hochschule hof

Fakultät Informatik

Vorlesung Informatik

„Datenbanken I“

Prof. Dr. Horst Heineck
Alfons-Goppel-Platz 1
95028 Hof/Saale

Raum: **FB 134**
Tel.: **09281/409-4440**
Fax: **09281/409-55-4440**
Email: **Horst.Heineck@hof-university.de**

Sprechstunde **Wintersemester 2019/2020**
Mittwoch: **13^{15} Uhr – 13^{45} Uhr**

Literaturverzeichnis

/Codd-69/ Codd, E. F.:
Derivability, redundancy and consistency of relations stored in large data banks, Technical Report Research Report RJ599, IBM, 1969

/Codd-70/ Codd, E. F.:
A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Vol 13, No. 6, p. 377-387, 1970

/CODASYL-75/ CODASYL:
Data Base Task Group Report 1971, Association for Computing Maschinery, 1975

/Chen-76/ Chen, P. P.:
The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems, Vol. 1, No. 1, p. 9-36, 1976

/ANSI-78/ ANSI/SPARC:
DBMS framework: Report on the study of data base management systems, Information Systems, 1978

/Codd-79/ Codd, E. F.:
Extending the relational model to capture more meaning, ACM Transactions on Database Systems, 1979

/Date-85/ Date, C. J.:
An Introduction to Database System, Vol. 1, Reading Massachusetts, 1985

/Jackson-89/ Jackson, G. A.:
Entwurf relationaler Datenbanken, Hanser Verlag München, Wien, 1989

/HMD-91/ HMD:
Theorie und Praxis der Wirtschaftsinformatik, verteilte Datenbanken, Forkel-Verlag, Heft 157, 1991

/Vossen-91/ Vossen, G., Witt, K.-U.:
Entwicklungstendenzen bei Datenbanksystemen, Oldenbourg Verlag München, Wien, 1991

/Sauer-95/ Sauer, H.:
Relationale Datenbanken, Addison-Wesley, 1995

/Schicker-96/ Schicker, E.:
Datenbanken und SQL, B.G. Teubner Stuttgart, 1996

/Meier-97/ Meier, A., Wüst, Th.:
Objektorientierte Datenbanken, dpunkt Verlag, 1997

/Kähler-99/ Kähler, W.-M.:
Relationales und objektrelationales SQL, Vieweg & Sohn Verlagsgesellschaft mbH, 1999

/Saake-99/ Saake, G., Heuer, A.:
Datenbanken – Implementierungstechniken, MITP Verlag GmbH, 1999

/Kofler-01/ Kofler, M.:
MySQL, Addison-Wesley Verlag, 2001

/Kuhlmann-01/ Kuhlmann, G., Müllmerstadt, F.:
SQL, Der Schlüssel zu relationalen Datenbanken, Rowohlt Taschenbuch Verlag, 2001

/Riccardi-01/ Riccardi, G.:
Datenbanksysteme mit Internet und Java-Applikationen, Addison-Wesley Verlag, 2001

/Eirund-02/ Eirund, H., Kohl, U.:
Datenbanken – leicht gemacht, 2. Auflage, Teubner Verlag, 2003

/Elmasri-02/ Elmasri, R., Navathe, S. B.:
Grundlagen von Datenbanksystemen, 3., überarbeitete Auflage, Verlag Pearson Studium, 2002

/Fritze-02/ Fritze, J., Marsch, J.:
Erfolgreiche Datenbankanwendung mit SQL3, 6. Auflage, Vieweg Verlag, 2002

/Rolland-02/ Rolland, F. D.:
Datenbanksysteme im Klartext, Verlag Pearson Studium, 2002

/Jarosch-03/ Jarosch, H.:
Grundkurs Datenbankentwurf, 2. Auflage, Vieweg Verlag, 2003

/Saake-03/ Saake, G., Sattler, K.-U.:
Datenbanken & Java, JDBC, SQLJ, ODMG und JDO, dpunkt.verlag, 2003

/Steiner-03/ Steiner, R.:
Grundkurs Relationale Datenbanken, 5. Auflage, Vieweg Verlag, 2003

/Date-04/ Date, C. J.:
An Introduction to Database System, Addison-Wesley Verlag, 2004
ftp://ftp.aw.com/cseng/authors/date/eight/AppD - Anhang D zu downloaden

/Gennick-04/ Gennick, J.:
Oracle SQL*Plus: The Definitive Guide, 2nd Edition, O'Reilly, 2004

/Greenwald-04/ Greenwald, R., Stackowiak, R., Stern, J.:
Oracle Essentials: Oracle Database 10g, Third Edition, O'Reilly Media Inc., 2004

/Moos-04/ Moos, A.:
Datenbank-Engineering, Analyse, Entwurf und Implementierung objektrelationaler Datenbanken, 3. Auflage, Vieweg Verlag, 2004

/Price-04/ Price, J.:
Oracle Database 10g SQL, Oracle Press Series, 2004

/Roy-04/ Roy-Fadermann, A., Koletzke, P., Dorsey P.:
Oracle JDeveloper 10g Handbook, Build Java 2 Platform, Enterprise Edition (J2EE) Applications, Oracle Press, 2004

/Schubert-04/ Schubert, M.:
Datenbanken: Theorie, Entwurf und Programmierung relationaler Datenbanken, B.G. Teubner Verlag Stuttgart, Leipzig, Wiesbaden, 2004

/Sybase-04/ Sybase:
Sybase PowerDesigner, General Features Guide, 2004

/Unland-04/ Unland, R.:
Wetterfest – SQL:2003: mit XML und Verwaltung externer Daten, IX, Heise Zeitschriften Verlag GmbH & Co. KG, 2004, Seite 44ff

/Alpar-05/ Alpar, P., Grob, H.L., Weimann, P., Winter, R.:
Anwendungsorientierte Wirtschaftsinformatik: Strategische Planung, Entwicklung und Nutzung von Informations- und Kommunikationssystemen, 4. Auflage, Vieweg Verlag, 2005

/Dröge-05/ Dröge, R., Raatz, M.:
Microsoft SQL Server 2005, Konfigurieren, Administrieren, Programmieren, Microsoft Press, 2005

/Feuerstein-05/ Feuerstein, St.:
Oracle PL/SQL, Programming, Fourth Edition, O'Reilly Media Inc., 2005

/Haas-05/ Haas, F.:
Oracle Tuning in der Praxis: Rezepte und Anleitungen für Datenbankadministratoren und –entwickler, Carl Hanser Verlag München Wien, 2005

/Held-05/ Held, A.:
Oracle 10g Hochverfügbarkeit: Die ausfallsichere Datenbank mit RAC, Data Guard und Flashback, Addison-Wesley Verlag, 2005

/Kline-05/ Kline, K.E., Kline, D., Hunt, B.:
SQL in a Nutshell, behandelt SQL Server, DB2, MySQL, Oracle & PostgreSQL, 2. Auflage, O'Reilly, 2005

/Loney-05/ Loney, K.:
Oracle Database 10g, Die umfassende Referenz, Hanser-Verlag, 2005

/Pollakowski-05/ Pollakowski, M.:
Grundkurs MySQL und PHP, So entwickeln Sie Datenbanken mit Open Source Software, Vieweg-Verlag, 2. Auflage, 2005

/Ahrends-06/ Ahrends, J., Lenz, D., Schwanke, P., Unbescheid, G.:
Oracle 10g für den DBA, Effizient konfigurieren, optimieren und verwalten, Addison-Wesley Verlag, 2006

/Linnemeyer-06/ Linnemeyer, L.C., Brown, B.D.:
Oracle Application Express Handbook, Develop Database-Centric Web Applications Quickly and Easily, McGraw-Hill/Osborne, 2006

/Saake-06/ Türker, C., Saake, G.:
Objektrelationale Datenbanken, Ein Lehrbuch, dpunkt-Verlag, 2006

/Faeskorn-07/ Faeskorn-Woyke, H., Bertelsmeier, B., Riemer, P., Bauer, E.:
Datenbanksysteme, Theorie und Praxis mit SQL2003, Oracle und MySQL, Pearson Studium, 2007

/Hart-07/ Hart, M., Freeman, R.G.:
Oracle Database 10g, RMAN Backup & Recovery, Oracle-Press, 2007

/Matthiessen-07/ Matthiessen, G., Unterstein, M.:
Relationale Datenbanken und Standard-SQL, 4. aktualisierte Auflage, Addison Wesley, 2007

/Fröhlich-08/ Fröhlich, L.:
Oracle 11g, Performance Forecast, Carl Hanser Verlag München Wien, 2008

/Greenwald-08/ Greenwald, R., Stackowiak, R., Stern, J.:
Oracle Essentials – Oracle Database 11g, Fourth Edition, O'Reilly Media Inc., 2008

/Held-08/ Held, A.:
Oracle 11g, New Features für DBAs und Software-Entwickler, Carl Hanser Verlag München Wien, 2008

/Kähler-08/ Kähler, W.M.:
SQL mit Oracle, Eine aktuelle Einführung in die Arbeit mit relationalen und objektrelationalen Datenbanken unter Einsatz von Oracle Express, 3. Auflage, Vieweg+Teubner Verlag, 2008

/Stern-08/ Stern, A.:
Keine Angst vor Microsoft Access, Datenbanken verstehen, entwerfen und entwickeln, Microsoft Press, 2008

/Greenwald-09/ Greenwald, R.:
Oracle Application Express, Wiley Publishing, 2009

/Harper-09/ Harper, S.:
Oracle SQL Developer 2.1, Packt Publishing, 2009

/Kemper-09/ Kemper, A., Eickler, A.:
Datenbanksysteme, Eine Einführung, 7. Auflage, Oldenburg Verlag, 2009

/Aust-10/ Aust, D., Kubicek, D., Pokolm, J.-C.:
Oracle APEX und Oracle XE in der Praxis, 1. Auflage,mitp-Verlag, 2010-03-11

/Beckmann-13/ Beckmann, R.:
Oracle Application Express in der Praxis, Mit Apex datenbankbasierte Webanwendungen entwickeln, Carl Hanser Verlag, 2013

/Greenwald-14/ Greenwald, G.:
Die globale Überwachung, Der Fall Snowden, Droemer Verlag, 2014

/OTN/ Oracle Technology Network:
http://www.oracle.com/technology/index.html

Ein herzlicher Dank geht an den Verlag Pearson Studium, der freundlicherweise die Bilder aus, z.B. /Elmasri-02/ oder /Rolland-02/ für Dozenten in elektronischer Form zur Verfügung stellt.

Für vergleichende Betrachtungen verschiedener Datenbanksysteme kann /Faeskorn-07/ empfohlen werden. Als besonderer Hinweis sei auf /Kline-05/ verwiesen. Hier werden hervorragend die Implementierungen der Sprache SQL durch verschiedene Hersteller mit dem neusten Standard SQL2003 verglichen und gegenübergestellt.

Inhaltsverzeichnis

1. Grundlegende Begriffe der Datenbanktechnologie

Datenbanken und Datenbanksysteme spielen eine wesentliche Rolle in verschiedenen Bereichen der modernen Gesellschaft. Die meisten von uns kommen täglich mit verschiedenen Aktivitäten in Berührung, die die eine oder andere Interaktion mit einem Datenbankmanagementsystem umfassen:

- ➔ Überweisung bei der Bank einreichen oder Onlinebanking,
- ➔ Reise buchen,
- ➔ in der Bibliothek ein Buch suchen oder
- ➔ im Supermarkt einkaufen

Diese Interaktionen sind Beispiele dessen, was unter traditionellen Datenbankanwendungen verstanden wird. Dabei werden die meisten Informationen entweder in Textform oder numerisch gespeichert, um einen Zugriff darauf zu ermöglichen.

In den letzten Jahren haben technologische und technische Fortschritte zu interessanten neuen Anwendungen für Datenbanksysteme geführt. Multimedia-Datenbanken können heute Bilder, Videoclips oder Sound-Nachrichten speichern. In **G**eografischen **I**nformations**S**ystemen (GIS) werden in Datenbanken Landkarten, Wetterdaten und Satellitenbilder gespeichert und analysiert. Für Data-Warehouse- und OLAP-Anwendungen (**O**n**l**ine **A**nalytical **P**rozessing) werden Datenbanksysteme eingesetzt, um Informationen aus sehr großen Datenmengen zu extrahieren und zu analysieren. Echtzeit- und aktive Datenbanksysteme werden zur Steuerung von Industrie- und Fertigungsprozessen eingesetzt.

Um die Grundlagen heutiger Datenbankmanagementsysteme zu verstehen, muss man sich mit den Grundlagen traditioneller Datenbankanwendungen beschäftigen

1.1. Informationssysteme und Datenbanksysteme

Gegenwärtig durchdringen moderne Informations- und Kommunikationstechnologien immer mehr Bereiche der Gesellschaft. Die Nutzung von Datenbanksystemen und rechnergestützten Informationssystemen ist an vielen Arbeitsplätzen unverzichtbar. Der Begriff ***„Informationssystem“*** ist allgemeiner gefasst als der Begriff ***„Datenbanksystem“***. Fast alle wichtigen Informationssysteme besitzen eine Datenbank als Bestandteil. Immer mehr Nutzer müssen sich deshalb mit der Begriffswelt der Datenbanktechnologie vertraut machen.

Einige wichtige Kategorien von Informationssystemen sind:

- ➔ Management-Informationssysteme / betriebliche Informationssysteme (an die betriebliche Struktur gebundene Kommunikationssysteme),
- ➔ Büroinformationssysteme (Systeme zur Bearbeitung, Archivierung und Wiederauffindung von Textdokumenten, auch Hypertext und Multimedia einbezogen),
- ➔ Informationsrecherchesysteme (typische Vertreter sind Patentrecherchesysteme, Literaturrechercheсysteme, aber auch Gesetzessammlungen und Gerichtsurteile sowie Kataloge von Museen),
- ➔ geographische Informationssysteme (sie verwalten regionale Informationen zur Geographie, Geodäsie, Umwelt, Demographie und zum Verkehr).

Spezialisierte Informationssysteme können im ***Echtzeitbetrieb*** (***„online“***) an bestimmte Prozesse gebunden sein, z.B. an die Produktionsüberwachung, Energieerzeugung und -verteilung. Weiterhin geben sie dem Dispatcher Hilfestellung bei Prozesseingriffen, wie Optimierung und Havariebeseitigung.

Auch Expertensysteme können als fachgebietsspezifische Informationssysteme betrachtet werden. Die große Anzahl feststehender Informationen (Fakten) wird dabei effektiv in einer Datenbank verwaltet, während situationsbezogenes Wissen meist in Regelform repräsentiert wird.

Der Umgang mit einer Datenbank erfordert eine andere Denkweise und eine andere Methode, als sie beispielsweise für die Aufstellung eines ***„C-Programms“*** erforderlich sind.

1.2. Von der Dateiverwaltung zur Datenbanktechnologie

In der frühen Phase der Rechentechnikentwicklung in den 50’er und 60’er Jahren war das Hauptinteresse auf das ***„Programm“*** gerichtet. Das Programmieren, d.h. die Umsetzung (***„Codierung“***) von Algorithmen, die vorwiegend numerische Daten verarbeiten, war Untersuchungsgegenstand. Den Daten als den eigentlichen Bearbeitungsobjekten wurde wenig Aufmerksamkeit gewidmet. Sie waren einfach Bestandteil des Programms. Zu dieser Zeit wurden Programme maschinennah codiert und die Hardwareentwicklung erlaubte nur die Bearbeitung relativ kleiner Datenmengen. Als die ersten höheren Programmiersprachen ***„FORTRAN“*** und ***„ALGOL“*** definiert wurden, waren diese algorithmisch orientiert und das Dateitypkonzept war schwach entwickelt.

Erst 1963, angeregt durch die Hardwareentwicklung (***„IBM/System 300*** mit großen peripheren Speichergeräten und Kanalkonzept), wurde ein ***„Dateikonzept“*** in die Programmiersprache ***„PL/1“*** aufgenommen. Eine ***„Datei“*** besteht aus einer ***Menge von Sätzen***, die wiederum eine Struktur (Format) besitzen. Die Dateibeschreibung enthielt nur diejenigen Angaben, die für das Betriebssystem des Rechners wichtig waren.

