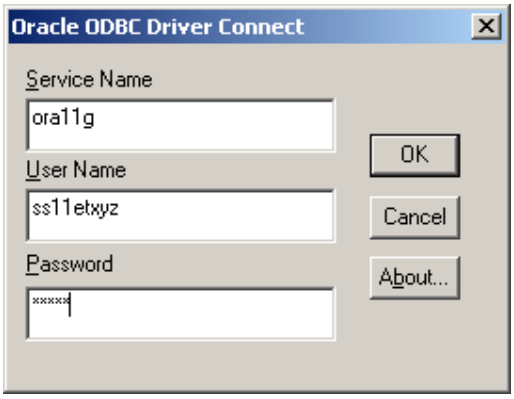
Für die Datenübernahme gibt es zwei Varianten:



In beiden Varianten muss anschließend die gewünschte Datenquelle ausgewählt oder gegebenenfalls zuvor neu angelegt werden:

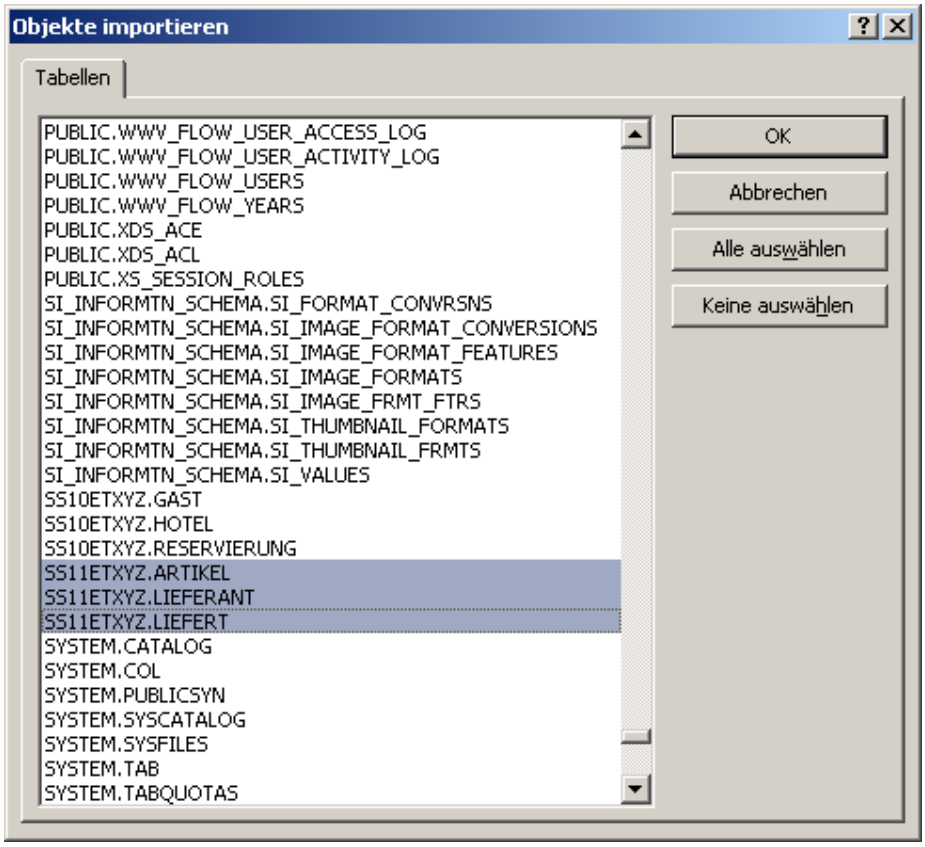


Nach Auswahl der ODBC-Datenquelle: **Oracle-Zugriff** meldet sich der Logon-Dialog zur Eingabe des Kennwortes zum Benutzer (an dieser Stelle können alle Vorgaben für den Anmeldedialog aber auch noch geändert werden):

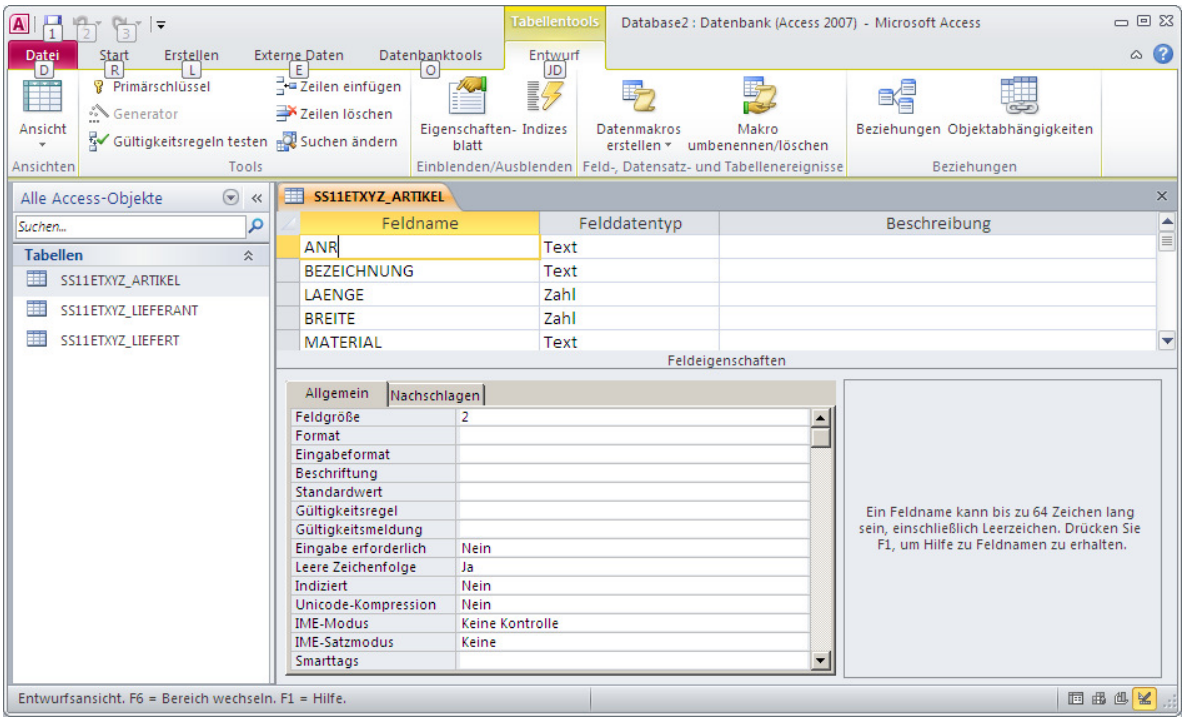


um anschliessend eine Liste alle Tabellen (und Views) anzubieten. Aus dieser Liste können eine oder auch mehrere Einträge ausgewählt (markiert) werden.

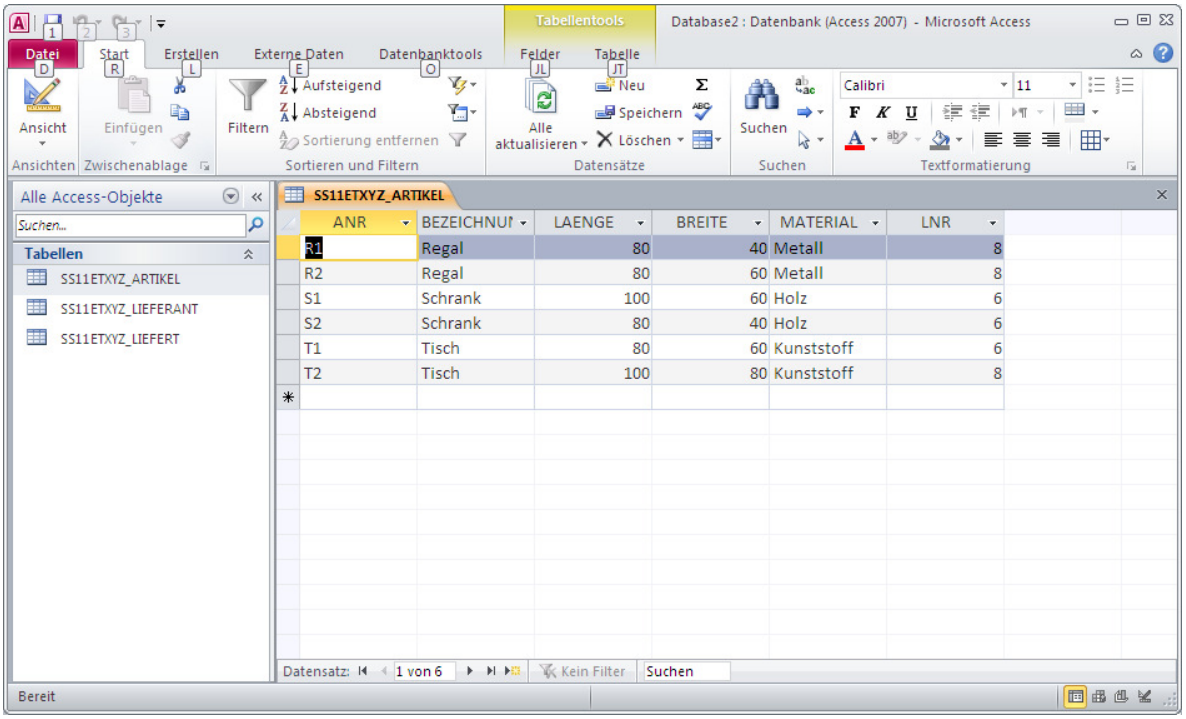
Hinweis:
Da allen Studierenden der lesende Zugriff auf die Basistabellen untereinander erlaubt wurde, ist die angebotende Liste sehr lang. Wenn die Liste den Fokus hat, kann durch (teilweise) Eingabe des Benutzernamens in der Liste positioniert werden. Im nachfolgenden Beispiel wurde nach Eingabe von „ss10“ auf den gewünschten Bereich der Liste positioniert und die zu importierenden Tabellen konnten ausgewählt werden:



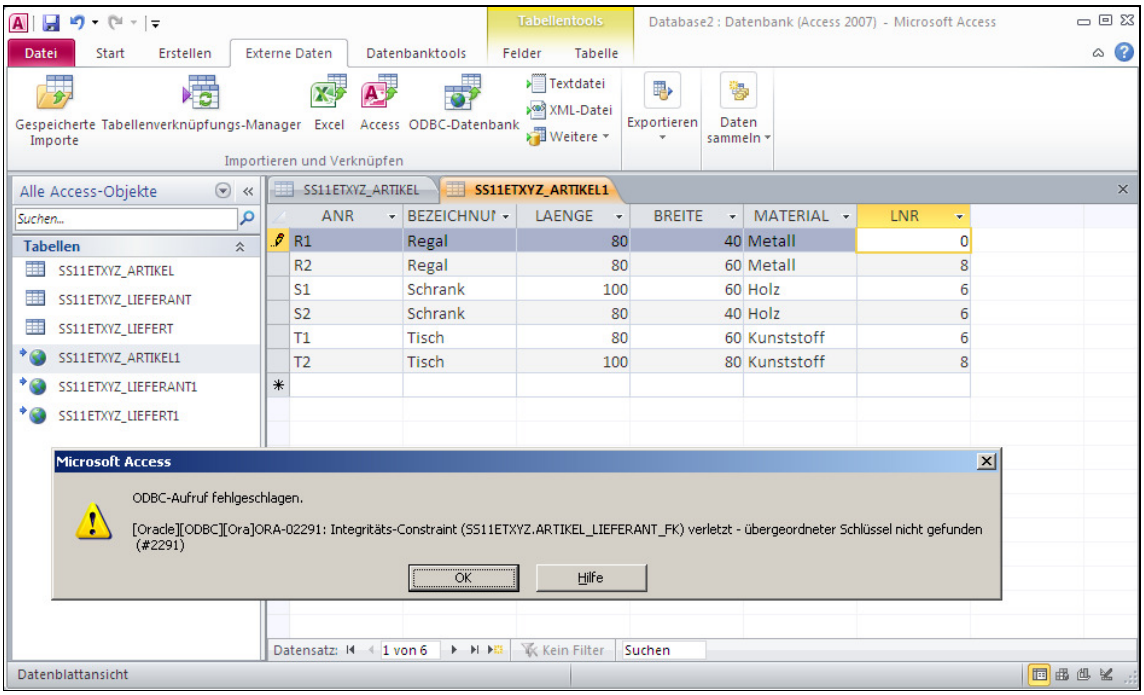
für die dann die entsprechenden Tabellen in der Access Datenbank angelegt werden.



Der Schema-Name wird dabei mit in den Tabellennamen übernommen, so dass erkennbar bleibt, welche Tabelle aus welchem Schema importiert wurde.



Werden die Tabellen nicht importiert, sondern verknüpft wurden, wird eine andere Darstellung verwendet, die DML-Operationen (insert, update, delete) und der Zugriff bei Abfragen werden direkt auf den Tabellen in der Oracle Datenbank ausgeführt. Eventuelle Fehlermeldungen kommen also direkt vom Oracle Server:

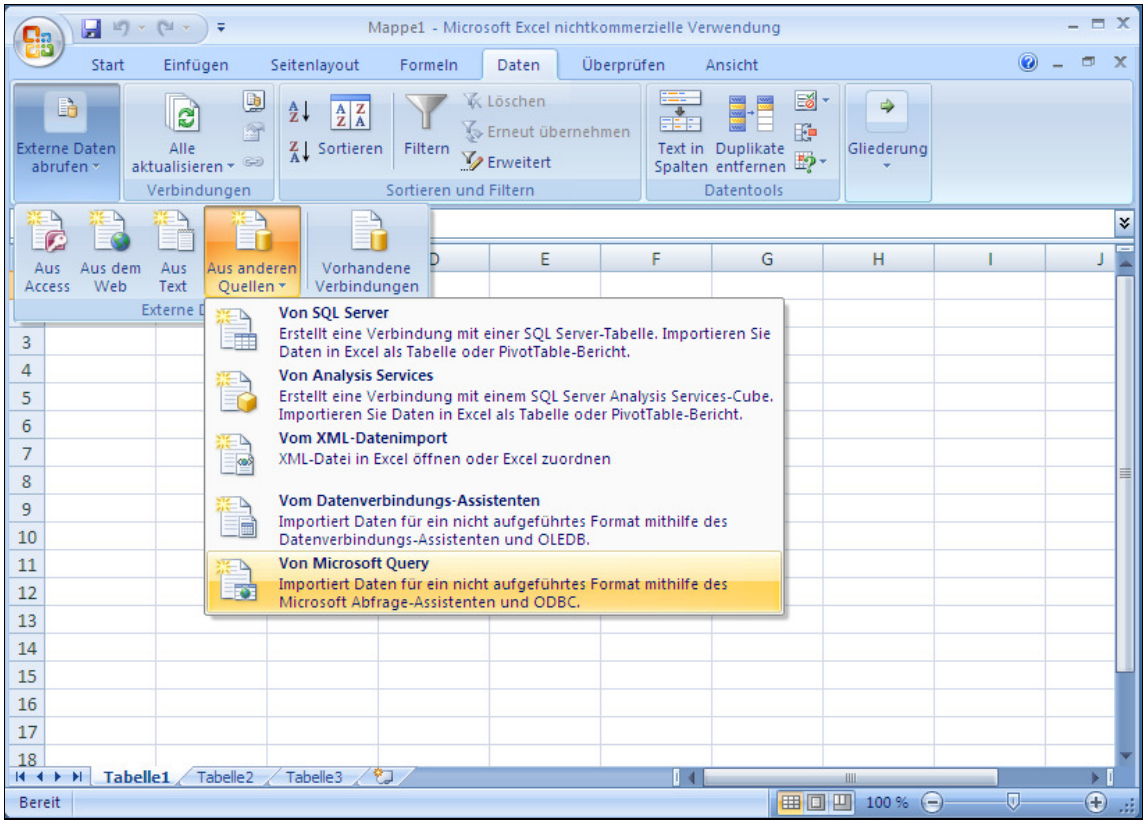


Hier wurden zusätzlich zu den importierten Tabellen auch noch dieselben Tabellen verknüpft (zur Unterscheidung wurde an den Namen ein 1 angehängt !) und sind in der Tabellenübersicht unterschiedlich gekennzeichnet. Der Versuch, die Lieferantennummer für den esrten Artikel auf 0 zu ändern, führte zu einer Oracle Fehlermeldung.

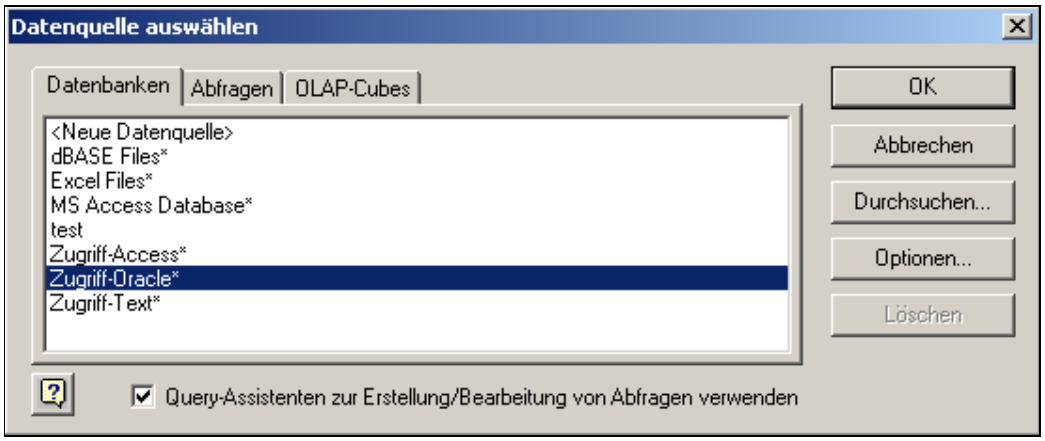
7.3 Datenimport über ODBC in MS Excel

Neben der Datenbanksoftware MS Access bieten auch die anderen Office-Produkte zur Tabellenkalkulation (MS Excel) und zur Textverarbeitung (MS Word) die Möglichkeit, über ODBC-Schnittstellen Daten aus Datenbanken in die entsprechenden Dokumente zu übernehmen. Dazu muss nur für die betreffende Datenbank ein ODBC-Treiber installiert sein (und natürlich der Zugriff auf die Datenbank technisch möglich sein).