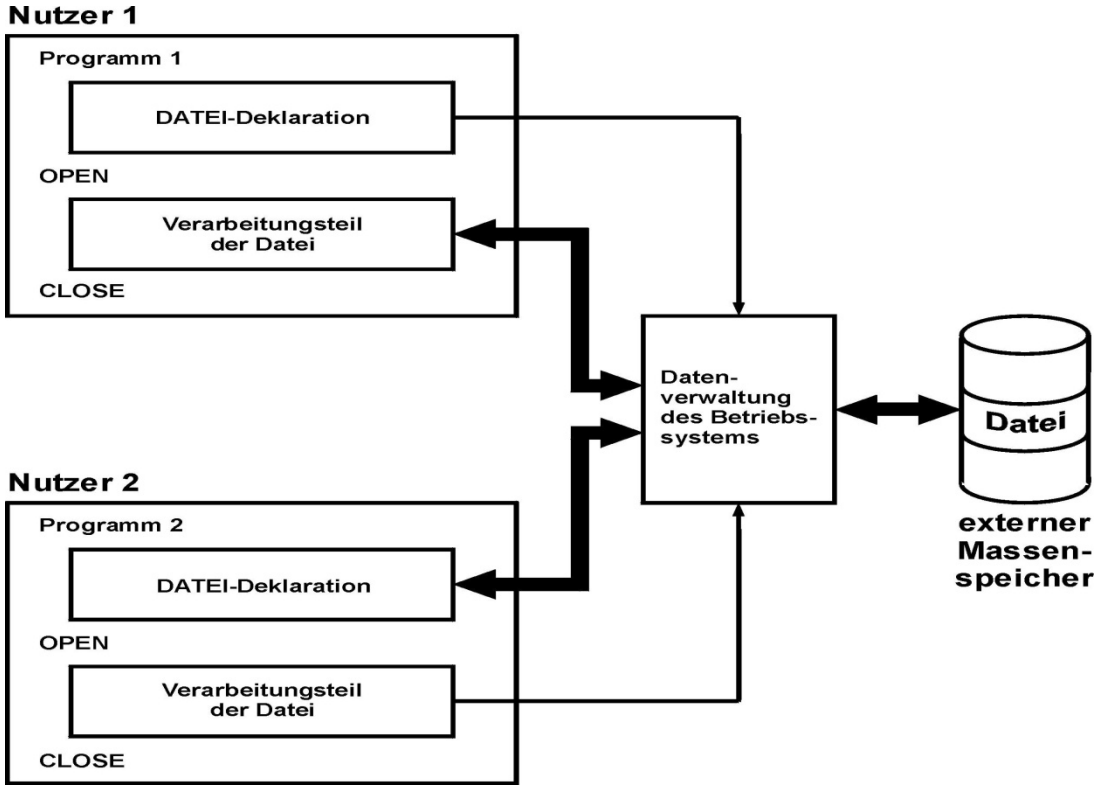
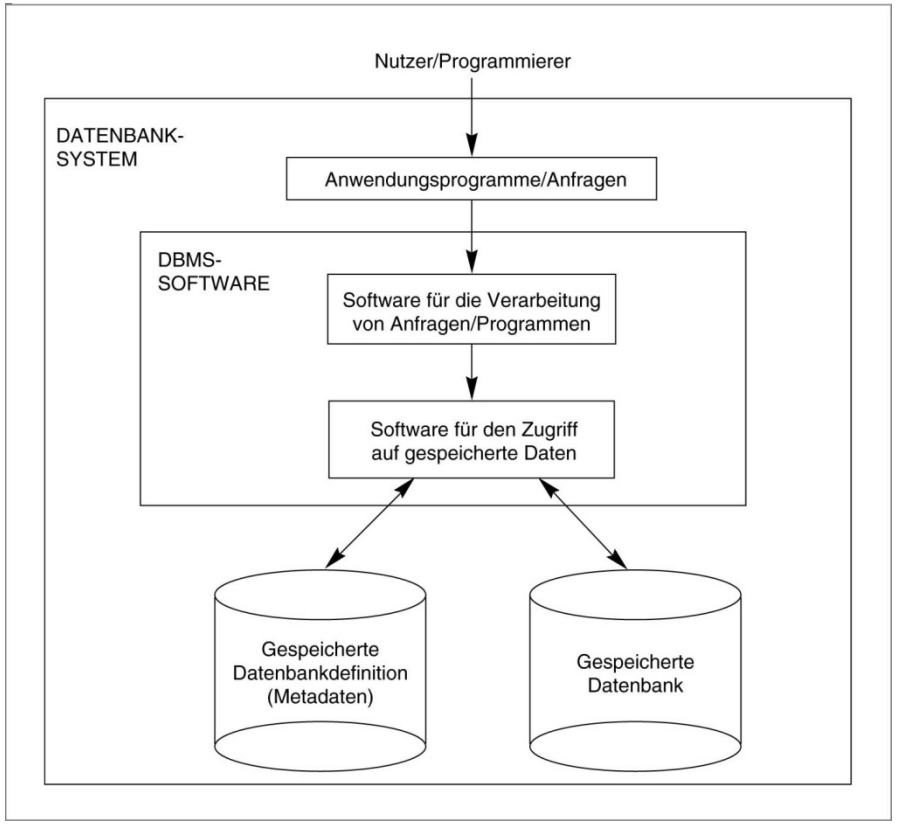


Abbildung 1.: Entwicklung eines Dateikonzeptes

Mit den vorwiegend physischen Angaben ist die Dateiverwaltung eines Betriebssystems in der Lage, die Dateien dem Nutzer verfügbar zu machen. Damit war es prinzipiell möglich, dass Daten durch unterschiedliche Nutzer in verschiedenen Programmen, die die gleichen Dateideklarationen enthielten, benutzt werden konnten, siehe Abbildung 1.

Die Dateien waren den Programmen fest zugeordnet, d.h. es musste unter Umständen mit einer Kopie einer Datei gearbeitet werden, oder bei einer überlappenden Nutzung musste eine Zugriffssynchronisation programmiert werden. Bei logisch zusammengehörenden, eventuell voneinander abhängigen Verarbeitungsprozeduren war somit eine Korrektheit des Dateiinhaltes nach der Bearbeitung nicht automatisch gesichert.



Der Übergang von der klassischen, programmzentrierten Sicht zur datenzentrierten Sicht (Datenbanktechnologie) wurde etwa 1968 bis 1970 vollzogen. Begleitet wurde diese Entwicklung von einer raschen Kapazitätszunahme externer Massenspeicher und von der Tatsache, dass die Daten immer wertvoller wurden.

Abbildung 2.: Datenbanksystemumgebung

Die Verfügbarkeit der Daten und der Zugriff im Dialog wurden wichtiger als umfangreiche Auswertungsalgorithmen. Das Datenmodell beschreibt die ***„logische Struktur der Datenbasis“***. Der Nutzer hat eine Datenbankabfragesprache ***„SQL“*** (***S***tructured ***Q***uery ***L***anguage) zur Wiedergewinnung von Informationen aus der Datenbasis zur Verfügung. Als Mittler zwischen Nutzer und Datenbasis arbeitet ein ***„Datenbankmanagementsystem“*** - ***DBMS*** (***D***ata***B***ase ***M***anagement ***S***ystem). Der Nutzer kann seine Abfragen in ***SQL*** im Dialog stellen, hat aber auch die Möglichkeit, bei der Nutzung von Programmiersprachen, wie z.B. ***„C“***, ***SQL***-Anweisungen einzubetten (***„embedded SQL***). Eine weitere Möglichkeit besteht für den Nutzer einer Datenbank darin, die ***„Entwicklungsumgebung“*** einer Datenbank zu verwenden, um komplexere Abfragen über der Datenbasis zu formulieren. Diese ***„4GL“***-Sprachen erlauben die Erstellung von kompletten (***SQL***-) Programmen, siehe Abbildung 3.

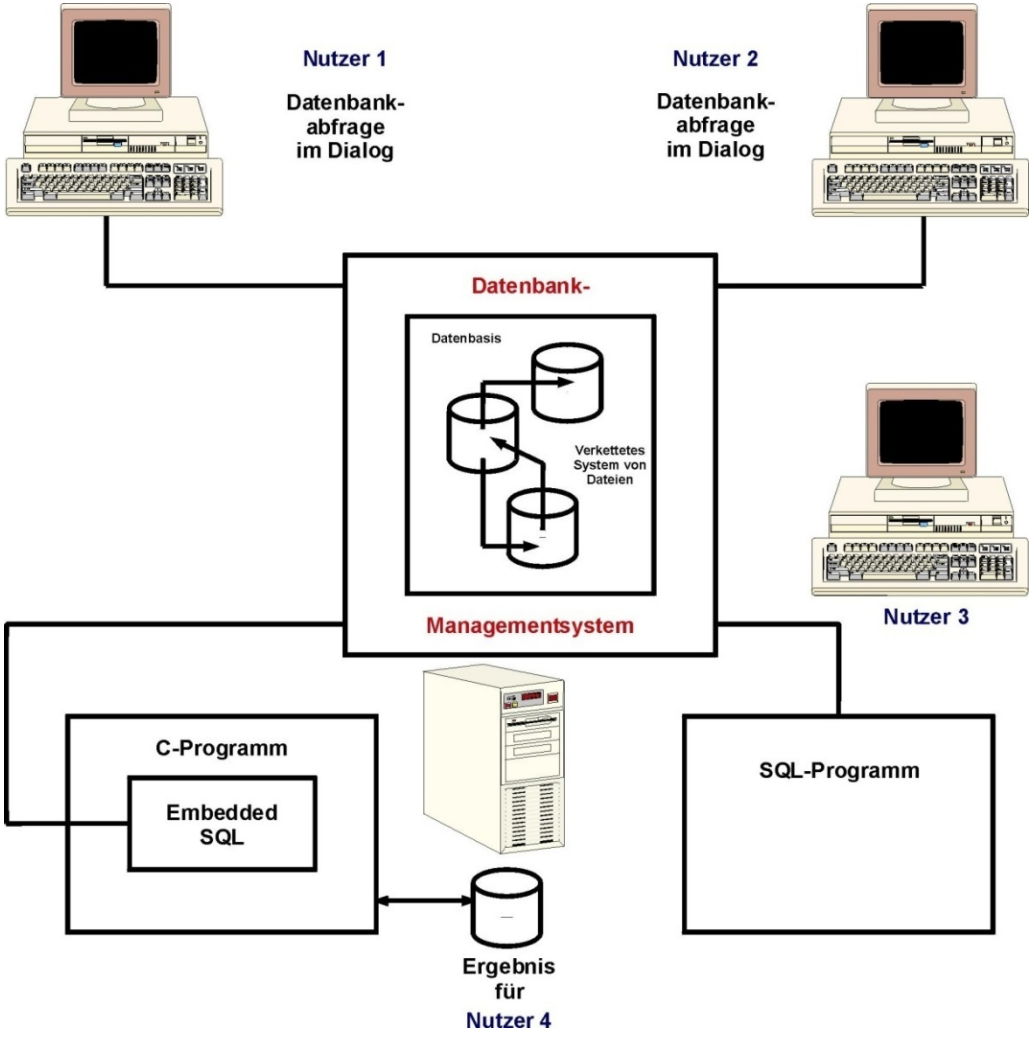


Abbildung 3.: Datenzentrierte Sicht in der Datenbanktechnologie

Definition 1 - Datenbanktechnologie:

Unter der Bezeichnung *Datenbanktechnologie* seien alle Aufgaben, Hilfsmittel und Lösungen zu verstehen, die notwendig sind für:

- **die Entwicklung von Datenbanksystemen,**
- **die Benutzung von Datenbanksystemen und dabei insbesondere**
- **die Entwicklung von Anwendersoftware auf der Basis von Datenbanksystemen.**

Dabei sind weitere Begriffe und ihre Bedeutung zu definieren:

Definition 2 - Datenbank:

Die Datenbank ist eine Ansammlung von Daten, die nach bestimmten Verfahren des Datenbankdesign aus der innerbetrieblichen Datenmodellierung strukturiert eingegeben wurden.
Dabei lassen sich folgende Anforderungen bezüglich der Abspeicherung der Daten formulieren:

1. **redundanzfreie, bzw. redundanzarme Abspeicherung der Daten,**
2. **konsistente Abspeicherung der Daten und**
3. **sichere Abspeicherung der Daten.**

Definition 3 - Datenbankmanagementsystem:

Ein Datenbankmanagementsystem umfasst die Gesamtheit von Software zur Datenverwaltung in einem Unternehmen. Als Basis dienen Systeme von so genannten Datenbankherstellern, z.B.:

- **ORACLE® – Oracle Server 12c Release 2,**
- **IBM – DB2, bzw. DB6 (für Unix/Linux),**
- **Microsoft – SQL Server 2017**

usw.

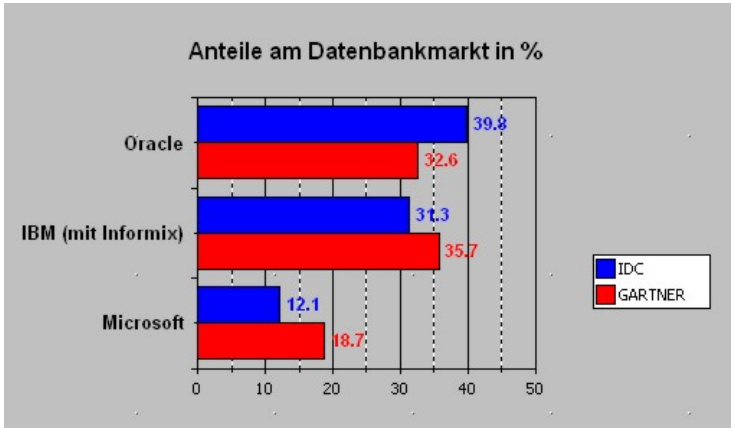


Abbildung 4.: Anteil am Datenbankmarkt 2003 (siehe COMPUTER Zeitung 25/2004/hd)

Definition 4 - Datenbanksystem:

Ein Datenbanksystem ist die Verbindung aus der Datenbank (DB) und dem Datenbank-managementsystem (DBMS)

DBS = DB + DBMS

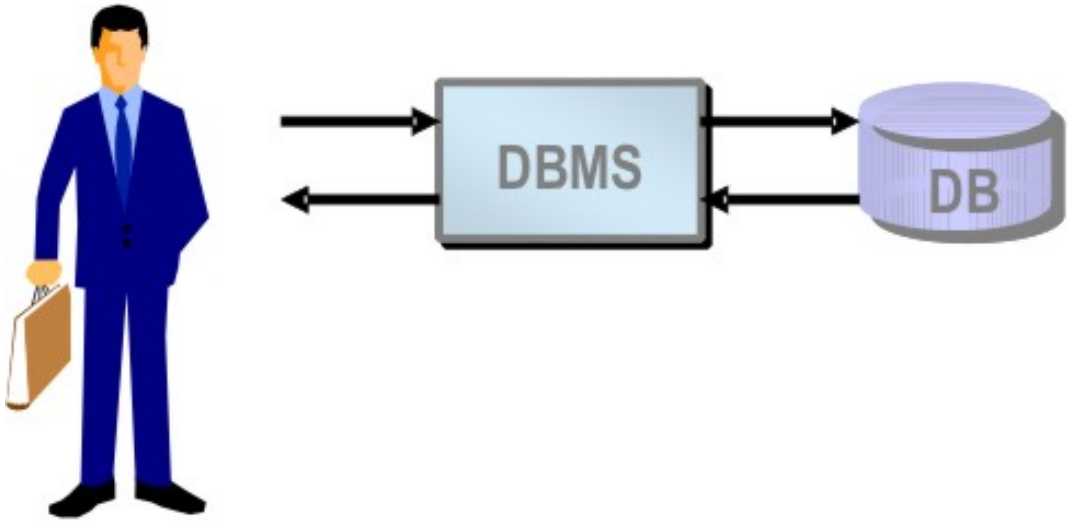


Abbildung 5.: Datenbanksystem

Der Umstand, dass ein DBMS von einem Hersteller nicht allein als Datenbankmanagementsystem für den Anwender ausreichend ist, stellt eine zentrale Überlegung dar.

Stand beim klassischen Softwarekonzept das Programm im Mittelpunkt, während die Daten eine eher untergeordnete Rolle spielten, so verhält es sich bei Datenbanksystemen (***„objektorientierte“*** Datenbanken ausgenommen) genau umgekehrt.
Dabei ist folgenden Problemen Aufmerksamkeit zu schenken:

Definition 5 - Redundanz:

> **Unter Redundanz versteht man „überflüssige“, d.h. mehrfach gehaltene Information, die auch mehrfach zu pflegen ist und daher früher oder später zur Inkonsistenz der Datenbank führt.**

Die bezieht sich auf die ***„logische“*** Datenhaltung. Redundanzen bei der ***„physikalischen“*** Datenhaltung kann die Fehleranfälligkeit des Systems reduzieren und ist deshalb nicht nur wünschenswert, sondern unvermeidbar.

Definition 6 - Inkonsistenz:

> **Die Dateninkonsistenz ist eine Widersprüchlichkeit, die aus den redundanten Informationen der Datenbank resultiert.**

Die postulierte Dominanz der Datenbank über die Programme macht die Unabhängigkeit von der Software für den Datenbankzugriff erforderlich.

Nicht selten liegt die erwartete Lebensdauer der in der Datenbank abzulegenden Datenbestände erheblich über der ***„Halbwertzeit“*** des Rechnersystems. So entsteht eine ungebührliche Bindung der Informationen an das Rechnersystem.