Innerhalb vom **MS Excel** wird der Dialog zum Einfügen externen Daten über
Daten→Externe Daten abrufen→Aus anderen Quellen→Von Microsoft Query
gestartet:

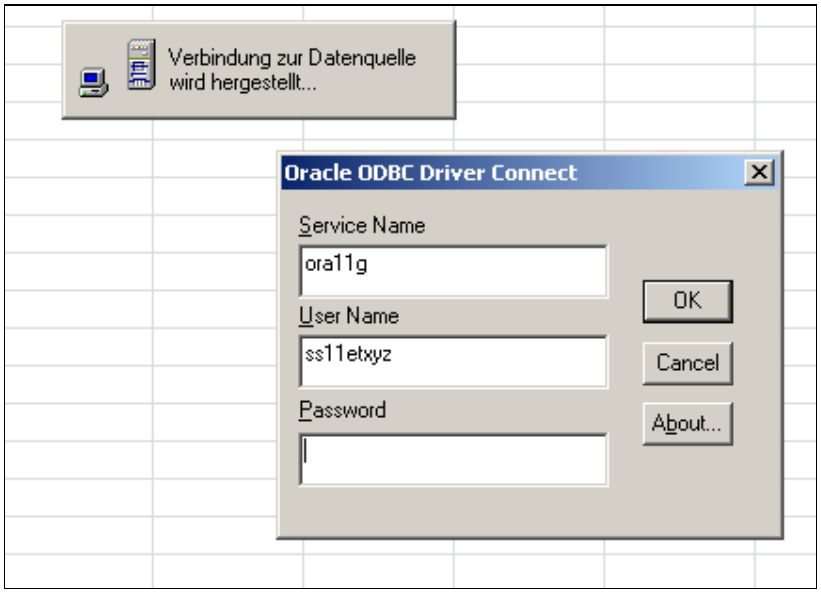


und führt im ersten Schritt zum Auswahldialog für eine ODBC-Datenquelle:

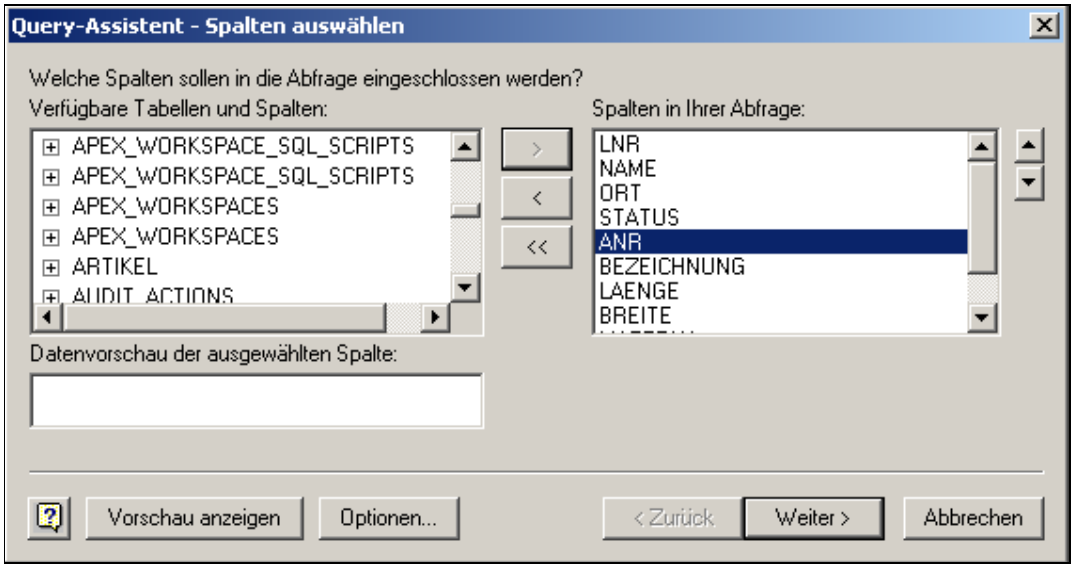


Für dieses Beispiel wählen wir auch wieder die Datenquelle **Zugriff-Oracle**, da wir wieder Daten aus der Oracle Datenbank in unser Excel Arbeitsblatt einfügen möchten:

Anschließend wird die Verbindung zur Datenquelle hergestellt, der Oracle-spezifische Anmeldedialog gestartet:



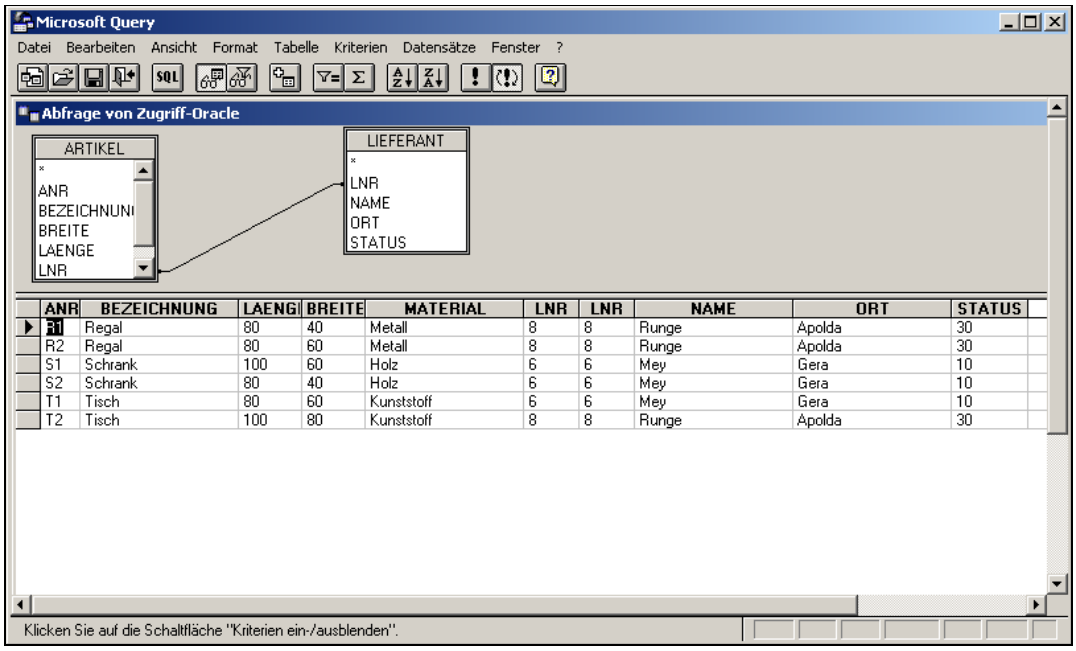
und ein Query-Assistent unterstützt in mehreren Schritten den Aufbau der Abfrage zur Bereitstellung der Daten:



Sind die vom Query-Assistenten angebotenen Gestaltungsmöglichkeiten ausreichend können im letzten Schritt die Daten direkt in das Arbeitsblatt eingefügt werden. Oft erscheint es aber einfacher, die Abfrage explizit zu gestalten, dies wird insbesondere dann der Fall sein, wenn die Definition der Abfrage noch nicht in allen Details feststeht und erst noch schrittweise erarbeitet werden soll:

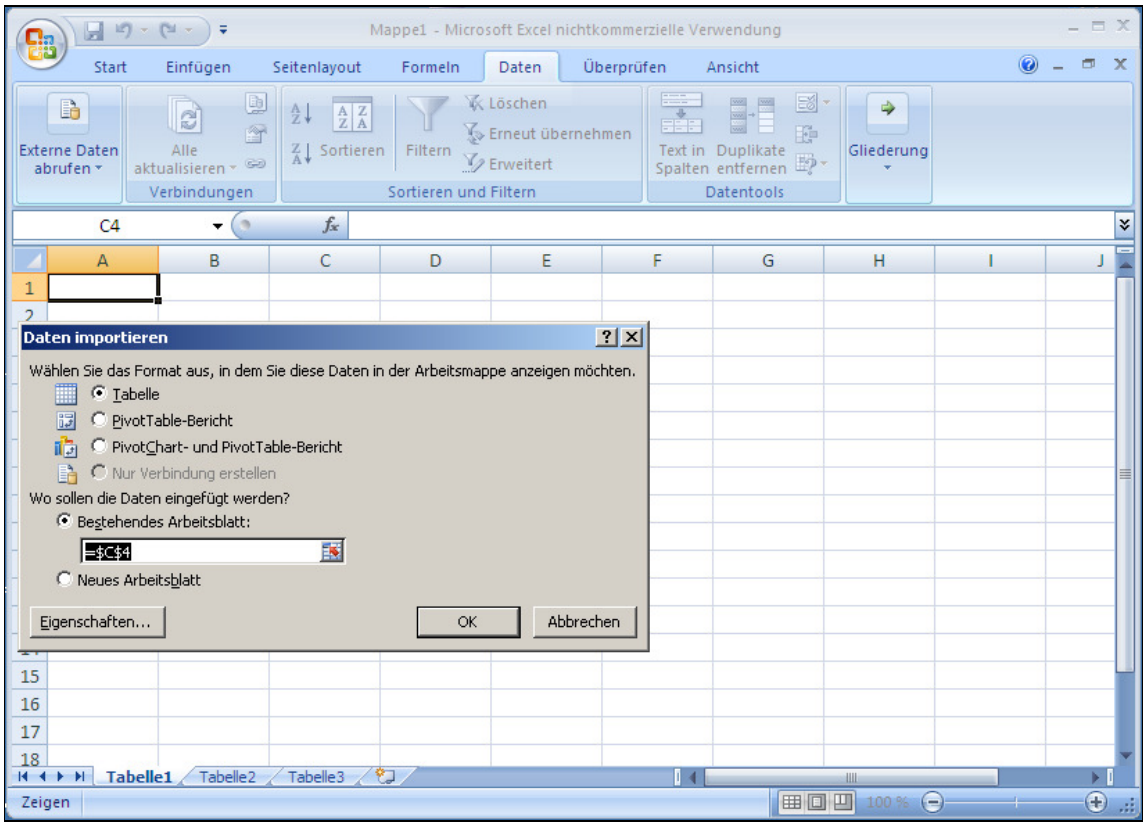
Um später erneut auf die Abfrage zurückgreifen zu können, kann die Abfrage auch gespeichert werden.

Der folgende Dialog im Programm **Microsoft Query** zur Gestaltung der Abfrage ist sehr stark an den Dialog im Programm Microsoft Access angelehnt:

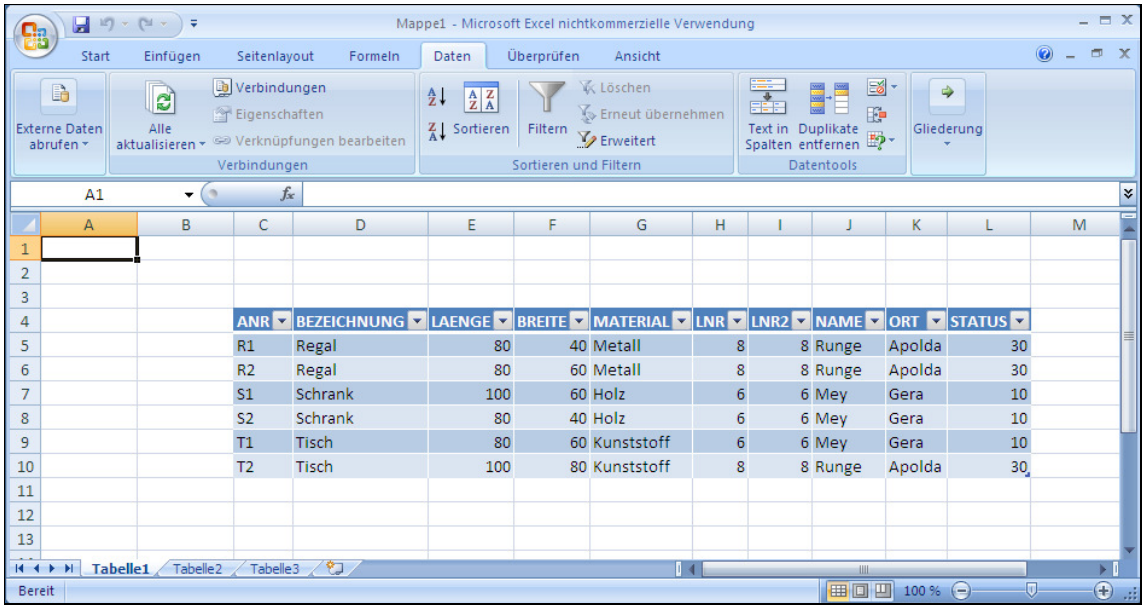


Am Ende des Gestaltungsprozesses können über den Menüpunkt:

die Daten in das Arbeitsblatt übernommen werden:

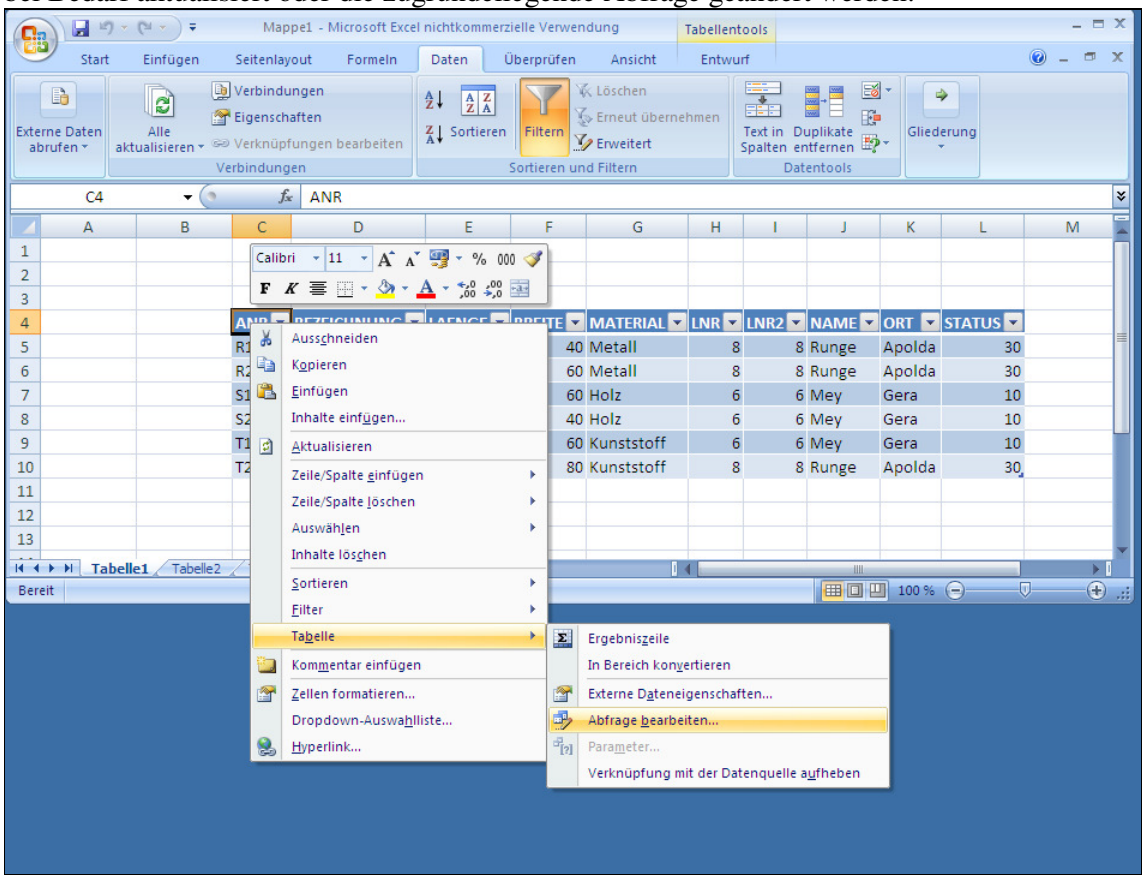


Dabei können noch abweichend von den Standardeinstellungen zahlreiche Eigenschaften des Datenimports festgelegt werden:



Im Endeffekt werden die durch die Abfrage bereitgestellten Daten aus der Datenbank als Teil eines Arbeitsblattes (MS Excel) in das aktuelle Dokument eingefügt.

Die Verbindung zur Datenquelle kann dabei beibehalten werden. Über das Kontextmenü werden zahlreiche Funktionen angeboten, z.B. können die Daten in dem betreffenden Bereich bei Bedarf aktualisiert oder die zugrundeliegende Abfrage geändert werden:

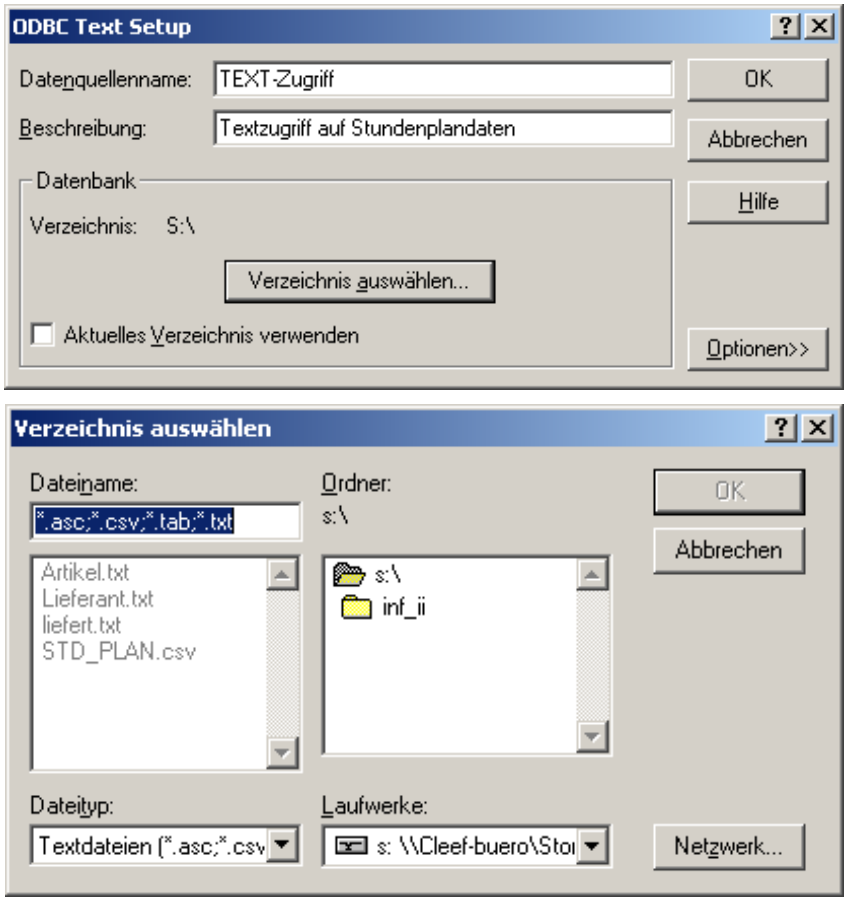


7.4 SQL-Zugriff auf Textdateien über ODBC

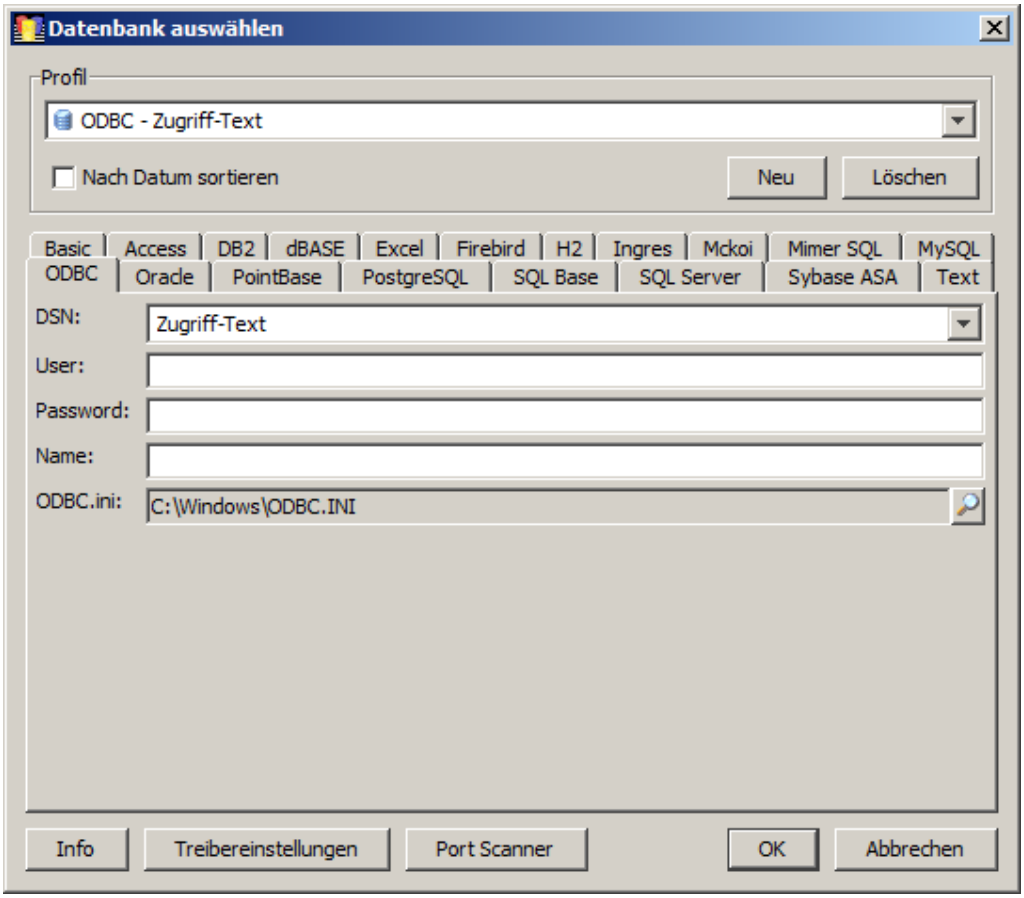
Beim Datenimport in eine Datenbanktabelle wurden Textdateien verwendet, die zeilenweise die einzelnen Tabellenzeilen enthalten und innerhalb einer Zeile die einzelnen Attributwerte durch speziell vereinbarte Trennzeichen unterscheiden:

```
1;Karcher;Jena;10
4;Schmidt;Weimar;20
6;Mey;Gera;10
8;Runge;Apolda;30
9;Todd;Jena;30
```

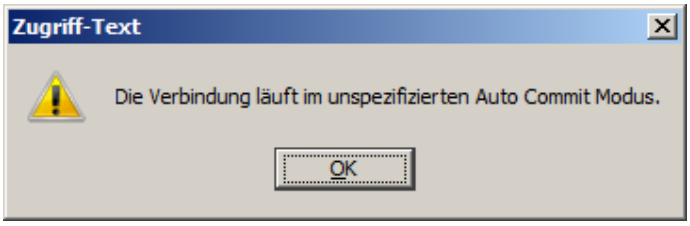
Für solche Textdateien zur Speicherung oder zum Austausch einfach strukturierter Daten wird häufig als Dateierweiterung das Kürzel **csv** (**Character Separated Values)** verwendet. Nach der Übernahme solcher Textdateien in eine Tabelle einer relationaler Datenbank können entsprechende Datenabfragen formuliert werden.
Mit dem ODBC-Treiber für Textdateien kann man sich die Übernahme der Daten in eine relationale Datenbank ersparen und stattdessen SQL-Abfragen direkt auf der Textdatei absetzen, d.h. mit Einsatz dieses ODBC-Treibers verhält sich eine solche Textdatei wie eine Tabelle in einer relationalen Datenbank und die Standardoperationen (Projektion, Selektion und Verbund) sind verfügbar. Wahlweise über die Anwendung **Oracle ODBC 32Bit Test** aus dem vorangegangenen Kapitel oder das **Aqua Data Studio** können entsprechende Abfragen formuliert werden.
Dabei wird im ODBC-Treiber ein Verzeichnis festgelegt, alle darin enthaltenen Textdateien (Dateierweiterung *.txt oder *.csv) stehen dabei als „Tabellen“ zu Verfügung.