Während beim konventionellen Programmieren eine ***vollständige formale Spezifikation*** zu Algorithmenentwicklung gefordert wird, kann das von einer Datenbankapplikation oft nicht verlangt wird. Die

Datenbankentwicklung ist also nicht als ***linearer, zirkelfreier*** und ***terminierender Prozess*** zu verstehen, sondern als ***periodisch auftretendes Phänomen*** gängiger Bestandteil der Datenbankpraxis.

Vorteile einer Datenbank:

- ➔ Verschiedenen Nutzergruppen steht eine ***Datenbasis*** für eine ***gemeinsame Nutzung*** zur Verfügung, die Nutzung erfolgt sowohl im Dialog als auch durch Programme bzw. Applikationen.

- ➔ Die Datenbasis als Modell eines Realitätsausschnittes erlaubt ***verschiedenen Nutzern eine unterschiedliche Sicht*** auf die Daten.

- ➔ Die ***realen Daten*** sind ***unabhängig von Nutzerprogrammen und -applikationen*** und damit von Verarbeitungsprozessen, und die ***Nutzung*** ist ***unabhängig von der physischen Speicherungsform.***

- ➔ Durch die ***zentralisierte Verwaltung*** (durch einen ***DBA*** – ***D***ata ***B***ase ***A***dministrator) wird Redundanz vermieden und inhaltliche Vollständigkeit ***„Integrität“*** sowie die logische Korrektheit ***„Konsistenz“*** gesichert, durch eine Nutzerverwaltung lassen sich ***„Zugriffsrechte“*** vergeben und überwachen - ***„Datenschutz“***.

1.3. Datenbankschichtenmodell

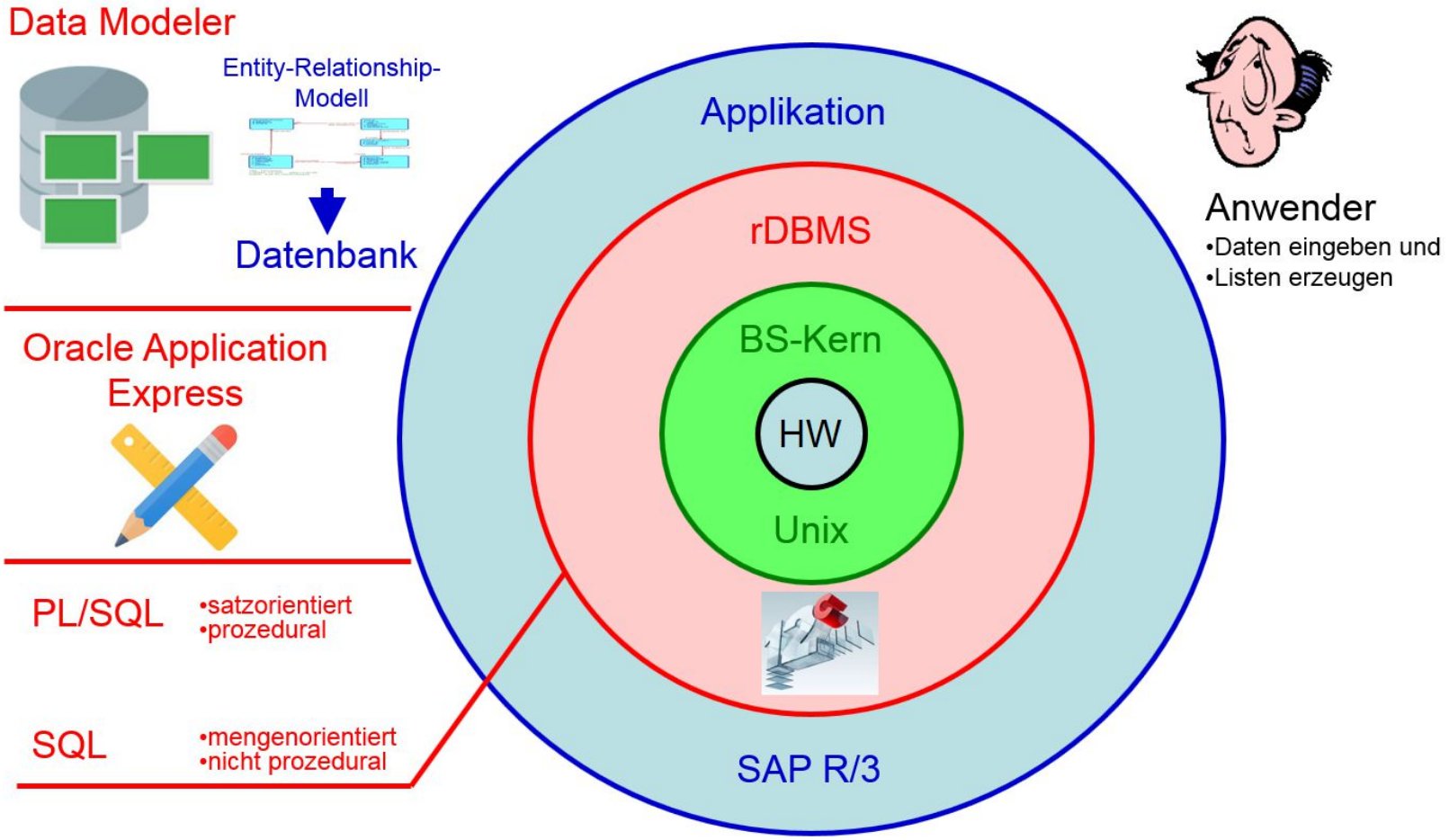


Abbildung 6.: Schalenmodell für den Datenbankeinsatz

Es ist ein Leistungsmerkmal des Datenbankmanagementsystems, den Benutzer von Hardware- und Systemdetails abzuschirmen und das Interface zu den physikalischen Daten herzustellen. Dabei sind vom System ***Datenbankintegrität, Recovery-Maßnahmen*** und ***Zugriffssynchronisation*** zu unterstützen.

In Abbildung 6 sind diese Schichten am Beispiel des relationalen Datenbankmanagementsystem **ORACLE®** in Verbindung mit einer Darstellung der gesamten Problematik des Einsatzes von Datenbankmanagementsystemen zusammengefasst.

Hinweis - prozedurale Spracherweiterungen sind nicht nur auf Oracle beschränkt:

Oracle	Oracle Datenbankserver	PL/SQL
Microsoft	SQL Server	Transact-SQL - (T-SQL)
IBM	DB2	UDB SQL

Abbildung 7 zeigt noch mal herausgehoben die Schichten, abhängig von der Sichtweise des Datenbankdesigner und des Datenbanksystems. Abhängig von dieser Sichtweise ergeben sich vier Ebenen, in Abwandlung der gängigen Methode, wie sie auch in Abbildung 6 enthalten ist, nur drei Ebenen zu unterscheiden.

➔ Konzeptionelles Schema, logische Gesamtsicht, rechnerunabhängig:
Hier kann man von einer Spezifikation des Datenmodells bzw. der formalen Beschreibung der realen Welt sprechen.

➔ Logisches Schema, DBS-abhängige Gesamtsicht:
Das logische Schema wird oft auch als ***„implementiertes konzeptionelles Schema“*** bezeichnet. Dies ist jedoch ***maschinenunabhängig*** zu verstehen, es bezieht sich auf das tatsächlich vorhandene Datenbanksystem.

Die Daten werden explizit in der (***relationalen***) Datenbank dargestellt. Dabei wird die Datenbank vom Benutzer als eine Sammlung von ***Tabellen*** wahrgenommen. Dies sagt jedoch nichts über die interne Struktur aus.

➔ Internes Schema , DBS-abhängige physische Datenorganisation:
Das interne Schema ist durch das Datenbankmanagementsystem festgelegt und für den Benutzer i.a. nicht zugänglich. Dies entlastet einerseits den Benutzer. Andererseits liegt hier auch ein Ansatzpunkt für ***Datenschutz*** und ***-sicherheit***. Das interne Schema ist aus Nutzersicht allenfalls für gewisse ***Tuning-Maßnahmen*** (***Index***, ***Cluster***, usw.) relevant.

➔ Externes Schema, DBS-abhängige Anwendersicht:
Eine spezielle Anwendersicht auf das logische Schema wird als externes Schema bezeichnet. Darin ist es besonders ausgeprägt möglich, Probleme des Zugriffsschutzes in Multiuser-Umgebungen zu behandeln.

Das konzeptionelle Schema wird durch ein ***„Datenmodell“*** (z.B. das ***relationale Datenmodell***) repräsentiert. Das Datenmodell stellt einen allgemeinen Begriffsapparat zur Verfügung, der es gestattet, Realitätsausschnitte ohne Bezug auf ein bestimmtes Sachgebiet zu modellieren.

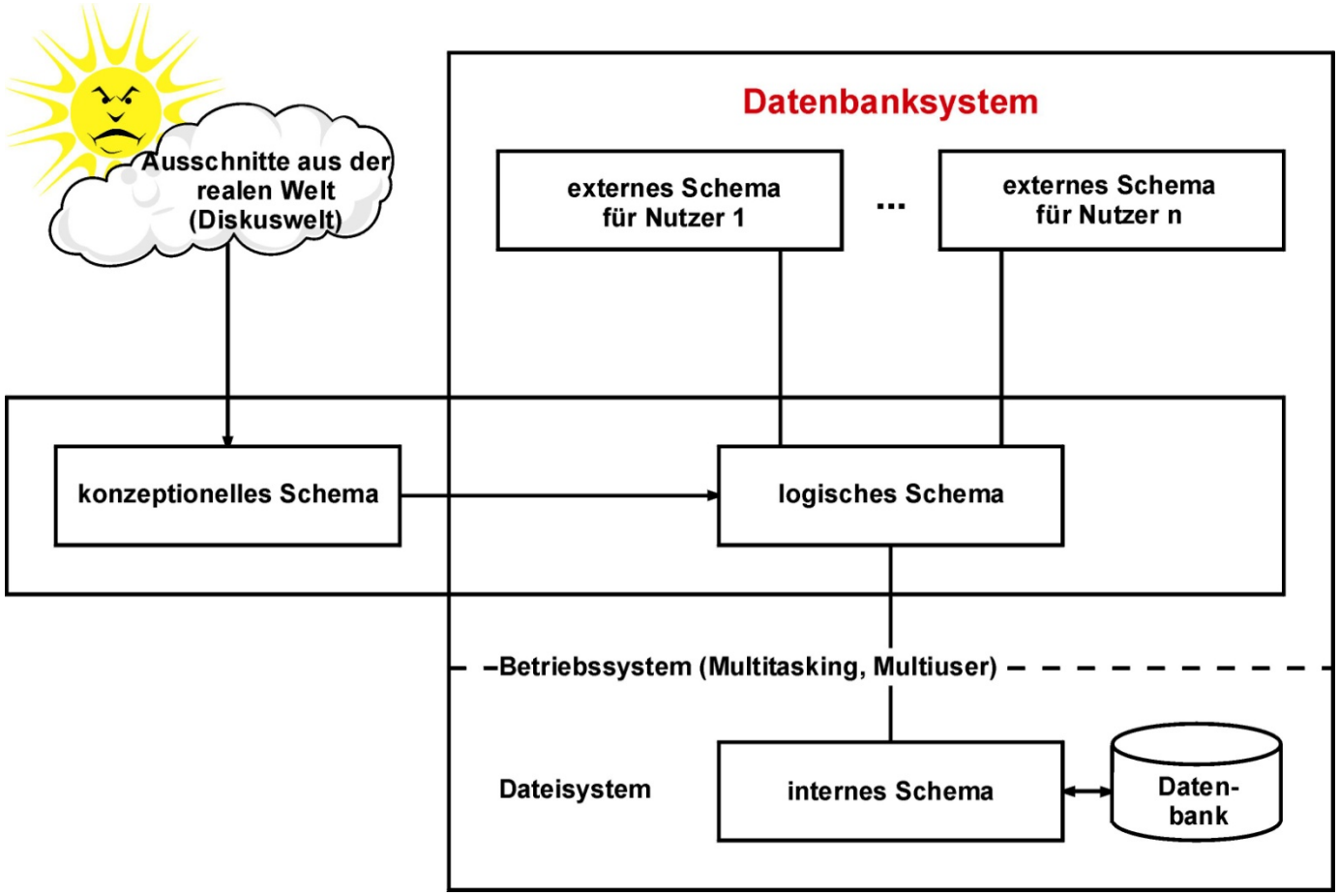


Abbildung 7.: Schichtenmodell von Datenbanksystemen

Das Prinzip des Schichtenmodells wurde bereits im Jahre 1975 von einer Arbeitsgruppe des ***„American National Standard Institute“*** (***ANSI/ X3/ SPARC***), die sich mit dem Aufbau von DBMS beschäftigte, entwickelt.

Diese Betrachtungsweise soll die Kluft zwischen der Sicht des Nutzers und der Sicht des Implementierens einengen helfen. Das Schichtenmodell wurde bei der Entwicklung der Datenbanktechnologie erarbeitet, ist aber so allgemein, dass es auf vielen Fachgebieten zur Modellierung von Realitätsausschnitten einsetzbar ist.

1.4. Datenunabhängigkeit

Die Drei-Schicht-Architektur kann zur Erklärung des Konzeptes der Datenunabhängigkeit verwendet werden. Der Definition zufolge beschreibt Datenunabhängigkeit die Möglichkeit, das Schema auf einer Ebene zu ändern, ohne das Schema der nächsthöheren Ebene ändern zu müssen. Datenunabhängigkeit kann auf zwei Arten definiert werden:

1. Logische Datenunabhängigkeit:
   Dies ist die Fähigkeit, das konzeptionelle Schema zu ändern, ohne externe Schemas oder Anwendungsprogramme ändern zu müssen. Es kann beispielsweise das konzeptionelle Schema geändert werden, um die Datenbank (durch Hinzufügen eines Datensatztyps oder Datenfeldes) zu erweitern oder aber (durch Entfernen eines Datensatztyps oder Datenfeldes) zu reduzieren. Im zweiten Fall sollte sich die Änderung nicht auf solche externen Schemas auswirken, die nur die nicht veränderten Teile des konzeptionellen Schemas referenzieren.

2. Physische Datenunabhängigkeit:
   Dieser Begriff beschreibt die Eigenschaft, ein internes Schema ändern zu können, ohne die konzeptionellen oder externen Schemas ändern zu müssen. Möglicherweise sind Änderungen des internen Schemas nötig, weil einige physische Dateien umorganisiert wurden, z.B. durch Erstellen zusätzlicher Zugriffsstrukturen, um den Zugriff auf die Daten zu beschleunigen. Änderungen des internen Schemas können auch dadurch entstehen, dass Hardwarekomponenten, von Festplattenlaufwerken bis hin zu neuen Clusterknoten ergänzt werden müssen. Wenn die gleichen Daten wie zuvor in der Datenbank verbleiben, muss das konzeptionelle Schema nicht geändert werden.

Definition 7 - physische Datenunabhängigkeit:

Die physische Datenunabhängigkeit der auf der Datenbank arbeitenden Programme ist zu gewährleisten, d.h., die Programme seien invariant gegenüber Änderungen in der konzeptionellen oder internen Ebene.

Definition 8 - logische Datenunabhängigkeit:

Die logische Datenunabhängigkeit der auf der Datenbank arbeitenden Programme ist zu gewährleisten, d.h., die Programme seien invariant gegenüber Änderungen und Erweiterungen von Struktur und Inhalt der Datenbank.

1.5. Datenbankentwurf

Die wesentlichen Entwicklungsschritte einer Datenbank sind vergleichbar mit den Stufen des allgemeinen Softwarelebenszyklus.

1. ***Analyse des Realitätsausschnittes***:
   Neben der Abgrenzung und Eingrenzung des zu modellierenden Realitätsausschnittes ist das Erkennen von Objekten und deren Beziehungen zwischen ihnen wichtig. Dieses Herausarbeiten der Grundobjekte für eine spätere Verarbeitung muss in enger Wechselwirkung mit der Zielstellung des Datenbankentwurfes, d.h. vor allem mit der funktionellen Spezifikation, erfolgen.

2. ***funktionelle Spezifikation:***
   Es ist festzulegen, in welcher Weise die Objekte zu bearbeiten sind. Es sind ggf. Nutzergruppen mit unterschiedlichen Anforderungsprofilen zu berücksichtigen. Umfang und Zielstellung des Datenbanksystems sind zu beschreiben.

3. ***Grobentwurf:***
   In der Phase des Grobentwurfes erfolgt die eigentliche Modellierung als Abstraktionsprozess. Die für die Verarbeitung wesentlichen Merkmale von Objekten und Beziehungen müssen bestimmt werden. In diesem Entwicklungsschritt wird die Modellierung mit dem ***„Entity-Relationship-Modell“*** bevorzugt.

4. ***Feinentwurf:***
   Der Feinentwurf erfolgt meist im relationalen Datenmodell. Es werden alle Details zu den ***„Relationen“*** der Datenbasis festgelegt. Es erfolgt eine Normierung der Relationen zur Vermeidung von Anomalien bei der Nutzung der Datenbank und zur Beseitigung von Redundanz.

5. ***Implementierungsentwurf:***
   Es werden Festlegungen zur Hardwarearchitektur (***integrierte*** oder ***verteilte Datenbank*** bzw. Nutzung einer Datenbank im Rechnernetz) getroffen. Entsprechend der Hardwarevoraussetzungen, der Datenmodellierung und den Anforderungen an die Funktionalität und die Leistungsparameter wird ein konkretes Datenbankmanagementsystem ausgewählt.

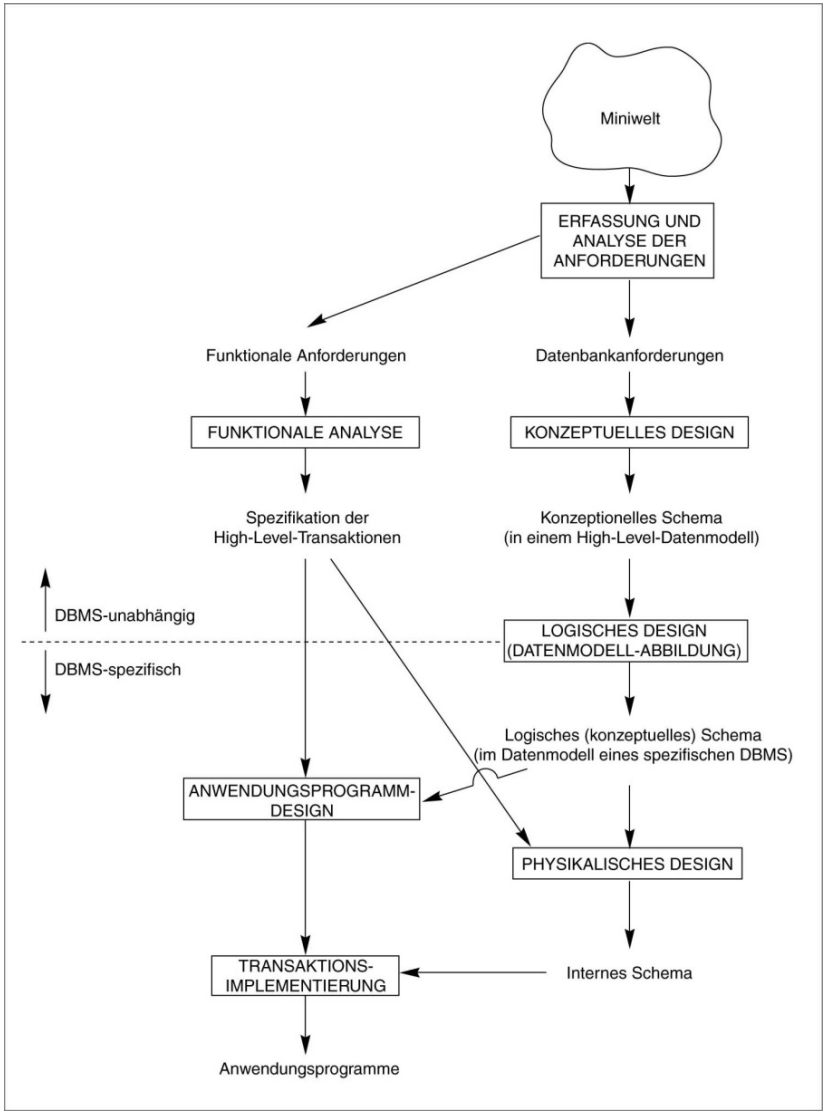


Abbildung 8.: Methodik des Datenbankentwurfs

6. ***Implementierung der Datenbank***
   Mit Hilfe der Datenbeschreibungssprache wird die Struktur der Datenbasis angelegt, und unter Nutzung der Datenmanipulationssprache werden die realen Daten eingespeichert und die Datenbasis aufgebaut.

7. ***Nutzung und Testung der Datenbank:***
   Mit der Implementierung eines Prototyps kann eine Nutzung als Testbetrieb beginnen. Schwachstellen und Fehler werden erkannt und beseitigt. Organisatorische Festlegungen zum Datenbankbetrieb, zum Datenschutz, zur Datensicherheit werden getroffen.

8. ***Regulärer Betrieb der Datenbank:***
   In dieser Phase wird vor allem die Leistungsfähigkeit der Datenbankimplementierung beobachtet. Auswertungen der ***„LOG“-Aufzeichnungen*** und Nutzerhinweise sollte der Datenbankadministrator zur Leistungsverbesserung nutzen.

2. Datenmodelle

Im Laufe der Entwicklung innerhalb der Datenbanktechnologie haben sich verschiedene Modellansätze entwickelt. Diese sollen im Folgenden beschrieben werden. In ihrer zeitlichen Entwicklungsreihenfolge können diese Modelle wie folgt eingeordnet werden:

Modellansatz	**Entwicklung ab**
Hierarchisches Modell	Anfang der sechziger Jahre des letzten Jahrhunderts
Netzwerkmodell	Ende der sechziger Jahre des letzten Jahrhunderts
Relationales Datenmodell	Anfang der siebziger Jahre des letzten Jahrhunderts
Objektorientiertes Datenmodell	Ende der achtziger Jahre des letzten Jahrhunderts

Tabelle 1.: chronologische Reihenfolge der Entwicklung der einzelnen Datenmodelle

Von besonderer Bedeutung, vor allem im kommerziellen Bereich ist das relationale Datenmodell. Dieses wird in diesem Abschnitt als letztes und ausführlicher behandelt, als die anderen Modellansätze.

2.1. Hierarchisches Datenmodell

Das **hierarchische Datenmodell** ist unmittelbar aus der konventionellen Datenverwaltung hervorgegangen und hat Bedeutung mit dem **DBMS IMS** (**I**nformation **M**anagement **S**ystem), einem **IBM**-Produkt, erlangt. Die dem hierarchischem Modell zugrunde liegenden Prinzipien sind im Wesentlichen aus diesem Produkt abgeleitet, das heute das vorherrschende hierarchische System in Banken, Versicherungsgesellschaften und Krankenhäusern sowie verschiedenen Regierungsbehörden ist.

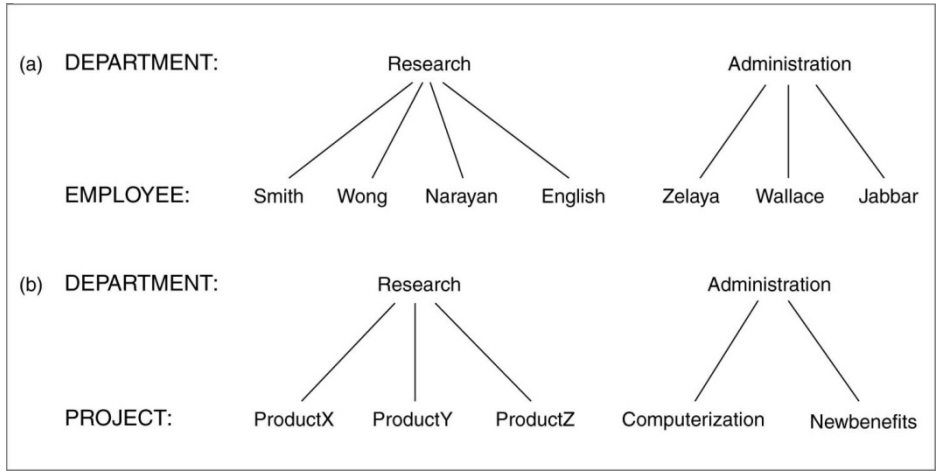
2.1.1. Eltern/Kind-Beziehungen und hierarchische Schemas

Im hierarchischen Datenmodell werden hauptsächlich zwei Datenstrukturkonzepte verwendet – Datensätze und Eltern/Kind-Beziehungen.

Ein **Datensatz** ist die Sammlung von **Feldwerten**, die Informationen über ein Objekt (Entität) oder Beziehungen zwischen den Objekten liefern. Datensätze des gleichen Typs werden zu **Datensatztypen** gruppiert. Ein Datensatztyp wird benannt und seine Struktur wird durch eine Sammlung benannter **Felder** oder **Datenexemplare** definiert. Jedes Feld besitzt einen bestimmten Datentyp (Zahl, Zeichenkette).

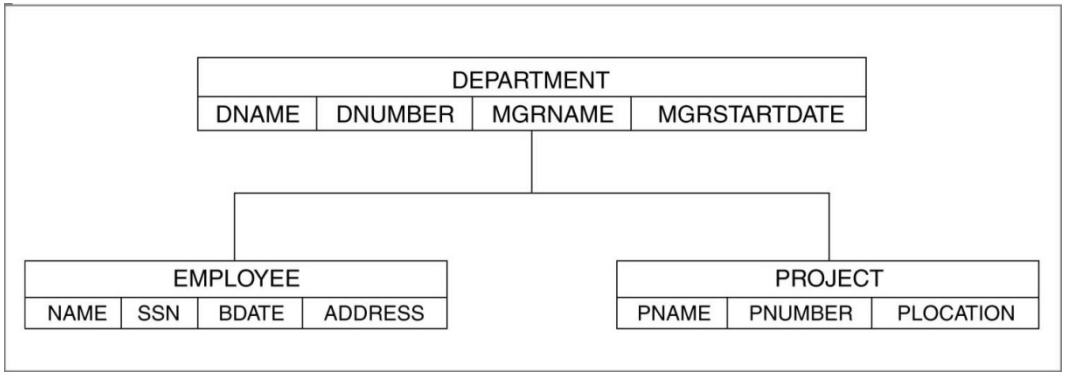
DEPARTMENT			
DNAME	DNUMBER	MGRNAME	MGRSTARTDATE

Abbildung 9.: Datensatztyp



Ein **Eltern/Kind-Beziehungstyp** (Parent-Child-Relationship, **PCR-Typ**) ist eine eins-zu-viele-Beziehung zwischen zwei Datensatztypen. Der Datensatztyp auf der eins-Seite ist der **Elterndatensatztyp** und derjenige auf der viele-Seite der **Kinderdatensatztyp** des PCR-Typs. Eine **Instanz** des PCR-Typs besteht aus einem Datensatz des Elterndatensatztyps und einer Reihe von Datensätzen des Kinddatensatztyps.

Abbildung 10.: Instanzen von Eltern/Kind-Beziehungen



Ein **hierarchisches Datenbankschema** besteht aus einer Reihe von hierarchischen Schemas. Jedes **hierarchische Schema** (oder **Hierarchie**) besteht aus einer Reihe von Datensatz- und PCR-Typen.

Abbildung 11.: Hierarchisches Schema

2.1.2. Eigenschaften eines hierarchischen Schemas

Ein hierarchisches Schema mit Datensatz- und PCR-Typen muss folgende Eigenschaften aufweisen:

1. Ein Datensatz, den man als **Wurzel** (root) des hierarchischen Schemas bezeichnet und der in keinem PCR-Typ als Kinddatensatztyp teilnimmt.

2. Jeder Datensatztyp, außer der Wurzel, nimmt als Kinddatensatztyp in genau einem PCR-Typ teil.

3. Ein Datensatztyp kann in einer beliebigen Anzahl (keine oder mehrere) PCR-Typen als Elterndatensatztyp teilnehmen.

4. Ein Datensatztyp, der in keinem PCR-Typ als Elterndatensatztyp teilnimmt, wird als **Blatt** (leaf) des hierarchischen Schemas bezeichnet.

5. Wenn ein Datensatztyp als Eltern in mehr als einem PCR-Typ teilnimmt, sind seine Kinddatensatztypen geordnet. Der Konvention zufolge wird die Ordnung in einem hierarchischen Diagramm von links nach rechts angezeigt.

Ein hierarchisches Schema wird als **Baumdatenstruktur** definiert. In der Terminologie von Baumstrukturen entspricht ein Datensatztyp einem **Knoten** und ein PCR-Typ einer **Kante** (oder **Bogen**) des Baumes.

Weiterhin gilt folgende Eigenschaft ➔ Es gibt keine Zyklen im Baum, d.h. keine Wege, für die gilt Anfangsdatensatztyp = Enddatensatztyp.

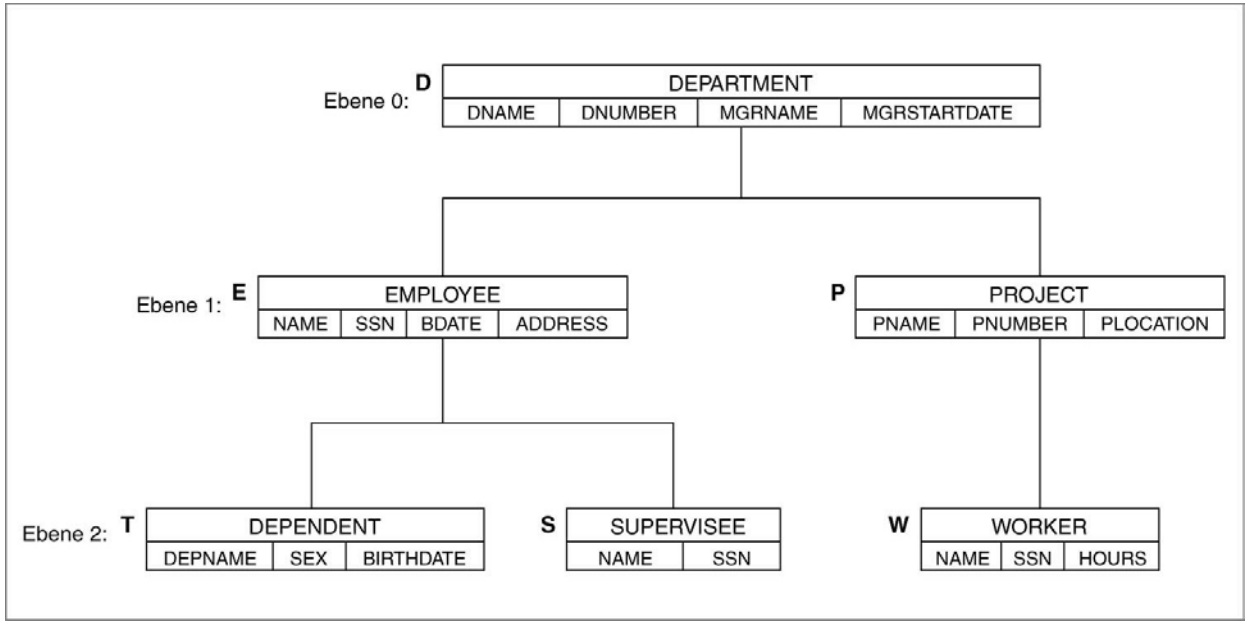


Abbildung 12.: Beispiel für ein hierarchisches Datenmodell

2.1.3. Integritätseinschränkungen im hierarchischen Modell

Bei der Spezifikation eines hierarchischen Schemas im hierarchischen Modell sind folgende Einschränkungen zu beachten:

1. Keine Datensatzinstanzen außer den Wurzeldatensätzen können ohne Beziehung zu einer Elterndatensatzinstanz existieren. Dies hat folgende Auswirkungen:

   a) Ein Kinddatensatz kann nur eingefügt werden, wenn er mit einem Elterndatensatz verknüpft ist.

   b) Ein Kinddatensatz kann unabhängig von seinem Elternteil gelöscht werden. Das Löschen eines Elternteils führt allerdings automatisch zum Löschen aller seiner Kind- und Nachkommendatensätzen.

   c) Die obigen Regeln gelten nicht für virtuelle Kind- und virtuelle Elterndatensätze.

2. Wenn ein Kinddatensatz zwei oder mehr Elterndatensätze des gleichen Datensatztyps hat, muss der Kinddatensatz einmal für jeden Elterndatensatz dupliziert werden.

3. Ein Kinddatensatz, der zwei oder mehr Elterndatensätze unterschiedlicher Datensatztypen hat, muss mindestens einen realen Elternteil haben, alle weiteren werden dann als **virtuelle Eltern** dargestellt. In IMS ist nur ein virtuelles Elternteil zulässig.

4. In IMS kann ein Datensatztyp der virtuelle Elternteil von nur einem virtuellen PCR-Typ sein. Das heißt, pro Datensatztyp ist nur ein **virtuelles Kind** zulässig.