Anschließend kann auf der Grundlage dieser Datenquelle im SQLDeveloper 2.3 eine weitere Verbindung vom Typ ODBC erstellt werden:



Login Name und Password werden hier nicht benötigt. Beim Öffnen dieser Verbindung erfolgt mit der Meldung:



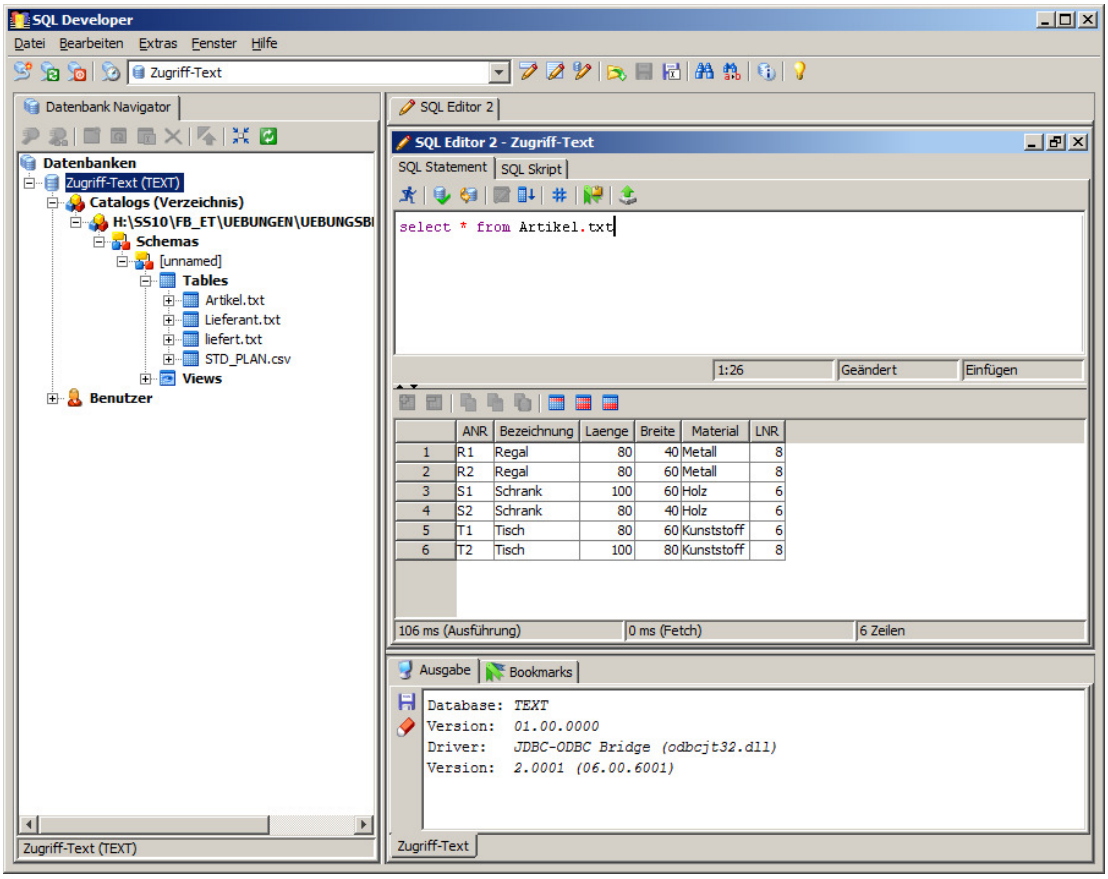
ein Hinweis auf eine Besonderheit der Verbindung.

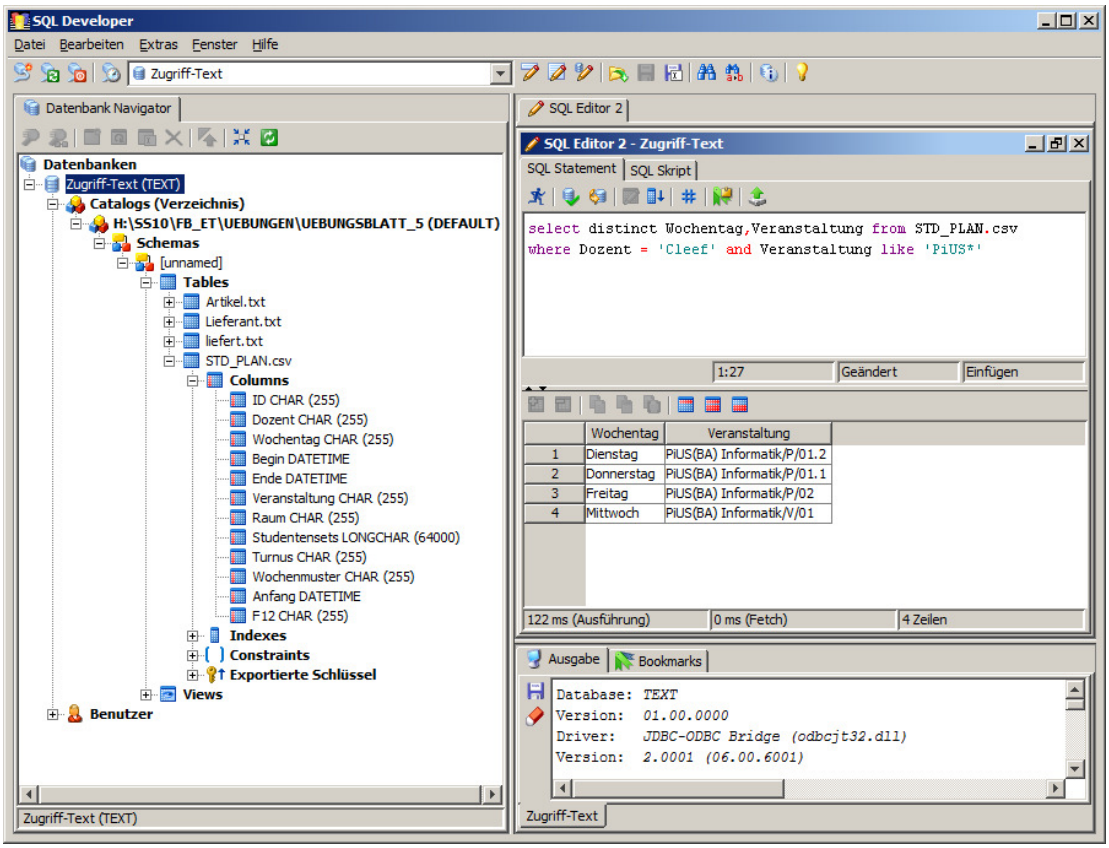
Für ODBC-Verbindungen für Text-Dateien gibt es einige Besonderheiten, da hier natürlich nicht mit einem relationalen Datenbank-Server kommuniziert wird.

So sind z.B. keine update- und delete-Anweisungen möglich, sondern nur einfache insert-Anweisungen. Die Zugriffe auf die Meta-Daten sind stark eingeschränkt. Eine solche Verbindung dient auch in erster Linie dazu, SQL-Abfragen auf den Textdateien auszuführen.

Normerweise sind die Dateien mit einer Erweiterung .txt oder .csv definiert, z.B. Artikel.txt.

Dies führt aber zu Problemen mit der SQL Notation, da die Datenamen jetzt gleichzeitig auch als Tabellennamen fungieren, wird die Erweitung .txt als Zugriff auf die Spalte txt interpretiert. Deshalb erscheint es ratsam, in den SQL-Anweisungen in der from-Klausel die vollständigen Dateinamen direkt mit einem entsprechenden Alias zu versehen.

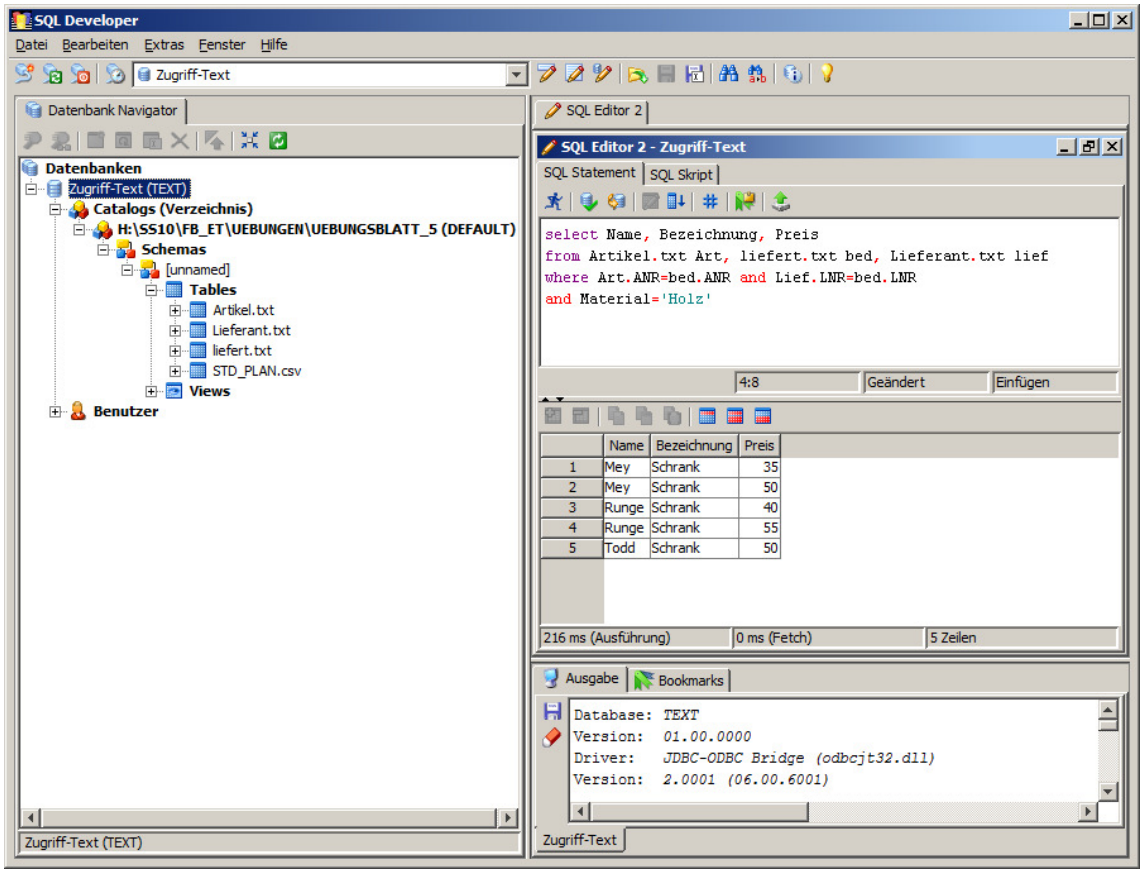

SQL Developer
Datei Bearbeiten Extras Fenster Hilfe
Zugriff-Text
Datenbank Navigator
SQL Editor 2
SQL Editor 2 - Zugriff-Text
SQL Statement SQL Skript
select * from Artikel.txt
1:26 Geändert Einfügen
Datenbanken
Zugriff-Text (TEXT)
Catalogs (Verzeichnis)
H:\SS10\FB_ET\UEBUNGEN\UEBUNGSBI
Schemas
[unnamed]
Tables
Artikel.txt
Lieferant.txt
liefert.txt
STD_PLAN.csv
Views
Benutzer
ANR Bezeichnung Laenge Breite Material LNR
1 R1 Regal 80 40 Metall 8
2 R2 Regal 80 60 Metall 8
3 S1 Schrank 100 60 Holz 6
4 S2 Schrank 80 40 Holz 6
5 T1 Tisch 80 60 Kunststoff 6
6 T2 Tisch 100 80 Kunststoff 8
106 ms (Ausführung) 0 ms (Fetch) 6 Zeilen
Ausgabe Bookmarks
Database: TEXT
Version: 01.00.0000
Driver: JDBC-ODBC Bridge (odbcjt32.dll)
Version: 2.0001 (06.00.6001)
Zugriff-Text (TEXT)
Zugriff-Text


SQL Developer
Datei Bearbeiten Extras Fenster Hilfe
Zugriff-Text
Datenbank Navigator
SQL Editor 2
SQL Editor 2 - Zugriff-Text
SQL Statement SQL Skript
select distinct Wochentag,Veranstaltung from STD_PLAN.csv
where Dozent = 'Cleef' and Veranstaltung like 'PiUS*'
1:27 Geändert Einfügen
Datenbanken
Zugriff-Text (TEXT)
Catalogs (Verzeichnis)
H:\SS10\FB_ET\UEBUNGEN\UEBUNGSBLATT_5 (DEFAULT)
Schemas
[unnamed]
Tables
Artikel.txt
Lieferant.txt
liefert.txt
STD_PLAN.csv
Columns
ID CHAR (255)
Dozent CHAR (255)
Wochentag CHAR (255)
Begin DATETIME
Ende DATETIME
Veranstaltung CHAR (255)
Raum CHAR (255)
Studentensets LONGCHAR (64000)
Turnus CHAR (255)
Wochenmuster CHAR (255)
Anfang DATETIME
F12 CHAR (255)
Indexes
Constraints
Exportierte Schlüssel
Views
Benutzer
Wochentag Veranstaltung
1 Dienstag PiUS(BA) Informatik/P/01.2
2 Donnerstag PiUS(BA) Informatik/P/01.1
3 Freitag PiUS(BA) Informatik/P/02
4 Mittwoch PiUS(BA) Informatik/V/01
122 ms (Ausführung) 0 ms (Fetch) 4 Zeilen
Ausgabe Bookmarks
Database: TEXT
Version: 01.00.0000
Driver: JDBC-ODBC Bridge (odbcjt32.dll)
Version: 2.0001 (06.00.6001)
Zugriff-Text (TEXT)
Zugriff-Text

Hierbei ist die Abfrage immer nur auf einer einzelne Tabelle (Artikel.txt bzw. STD_PLAN.csv) formuliert.

Hinweis: Die Joker-Zeichen für die Musterangabe bei **like** sind hier * und ?

Es sind aber auch mehrere Tabellen möglich, dabei ist die Verwendung eines Alias-Namens dringend geboten, da sonst die Datei-Erweiterung als Spaltenname verstanden werden kann:



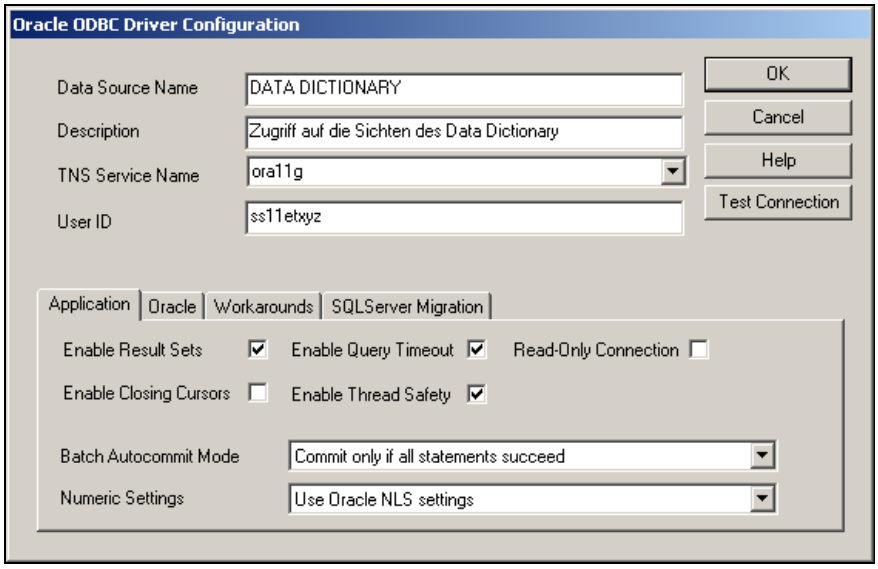
Eine Referenz auf die einzelnen Spalten über einen Spaltennnamen ist dabei aber nur möglich, wenn die ursprünglichen Textdateien um eine zusätzliche Zeile als erste Zeile ergänzt wurde, in der die Spaltennamen hinterlegt sind:

```
LNR;Name;Ort;Status
1;Karcher;Jena;10
4;Schmidt;Weimar;20
6;Mey;Gera;10
8;Runge;Apolda;30
9;Todd;Jena;30
```

Fehlt diese Angabe, werden die Spalten einfach mit F1, F2, ….. bezeichnet.

7.5 Beispielzugriffe auf das Data Dictionary

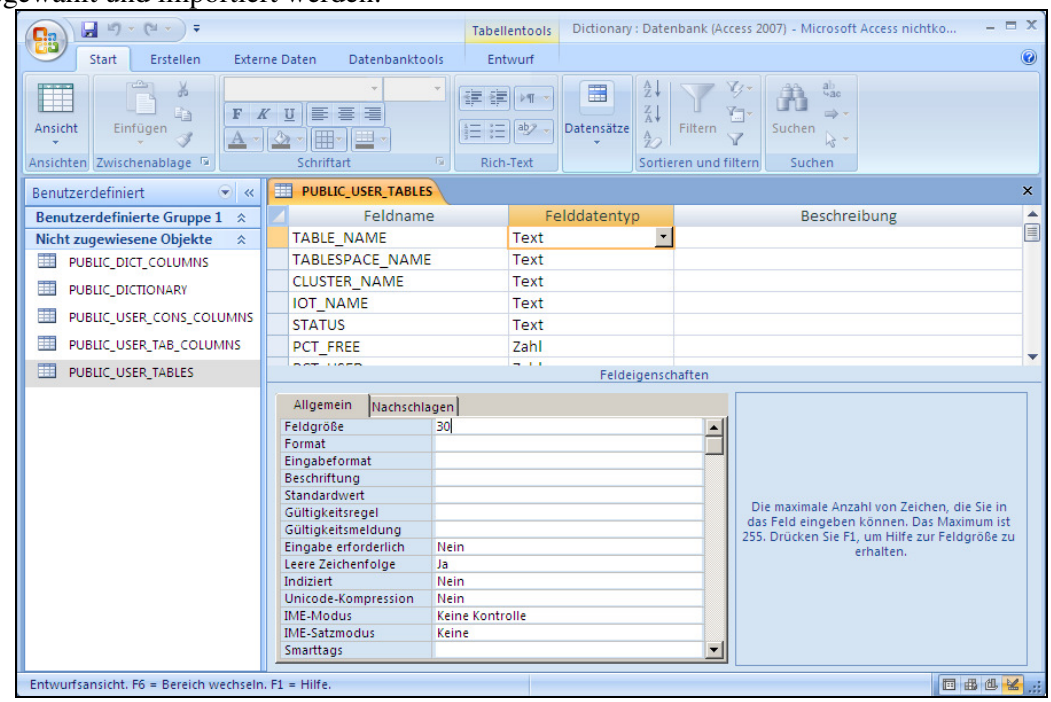
Als ein Beispiel für die Nutzung von ODBC kann innerhalb von MS Access auf die Ansichten des Data Dictionary einer Oracle-Datenbank zugegriffen werden. Anschließend können dann verschiedene Übersichten als Report erstellt und ausgegeben werden. Dabei kann über entsprechende Abfragen der Zugriff auf die Data Dictionary Ansichten sinnvoll eingeschränkt werden.



In der danach angebotenen Auswahl sind auch die öffentlichen Data Dictionary Sichten enthalten. Hier können jetzt zum Beispiel

PUBLIC.DICTIONARY
PUBLIC.DICT_COLUMNS
PUBLIC.USER_TABLES
PUBLIC.USER_TAB_COLUMNS
PUBLIC.USER_CONS_COLUMNS

ausgewählt und importiert werden:



Das so genannte Data Dictionary einer Oracle Datenbank enthält die Strukturinformationen aller in der Datenbank gespeicherten „Objekte“, und stellt diese Informationen alle Benutzern öffentlich zur Verfügung. Über die Tabelle PUBLIC_USER_TABLES können z.B. alle in einem Schema angelegten Tabellen (z.B. für den Benutzer SS11ETXYZ) angezeigt werden, die Tabelle PUBLIC_USER_TAB_COLUMNS enthält dazu die Spalten(Beschreibungen) der Tabellen.