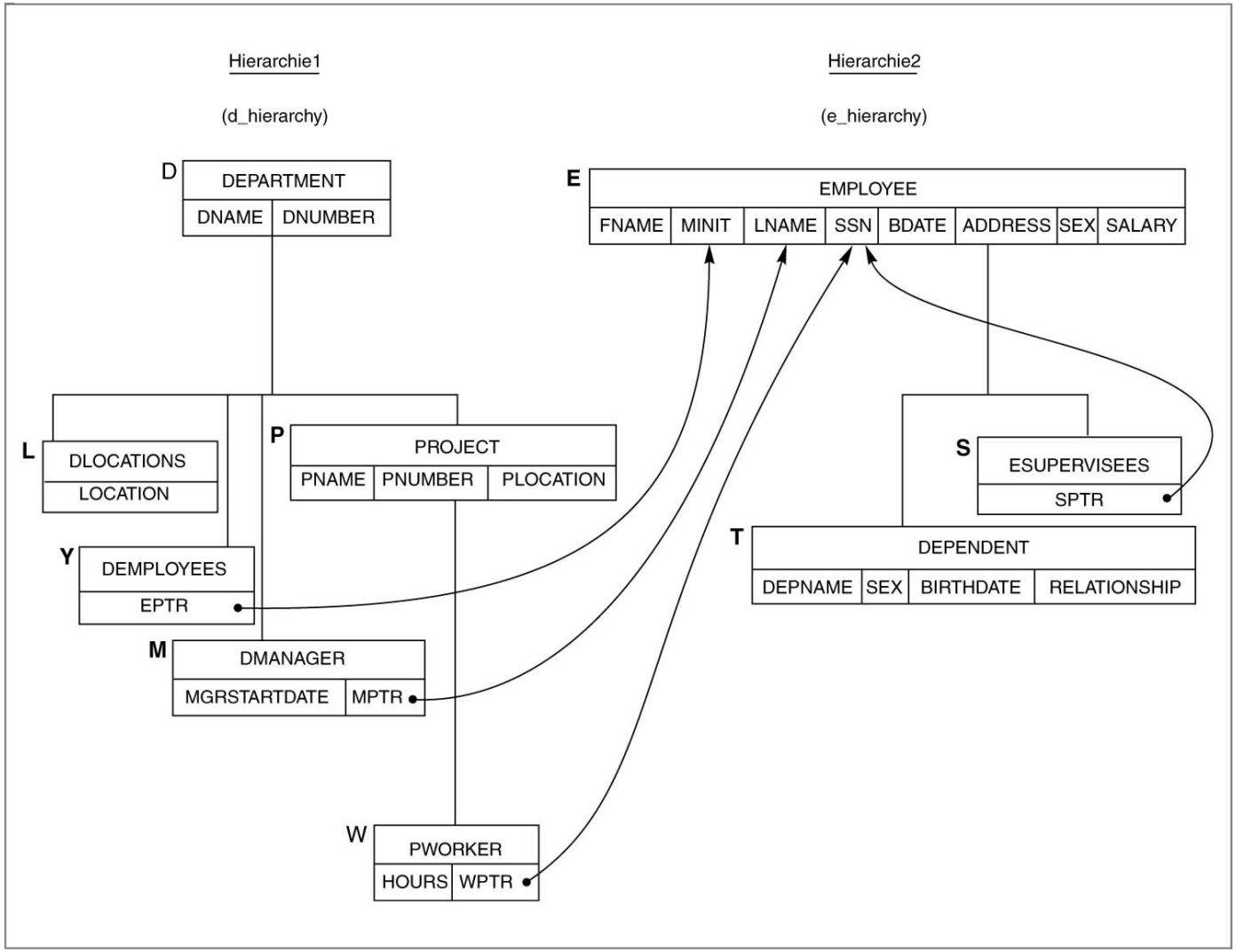


Abbildung 13.: Verwenden von vPCR-Typen zur Vermeidung von Redundanzen in der Datenbank

2.2. Netzwerkmodell

Das ursprüngliche Netzwerkmodell und die Sprache wurden im Bericht der ***CODASYL*** (***Co***nference on ***Da***ta ***Sy***stems ***L***anguage) ***D***ata ***B***ase ***T***ask ***G***roup (***DBTG***) 1971 vorgestellt. Deshalb wird es auch als ***DBTG***-Modell bezeichnet. Neuere Konzepte wurden in die revidierten Berichte von 1978 und 1981 eingepflegt.

Im ersten ***CODASYL/DBTG***-Bericht wurde die Programmiersprache ***COBOL*** als Wirtssprache (Host Language) benutzt.

Mit dem Bericht von 1978 erfolgte eine leichte Modifikation sowie die Einführung einer ***DSDL*** - ***D***ata ***S***torage ***D***escription ***L***anguage zur Beschreibung von internen Schemata.

Im Netzwerkmodell gibt es zwei grundlegende Datenstrukturen – Datensätze und Mengen:

2.2.1. Datensätze, Datensatztypen und Datensatzexemplare

Daten werden in **Datensätzen** gespeichert. Jeder **Datensatz** besteht aus einer Gruppe zusammenhängender **Datenwerte**. Datensätze werden nach **Datensatztypen** klassifiziert, wobei jeder **Datensatztyp** die Struktur einer Gruppe von Datensätzen beschreibt, in denen der gleiche Informationstyp gespeichert wird. Jeder Datensatztyp erhält einen Namen. Weiterhin wird jedem Datenexemplar (oder Attribut) im Datensatztyp ein Name und ein Format (**Datentyp**) zugeordnet:

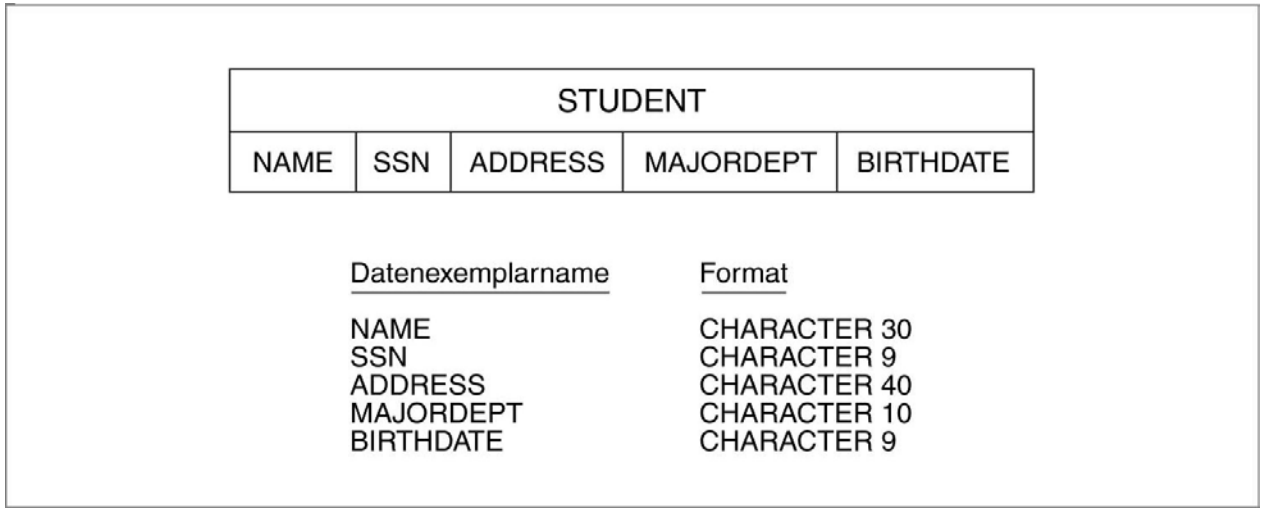


Abbildung 14.: Datensatztyp STUDENT mit den Datenexemplaren NAME, SSN, ADDRESS, …

2.2.2. Mengentypen und ihre grundsätzlichen Eigenschaften

Ein Mengentyp (Set Type) ist die Beschreibung einer eins-zu-viele-Beziehung (**1:m Beziehung**) zwischen zwei Datensatztypen, siehe Abbildung 15.

Diese Art der Darstellung in Diagrammform wird als **Bachman-Diagramm** bezeichnet. Jede Mengentypdefinition besteht aus drei Basiselementen:

- Name des Mengentyps
- Ein Eigentümerdatensatztyp
- Ein Mitgliederdatensatztyp

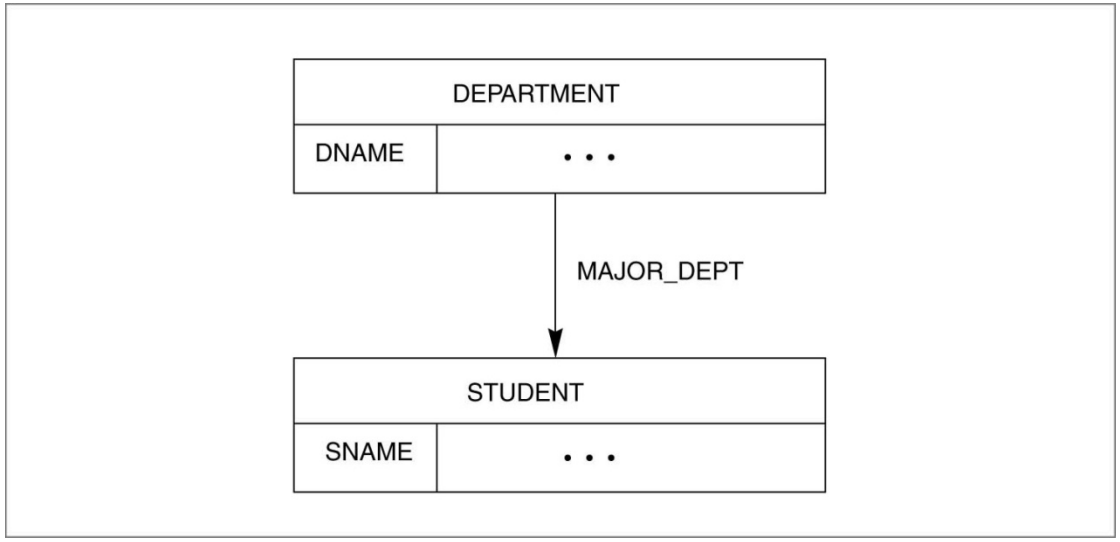


Abbildung 15.: Mengentyp MAJOR_DEPT

Der Mengentyp in Abbildung 15 ist **MAJOR_DEPT**, **DEPARTMENT** ist der **Eigentümerdatensatztyp** und **STUDENT** ist der **Mitgliederdatensatztyp**. Dies ist die eins-zu-viele-Beziehung (**1:m Beziehung**) zwischen akademischen Departments und Studenten mit Hauptfach in diesen Departments.

In der Datenbank selbst gibt es viele Mengenelemente (oder Mengeninstanzen), die einem Mengentyp entsprechen. Jede Instanz bezieht sich auf einen Datensatz des Eigentümerdatensatztyps – im Beispiel ein **DEPARTMENT** Datensatz – auf die Datensatzmenge des entsprechenden Mitgliedsdatensatztyps – die Menge von **STUDENT** Datensätzen für Studenten mit Hauptfach im entsprechenden Department.

Folglich setzt sich jedes Mengenelement wie folgt zusammen:

- Ein Eigentümerdatensatz des Eigentümerdatensatztyps
- Eine Reihe zusammenhängender Mitgliedsdatensätze (keiner oder mehrere) des Mitgliedsdatensatztyps

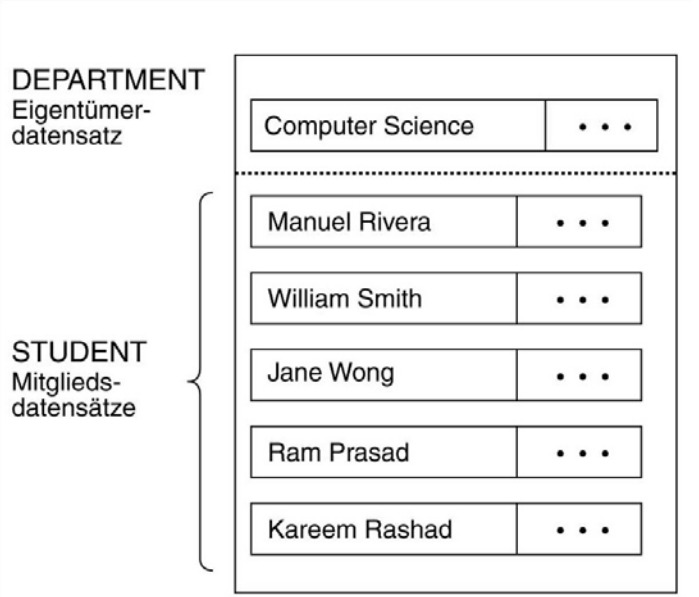


Abbildung 16.: Mengeninstanz

Im Netzwerkmodell sind die Mitgliedsdatensätze einer Mengeninstanz geordnet, während die Ordnung von Elementen in einer mathematischen Menge keine Rolle spielt. Es kann also auf den ersten, zweiten, i-ten Mitgliedsdatensatz einer Mengeninstanz Bezug genommen werden.

In Abbildung 17. ist eine alternative Darstellung der Mengeninstanz aus Abbildung 16. als **einfach verkette Liste** enthalten. Der Datensatz von „Manuel Rivera“ ist der erste **STUDENT** Mitgliedsdatensatz in der Menge „Computer Science“ und der von „Kareem Rashad“ ist der letzte Mitgliedsdatensatz.

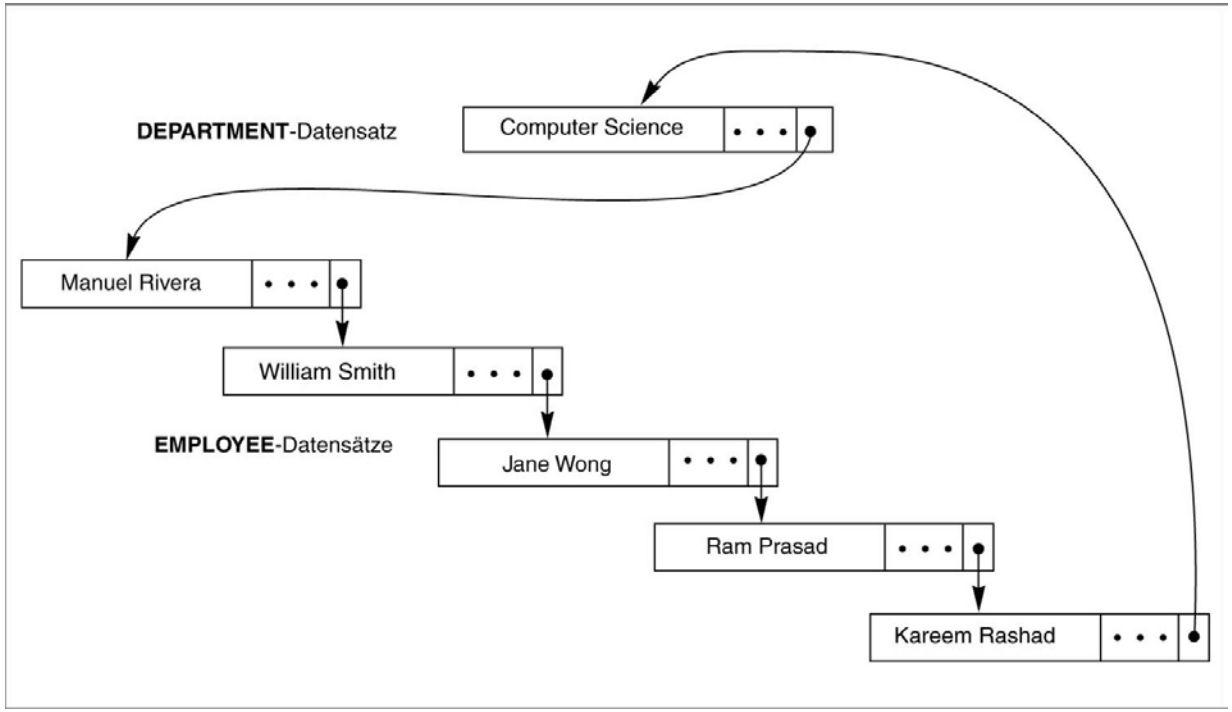
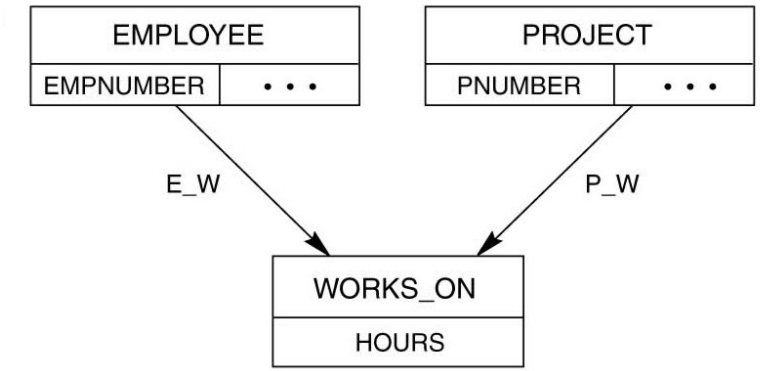


Abbildung 17.: Alternative Darstellung einer Mengeninstanz als verkettete Liste

2.2.3. Darstellung von m:n Beziehungen mittels Mengen

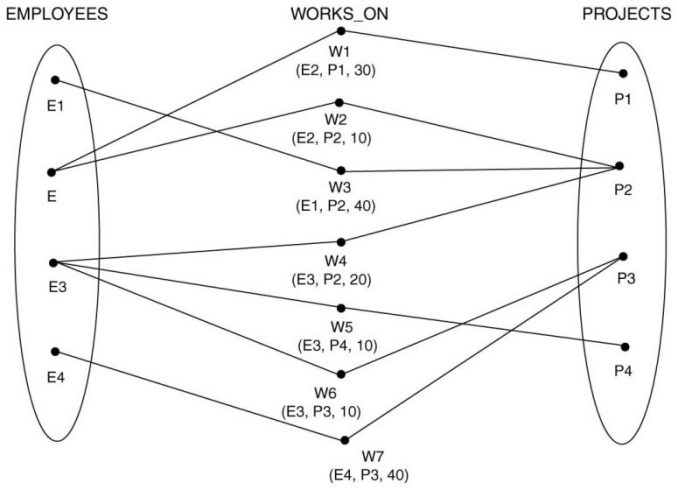
Ein Mengentyp stellt eine **1:m Beziehung** zwischen zwei Datensatztypen dar. Dies bedeutet, dass ein Datensatz des Mitgliedsdatensatztyps nur in einer Mengeninstanz vorkommen kann. Diese Einschränkung wird beim Netzwerkmodell automatisch vom **Datenbankmanagementsystem** umgesetzt.