7.6 Vorteile/Nachteile und typische Einsatzgebiete

Vorteile:
- **Anwender** können Datenbanksysteme weitgehend frei wählen.
- **Entwickler** brauchen sich nicht mit verschiedenen API's auszukennen, sondern nur mit einer Datenbankschnittstelle (Erhebliche Erleichterung der Programmierung)
- **Datenbankanbieter** haben den Vorteil, dass beliebige Anwendungsprogramme mit ihrem Datenbankmanagementsystem zusammenarbeiten können, wenn beide ODBC unterstützen.
- Für nahezu alle Anwendungs-/Datenbanksysteme sind ODBC-Treiber verfügbar.
- Es wird keine „Client-Software“ des Datenbankbieters benötigt, der ODBC Treiber kann außer vom Datenbankanbieter auch von sonstigen Unternehmen angeboten werden oder bereits Teil der Anwendungssoftware ein.

Nachteile:
- Verwaltungsaufwand der ODBC-Schnittstelle
- Geschwindigkeitseinbußen bei Datenbankabfragen, durch den zusätzlichen Ressourcenverbrauch der ODBC-Schnittstelle
- Eingeschränkte Funktionalität gegenüber Datenbank-spezifischen Zugriffsmethoden

8. JDBC

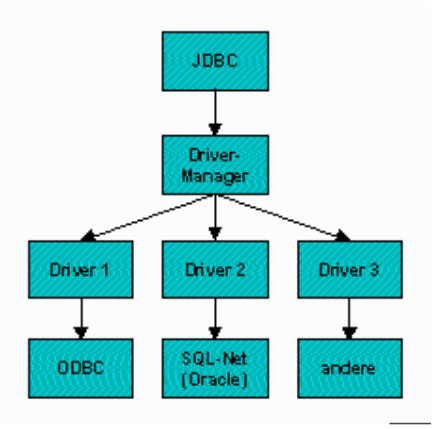
8.1 Grundlagen

JDBC definiert Objekte und Methoden, mit deren Hilfe es möglich wird, in Applikationen den Zugriff auf darunter liegende Datenbanken zu realisieren. Der Zugriff erfolgt in der folgenden Weise:

- Herstellung einer Verbindung zur Datenbank über den entsprechenden JDBC-Treiber für das verwendete DBMS
- Erstellung eines Statement-Objektes
- Weitergabe des auszuführenden Statements über das Statement-Objekt an das darunter liegende DBMS
- Abholung der Ergebnisse über die Ergebnisdatensätze
- Schließen der Verbindung zur Datenbank

Die Java-Applikation läuft dabei typischerweise auf einem Client-Rechner, der eine Verbindung zu einer (oder mehreren) auf einem Remote-Server liegenden Datenbank(en) aufbaut.

Die Verwaltung der Datenbankverbindungen, die in Form von Java-Objekten erzeugt werden, wird vom JDBC-Treiber-Manager übernommen. Dieser kann mehrere Verbindungen zu unterschiedlichen Datenbanken gleichzeitig verwalten und ermöglicht somit auch den Zugriff auf verteilte Datenbanken.



JDBC-Treiber-Manager

Über JDBC wird ein Objekt der Klasse "java.sql.DriverManager" erzeugt, welches über die Methode "DriverManager.getConnection()" ein oder mehrere Objekte vom Typ "java.sql.Connection" erstellt, über die dann die Verbindung zu (verschiedenen) Datenbanken hergestellt werden kann.

8.2 Treibertypen

Die Anwendung von JDBC verlangt den Einsatz eines JDBC-Treibers für die verwendete Datenbank, der über den sogenannten JDBC-Treiber-Manager verwaltet wird. Die JDBC-Treiber lassen sich in vier verschiedene Kategorien unterteilen (Typ-1, Typ-2, Typ-3, Typ-4). Die Unterschiede in den Architekturen der einzelnen Treiber werde ich im nachfolgenden erläutern und anhand von Abbildungen veranschaulichen.

- **Typ-1:**

  Die Java-Applikation bzw. das Applet binden über den JDBC-Treiber-Manager den JDBC-ODBC-Bridge-Treiber ein, der seinerseits über native Schnittstellen (z.B. Windows-DLLs) den ODBC-Treiber-Manager in Verbindung mit einem lokal installierten ODBC-Treiber zur Datenbankanbindung anspricht.

- **Typ-2:**

  Hier wird der native ODBC-Treiber durch einen nativen herstellerabhängigen Treiber (zum Beispiel einen Oracle-Treiber) ersetzt. Auch dieser Treiber muss lokal beim Client installiert werden.

- **Typ-3:**

  Diese Architektur ersetzt die lokale Installation eines Treibers beim Client durch eine serverseitige Middleware-Installation, welche den Datenbankzugriff realisiert. Das bietet den Vorteil, dass beim Client keinerlei Installationen mehr notwendig ist, da der universelle Treiber vom Server geladen werden kann und der "echte" Treiber auf dem Server ausgeführt wird.

  Die Java-Applikation bzw. das Applet binden über den JDBC-Treiber-Manager einen universellen Java-JDBC-Treiber ein, welcher die Aufrufe in ein DBMS-unabhängiges Protokoll übersetzt und anschließend an eine auf dem Server installierte Middleware-Applikation sendet. Diese wandelt die Aufrufe entsprechend in datenbankspezifische Aufrufe um und realisiert die Datenbankanbindung.

- **Typ-4:**

  Bei dieser Möglichkeit werden Treiber verwendet, welche in reinem Java-Code programmiert sind. Die Anbindung von nativem Code entfällt. Diese Alternative wird als die modernste betrachtet, da sie durch die fehlende Einbindung von nativem Code auf der einen Seite die Plattformunabhängigkeit von Java unterstützt und auf der anderen Seite den Download der Treiber vom Webserver auf den Client ermöglicht.
  Die Java-Applikation bzw. das Applet binden über den JDBC-Treiber-Manager einen nativen DBMS-spezifischen JDBC-Treiber ein, welcher auf direktem Wege die Datenbankverbindung herstellt

Datenbank	URL	Treiber-Klasse
IBM DB2	jdbc:db2://<HOST>:<PORT>/<DB>	COM.ibm.db2.jdbc.app.DB2Driver
Microsoft SQL Server	jdbc:weblogic:mssqlserver4:<DB>@<HOST>:<PORT>	weblogic.jdbc.mssqlserver4.Driver
JDBC-ODBC Bridge	jdbc:odbc:<DB>	sun.jdbc.odbc.JdbcOdbcDriver
Oracle Thin	jdbc:oracle:thin:@<HOST>:<PORT>:<SID>	oracle.jdbc.driver.OracleDriver
IDS Server	jdbc:ids://<HOST>:<PORT>/conn?dsn='<ODBC_DSN_NAME>'	ids.sql.IDSDriver
Informix Dynamic Server	jdbc:informix-sqli://<HOST>:<PORT>/<DB>:INFORMIXSERVER=<SERVER_NAME>	com.informix.jdbc.IfxDriver
Microsoft SQL Server (JTurbo Driver)	jdbc:JTurbo://<HOST>:<PORT>/<DB>	com.ashna.jturbo.driver.Driver
Microsoft SQL Server 2000 (Microsoft Driver)	jdbc:microsoft:sqlserver://<HOST>:<PORT>[;DatabaseName=<DB>]	com.microsoft.sqlserver.jdbc.SQLServerDriver
MySQL (MM.MySQL Driver)	jdbc:mysql://<HOST>:<PORT>/<DB>	org.gjt.mm.mysql.Driver
Oracle OCI 9i	jdbc:oracle:oci:@<SID>	oracle.jdbc.driver.OracleDriver
PostgreSQL (v7.0 and later)	jdbc:postgresql://<HOST>:<PORT>/<DB>	org.postgresql.Driver
Sybase (jConnect 5.2)	jdbc:sybase:Tds:<HOST>:<PORT>	com.sybase.jdbc2.jdbc.SybDriver

8.3 Klassenüberblick

In den vorhergehenden Abschnitten wurde erläutert, wie der allgemeine Ablauf einer Datenbankanfrage über JDBC abläuft. In diesem Abschnitt soll ein erster Überblick über die benutzten Klassen gegeben werden. Die ausführlichen Klassenbeschreibungen im Stil von javadoc findet man unter:

http://java.sun.com/j2se/1.4.2/docs/api/

dort findet man neben dem allgemeinen Teil der (englischsprachigen) Klassenbeschreibung auch die speziellen Angaben zu den Felder, Konstruktoren und Methoden angegeben.

Hier wird nun auf die spezielle Implementation innerhalb eines Programms eingegangen (im nächsten Abschnitt sind dann die vollständigen Quelltexte für ein entsprechendes Beispielprogramm gegeben):

- **Importieren der notwendigen Klassen**

  Die für die Ausführung von JDBC notwendigen Klassen liegen allesamt im Package java.sql vor, das Bestandteil des von SUN ausgelieferten JDK ist. Diese Klassen müssen vor der Anwendung innerhalb eines Java-Programms (zu Beginn des Programms) importiert werden. Durch den Platzhalter (*) kann man erreichen, dass alle Klassen des angegebenen Pakets importiert werden.

  Die Syntax für den Import der JDBC-Klassen lautet also wie folgt:

  import java.sql.*;

- **Laden des JDBC-Treibers**

  Zur Ausführung der JDBC-Befehle muss ein Datenbanktreiber geladen werden, der die Anweisungen in eine Form umsetzt, die vom speziellen Datendatenbanksystem verstanden wird. Ein solcher Treiber kann z.B. auch die JDBC-ODBC-Bridge sein, die in Verbindung mit einem lokal installierten und eingerichteten ODBC-Treiber jede ODBC-Datenbank ansprechen kann, oder ein Treiber für ein spezielles DBMS (zum Beispiel der Oracle-JDBC-Treiber). Der Treiber wird mit dem Java Klassenlader (class loader) über die Methode "class.forName()" geladen oder alternative über die Klassenmethode registerDriver der Klasse Driver Manager.

  Die Syntax für das Laden des Oracle-JDBC-Treibers lautet somit:

  Class.forName("oracle.jdbc.driver.OracleDriver");
  oder
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

  Eine Übersicht über die jdbc-Treiber für häufig eingesetzte Datenbanken mit den Namen für die jeweilige Treiberklasse findet man unter 8.2.

- **Connect zur Datenbank**

  Spätestens hier kommt der bereits erwähnte JDBC-Driver-Manager "ins Spiel". Es handelt sich auch hierbei um eine Klasse, die sich java.sql-Package befindet.

  Die Datenbankverbindung wird über die Methode "getConnection()" hergestellt. Diese Methode erwartet bis zu drei Parameter:

- String IN » URL der Datenbank, zu der die Verbindung aufgenommen werden soll
- String IN » Anmeldename auf der Datenbank
- String IN » Passwort für die Datenbankanmeldung

Die beiden letzten Parameter sind optional und können ggf. auch als Liste übergeben werden, auf diesen Spezialfall möchte ich hier jedoch nicht eingehen. Eine Übersicht über die jdbc-Treiber für häufig eingesetzte Datenbanken mit der jeweiligen url für die getConnection-Methode findet man unter 8.2.

Bei fehlerfreier Ausführung wird eine geöffnete Datenbankverbindung (vom Typ Connection) zurückgeliefert:

- Connection OUT » Offene DB-Verbindung

Der Aufruf im Programm hat demzufolge die folgende Syntax:

Connection my_con = DriverManager.getConnection(db_url, username, passwort);

- **Erzeugen eines Statements**

  Es gibt drei unterschiedliche Statementklassen:

  Statement: Ausführen von einfachen select-Anweisungen ohne Parameter bzw. einfachen DDL- oder DML-Anweisungen. Die Syntax zum Erzeugen eines Statement-Objektes lautet: Statement my_stmt = my_con.createStatement(); Zu beachten ist, dass diese Methode für eine bestehende (!) Datenbankverbindung (s.o.) erfolgen muss.

  PreparedStatement: Ausführen einer „vorübersetzten“ Select-Anweisung ohne oder insbesondere mit Parametern, d.h. Objekte dieser Klasse werden insbesondere dann eingesetzt, wenn eine Select-Anweisung in der Struktur unverändert bleibt, aber einzelne Werte als Parameterwert immer wieder verändert wird.
  Die Syntax zum Erzeugen eines PreparedStatement-Objektes lautet: PreparedStatement my_prepstmt = my_con.prepareStatement(String sql); In der SQL-Anweisung werden die Parameter durch „?“ angegeben und können anschließend mit speziellen Methoden durch die aktuellen Parameterwerte überschrieben werden. Zu beachten ist ist auch hier, dass diese Methode für eine bestehende (!) Datenbankverbindung (s.o.) erfolgen muss.

  CallableStatement: Ausführen einer PL/SQL-Blocks oder einer gespeicherten Prozedur, d.h. Objekte dieser Klasse werden insbesondere dann eingesetzt, wenn an Stelle einer einzelnen SQL-Anweisung ein anonymer PL/SQL-Block oder eine zuvor gespeicherte Prozedur ausgeführt werden soll
  Die Syntax zum Erzeugen eines CallableStatement-Objektes lautet: CallableStatement my_callstmt = my_con.prepareCall(String sql); Auch hier können wieder für symbolische Parameter, notiert durch ? entsprechende aktuelle Parameterwerte eingesetzt werden. Zu beachten ist auch hier, dass diese Methode für eine bestehende (!) Datenbankverbindung (s.o.) erfolgen muss.

- **Ausführen eines Statements**

Ein einfaches Statement wird mit einer der beiden Methoden executeQuery (für Abfragen) oder executeUpdate() (bei Update / Insert / Delete) ausgeführt. Die Methode bezieht sich immer auf ein bestehendes Statement und erwartet als Eingabeparameter den String für das auszuführende Statement. Der Rückgabewert ist entweder vom Typ ResultSet (Ergebnistabelle bei Abfragen, Auswertung s.u.) oder vom Typ int (Anzahl der geänderten Datensätze bei Update).

Die Syntax lautet demnach wie folgt:

```
ResultSet my_result = my_stmt.executeQuery("select * from Artikel");
```

oder

```
int lv_result = my_stmt.executeUpdate("update Artikel set  ...");
```

- **Ausführen eines PreparedStatements**

  Ein PreparedStatement wird auch im Prinzip vollkommen analog zum einfachen Statement mit einer der beiden Methoden executeQuery (für Abfragen) oder executeUpdate() (bei Update / Insert / Delete) ausgeführt. Dabei müssen aber vor der Ausführung allen formalen Parametern aktuelle Parameterwerte zugewiesen werden. Hierzu stehen typgerechte Methoden (setInt(….), setString(….) …) zur Verfügung, die Werte selbst können als konstante Größen oder als Inhalt (lokaler) Variablen angegeben werden.

- **Auswerten des Ergebnisses**

  Wie bereits erwähnt wird das Ergebnis einer Abfrage (executeQuery) in Form einer Ergebnistabelle vom Typ ResultSet zurückgeliefert und kann im Programm dann Zeile für Zeile bearbeitet werden (Methode next positioniert auf die nächste Zeile oder signalisiert, wenn es keine weiteren Zeilen gibt). Eine solche Ergebnismenge entspricht im Prinzip dem Cursor-Konzept in PL/SQL.

  Da der Inhalt und der Aufbau dieser Ergebnistabelle von der/den zugrunde liegenden Datenbanktabelle(n) der Abfrage abhängt, ist sie nicht mit einfachen Mitteln auszugeben, sondern muss vielmehr mit den entsprechenden (vom Interface ResultSet zur Verfügung gestellten) Methoden ausgewertet werden. Zum Beispiel kann die Ergebnistabelle der folgenden Abfrage

  ```
  SELECT LNR, Name FROM Lieferant;
  ```

  welche einen numerischen Wert (die Lieferantennummer) sowie einen String (den Namen) zurückliefert, mit den beiden folgenden Methoden ausgewertet und die Werte lokalen Variablen zugeordnet werden:

  ```
  int       lv_nrr    = my_result.getInt("LNR");
  String    lv_name= my_result.getString("Name");
  ```

  Das Interface ResultSet bietet für jeden Datentyp eine entsprechende Methode der Art get<Typ>(….) an, mittels derer die entsprechenden Ergebnisspalten ausgelesen und lokalen Variablen zugewiesen werden können.

Ist der Aufbau der zugrundeliegenden Datenbanktabelle nicht bekannt (Beispiel: "Select * from Artikel"), muss erst über die Methode Connection.GetMetaData für eine bestehende Verbindung die Struktur der Datenbank (bzw. der Tabelle) ausgelesen und mit den entsprechenden Methoden ausgewertet werden.

- **Abmelden von der Datenbank**

  Für das Abmelden von der Datenbank (also dem Schließen der Verbindung) wird die entsprechende Methode close() der Klasse Connection benutzt, die für eine bestehende Verbindung (!) und ohne Parameter aufgerufen wird.

  Die Syntax lautet:

  my_con.close();

8.4 Beispiel für JDBC-Zugriffe Oracle, MySQL

```
import java.sql.*;

public class Database_Demo
{

  public static void main(String[] args)
  {
    Statement stmt = null;
    Connection conn = null;

    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        conn = DriverManager.getConnection
        ("jdbc:oracle:thin:@gw3svr32.gw.net.fh-jena.de:1521:oracle11","ss11etxyz", "geheim");

        System.out.println("Verbindung zur Datenbank hergestellt !");

        stmt = conn.createStatement ();

        stmt.execute ("insert into Lieferant Values (10, 'Meyer,'Paul',10)"))
        System.out.println("Neuer Lieferant wurde eingefuegt !");

    }
    catch (SQLException e)     {      System.out.println(e);      }

    try
    {
        pstmt.close();
        conn.close();
        System.out.println("Verbindung zur Datenbank beendet !");
    }
    catch (SQLException e)
    {
        System.out.println(e);
    }

  }

}
```

Beim Zugriff auf andere Datenbanken müssen nur die beiden Zeilen:

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

conn = DriverManager.getConnection
("jdbc:oracle:thin:@gw3svr32.gw.net.fh-jena.de:1521:oracle11","ss11etxyz", "geheim");

angepasst werden.

Oracle Datenbank: ora11g

Treiberklasse:
oracle.jdbc.driver.OracleDriver
Verbindungsdaten:
“jdbc:oracle:thin:@gw3svr32.gw.net.fh-jena.de:1521:oracle11", "ss11etxyz", "geheim"

MySQL Datenbank:

Treiberklasse:
org.gjt.mm.mysql.Driver
Verbindungsdaten:
"jdbc:mysql://gw3svr32.gw.net.fh-jena.de:3306/ss11etxyz","ss11etxyz","geheim"

ODBC Datenbank: JDBC-ODBC-Bridge

Treiberklasse:
sun.jdbc.odbc.JdbcOdbcDriver
Verbindungsdaten: (auf Access-Datenbank ohne Benutzer, Kennwort)
"jdbc:odbc:ZUGRIFF"
Verbindungsdaten: (auf MySQL-Datenbank mit Benutzer, Kennwort)
"jdbc:odbc:Zugriff-MYSQL",”ss11etxyz”,”geheim”

Auch hier sind wieder die spezifischen Angaben (Benutzername, Kennwort, MySQL-Datenbank, Servername, SID und Datenquellenname) für den jeweiligen Teilnehmer anzupassen.

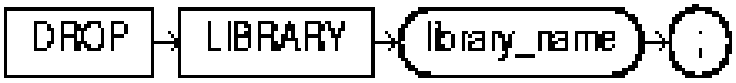
Anhang

A1 Syntaxdiagramme SQL

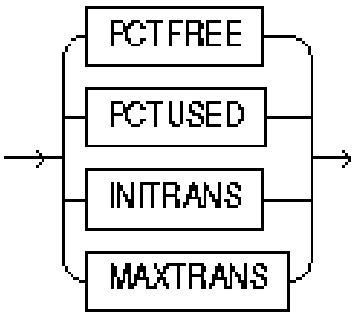
Die Syntax der einzelnen Elemente der Sprache kann in Form von sogenannten Syntaxdiagrammen gegeben werden. Syntaxdiagramme sind eine graphische Darstellung zur Illustration der gültiger SQL-Syntax. Die Syntaxdiagramme werden dabei von links nach rechts gelesen, die Richtung wird dabei durch kleine Pfeile angegeben

- Befehle und andere Schlüsselwörter der Sprache sind in Großbuchstaben innerhalb von Rechtecken notiert, sie sind genauso zu übernehmen, können allerdings dabei auch in Kleinbuchstaben geschrieben werden.
- Parameter dagegen erscheinen in Kleinbuchstaben in Softboxen (Rechecke mit Halbkreisen als rechter bzw. linker Abschluß) und müssen wie formale Parameter durch entsprechende Zeichenkonstanten ersetzt werden oder sie sind durch ein weiteres Syntaxdiagramm illustriert.
- Operatoren, Trennzeichen und sonstige Sonderzeichen erscheinen in Kreisen und müssen genauso übernommen werden.

Erforderliche Schlüsselwörter können einzeln oder als Alternative in einer vertialen Liste erscheinen. Einzelne, erforderliche Schlüsselwörter und Parameter erscheinen auf der „Hauptlinie“, d.h. direkt auf der horizontalen Hauptlinie, der man zur Ableitung der Syntax gerade folgt.

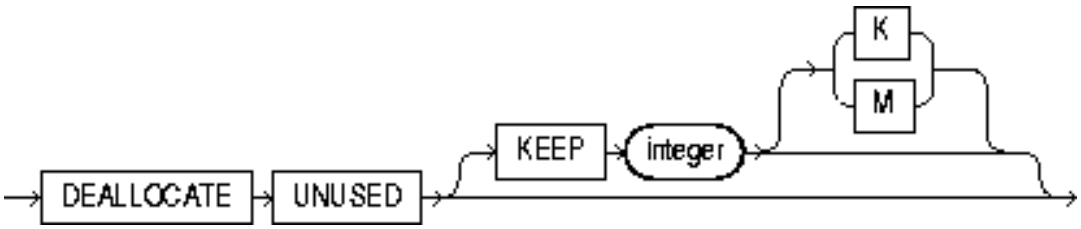


Der Befehl zum Löschen einer Bibliothek besteht demnach aus den erforderlichen Schlüsselwörtern DROP und LIBRARY (in dieser Reihenfolge) gefolgt von einem erforderlichen Parameter für den aktuellen Namen der zu löschenden Bibliothek und wird vom Semikolon abgeschlossen.
Wenn ein Syntaxdiagramm mehr als einen Weg beschreibt, so kann man allen möglichen Wegen folgen, um zu gültiger SQL-Syntax zu gelangen. Wenn man dabei die Wahlmöglichkeit zwischen einem/mehreren Schlüsselwörtern, einem/mehreren Operatoren oder einem/mehreren Parametern hat, so erscheinen die Auswahlmöglichkeiten in einer vertikalen Liste.



An der betreffenden Stelle muss genau eines der vier angegebenen Schlüsselwörter der vertikalen Liste eingesetzt werden.

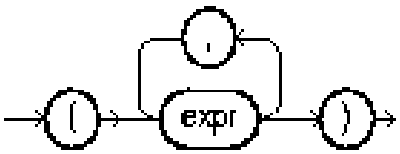
Optionale Schlüsselwörter und Parameter erscheinen in vertikalen Listen oberhalb der Hauptlinie, d.h. man kann der Hauptlinie weiter folgen, oder einer parallelen Nebenlinie mit den optionalen Schlüsselwörter oder Parameter folgen:



Aus diesem Diagramm ergeben sich folgende syntaktisch richtigen Möglichkeiten:

DEALLOCATE UNUSED
DEALLOCATE UNUSED KEEP 10000
DEALLOCATE UNUSED KEEP 100K
DEALLOCATE UNUSED KEEP 4M

d.h. es ist ein Variante ohne Größenangabe, sowie jeweils eine mit Größenangabe in Bytes, in Kilobytes oder in Megabytes möglich.
Die Syntaxdiagramme können auch Schleifen enthalten. Dadurch wird die Möglichkeit geboten, bestimmte Teile des Syntaxdiagramms beliebig oft zu wiederholen.



In diesem Diagramm ist eine Ausdrucksliste dargestellt, eine solche Liste beginnt stets mit einer normalen öffnenden Klammer, endet stets mit einer normalen schließenden Klammer und muss mindestens einen Ausdruck (englisch:expression) enthalten, kann aber auch mehrere Ausdrücke hintereinander, jeweils durch ein Komma getrennt enthalten. Sind

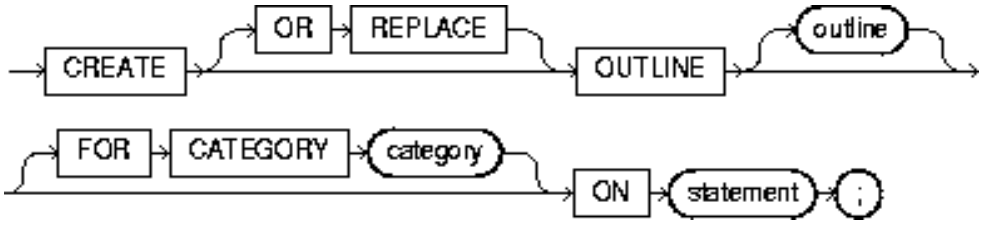
a + b 3.9 * c NULL

jeweils syntaktisch gültige Ausdrücke, so wäre nach demnach auch

(a + b , 3.9 *c , NULL)

nach diesem Diagramm ein syntaktisch gültiger Ausdruck.

Wenn ein komplexes Syntaxdiagramm nicht in einer Zeile Platz findet, so wird die Hauptlinie des Diagramms an geeigneter Stelle aufgetrennt und auf zwei oder mehr Zeilen verteilt. Die Hauptlinie endet dann nicht am Ende einer Zeile, sondern wird am Anfang der nächsten Zeile fortgesetzt, dies ist jeweils durch einen kleine Pfeilspitze am Ende der Zeile notiert.



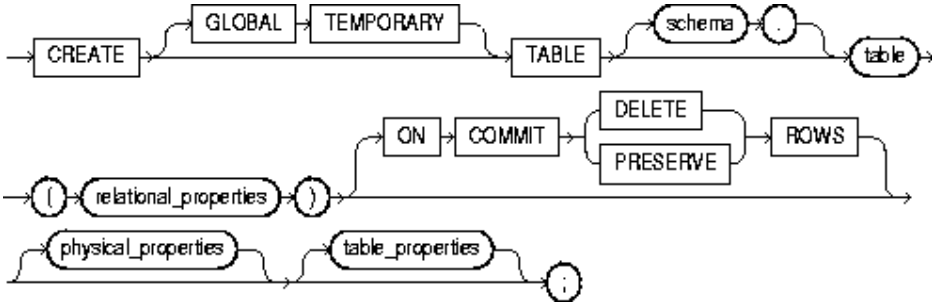
Die Namen für Datenbankbezeichner, wie Tabellen, Spalten oder auch Benutzer dürfen höchstens 30 Stellen lang sein, müssen mit einem Buchstaben beginnen und dürfen im Rest aus einer beliebigen Kombination von Buchstaben, Ziffern sowie den Sonderzeichen $ # _ bestehen. Wird ein Datenbankbezeichner in Hochkomma "....." eingeschlossen, so kann er aus einer beliebigen Kombination aus Buchstaben, Ziffern und Sonderzeichen (außer dem Hochkomma) bestehen.
Die allgemeine Syntax der Kommandos ist oft sehr umfangreich und damit komplex und enthält oft auch noch Komponenten oder Varianten, die im Rahmen dieser Lehrveranstaltung nicht behandelt werden. Im folgenden sind die kompletten Syntaxdiagramme angegeben, die Teile die nicht weiter verfolgt werden, sind genannt und können ignoriert werden.

A1.1 Überblick DDL-Befehle

CREATE TABLE
DROP TABLE
ALTER TABLE

Die Arbeit mit den Syntaxdiagrammen wird nacheinander CREATE-TABLE-Befehl, DROP-TABLE-Befehl und ALTER-TABLE-Befehl dargestellt und erläutert:

A1.1.1 CREATE-TABLE-Befehl

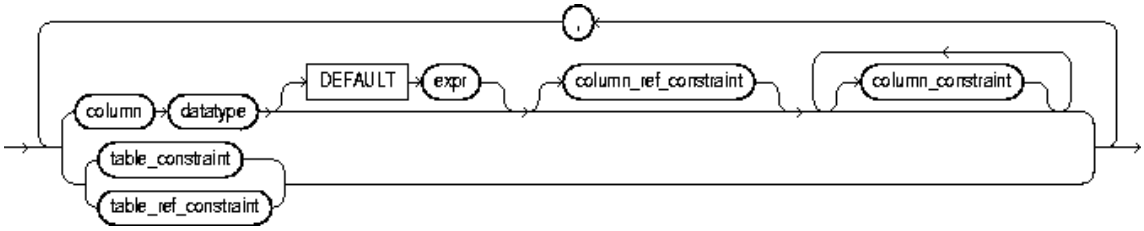


Die Schlüsselwörter GLOBAL, TEMPORARY können hier ebenso ignoriert werden wie die Angaben ON COMMIT DELETE ROWS bzw. ON COMMIT PRESERVE ROWS und die physical_properties.

schema
: steht dabei für den Benutzernamen (z.B. studetpm) , in dessen Schema die Tabelle angelegt werden soll.

table
: steht dabei für den Namen der Tabelle (z.B. liefert), die angelegt werden soll.

relational_properties



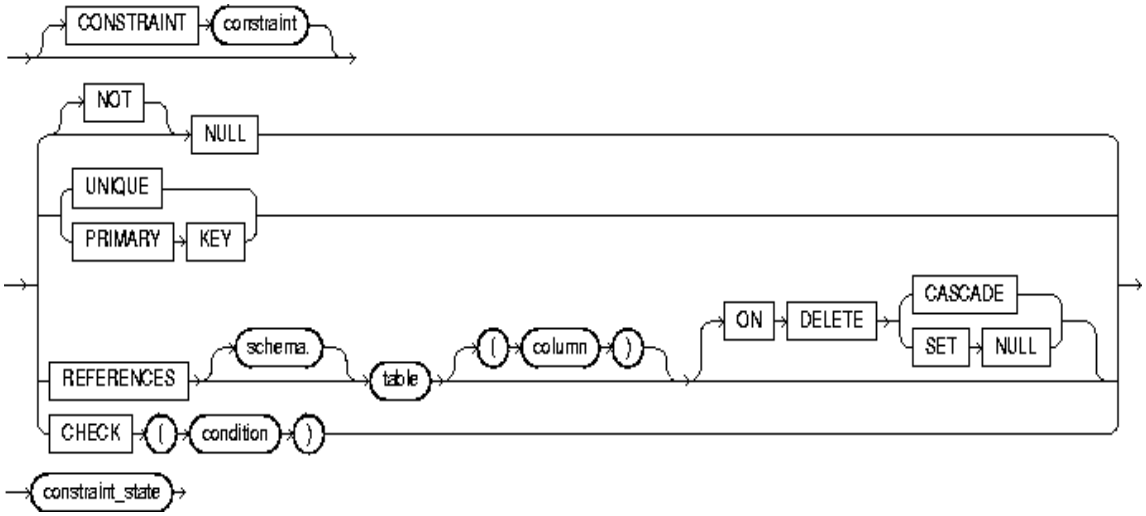
Die Parameter column_ref_constraint und table_ref_constraint können hier ignoriert werden.

column
: steht dabei für den Namen der Spalte (z.B. Preis)

datatype
: steht dabei für den Datentyp (z.B. CHAR(2), VARCHAR2(6), NUMBER(5,2))

expr
: steht dabei für einen gültigen Ausdruck

column_constraint



Der Parameter constraint_state kann hier ignoriert werden.

constraint
: steht dabei für einen schemaweit eindeutigen Namen für die Bedingung. Dieser Name wird bei Verletzung der Bedingung als Referenz genannt

schema
: steht dabei für den Benutzernamen (z.B. studetab), in dessen Schema sich die referenzierte Tabelle befindet

table
: steht dabei für den Namen der Tabelle (z.B. Lieferant), die in einer Fremdschlüsselbeziehung referenziert wird

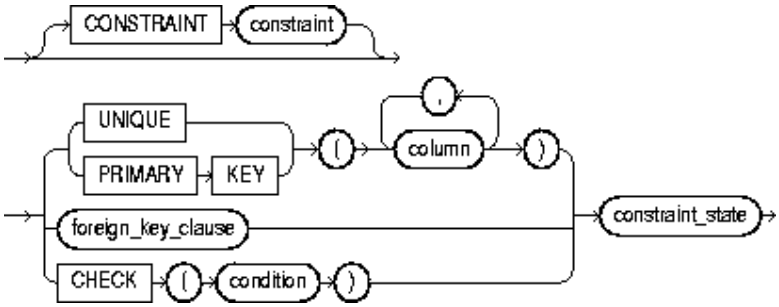
column
: steht dabei für den Namen der Spalte (z.B. LNR), die in einer Fremdschlüsselbeziehung referenziert wird.

condition
: Steht hier für eine einfache Bedingung (z.B. Status IN (10, 20, 30)), die für die betreffende Spalte überwacht werden soll.

ON DELETE CASCADE
: Steht hier für die Möglichkeiten, beim Löschen entlang der Fremdschlüsselbeziehungen die abhängigen Zeilen ebenfalls zu löschen.

ON DELETE SET NULL
: Steht hier für die Möglichkeiten, beim Löschen in den abhängigen Zeilen entlang der Fremdschlüsselbeziehungen den Bezug auf den speziellen Wert NULL zu setzen.

Table_constraint



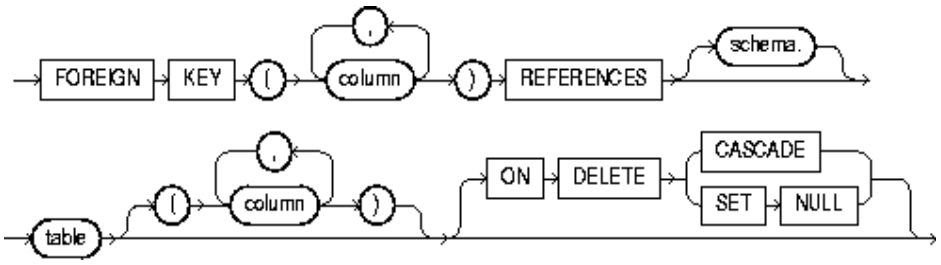
Der Parameter constraint_state kann hier ignoriert werden.

constraint
: steht dabei für einen schemaweit eindeutigen Namen für die Bedingung. Dieser Name wird bei Verletzung der Bedingung als Referenz genannt

column
: steht dabei in der Liste für den Namen der Spalte (z.B. LNR) der aktuellen Tabelle, auf die sich die Angabe UNIQUE bzw. PRIMARY KEY beziehen.

condition
: Steht hier für eine einfache Bedingung (z.B. Status IN (10, 20, 30)), die für die betreffende Spalte überwacht werden soll.

foreign_key_clause



column
: steht dabei in der Liste für den Namen der Spalte (z.B. LNR) in der aktuellen Tabelle, für die eine Fremdschlüsselbeziehung festgelegt wird

schema
: steht dabei für den Benutzernamen (z.B. studetab), in dessen Schema sich die referenzierte Tabelle befindet

table
: steht dabei für den Namen der Tabelle (z.B. Lieferant), die in einer Fremdschlüsselbeziehung referenziert wird

column
: steht dabei in der Liste für den Namen der Spalte (z.B. LNR) in der referenzierten Tabelle, auf die in der Fremdschlüsselbeziehung Bezug genommen wird.

ON DELETE CASCADE
: Steht hier für die Möglichkeiten, beim Löschen entlang der Fremdschlüsselbeziehungen die abhängigen Zeilen ebenfalls zu löschen.

ON DELETE SET NULL
: Steht hier für die Möglichkeiten, beim Löschen in den abhängigen Zeilen entlang der Fremdschlüsselbeziehungen den Bezug auf den speziellen Wert NULL zu setzen.

Beispiel:

Tabelle: **Hersteller (Primärschlüssel: HNR)**

HNR	Name
1	Badendorf
2	Gondi
3	Hankel
4	KMEX
9	Baff
10	Plendax
12	Boyer

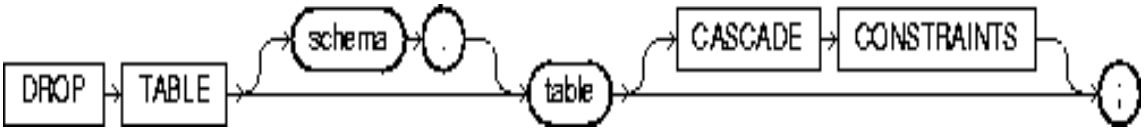
Tabelle: **Erzeugnis (Primärschlüssel: ANR, Fremdschlüssel HNR, Referenz auf Hersteller)**

ANR	Bezeichnung	Preis	HNR
8	Parfüm	3.40	2
9	Rasierwasser	5.30	3
11	Zahncreme	1.20	10
13	Haarspray	7.40	1
15	Haarcreme	1.50	4
36	Shampoo	6.20	12
37	Shampoo	5.50	9

```
CREATE TABLE Hersteller
(
    HNR  NUMBER(2)  CONSTRAINT  Hersteller_PK  PRIMARY KEY,
    Name  VARCHAR2(20)
);

CREATE TABLE Erzeugnis
(
    ANR             NUMBER(2),
    Bezeichnung     VARCHAR2(20),
    Preis           NUMBER(5,2),
    HNR             NUMBER(2),
    CONSTRAINT  Erzeugnis_PK  PRIMARY KEY (ANR),
    CONSTRAINT  Erzeugnis_Hersteller_FK FOREIGN KEY ( HNR )
                REFERENCES Hersteller
);
```

A1.1.2 DROP-TABLE-Befehl

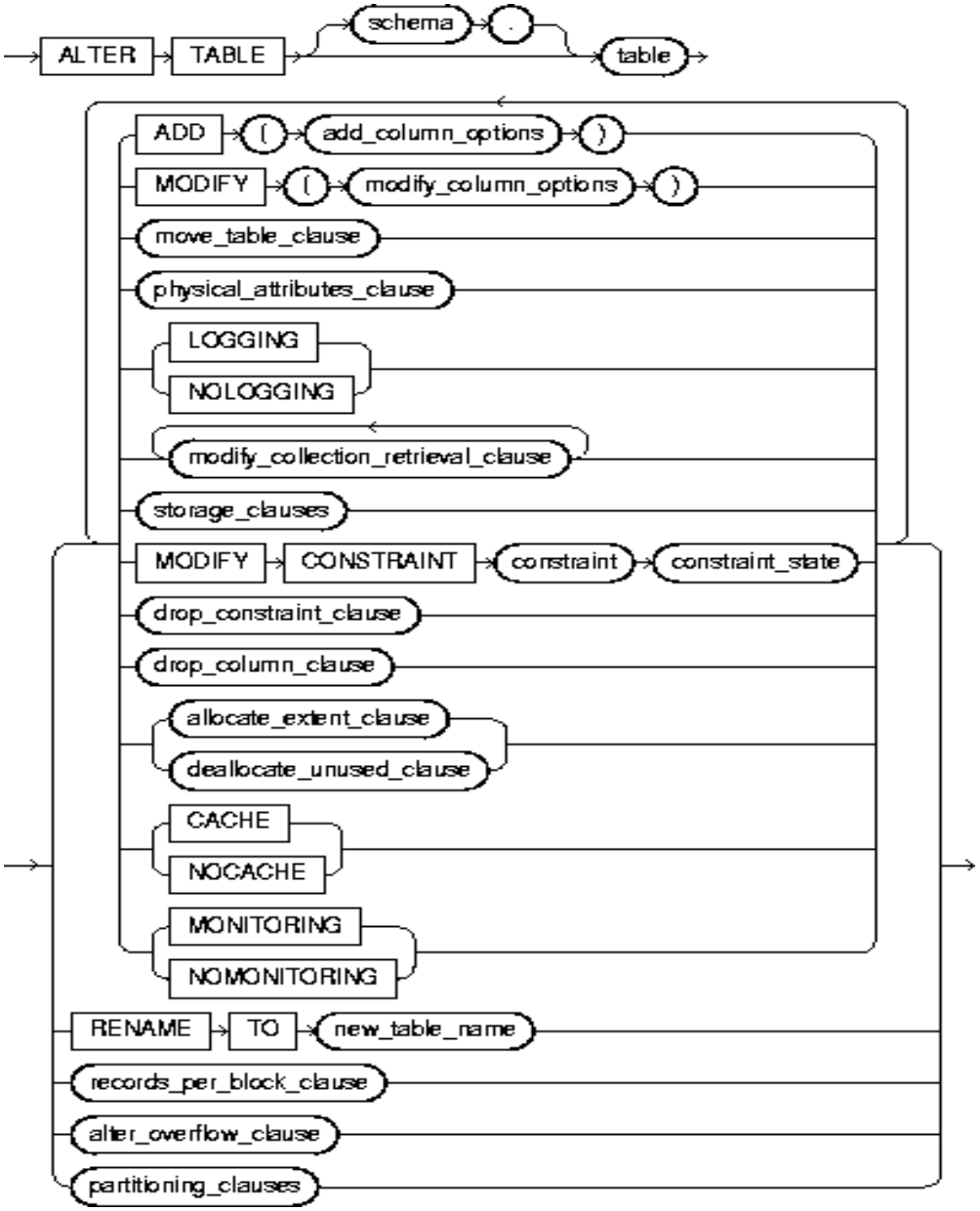


Die Schlüsselwörter CASCADE CONSTRAINTS können hier ignoriert werden.

schema
: steht dabei für den Benutzernamen (z.B. studetpm), in dessen Schema die Tabelle gelöscht werden soll.

table
: steht dabei für den Namen der Tabelle (z.B. liefert), die gelöscht werden soll

A1.1.3 ALTER-TABLE-Befehl



schema
steht dabei für den Benutzernamen (z.B. studetpm), in dessen Schema die Tabelle geändert werden soll.
table
steht dabei für den Namen der Tabelle (z.B. liefert), die geändert werden soll
ADD (add_column_options)
wird zum hinzufügen weiterer Spalten benutzt
MODIFY (modify_column_options)
wird zum ändern vorhandener Spalten benutzt
drop column_clause
wird zum löschen einer vorhandenen Spalte benutzt
MODIFY CONSTRAINT constraint constraint_state
wird zum aktivieren bzw. deaktivieren einer vorhandenen Bedingung benutzt
RENAME TO new_table_name
wird zum umbenennen der Tabelle benutzt
drop_constraint_clause
wird zum löschen einer vorhandenen Tabellen-Bedingung benutzt

Die restlichen Angaben bieten sehr umfangreiche Möglichkeiten, Einfluß auf die physische Speicherung der Tabelle zu nehmen und können hier ignoriert werden. Auf die Details soll hier verzichtet werden und nur einige Möglichkeiten an Beispielen dargestellt werden.