Um eine **1:1 Beziehung** darzustellen, muss die 1:1 Einschränkung zusätzlich vom Anwendungsprogramm auferlegt werden.



Eine **m:n Beziehung** zwischen zwei Datensatztypen kann nicht durch einen einzigen Mengentyp dargestellt werden. Betrachtet man die Beziehung **WORKS_ON** in Abbildung 18 zwischen **EMPLOYEE** und **PROJECT**, so kann festgestellt werden, dass ein Angestellter (**EMPLOYEE**) an mehreren Projekten gleichzeitig arbeiten kann, und dass an einem Projekt (**PROJECT**) ebenfalls mehrere Angestellte arbeiten können.

Abbildung 18.: verkettender Datensatztyp WORKS_ON



Die korrekte Methode für die Darstellung einer **m:n Beziehung** im Netzwerkmodell ist die Verwendung zweier Mengentypen (**EMPLOYEE** und **PROJECT**) und eines zusätzlichen Datensatztyp (**WORKS_ON**). Dieser zusätzliche Datensatztyp wird als verkettender Datensatztyp. Jeder Datensatz vom Datensatztyp **WORKS_ON** muss durch die **E_W** Menge zu einem **EMPLOYEE** Datensatz und durch die **P_W** Menge zu einem **PROJECT** Datensatz gehören. Er dient der Herstellung einer Beziehung.

Abbildung 19.: Mengendarstellung der m:n Beziehung

Während in Abbildung 19 eine Mengendarstellung der **m:n Beziehung** enthalten ist, ist in Abbildung 20 eine verkettete Darstellung der gleichen Beziehung enthalten.

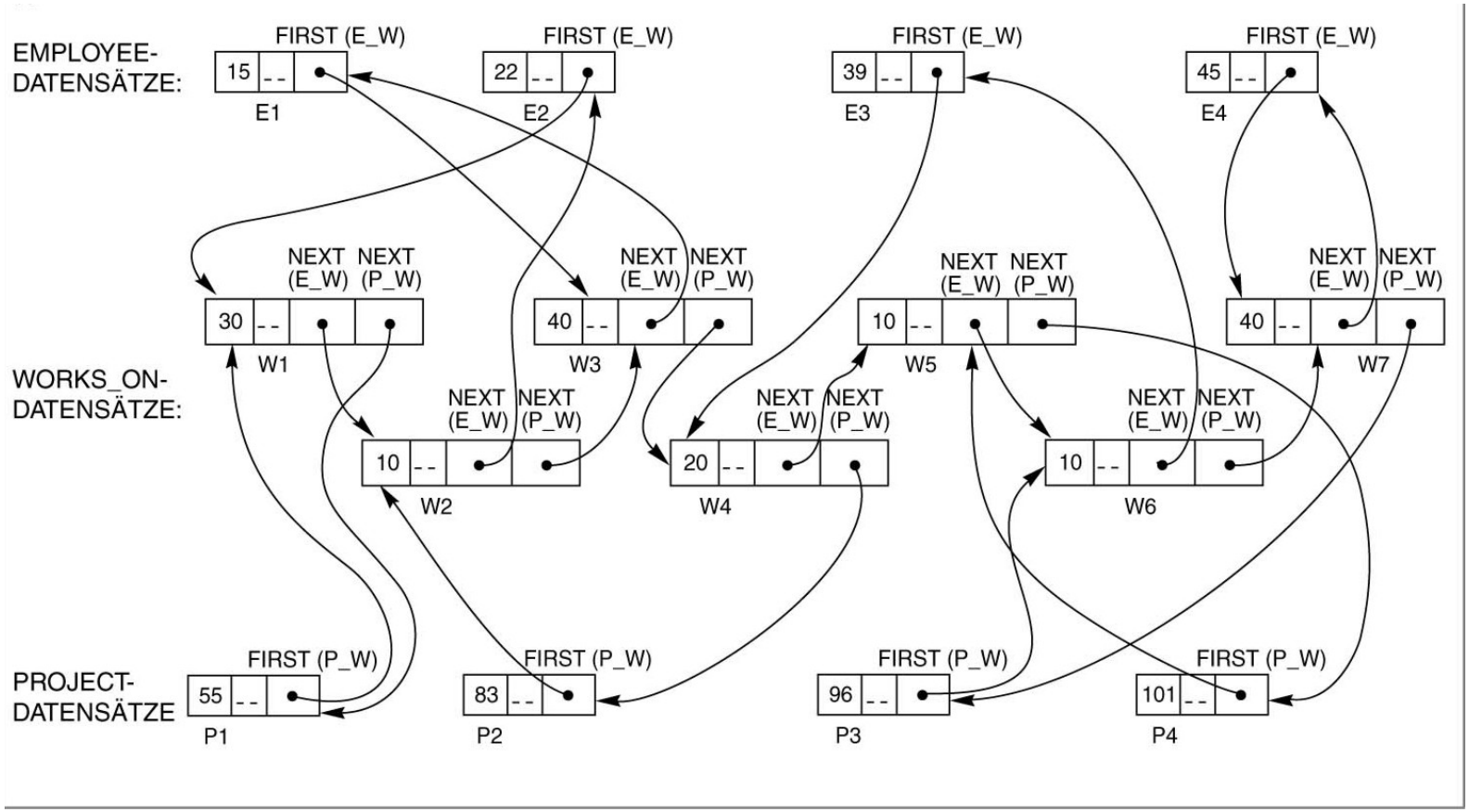


Abbildung 20.: verkettete Darstellung der m:n Beziehung

2.3. Objektorientiertes Datenmodell

Der mit **OO** abgekürzte Begriff objektorientiert hat seine Ursprünge in **OO-Programmiersprachen** (**OOPL**s). Heute werden OO-Konzepte in den Bereichen Datenbanken, Software-Engineering, Wissensbasen, künstliche Intelligenz und Computersysteme im Allgemeinen angewandt. Die in den siebziger Jahren entwickelte Programmiersprache **Smalltalk** war eine der ersten Sprachen, die ausdrücklich zusätzliche **OO-Konzepte** enthielt, wie z.B. Nachrichten (Message Passing) und Vererbung. Sie gilt als reine OO-Programmiersprache, was bedeutet, dass sie ausdrücklich auf dem objektorientierten Entwurf basiert. Dies steht im Gegensatz zu hybriden OO-Programmiersprachen, die OO-Konzepte in eine bereits bestehende Sprache integrieren. Ein typisches Beispiel hierfür ist **C++**, wo OO-Konzepte in die Programmiersprache **C** integriert wurden.

2.3.1. Komplexe Objekte

Ein Objekt hat normalerweise zwei Komponenten – einen **Zustand** (**Werte** bzw. **Attribute**) und ein **Verhalten** (**Operationen** bzw. **Methoden**) und wird über einen „Objektnamen“ angesprochen. Folglich ähnelt es im gewissen Sinne einer Variablen in einer Programmiersprache, außer dass es generell eine komplexe Datenstruktur und spezifische Operationen hat, die vom Programmierer definiert werden. In OOPL existieren Objekte nur während der Programmausführung und werden daher als **transistente Objekte** bezeichnet. Ein **OO-Datenbankmanagementsystem** kann die Existenz von Objekten durch deren permanente Speicherung erweitern, so dass es **persistente Objekte** sind, d.h. über die Programmbeendigung hinaus existieren und später von anderen Programmen benutzt werden können.

Abbildung 21.: Objekt

Eines der Ziele von OO-Datenbankmanagementsystemen ist die Wahrung einer direkten Entsprechung zwischen Objekten der realen Welt und der Datenbank, so dass Objekte nicht ihre Integrität und Identität verlieren sowie leicht identifiziert und verwendet werden. Folglich bieten OO-Datenbankmanagementsysteme einen eindeutigen, vom System generierten **Objektidentifikator** (**OID**) für jedes Objekt. Die Identifikation eines Objektes muss systemweit (oder mindestens datenbankweit) eindeutig und unveränderbar sein. Insbesondere muss sie unabhängig vom Ort der Speicherung bleiben.

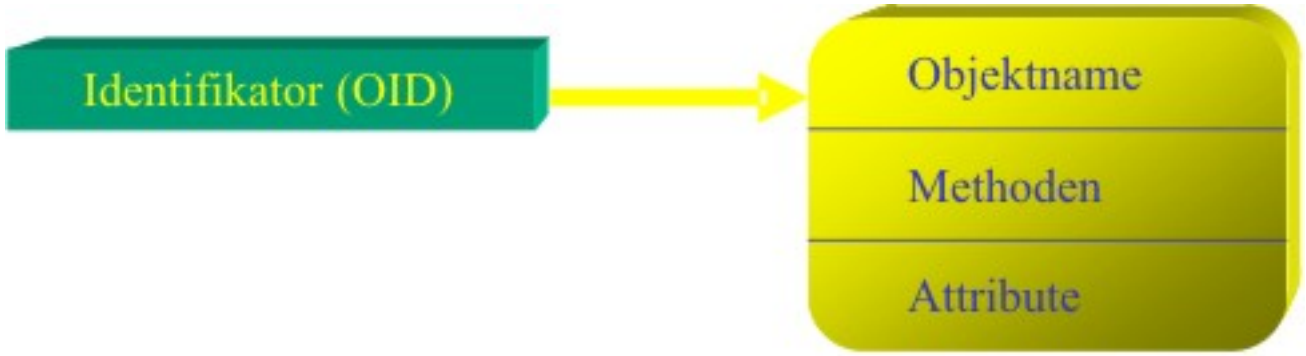


Abbildung 22.: eindeutige Identifikation der Objekte

Als weiteres Merkmal von OO-Datenbankmanagementsystemen können Objekte eine ***Objektstruktur beliebiger Komplexität*** haben, um alle erforderlichen Daten aufnehmen zu können, die das Objekt beschreiben.

Demgegenüber sind Informationen über ein komplexes Objekt in (objekt-)**relationalen Datenbankmanagementsystemen** oft über viele Relationen oder Datensätze verstreut, was zum Verlust der direkten Entsprechung zwischen Objekten der realen Welt und ihrer Darstellung in der Datenbank führt.

Abbildung 23.: komplexe Objektstrukturen

2.3.2. Datenkapselung, Typen- und Klassenhierarchien

Objektorientierte Systeme erlauben die Definition von Operationen oder Funktionen (Verhalten), die auf Objekte eines bestimmten Typs angewandt werden können. Einige OO-Modelle bestehen darauf, dass alle Operationen, die ein Benutzer auf ein Objekt anwenden kann, vordefiniert sein müssen. Dies erzwingt die **Kapselung** von Objekten.

Zum Zwecke der Kapselung wird eine Operation in zwei Teilen definiert. Der erste als **Signatur** oder **Schnittstelle** der Operation bezeichnete Teil spezifiziert der Operationsnamen und Argumente (Parameter).

Der zweite, als **Rumpf** oder **Body** bezeichnete Teil spezifiziert die Implementierung der Operation. Operationen können dadurch aufgerufen werden, dass eine Nachricht (**Message**) an ein Objekt gesandt wird, die den Operationsnamen und die Parameter enthält. Das Objekt führt dann die Methode der betreffenden Operation aus. Diese Verkapselung erlaubt die Modifikation der internen Struktur eines Objektes sowie die Implementierung ihrer Operationen, ohne dass die externen Programme, die diese Operationen aufrufen, in ihrer Funktion beeinträchtigt werden. Folglich bietet Kapselung eine gewisse Unabhängigkeit der Daten und Operationen.

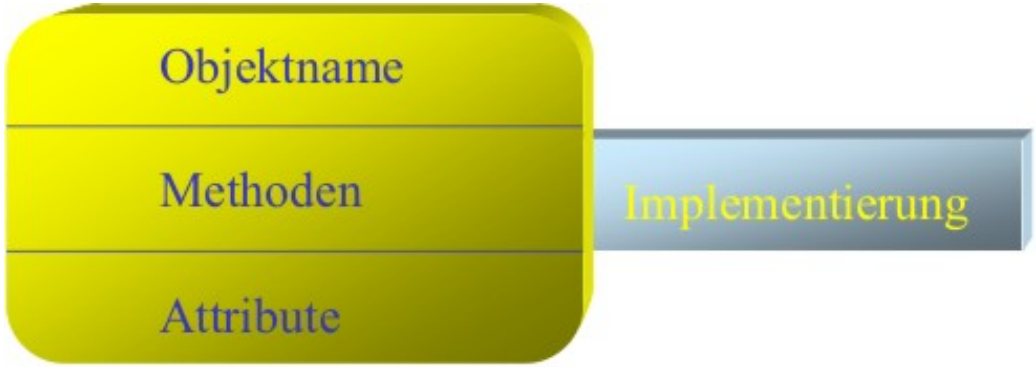


Abbildung 24.: Datenkapselung

Jedes **OO-Datenbankmanagementsystem** sollte **Typen und Klassen** unterstützen. Unter Typen sind klassische Datentypen aus Programmiersprachen zu verstehen, die zur Übersetzungszeit einer Typenprüfung unterworfen sind. Bei einem OO-Datenbankmanagementsystem beschreibt der Typ lediglich den strukturellen Teil einer Klasse.

Objekte mit äquivalenter Struktur und gleichem Verhalten werden zu Klassen zusammengefasst. Klassen werden benutzt, um Instanzen zur Laufzeit generieren zu können. Ein Klassenkonzept ist somit meistens dynamischer als ein Typkonzept.

Abbildung 25.: Zuordnung von Objekten zu Klassen

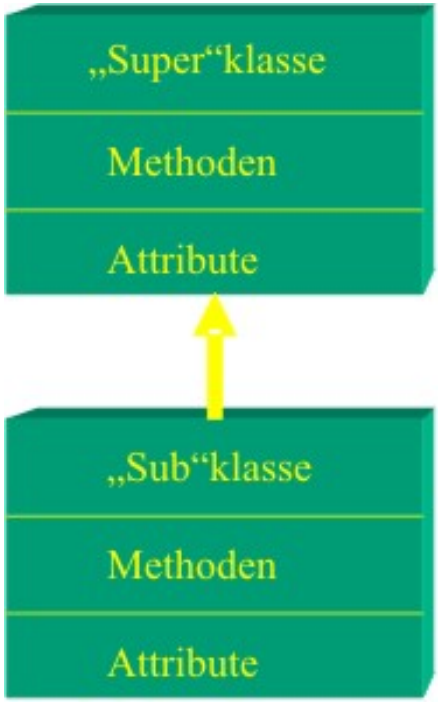


Abbildung 26.: Vererbung

2.3.3. Vererbung und Polymorphismus

Vererbung von Struktur und Verhalten muss durch das OO-Datenbankmanagementsystem ermöglicht werden. Dies erlaubt die Spezifikation neuer Typen oder Klassen, die einen Großteil ihrer Struktur und Operationen von zuvor definierten Typen oder Klassen erben. Die Spezifikation von Objekten kann also systematisch verlaufen. Dies vereinfacht die inkrementelle Entwicklung der Datentypen eines Systems und die Wiederverwendung existierender Typdefinitionen bei der Erstellung neuer Objekttypen.

Eine Subklasse (**Unterklasse**) in einer Klassenhierarchie kann Eigenschaften (Attribute und Methoden) von einer übergeordneten Klasse (**Oberklasse**) erben. Daneben darf die Unterklasse eigene Eigenschaften, seien dies Attribute oder Methoden, hinzufügen.