Beispiele: (Tabellen aus Beispiel in 3.1.1)

```
ALTER TABLE  Erzeugnis  MODIFY
(
    Preis           NUMBER(6,2)     DEFAULT  1.00
                    CONSTRAINT   Preis_positiv  CHECK (Preis > 0.00)
);

ALTER  TABLE  Hersteller  MODIFY   Erzeugnis_Hersteller_FK   DISABLE;

ALTER  TABLE  Hersteller  MODIFY   Erzeugnis_Hersteller_FK   ENABLE;

ALTER  TABLE  Hersteller  RENAME TO  Produzent;
```

A1.2 Überblick DML-Befehle

INSERT INTO .....
DELETE FROM ..... WHERE .....
UPDATE ..... WHERE .....

Wir verzichten hier auf die Wiedergabe der Syntaxdiagramme und geben eine vereinfachte Darstellung der Syntax. Auf die vollständigen Syntaxdiagramme für diese Befehle sei verwiesen.

A1.2.1 INSERT-Befehl

Hier wird zunächst nur der einfache Fall betrachtet, dass die einzugebenden Zeilen als Werteliste gegeben sind:

INSERT INTO table (<Liste der Spalten>) VALUES (<Liste der Werte>);

Jede einzelne Befehl fügt genau eine Zeile in die betreffende Tabelle **table** ein.

<Liste der Spalten>
ist dabei eine durch Komma getrennte Aufzählung der Spalten(namen)

<Liste der Werte>
ist dabei eine durch Komma getrennt Aufzählung konstanter Werte. Die Werteliste muss bezgl. Anzahl, Reihenfolge und Datentyp zur <Liste der Spalten> passen, d.h. in der einzufügenden Zeile wird der erste Wert der < Liste der Werte> in die erste Spalte der <Liste der Spalten> übernommen, der zweite Wert in die zweite Spalte usw.
Entsprechend dem Datentyp der Spalte muss sich an der entsprechenden Position in der <Liste der Werte> ein numerischer Wert (z.B. eine Konstante 5.2) oder eine Zeichenkette (z.B. 'Müller') befinden, die Zeichenkette muss dabei in einfache Hochkomma eingeschlossen sein

Die <Liste der Spalten> kann weggelassen werden, dann werden implizit alle Spalten in der Reihenfolge genommen, in die sie beim CREATE-TABLE-Befehl erstellt worden sind.
Ist für eine Spalte ein DEFAULT-Wert angegeben oder ist NULL ein zulässiger Wert, kann beim INSERT-Befehl auf diese Spalte (und ihren Wert) verzichtet werden, für sie wird dann automatisch der DEFAULT-Wert oder NULL übernommen.

Beispiele: (Tabellen aus Beispiel in 5.1)

```
INSERT INTO Lieferant (LNR, Status, Ort, Name)
            VALUES    (10 , 10 , 'Apolda' , 'Schulze');

INSERT INTO Lieferant VALUES (20 , 'Schneider' , 'Weimar' , 10 );
```

A1.2.2 DELETE-Befehl

Mit einem DELETE-Befehl können wahlweise alle oder ein Teil der Zeilen gelöscht werden.

DELETE FROM table;

DELETE FROM table WHERE condition;

Die erste Variante löscht alle Zeilen der Tabelle **table**, die zweite Variante löscht nur die Zeilen in der Tabelle **table**, die die angegebene Bedingung **condition** erfüllen.

Beispiele: (Tabellen aus Beispiel in 5.1)

DELETE FROM Lieferant;

DELETE FROM liefert where ANR IN (‘R1‘,‘R2‘);

A1.2.3 UPDATE-Befehl

Mit einem UPDATE-Befehl können eine oder mehrere Spalten für alle oder einen Teil der Zeilen einer Tabelle geändert werden. Analog zum INSERT-Befehl wird hier zunächst nur der einfache Fall betrachtet, dass die die zu ändernden Spalten einzeln auf neue Werte gesetzt werden. Für jede zu ändernde Spalte wird in einer Art Zuweisung (SET_Befehl) der neue Wert ermittelt und zugewiesen, dabei kann auf den Wert vor der Änderung zurückgegriffen werden, z.B. werden mit

SET Preis = Preis * 1.05

die Werte in der Spalte Preis geändert und zwar um 5 % erhöht. Die Angabe Preis auf der rechten Seite des Gleichheitszeichens greift auf die Werte in Preis vor der Änderung zu, während die Angabe Preis auf der linken Seite des Gleichheitszeichen für den Wert nach der Änderung steht. Alle Änderungen werden in einer SET-Clause zusammengefaßt.

SET-Clause

SET column1 = expr1, column2 = expr2, column3 = expr4, ...

Analog zum DELETE-Befehl können auch die zu ändernden Zeilen durch eine Bedingung eingeschränkt werden:

UPDATE table SET-Clause;

UPDATE table SET-Clause WHERE condition;

Die erste Variante ändert alle Zeilen der Tabelle **table**, die zweite Variante ändert nur die Zeilen in der Tabelle **table**, die die angegebene Bedingung **condition** erfüllen.

Beispiele: (Tabellen aus Beispiel in 5.1)

UPDATE Artikel SET LAENGE = 90, BREITE = 90 WHERE ANR = ‘T2‘;

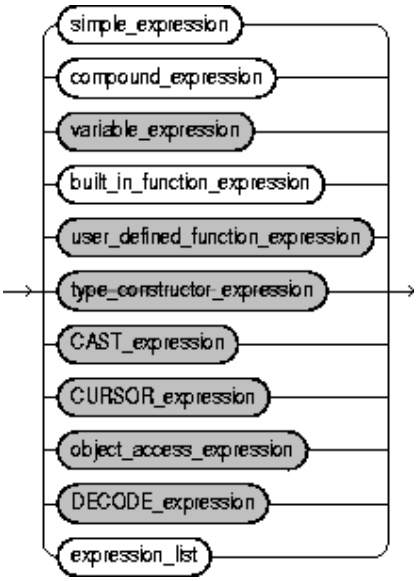
A1.3 Select-Befehl

Mit dem Select-Befehl können externe Sichten (englisch: views) definiert werden. Der Select-Befehl erstellt unter Verwendung der Standardoperationen (Projektion, Selektion und Verbund) eine Teilabfrage (englisch: subquery), die als Ergebnis eine Tabelle (Relation) liefert. Diese Tabelle ist aber nicht statisch gespeichert, sondern wird bei jedem Zugriff dynamisch aus den verbundenen Tabellen erzeugt, deshalb nennt man sie eine Sicht (view).
Zur Beschreibung der Spalten einer Sicht werden Ausdrücke (englisch: expressions) verwendet, ebenso wie für die Formulierung von Bedingungen (englisch: conditions) bei der Selektion.

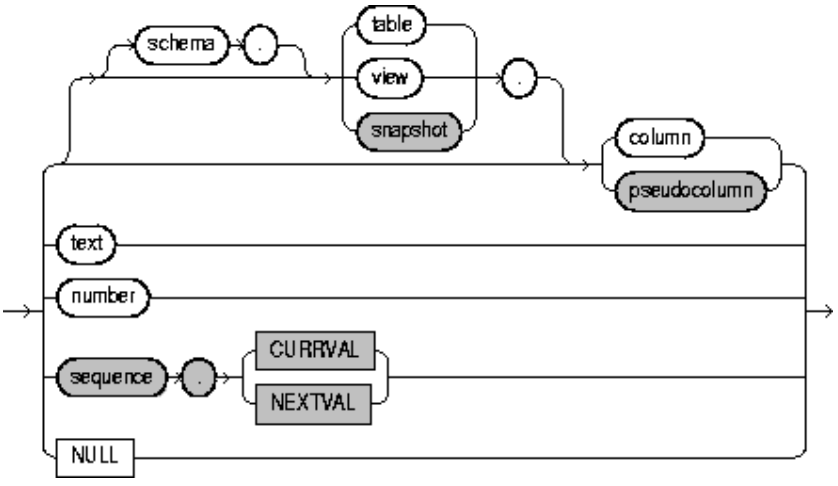
A1.3.1 Ausdrücke und Bedingungen

In den folgenden Syntax-Diagrammen sind die Teile, die im Rahmen der Vorlesung nicht behandelt werden, grau unterlegt und können ignoriert werden.

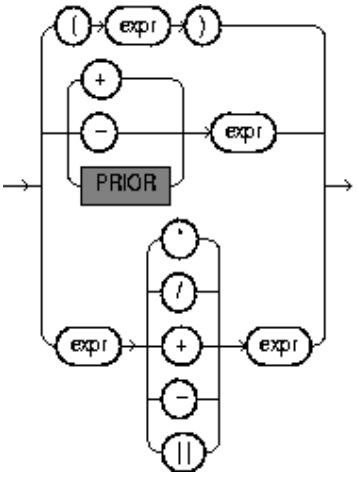
Syntax-Diadramm: **expr::=**



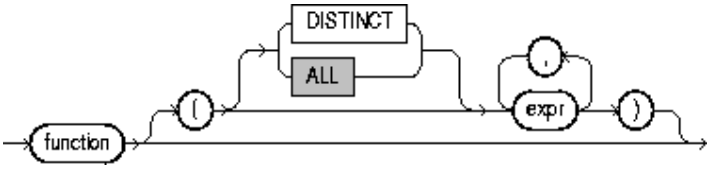
Syntax-Diadramm: **simple_expression::=**



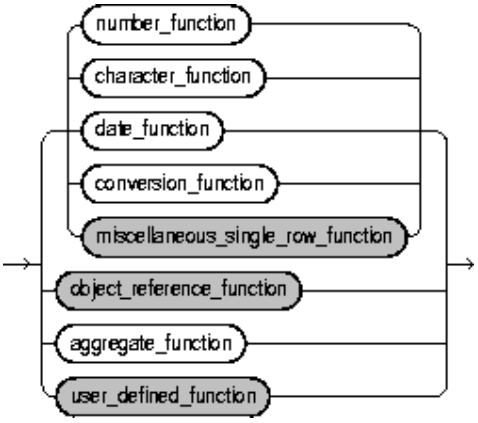
Syntax-Diadramm: **compound_expression::=**



Syntax-Diadramm: **build_in_function_expression::=**



Syntax-Diadramm: **function::=**



Beispiele für numerische Funktionen (number_function):

COSH	CEIL	MOD	SQRT
ABS	COS	POWER	TAN
ACOS	EXP	ROUND (Number Function)	TANH
ADD_MONTHS	FLOOR	SIGN	TRUNC (Number Function)
ATAN	LN	SIN	
ATAN2	LOG	SINH	

Beispiele für Zeichenketten-Funktionen (character_function):

CHR	NLS_INITCAP	RTRIM	UPPER
CONCAT	NLS_LOWER	SOUNDEX	ASCII
INITCAP	NLSSORT	SUBSTR	INSTR
LOWER	NLS_UPPER	SUBSTRB	INSTRB
LPAD	REPLACE	TRANSLATE	LENGTH
LTRIM	RPAD	TRIM	LENGTHB

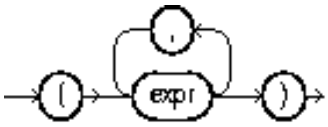
Beispiele für Datum-/Konvertierungs-Funktionen (date_function/conversion_function):

ADD_MONTHS	SYSDATE	CHARTOROWID	TO_DATE
LAST_DAY	TRUNC	CONVERT	TO_LOB
MONTHS_BETWEEN		HEXTORAW	TO_MULTI_BYTE
NEW_TIME		ROWIDTOCHAR	TO_NUMBER
NEXT_DAY		TO_CHAR (date conversion)	TO_SINGLE_BYTE
ROUND (Date Function)		TO_CHAR (number conversion)	TRANSLATE ... USING

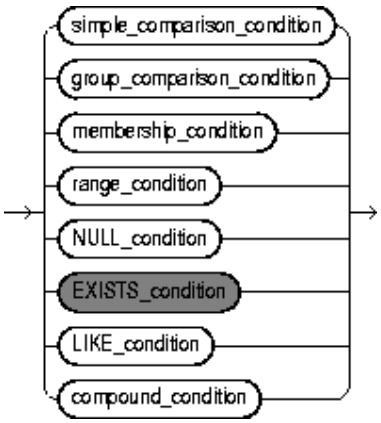
Beispiele für Aggregate-Funktionen (aggregate_function):

AVG	GROUPING	MIN	SUM
COUNT	MAX	STDDEV	VARIANCE

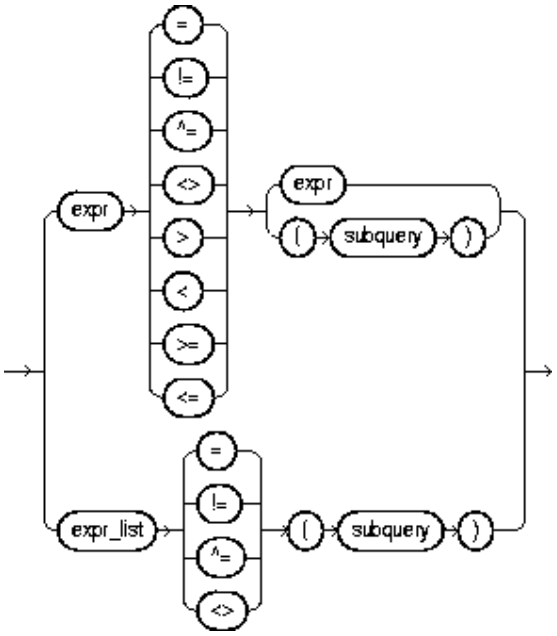
Syntax-Diadramm: **expression_list::=**



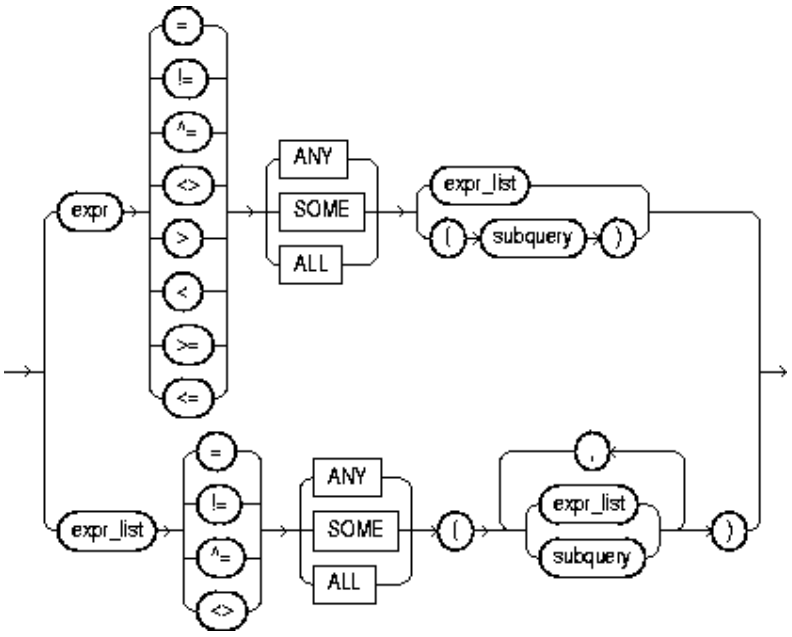
Syntax-Diadramm: **condition::=**



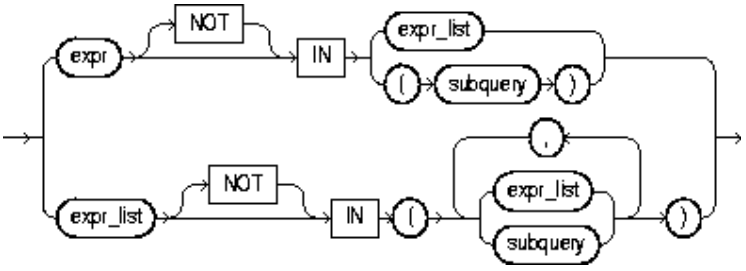
Syntax-Diadramm: **simple_comparison_condition::=**



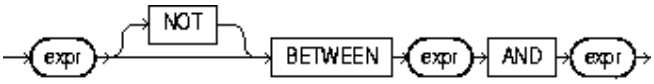
Syntax-Diadramm: **group_comparison_condition::=**



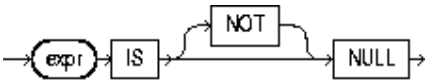
Syntax-Diadramm: **membership_condition::=**



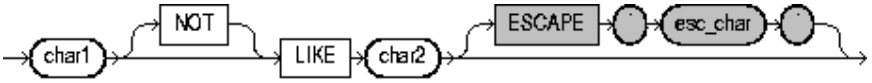
Syntax-Diadramm: **range_condition::=**



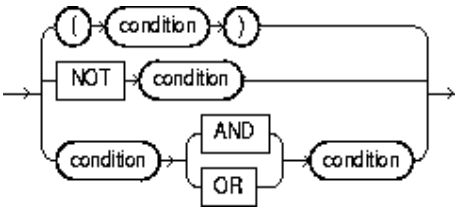
Syntax-Diadramm: **null_condition::=**



Syntax-Diadramm: **like_condition::=**



Syntax-Diadramm: **compound_condition::=**



Beispiele:

Ausdrücke:

((studetxy.erzeugnis.preis * 100) * globals.faktor)

(Laenge, 'Laenge', 100, NULL)

CONCAT(Vorname, Nachname)

TO_CHAR(Sysdate,'DD.MM.YYYY')

COUNT(HNR)

Bedingungen:

Artikel.Laenge > Artikel.Breite

Wert IN (10, 20, 30)

Status IS NULL

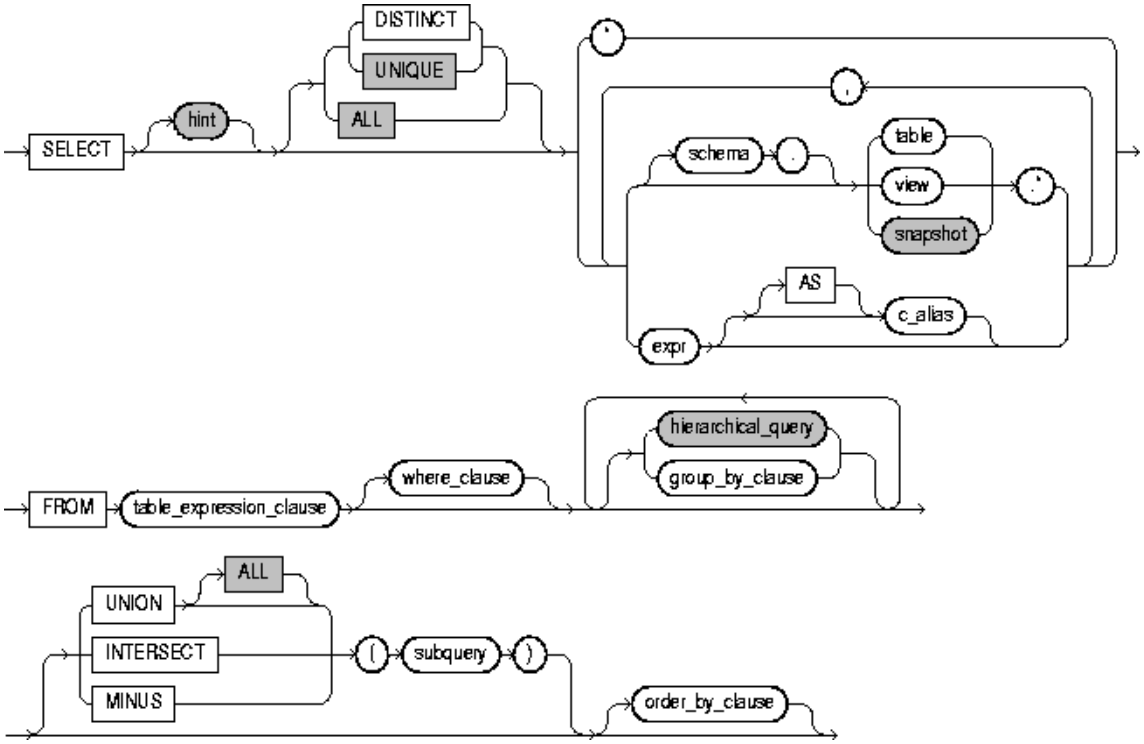
Wert BETWEEN (Laenge – 10) AND (Laenge + 10)

Status > 10 OR UPPER(Name) LIKE '%FH-JENA%'

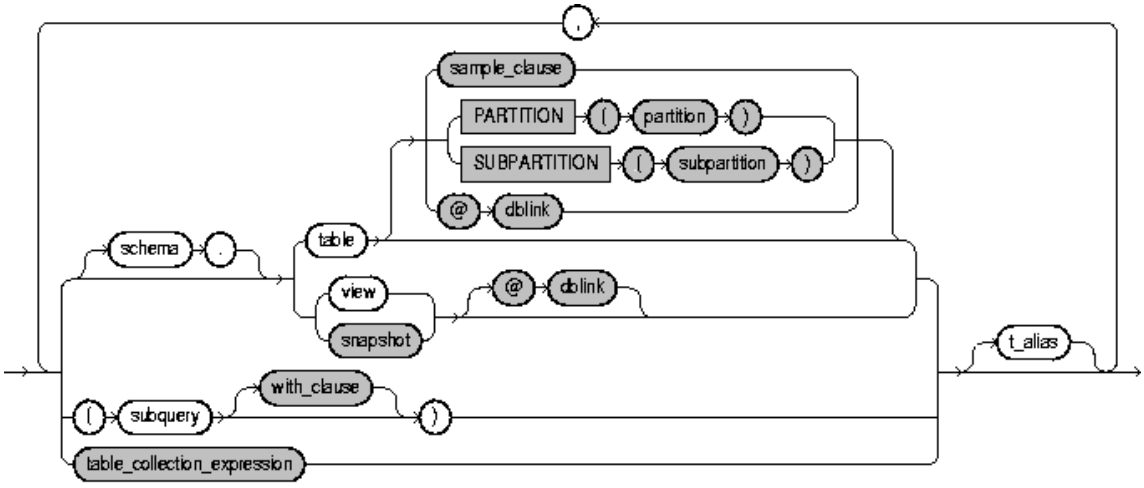
A1.3.2 Teilabfragen mit select

In den folgenden Syntax-Diagrammen sind die Teile, die im Rahmen der Vorlesung nicht behandelt werden, wieder grau unterlegt und können ignoriert werden.