Dieselben Methodennamen können auf Objekte unterschiedlicher Klassen vielgestaltig (**polymorph**) angewendet werden, obwohl die entsprechenden Methoden je nach Klassenzugehörigkeit abweichende Implementierungen aufweisen. In einer solchen Situation kann sich ein Operationsname (Methode) je nach den betroffenen Objekttypen auf mehrere unterschiedliche Implementierungen beziehen.

Ist zur Übersetzungszeit die notwendige Implementierung festgelegt, so kann statisch gebunden werden; andernfalls muss zur Laufzeit der auszuführende Code bestimmt werden, was als spätes Binden bezeichnet wird.

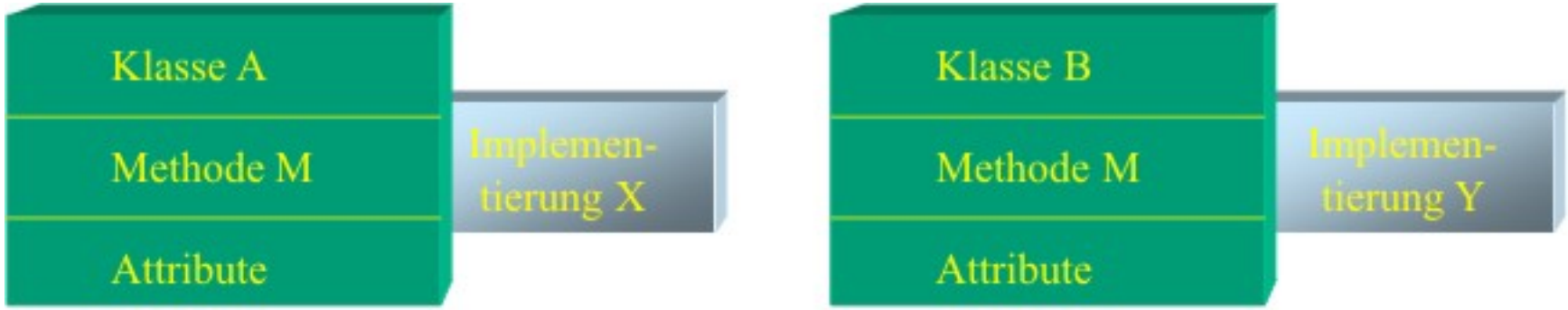


Abbildung 27.: Polymorphismus

2.4. Relationales Datenmodell

Als Vater des relationalen Datenmodells gilt ***E. F. Codd***. Er veröffentlichte 1970 einen Artikel in der Zeitschrift ***Communications of the ACM***: „A Relational Model of Data for Large Shared Data Banks“, dem dann noch weitere Artikel folgten.

Dieses Datenmodell hat viele Diskussionen ausgelöst, war Gegenstand zahlreicher wissenschaftlicher Arbeiten und hat für die Datenbanktechnologie eine sprunghafte Entwicklung ausgelöst. In erster Linie werden seine Einfachheit und Flexibilität dafür verantwortlich gemacht.

2.4.1. Mathematische Grundlagen

Für die Behandlung des relationalen Datenmodells und auch darüber hinaus werden einige mathematische Grundbegriffe benutzt.

Definition 9 - Menge:

Eine Menge als Zusammenfassung von bestimmten, wohlunterschiedenen Objekten zu einem Ganzen ist ein fundamentaler Begriff in der Mathematik:

$$A = \{a_1, a_2, a_3, \ldots, a_n\}$$

und nennen die a_i, die durch Komma getrennt aufgezählt werden, Elemente der Menge A. Also ist z.B. a_2 Element der Menge A, d.h. $a_2 \in A$.

In der Menge spielt die Reihenfolge der Elemente keine Rolle, und ebenso ist wichtig, dass es keine Elementeswiederholung gibt.

Auf dieser Grundlage lässt sich dann auch das kartesische Produkt zweier Mengen wie folgt definieren:

Definition 10 - kartesisches Produkt:

Seien a und b beliebige Elemente. Dann bezeichnet (a,b) ein geordnetes Paar, bestehend aus erster und zweiter Koordinate.

Das kartesische Produkt zweier Mengen A und B ist die Menge aller geordneten Paare, deren erste Koordinate ein Element von A und deren zweite Koordinate Element von B ist.

$$A \times B = \{(a,b) \mid a \in A \text{ und } b \in B\}$$

Zur Demonstration seien zwei reduzierte Mengen aus der **ORACLE®**-Beispieldatenbank eingeführt:

$$A = \{\text{Hartstein, Higgins, Hunold, Ernst}\}$$
$$B = \{\text{Marketing, Accounting, IT}\}$$

Eine Menge hat 4 Elemente und die zweite Menge 3 Elemente. Entsprechend der Definition ergibt sich das kartesische Produkt aus der Menge aller geordneten Paare:

$$A \times B = \{(\text{Hartstein, Marketing}),$$
$$(\text{Hartstein, Accounting}),$$
$$(\text{Hartstein, IT}),$$
$$(\text{Higgins, Marketing}),$$
$$(\text{Higgins, Accounting}),$$
$$(\text{Higgins, IT}),$$
$$(\text{Hunold, Marketing}),$$
$$(\text{Hunold, Accounting}),$$
$$(\text{Hunold, IT}),$$
$$(\text{Ernst, Marketing}),$$
$$(\text{Ernst, Accounting}),$$
$$(\text{Ernst, IT})\}$$

Die Anzahl der Elemente, genannt ***Kardinalität*** oder Mächtigkeit einer Menge, der resultierenden Menge ergibt sich zu:

$$\text{card}\ (A \times B) = \text{card}(A) \bullet \text{card}(B)$$

Im Beispiel bedeutet das, die Kardinalität $\text{card}(\text{A} \times \text{B}) = \text{card}(\text{A}) * \text{card}(\text{B}) = 3 * 4 = 12$. D.h., die Ergebnismenge des kartesischen Produktes sind 12 geordnete Paare.

Definition 11 - Relation:

Eine Relation zwischen zwei Mengen A und B ist eine Teilmenge R des kartesischen Produkts der beiden Mengen.

$$R \subseteq A \times B$$

ist eine zweistellige Relation, da zwei Mengen bei der Bildung beteiligt sind.

Auf dem Begriff der Relation aufbauend, kann an dem eingeführten Beispiel die Relation wie folgt gebildet werden, die der modellierten Realität entspricht:

$$\begin{aligned} \text{R} \subseteq \text{A} \times \text{B} = \ & \{(\text{Hartstein, Marketing}), \\ & (\text{Higgins, Accounting}), \\ & (\text{Hunold, IT}), \\ & (\text{Ernst, IT})\} \end{aligned}$$

Es kann verallgemeinert werden:

n-stellige Relation:

$$R \subseteq M_1 \times M_2 \times M_3 \times \ldots \times M_n$$

Die Menge R besteht dann nicht aus geordneten Paaren, sondern aus geordneten n-Tupeln:

$$R = \{(x_1, x_2, x_3, \ldots, x_n) \mid x_i \in M_i \text{ und } 1 \leq i \leq n\}$$

Betrachtet man noch einmal die zweistellige Relation $R \subseteq A \times B$. Da kann gesagt werden:

Eine zweistellige Relation $R \subseteq A \times B$ ist eine Abbildung aus der Menge A in die Menge B. Durch die geordneten Paare wird dem Element a das Element b zugeordnet.

Für die Abbildung sind mathematisch so genannte ***Zuordnungstypen***, in der Datenbanktheorie als ***Beziehungstypen*** bezeichnet, bekannt, die in Abbildung 28 veranschaulicht werden:

Da Beziehungstypen die gegenseitige Zuordnung beschreiben, sind sie die Kombination aus zwei gegenseitigen ***Assoziationstypen***. Es sind bei der Betrachtung der Beziehungen zwischen den einzelnen Elementen, im einfachsten Fall, zweier Mengen, die Standpunkte ausgehend von jeweils einer Menge anzugeben:

Assoziationstyp	***Bedeutung***
1 - einfache Assoziation	jedem Element aus A ist ein Element aus B zugeordnet
c - konditionelle Assoziation	jedem Element aus A ist kein, oder ein Element aus B zugeordnet
m - multiple Assoziation	jedem Element aus A ist mindestens ein Element aus B zugeordnet
mc - multiple konditionelle Assoziation	jedem Element aus A ist kein, oder mindestens ein Element aus B zugeordnet

Tabelle 2.: Assoziationstypen

Bezüglich der Abbildung eines Elementes einer Menge A in die Menge B gibt es folgende Möglichkeiten:

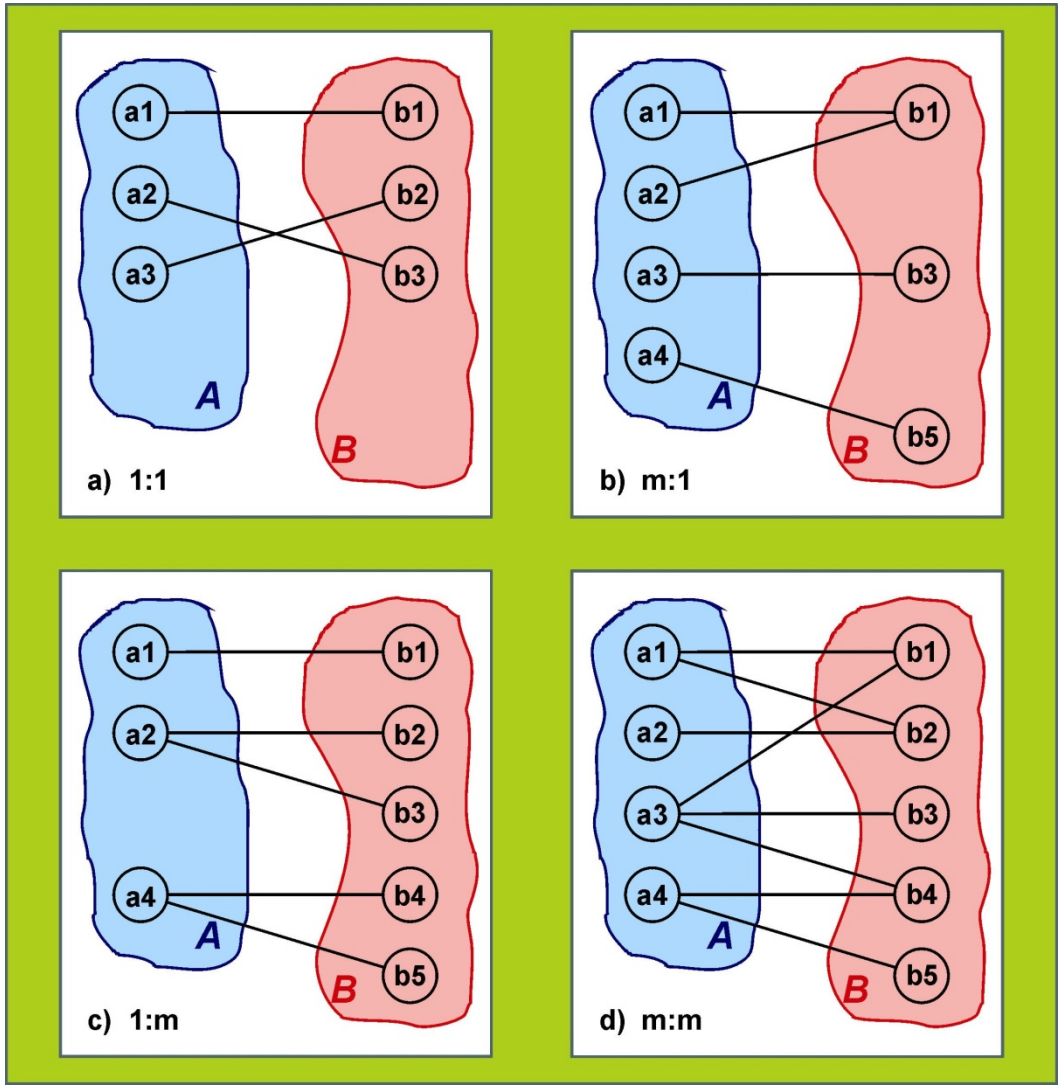


Abbildung 28.: Beziehungstypen

Werden diese Assoziationstypen, die nur eine Richtung der Abbildung beschreiben, verwendet, die Beziehungstypen zu definieren, so erhält man folgende Übersicht:

Beziehungstypen					***Bedeutung***
	1	**c**	**m**	**mc**	
1	**1:1**	**c:1**	**m:1**	**mc:1**	**hierarchische Beziehung**
c	**1:c**	**c:c**	**m:c**	**mc:c**	**konditionelle Beziehung**
m	**1:m**	**c:m**	**m:m**	**mc:m**	**netzförmige**
mc	**1:mc**	**c:mc**	**m:mc**	**mc:mc**	**Beziehung**

Tabelle 3.: Beziehungstypen

***Achtung:* netzförmige Beziehungen können nicht im relationalen Datenmodell abgebildet werden!!!**

Für die weiteren Betrachtungen soll noch der Begriff ***Funktion*** definiert werden

Definition 12 - Funktion:

Eine Funktion F ist eine spezielle Abbildung, nämlich eine eindeutige Abbildung aus der Menge A in die Menge B:

$$F = \{(a,b) \mid a \in A \text{ und } b \in B\}$$

Eindeutigkeit heißt, dass aus

$(a,b_1) \in F$ und $(a,b_2) \in F$ folgt $b_1 = b_2$

Mit folgender Schreibweise:

$F : A \rightarrow B$

A wird *Definitionsbereich* und B der *Wertebereich* von F genannt. B ist funktional abhängig von A.

2.4.2. Terminologie und Notation

Für das weitere Verständnis müssen weitere Begriffe eingeführt werden. Diese lassen sich aus dem Entwurfsprozess ableiten und sind ihrerseits Bestandteil des ***Entity-Relationship-Modells***:

Definition 13 - Entität:

Eine Entität (entity) ist ein individuelles und identifiziertes (Eindeutigkeit) Exemplar von Dingen, Personen oder Begriffen der realen oder der Vorstellungswelt.

Die Informationen über das Objekt sind Beschreibungen ausgewählter ***Merkmale*** des Objektes. Ein Objektmerkmal wird durch ein ***Attribut*** repräsentiert.

Definition 14 - Attribut:

Das Attribut ist durch einen *Namen* und einen *Wert* innerhalb eines bestimmten *Wertebereichs* charakterisiert.

Für eine Entität, die durch n Attribute charakterisiert sein soll, existieren damit n Wertebereiche:

$$W_1(A_1), W_2(A_2), W_3(A_3), ..., W_n(A_n) .$$

Da diese Wertebereiche Mengen (von Werten) sind, ergibt sich als Menge der möglichen Wertekombinationen das kartesische Produkt

$$W_1(A_1) \times W_2(A_2) \times W_3(A_3) \times ... \times W_n(A_n) ,$$

und Sinnvollerweise wird zugelassen, dass auch nur eine Teilmenge dieses Produktes möglich ist:

$$R \subseteq W_1(A_1) \times W_2(A_2) \times W_3(A_3) \times ... \times W_n(A_n) .$$

Damit ist R eine n-stellige Relation zwischen n Wertebereichen. ***E. F. Codd*** nannte diese Wertebereiche ***domains***, so dass in einigen Veröffentlichungen eingedeutscht auch der Begriff ***Domäne*** auftaucht und zumeist eingebürgert ist.

Schreibweise	***Begriff***
A_i	Attribut (Eigenschaft, Merkmal)
$W_i(A_i)$	Wertebereich des Attributes
$a_i \in W_i(A_i)$	Wert des Attributs A_i
$(a_1, a_2, a_3, ..., a_n)$	Tupel
$(A_1, A_2, A_3, ..., A_n)$	Tupeltyp
R	Name der Relation
$R(A_1, A_2, A_3, ..., A_n)$	Relation, basierend auf Tupeltyp

Tabelle 4.: Begriffe und Schreibweisen für das relationale Datenmodell

Eine n-stellige Relation ist eine Menge von geordneten n-Tupeln. Daraus folgt, dass eine ***Entität*** durch eine konkrete Wertebelegung, d.h. einen ***Wert*** je ***Attribut***, charakterisiert wird:

$$r = (a_1, a_2, a_3, ..., a_n) ; r \in R .$$

Tabelle 4 fasst die bisher eingeführten Begriffe und Schreibweisen zusammen. Damit ist der grundlegende ***Coddsche Ansatz*** eingeführt und damit die Namensgebung für das ***relationale Datenmodell***:

Demzufolge wird die Relation als eine ***Tabelle*** betrachtet. Als Spaltenüberschriften dieser Tabelle fungieren die ***Attributnamen***. Eine Bezeichnung der Zeilen ist im relationalen Datenmodell nicht vorgesehen.