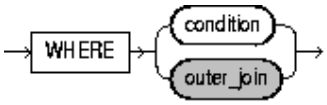
Syntax-Diadramm: **subquery::=**



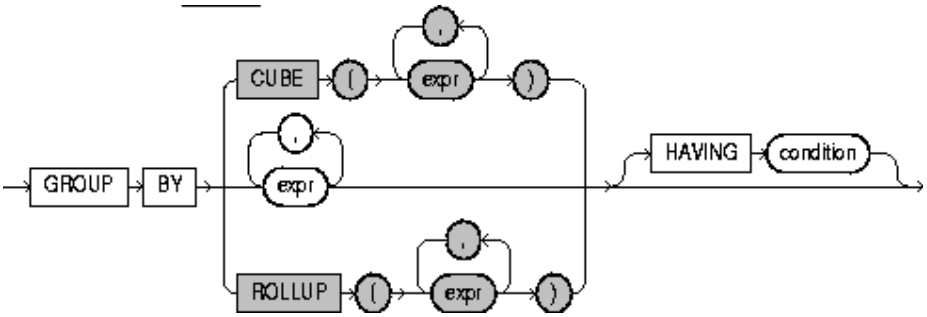
Syntax-Diadramm: **table_expession_clause::=**



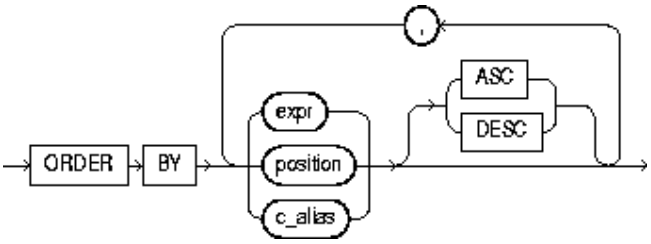
Syntax-Diadramm: **where_clause::=**



Syntax-Diadramm: **group_by_clause::=**



Syntax-Diadramm: **order_by_clause::=**



Beispiele:

Tabelle: **Hersteller (Primärschlüssel: HNR)**

HNR	Name
1	Badendorf
2	Gondi
3	Hankel
4	KMEX
9	Baff
10	Plendax
12	Boyer

Tabelle: **Erzeugnis (Primärschlüssel: ANR, Fremdschlüssel HNR, Referenz auf Hersteller)**

ANR	Bezeichnung	Preis	HNR
8	Parfüm	3.40	2
9	Rasierwasser	5.30	3
11	Zahncreme	1.20	10
13	Haarspray	7.40	1
15	Haarcreme	1.50	4
36	Shampoo	6.20	12
37	Shampoo	5.50	9

Abfrage:

```
SELECT Erzeugnis.*, Name
FROM   Erzeugnis, Hersteller
WHERE Erzeugnis.HNR = Hersteller.HNR
   AND  Preis < 5.00;
```

Ergebnis:

ANR	Bezeichnung	Preis	HNR	Name
8	Parfüm	3.40	2	Gondi
11	Zahncreme	1.20	10	Plendax
15	Haarcreme	1.50	4	KMEX

Abfrage:

```
SELECT Name
FROM   Hersteller
WHERE HNR  IN (    SELECT     HNR
                              FROM       Erzeugnis
                              WHERE      Bezeichnung='Shampoo' );
```

Ergebnis:

Name
Baff
Boyer

Abfrage:

```
SELECT      Bezeichnung, COUNT(HNR) ANZ_HS, AVG(Preis) AVG_Preis
FROM        Erzeugnis
GROUP  BY   Bezeichnung
HAVING      COUNT(HNR)  > 1;
```

Ergebnis:

Bezeichnung	ANZ_HS	AVG_Preis
Shampoo	2	5.85

Abfrage:

```
SELECT      Produzent.Name  Hersteller
FROM        (Select * from studetxy.Hersteller)  Produzent
WHERE       Produzent.HNR > 9;
```

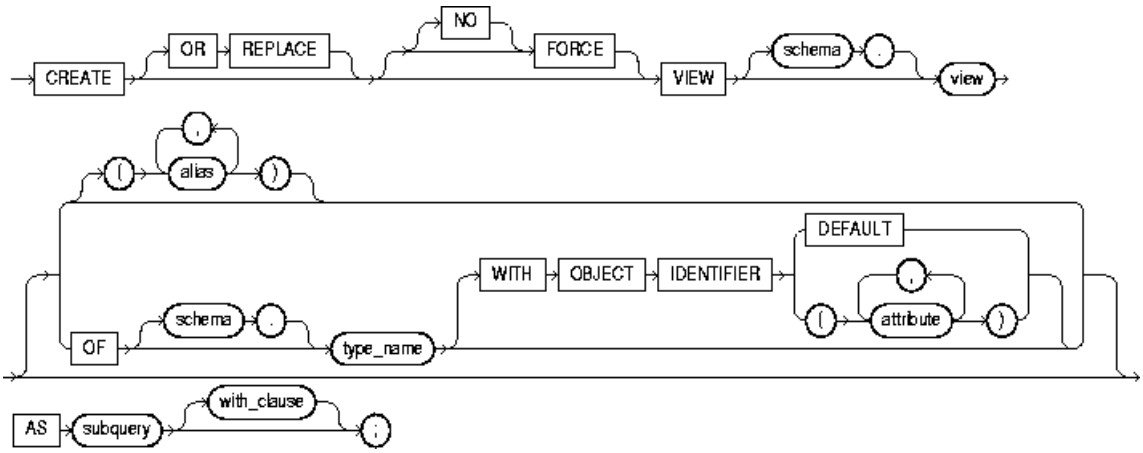
Ergebnis:

Hersteller
Plendax
Boyer

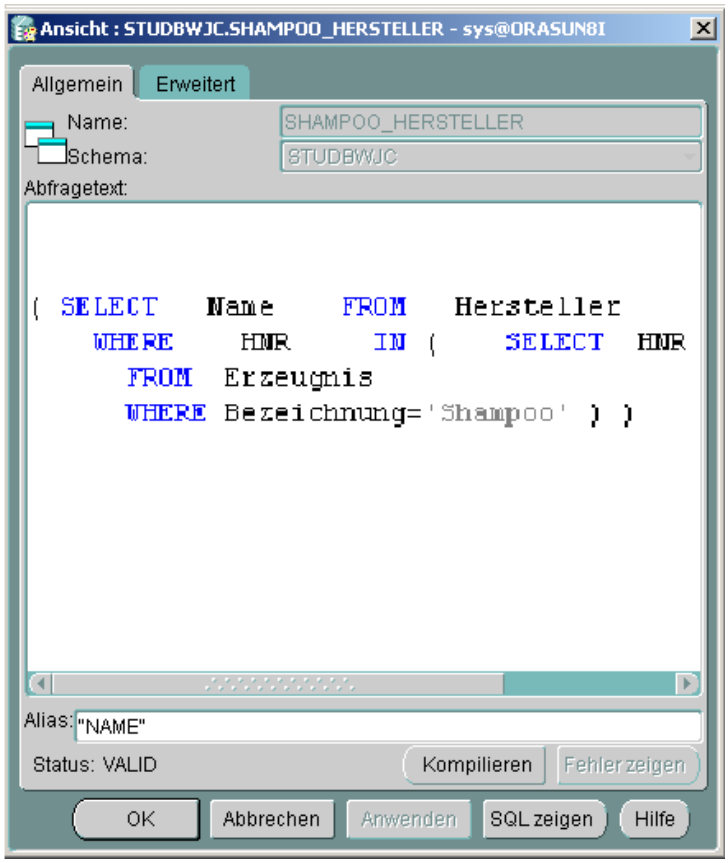
A1.3.4 CREATE VIEW-Befehl

Analog zum SQL-Befehl zum erstellen einer statischen Tabelle (CREATE TABLE ....) gibt es auch einen entsprechenden SQL-Befehl um eine dynamische Sicht zu erstellen, in diesem Fall wird aber nicht die Ergebnistabelle gespeichert, sondern der zugehörige Select-Befehl, d.h. mit dem CREATE-VIEW-Befehl wird eine benannte Subquery erstellt, diese kann danach dann analog zu (Basis)Tabellen direkt über den namen angesprochen werden und z.B. in der from Klausel des Select-Befehls verwendet werden.

Syntax-Diadramm: **create_view::=**



Beispiel:



SQL-Syntax:

```
CREATE OR REPLACE   Shampoo_Hersteller AS
(  SELECT   Name    FROM   Hersteller
   WHERE    HNR     IN (    SELECT  HNR
                            FROM  Erzeugnis
                            WHERE Bezeichnung='Shampoo' ) );
```

A2 COM (ADO.NET)

A2.1 Grundlagen

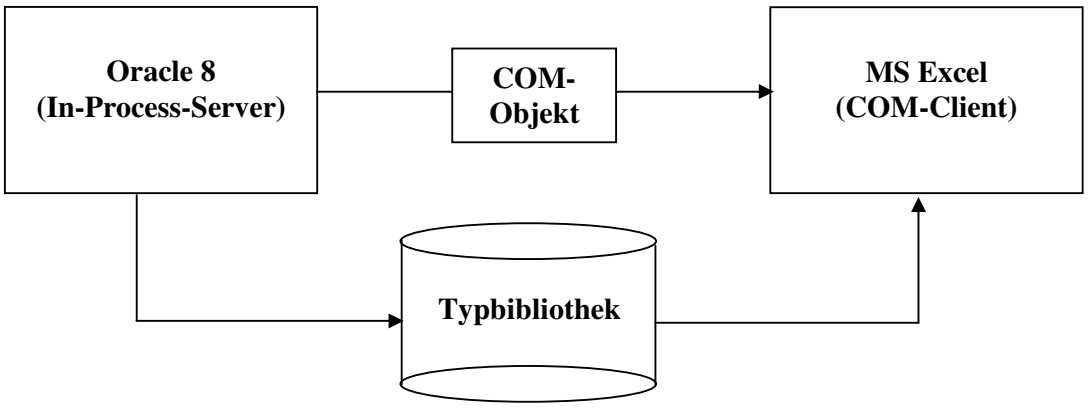
Eine Alternative zu ODBC ist die Nutzung einer anderen genormten Schnittstelle COM, sofern der betreffende Datenbankserver diese Technologie unterstützt.

COM-Server:

Unter Automatisierung wird hier die Fähigkeit einer Anwendung verstanden, Objekte in einer anderen Anwendung über die Programmierung zu steuern, in diesem Sinne sind COM-Server Objekte, die zur Laufzeit programmiert werden können, d.h. ein COM-Server ist eine Anwendung oder eine Bibliothek und besteht aus mehreren COM-Objekten, jedes COM-Objekt stellt dabei eine Gruppe von Eigenschaften und Methoden dar. Über die COM-Objekte stellt der COM-Server einer Client-Anwendung oder einer Client-Bibliothek Dienste zur Verfügung. Die vom COM-Server angebotenen COM-Objekte sind mit ihren Eigenschaften und Methoden in einer Typbibliothek zusammengefasst.

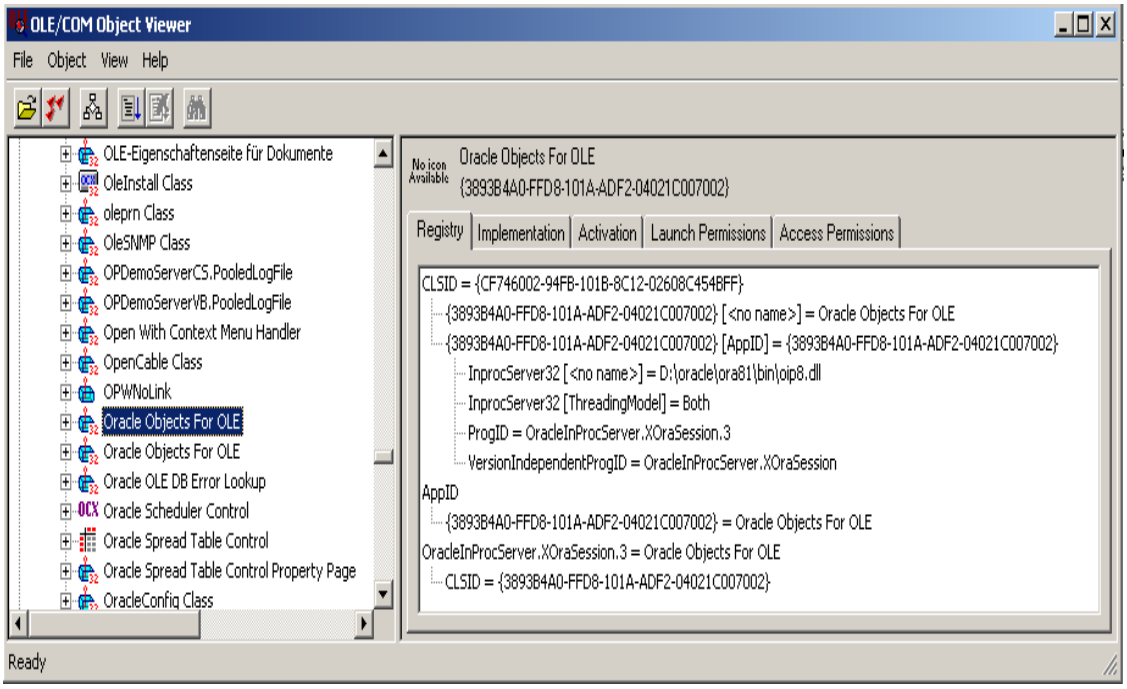
COM-Client:

Eine Client Anwendung, die die COM-Technologie unterstützt, die also die von COM-Servern angebotenen COM-Objekten nutzen kann.



Über diese Schnittstelle kann nun direkt aus MS-Excel heraus z.B. auf einen Oracle Datenbank-Server zugegriffen werden. Dieselbe Technik kann auch z.B. mit MS Word eingesetzt werden, um z.B. direkt aus den Tabellen einer Oracle Datenbank ein Word-Dokument zu erstellen.

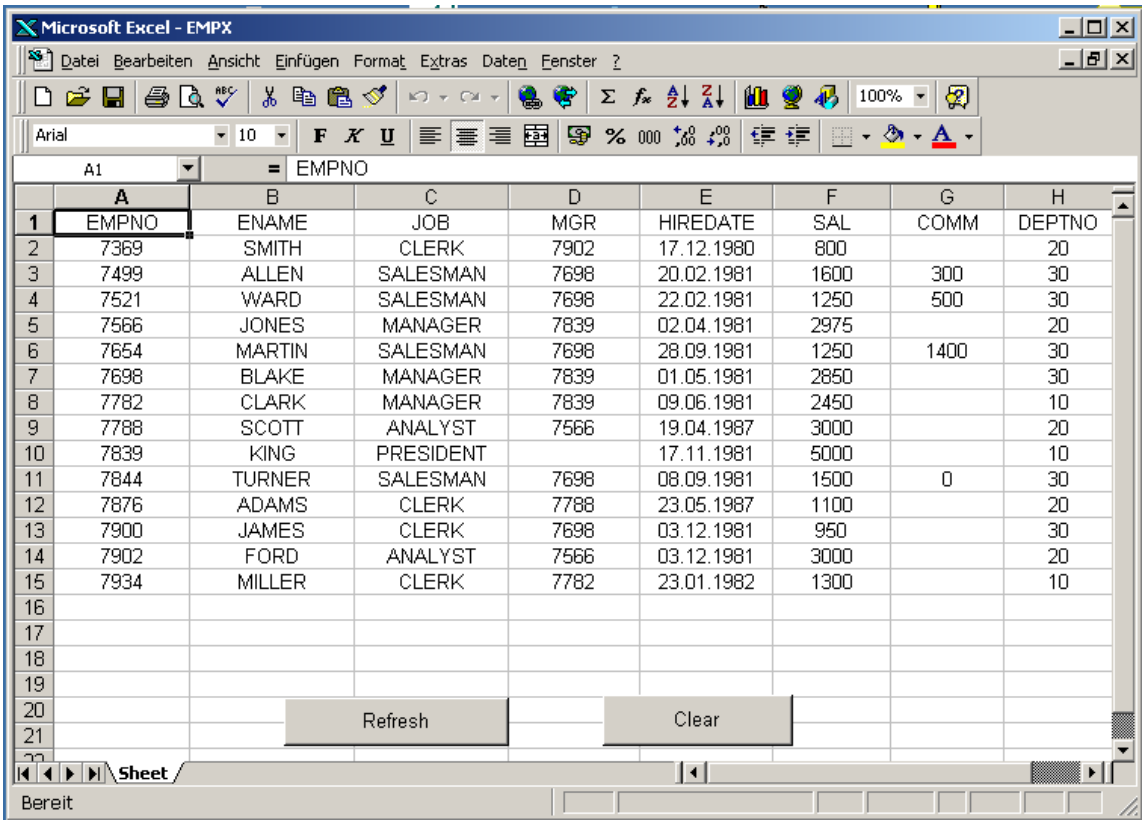
Jeder COM-Server muss zunächste registriert werden. In der Clientanwendung werden dann alle registrierten COM-Server aufgelistet, man kann dann dort die auswählen, die man benutzen möchte.
Unabhängig von einer Clientanwendung kann man sich mit der Utility **OleView** (Gibt es von Microsoft zum Download) zunächst einen Überblick über alle registrierten COM-Server verschaffen und weiter dann für jeden COM-Server die Details nachschlagen. Hier findet man dann auch die sogenannte ProgId, unter der der COM-Server angesprochen werden kann.
Nachfolgend ist der Eintrag für den COM-Server für den Zugriff auf Oracle Datenbanken dargestellt.



OleView (Eintrag für Oracle OLE-Server)

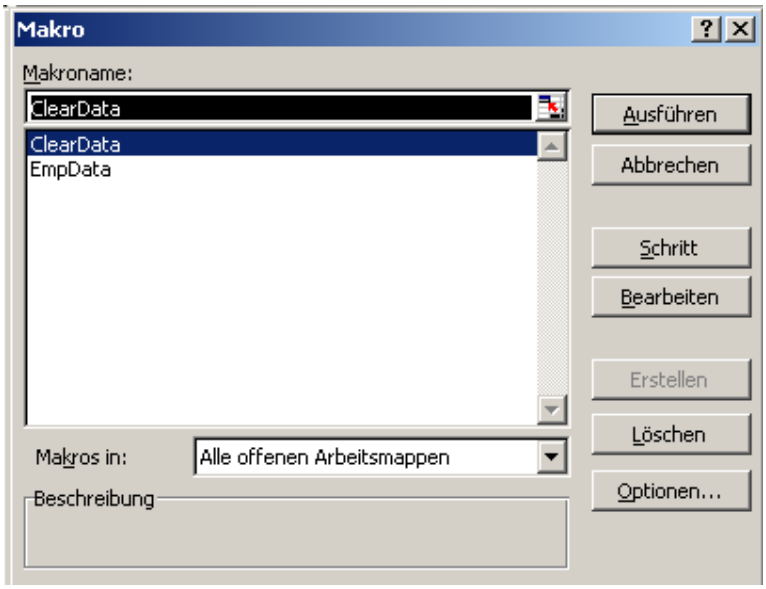
.

Beispiel (Füllen einer Excel-Tabelle aus einer Oracle DB-Tabelle)



	A	B	C	D	E	F	G	H
1	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
2	7369	SMITH	CLERK	7902	17.12.1980	800		20
3	7499	ALLEN	SALESMAN	7698	20.02.1981	1600	300	30
4	7521	WARD	SALESMAN	7698	22.02.1981	1250	500	30
5	7566	JONES	MANAGER	7839	02.04.1981	2975		20
6	7654	MARTIN	SALESMAN	7698	28.09.1981	1250	1400	30
7	7698	BLAKE	MANAGER	7839	01.05.1981	2850		30
8	7782	CLARK	MANAGER	7839	09.06.1981	2450		10
9	7788	SCOTT	ANALYST	7566	19.04.1987	3000		20
10	7839	KING	PRESIDENT		17.11.1981	5000		10
11	7844	TURNER	SALESMAN	7698	08.09.1981	1500	0	30
12	7876	ADAMS	CLERK	7788	23.05.1987	1100		20
13	7900	JAMES	CLERK	7698	03.12.1981	950		30
14	7902	FORD	ANALYST	7566	03.12.1981	3000		20
15	7934	MILLER	CLERK	7782	23.01.1982	1300		10
16								
17								
18								
19								
20								
21								

Den beiden Schaltflächen „Refresh“ und „Clear“ ist jeweils eine sogenannte Visual Basic Prozedur zugewiesen, d.h. ein in der Programmiersprache Visual Basic geschriebenes Programm, das eigenständig ausgeführt werden kann. Ein Mausklick auf die Schaltfläche startet die betreffende Visual Basic Prozedur.
Eine Tabellenkalkulation MS Excel kann ein oder mehrere Visual Basic Prozeduren enthalten. Die Visual Basic Prozeduren werden (zusammen mit sogenannten Makros) unter Extras::Macro::Macros...verwaltet:



Macro EmpData:

```
Sub EmpData()

  Dim OraSession  As Object
  Dim OraDatabase As Object
  Dim EmpDynaset  As Object

  Dim felder()    As Object
  Dim feldcount   As Integer

  Set OraSession = CreateObject("OracleInProcServer.XOraSession")
  Set OraDatabase = OraSession.OpenDatabase("ora10g", "ss09etxyz/xaver", 0&)
  Set EmpDynaset = OraDatabase.CreateDynaset("select * from Artikel", 0&)

  Range("A1:H15").Select
  Selection.ClearContents

  feldcount = EmpDynaset.Fields.Count

  ReDim felder(0 To feldcount - 1)

  For Colnum = 0 To feldcount - 1
    Set felder(Colnum) = EmpDynaset.Fields(Colnum)
  Next

  For Colnum = 0 To feldcount - 1
    ActiveSheet.Cells(1, Colnum + 1) = felder(Colnum).Name
  Next

  For Rownum = 2 To EmpDynaset.RecordCount + 1
    For Colnum = 0 To feldcount - 1
      ActiveSheet.Cells(Rownum, Colnum + 1) = felder(Colnum).Value
   Next
   EmpDynaset.DbMoveNext
  Next

Range("A1:A1").Select

End Sub
```

A2.2 Klassenüberblick

Die bereitgestellten Klassen sind natürlich vom OleServer abhängig, für einen OleServer, der die Funktionalität eines Datenbankservers bereitstellt, nehmen wir stellvertretend den für Oracle-Datenbanken bereitgestellten OleServer „OracleInProcServer“:

OraSessionClass

repräsentiert das Basisobjekt für den Zugriff auf den Oracle OleServer und bietet neben vielen anderen Methoden auch eine zum Öffnen einer Datenbank(-Verbindung)

OraConnection

erlaubt den Zugriff auf die bereitgestellte Datenbankverbindung und dient im Wesentlichen der Transaktionskontrolle (begin, commit, reset)

OraDatabase

repräsntiert eine Datenbank und bietet die verschiedenen Möglichkeiten SQL-Befehle abzusetzten. Dabei wird wieder unterschieden, ob der SQL-Befehl eine Ergebnismenge (die hier Dynaset genannt wird) liefert oder nur eine ganze Zahl. Darüberhinaus gibt es noch die Möglichkeit PL/SQL-Blöcke auszuführen.

OraDynaset

repräsentiert die Ergebnismenge einer Select-Anweisung, bietet aber zahlreiche Möglichkeiten zur Navigation (first, last, next, match, …) innerhalb der Zeilen der Ergebnismenge sowie die üblichen Operationen für die positionierte Tabellenzeile.

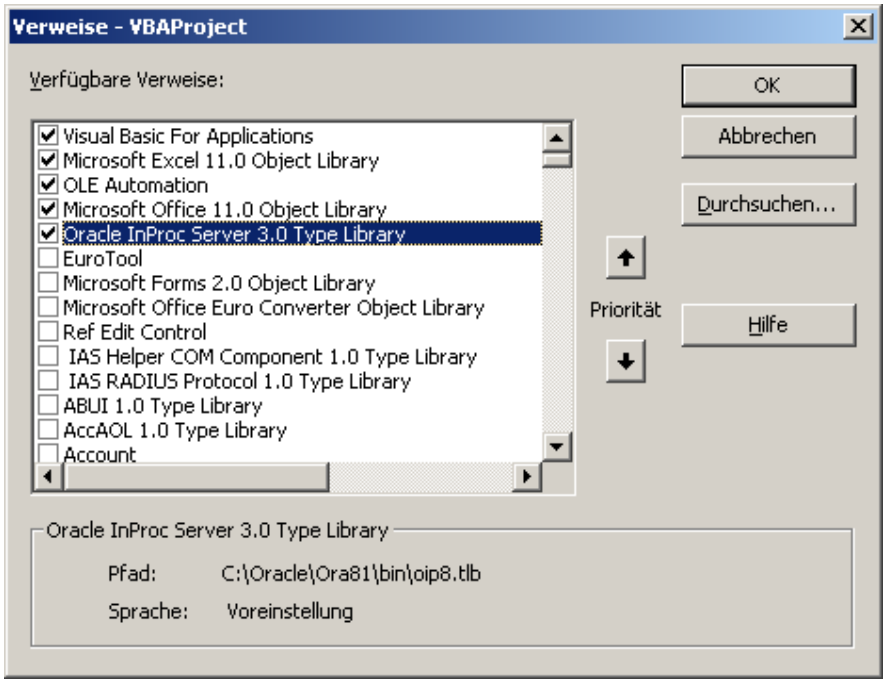
OraField

repräsentiert ein einzelnes Feld der Ergebnismenge und bietet z.B. den Zugriff auf den (Spalten)Namen, den (aktuellen) Inhalt und den Datentyp

Im Visual Basic Editor kann ein Objektkatalog zur Ansicht gewählt werden, der eine Übersicht über die verfügbaren OleServer-Klassen bereitstellt. Diese werden aus der Typbibliothek abgeleitet.

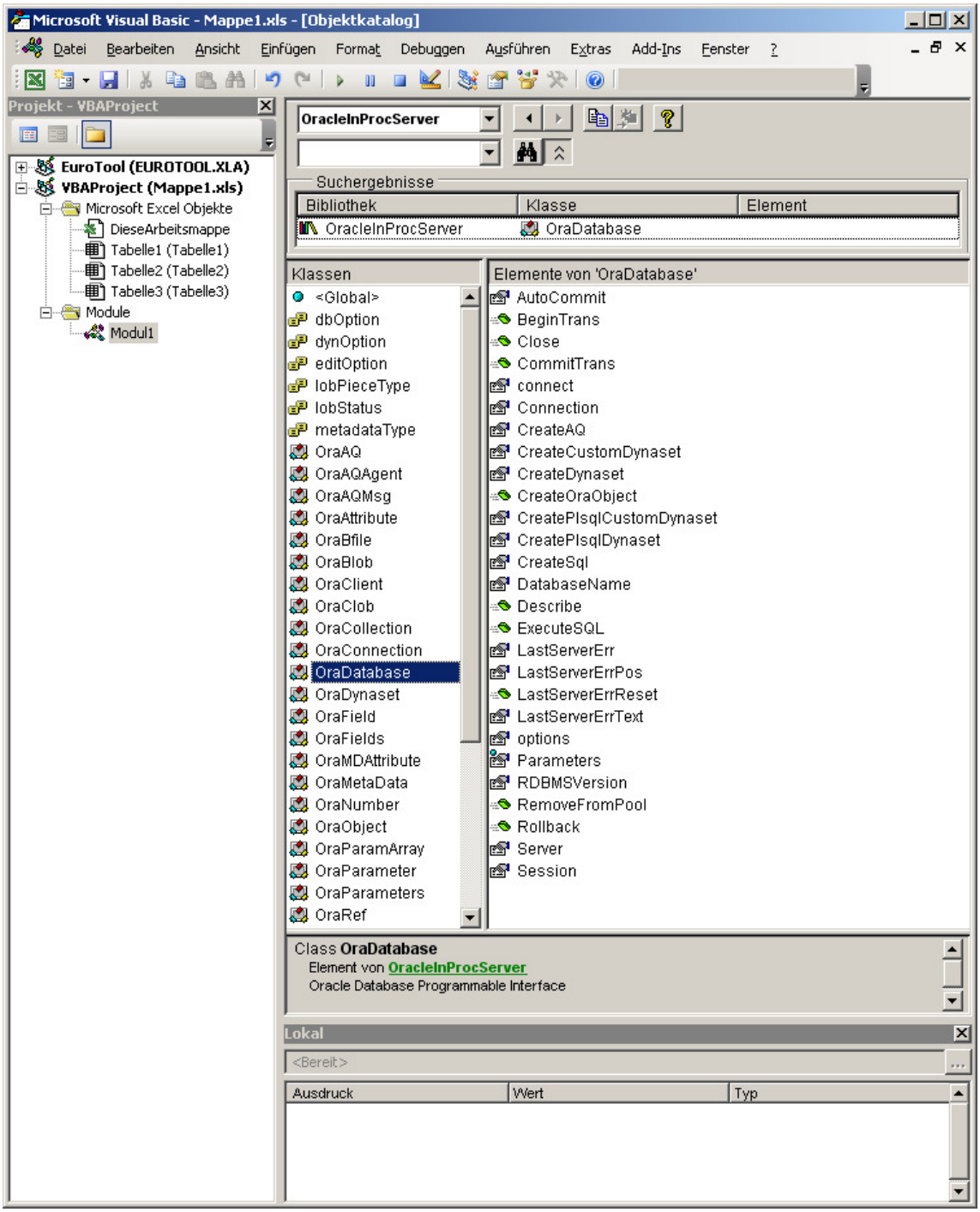
A2.3 OleServer und COM-Zugriffe

Der OleServer muss registriert sein und der Visual Basic Anwendung bekannt gemacht werden. Die Verwaltung der bekannt gemachten OleServer erfolgt über **Extras::Verweise….**



Der OleServer für eine Oracke Datenbank ist ausgewählt und markiert. Es wird eine sogenannte Typ-Bibliothek bereitgestellt.

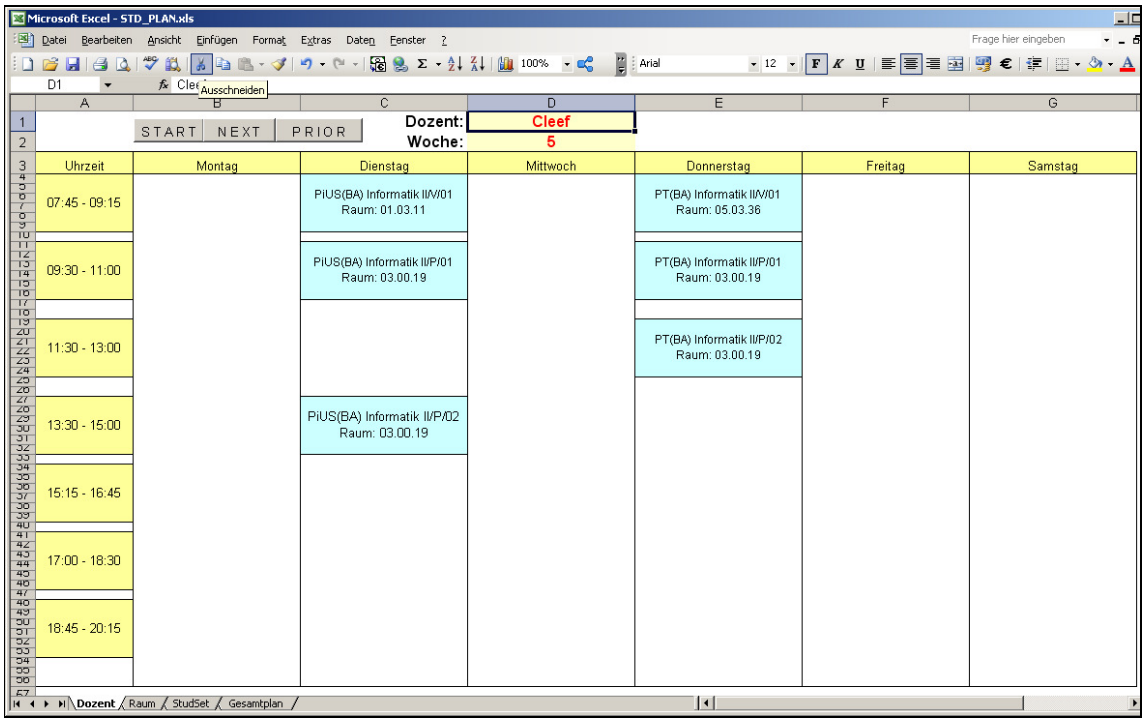
Anschließend werden die von den OleServern bereitgestellten Klassen im Objektkatalog aufgelistet und können innerhalb der Visual Basic Prozeduren wie gewöhnliche Klassen verwendet werden (siehe nächste Seite).



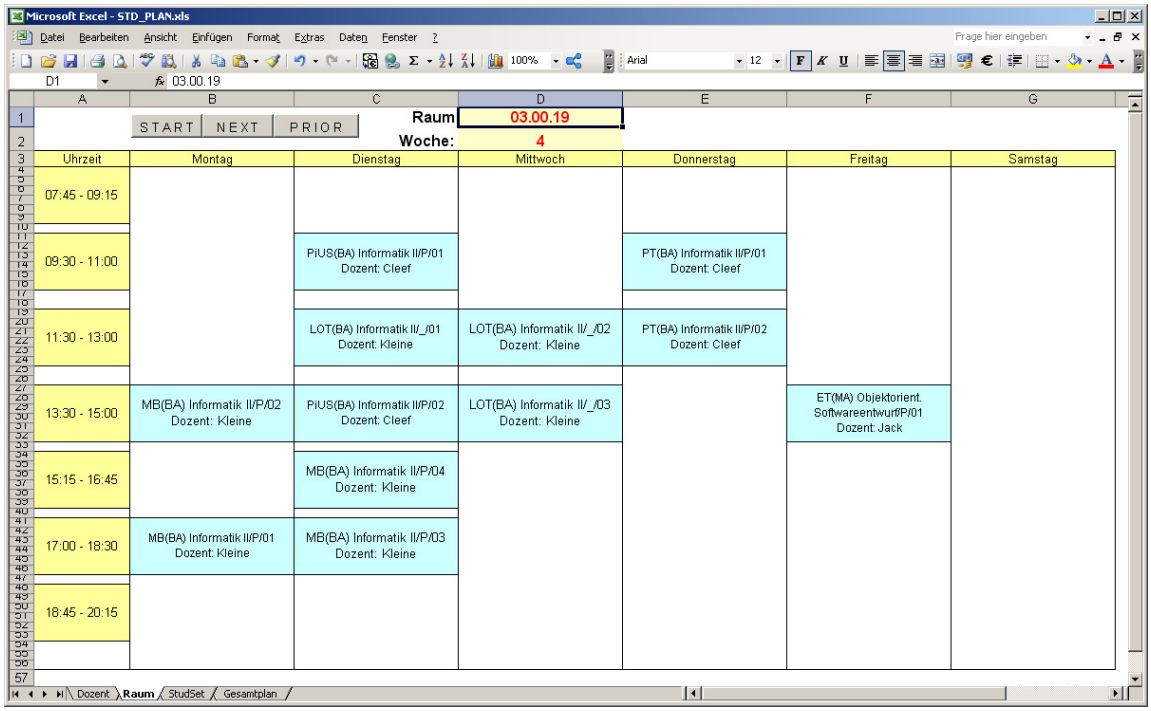
Neben den Zugriffen auf registrierte Datenbankserver, die die OLE-Technik unterstützen, kann diese Technik auch genutzt werden, um z.B. auf speziell formatierte Textdateien (csv = character separated values) zu zugreifen. Ist der Stundenplan in der in 2.2.4 beschriebenen Art in eienr datei STD_PLAN.csv gegeben, so kann z.B. innerhalb eines Excel-Dokuments auf diese daten zugegriffen werden und eine entsprechende Darstellung des Stundenplann erzeugt werden:

Stundenplan-Ausgabe (Es werden die Stundenplandaten für das SS 2008 verwendet !):

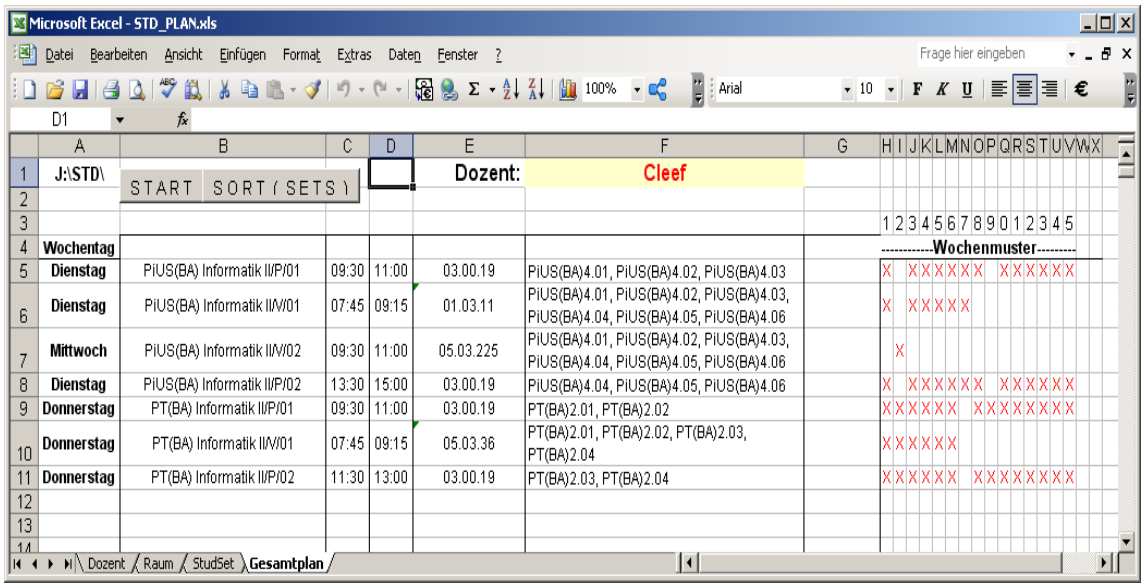
(1) Veranstaltungen für die 5. Vorlesungswoche und den Dozenten Cleef



(2) Raumbelegung für die 4. Woche und den Raum 03.00.19



(3) Semester-Gesamtübersicht für den Dozenten Cleef



A2.4 Vorteile/Nachteile und typische Einsatzgebiete

Vorteile:

- ⇒ **Entwickler** können prozedurale Möglichkeiten von VB (oder C#) nutzen und sind damit nicht auf die Möglichkeiten von SQL allein festglegt.
- ⇒ **Entwickler** können auch beim Datenbankzugriff im objektorientierten Umfeld bleiben, der Zugriff auf die Datenbank erfolgt über Klassen und deren Instanzen.
- ⇒ Für zahlreiche (objektorientierte) Anwendungen wird eine COM-Schnittstelle angeboten, damit lassen sich dann schnell Zugriffe aus anderen Anwendungen (z.B. Office Anwendungen) heraus realisieren.

Nachteile:

- ⇒ Verwaltungsaufwand für den Aufbau der Ole-Verbindung
- ⇒ Geschwindigkeitseinbußen bei Datenbankabfragen, durch den Umweg über die Klassen des OleServers
- ⇒ Abhängigkeit von den Klassen des OleServers, häufig ergeben sich hier Änderungen beim Übergang auf eine neue Version, deshalb sind dann oft Änderungen an den Anwendungen erforderlich, die die COM-Schnittstelle nutzen.

OleServer müssen häufig gesondert registriert werden.