Aufgrund dieser anschaulichen Interpretation des relationalen Ansatzes werden häufig auch diese an der Tabelle orientierten Begriffsbildungen verwendet. Tabelle 5 soll als Orientierungshilfe im Sprachgebrauch zum relationalen Datenmodell dienen.

Relation	Tabelle
Relationsname	Tabellenname
Tupel	Zeile (row, record)
Attribut	Spalte (column)
Wertebereich des Attributes	Wertebereich der Eigenschaft

Tabelle 5.: Unterschiedliche Begriffe zum relationalen Datenmodell

Es hat sich aber auch eine noch einfachere, dem mathematischen Hintergrund versteckende Form der Behandlung des relationalen Datenmodells verbreitet. Abbildung 29 zeigt die Basis dieses gedanklichen Ansatzes:

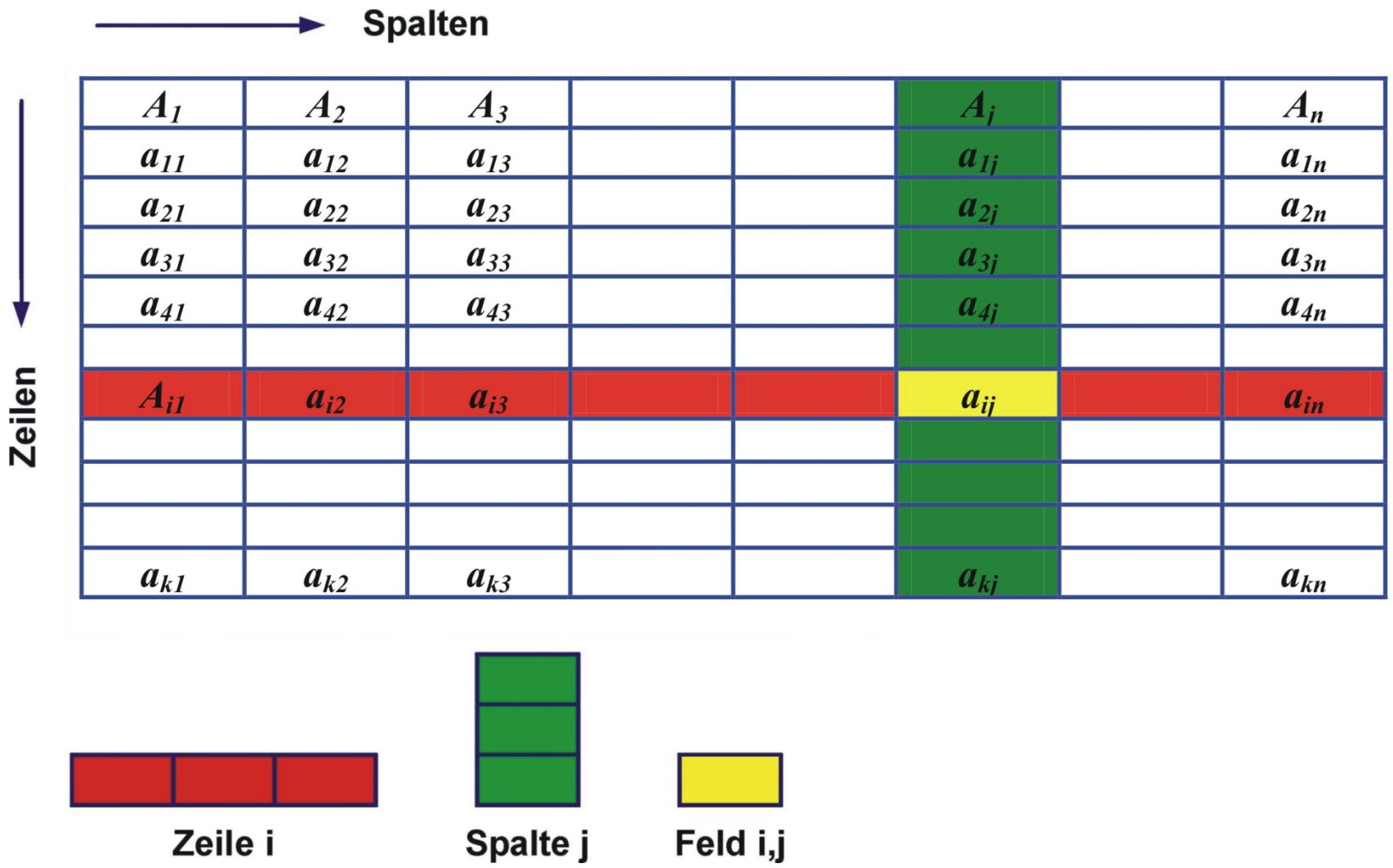


Abbildung 29.: Relation als Tabelle

In Mengen gibt es keine Wiederholungen. Daraus folgt, dass in einer Relation jedes Tupel in bestimmtem Sinne „***eindeutig“*** ist und auch als solches identifizierbar sein muss.

Es muss also ein Attribut oder allgemeiner mehrere Attribute (Attributkombination) geben, deren Wertebelegung die eindeutige Identifikation des Tupels und damit seiner Unterscheidung von anderen Tupeln gestattet.

Definition 15 - identifizierende Attributkombination:

Es seien gegeben:
$R(A_1, A_2, A_3, ..., A_n)$ **und eine Attributkombination**
$(A_{k1}, A_{k2}, A_{k3}, ..., A_{km})$ **mit** $(1 \le m \le n)$ **und**
$(1 \le k1 \le k2 \le k3 \le ... \le n)$
d.h. gleiche Ordnung wie im Tupel $(A_1, A_2, A_3, ..., A_n)$ **.**
Diese Attributkombination heißt identifizierende Attributkombination, wenn für je zwei r_i **und** r_j $(r_i, r_j \in R)$ **gilt.**
Wenn $(a_{ik1}, a_{ik2}, a_{ik3}, ..., a_{ikm}) = (a_{jk1}, a_{jk2}, a_{jk3}, ..., a_{jkm})$ **dann** $r_i = r_j$ **.**

Definition 16 - Schlüssel:

Ist eine identifizierende Attributkombination minimal in dem Sinne, dass das Weglassen eines ihrer Attribute die Identität zwischen Tupeln r_i und r_j nicht mehr folgt, dann wird sie Schlüssel genannt.

Es kann mehrere Schlüssel bezüglich einer Relation geben, so dass es Aufgabe der Datenmodellierung ist, einen dieser Schlüssel als ***Primärschlüssel*** auszuweisen.

Für die Schreibweise eines Primärschlüssels wird die Unterstreichung bei der Relationendarstellung verwendet:

$$R(\underline{A_1, A_2}, A_3, ..., A_n),$$

mit der Attributkombination (A_1, A_2) als Primärschlüssel. Es hat sich aber auch eine alternative Schreibweise herausgebildet, die den gleichen Sachverhalt dokumentiert:

$$R(\#A_1, \#A_2, A_3, ..., A_n).$$

Die Primärschlüssel sind die entscheidenden Elemente bei der Modellierung von Beziehungsmengen im relationalen Datenmodell. Betrachtet man die Entitätstypen R und S und deren Darstellung als Relationen

$$R(\#A_1, \#A_2, A_3, ..., A_n) \text{ und } S(\#B_1, B_2, B_3, ..., B_m).$$

Besteht zwischen beiden eine Beziehung, die als Beziehungstyp T modelliert werden soll, dann ergibt sich:

$$T(\#A_1, \#A_2, \#B_1)$$

oder, falls die Beziehung durch bestimmte Eigenschaften (Attribute) charakterisiert ist:

$$T(\#A_1, \#A_2, \#B_1, C_1, C_2, C_3, ..., C_k).$$

Aus dem Genannten abgeleitet kann festgestellt werden:

- ➔ Die Zuordnung der Elemente erfolgt über die Primärschlüssel, die ja zur eindeutigen Identifikation der Elemente ausgewiesen sind.

- ➔ Die Beziehungsmenge wird eine Relation des relationalen Schemas.

- ➔ Der Primärschlüssel dieser Relation ist die Vereinigung der Attributkombinationen, die Primärschlüssel der beteiligten Relationen sind.

Damit sind die Grundlagen des relationalen Datenmodells bekannt und können zusammengefasst werden:

- ➔ Das ***relationale Datenmodell*** stellt für die Datenmodellierung genau einen Strukturtyp zur Verfügung, die ***Relation***.

- ➔ ***Entitätsmengen*** und ***Beziehungsmengen*** sind demnach als Relationen darzustellen.

- ➔ ***Attribute*** der Entitäts- und Beziehungsmengen werden ***Attribute der Relation***.

- ➔ Für jede Relation ist ein Primärschlüssel zu definieren.

Da Relationen Mengen im mathematischen Sinne sind, gibt es kein Mehrfachauftreten von Tupeln und keine Ordnung zwischen den Tupeln.

2.4.3. Codd´sche Regeln

Nach Festlegung von ***E. F. Codd*** sind Regeln formulierbar, nach deren Bewertung jedes Datenbankmanagementsystem validierbar ist. Die folgenden 12 Regeln müssen erfüllt sein, wenn ein DBMS das relationale Datenmodell unterstützen soll.

1. ***Informationsregel:***
   Alle Informationen werden in der Form von ***zweidimensionalen Tabellen*** dargestellt.

2. ***Garantierter Zugriff:***
   Auf jeden atomaren Wert der ***Datenbank*** kann mittels ***Tabellenname***, ***Spaltenname*** und Wert des Identifikationsschlüssels (***Primärschlüssel***) zugegriffen werden. Es wird nicht über Speicheradressen gearbeitet, sondern ***wirklich nur*** über ***Namen*** von ***Tabellen*** und ***Spalten***. Der Benutzer muss nicht wissen, wo die Daten physisch gespeichert sind.

3. ***Systematische Behandlung von „NULL“-Werten:***
   „NULL“-Werte werden explizit unterstützt. Sie stellen ***„fehlende“*** Informationen dar. Damit wird ***unabhängig*** vom Datentyp ein Wert bereitgestellt, der das Nichtvorhandensein einer Wertebelegung symbolisiert. Undefinierte Werte müssen sich von leeren Strings und arithmetischen Nullen unterscheiden und bedingen eine Sonderbehandlung. D.h., alle vorhandenen Operationen müssen die ***„NULL“-Werte*** unterstützen. Außerdem muss eine Spalte mit (dem ***Constraint***) ***NOT NULL*** deklarierbar oder die Eingabe von ***„NULL“-Werten*** möglich sein.

4. ***Data Dictionary auf Basis des relationalen Datenmodells:***
   Alle Angaben über die Datenbank selbst müssen ebenfalls in Tabellen abgelegt werden. Damit können ***„autorisierte“*** Benutzer genau dieselbe Abfragesprache zur Untersuchung dieser Daten wie auch bei regulären Daten anwenden. Grund für diese Regel ist, dass jeder Benutzer

(***Anwendungsentwickler***, ***Endanwender*** und ***Datenbankadministrator***) nur ein Datenmodell und somit nur eine Sprache lernen muss. Diese Verwaltungsdatenbank wird als ***„Systemkatalog“*** bzw. das ***„Data Dictionary“*** bezeichnet.

5. ***Umfassende Sprache:***
   Zum Zugriff auf die Daten in der Datenbank wird eine standardisierte Programmiersprache zur Verfügung gestellt. Dies ist heute die Sprache ***SQL - S***tructured ***Q***uery ***L***anguage. Diese Sprache umfasst folgende Bereiche:

   - ➔ ***D***ata ***D***efinition ***L***anguage - ***DDL***
   - ➔ ***D***ata ***M***anipulation ***L***anguage - ***DML***
   - ➔ ***D***ata ***C***ontrol ***L***anguage - ***DCL***

   Unabhängig davon, ob interaktiv oder aus einem Programm in einer anderen Programmiersprache heraus mit dieser Sprache gearbeitet wird, ist sie immer gleich.

6. ***Datenänderung über Views:***
   Datenmanipulationen der Daten müssen sowohl über die Tabellen, als auch über so genannte Nutzersichten (***Views***) möglich sein.

7. ***High Level Datenänderungen:***
   Mengenoperationen sollen nicht nur für Abfragen möglich sein, sondern auch für Änderungsoperationen.

8. ***Physische Datenunabhängigkeit:***
   Anwendungsprogramme bleiben logisch unbeeinträchtigt, wenn Änderungen an der Speicherstruktur oder der Zugriffsmethode vorgenommen werden. Es muss eine klare Trennung zwischen der

logischen Sicht der Daten und der physischen Sicht geben, sonst kann diese Forderung nicht erfüllt werden. D.h., die Dateien, in denen die Daten abgelegt werden, können auf einem anderen Speichermedium liegen. Trotzdem müssen Programme, die diese Daten nutzen, nicht verändert werden, da die Programme auf Tabellen (***logische Sicht***) zugreifen und nicht direkt auf Dateien (***physische Sicht***).

9. ***Logische Datenunabhängigkeit:***
   Anwendungsprogramme bleiben logisch unbeeinträchtigt, wenn informationserhaltende Änderungen an den Basistabellen vorgenommen werden. Diese Forderung kann über den Einsatz von Views im Anwendungsprogramm erfüllt sein. So sind nachträgliche Änderungen am logischen ***Datenbank-design*** möglich, ohne schon existierende Programme zu beeinträchtigen.

10. ***Integritätsunabhängigkeit:***
    Integritätsbedingungen, die spezifisch für eine Datenbank sind, müssen mit Hilfe der relationalen Sprache (***SQL***) definierbar sein und im ***Data Dictionary*** abgelegt werden. Sie sollten nicht im Anwenderprogramm definiert werden müssen.

11. ***Verteilungsunabhängigkeit:***
    Unabhängig davon, wo die Daten physisch liegen, bleiben die Anwendungsprogramme unbeeinträchtigt. Auch wenn Datendateien auf anderen Plattenlaufwerken verteilt werden, muss der Programmierer nicht wissen, wo die Daten liegen. Wichtig ist ***„WAS“*** gesucht wird. Nicht ***„WIE“*** und ***„WO“*** Daten gesucht werden.

12. ***Unterwanderungsverbot:***
    Keine der aufgestellten Regeln darf mit Hilfe einer anderen Sprache umgangen werden. D.h., es darf nur über die relationale Sprache (***SQL***), die diese Regeln erfüllt, mit der Datenbank gearbeitet werden. Eine andere Möglichkeit gibt es nicht.

Für die weiteren Betrachtungen soll an dieser Stelle die Beispiel-Datenbank vorgestellt werden, die in der Firma **ORACLE®** zu Schulungszwecken weltweit verwendet wird.

Es handelt sich um die Relationen:

EMPLOYEES siehe Tabelle 7,
DEPARTMENTS siehe Tabelle 8,
LOCATIONS siehe Tabelle 9 und
JOBS siehe Tabelle 10**.**
Die Strukturen sind in Tabelle 6 dargestellt:

Name	Null?	Typ
EMPLOYEE_ID		NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Name	Null?	Typ
DEPARTMENT_ID		NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

Name	Null?	Typ
LOCATION_ID		NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

Name	Null?	Typ
JOB_ID		VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

Tabelle 6.: Tabellenstrukturen aller Tabellen

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DAT	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
200	Jennifer	Whalen	JWHALEN	515.123.4444	17.09.87	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17.02.96	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17.08.97	MK_REP	6000		201	20
205	Shelley	Higgins	SHIGGINS	515.123.8080	07.06.94	AC_MGR	12000		101	110
206	William	Gietz	WGIETZ	515.123.8181	07.06.94	AC_ACCOUNT	8300		205	110
100	Steven	King	SKING	515.123.4567	17.06.87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21.09.89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03.01.90	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	21.05.91	IT_PROG	6000		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07.02.99	IT_PROG	4200		103	60
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16.11.99	ST_MAN	5800		100	50
141	Trenna	Rajs	TRAJS	650.121.8009	17.10.95	ST_CLERK	3500		124	50
142	Curtis	Davies	CDAVIES	650.121.2994	29.01.97	ST_CLERK	3100		124	50
143	Randall	Matos	RMATOS	650.121.2874	15.03.98	ST_CLERK	2600		124	50
144	Peter	Vargas	PVARGAS	650.121.2004	09.07.98	ST_CLERK	2500		124	50
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29.01.00	SA_MAN	10500	,2	100	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11.05.96	SA_REP	11000	,3	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24.03.98	SA_REP	8600	,2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24.05.99	SA_REP	7000	,15	149	

20 Zeilen ausgewählt.

Tabelle 7.: Inhalt von Tabelle EMPLOYEES

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 Zeilen ausgewählt.

Tabelle 8.: Inhalt von Tabelle **DEPARTMENTS**

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTR
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

Tabelle 9.: Inhalt von Tabelle **LOCATIONS**

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000

12 Zeilen ausgewählt.

Tabelle 10.: Inhalt von Tabelle JOBS

Diese Beispiel-Datenbank wird zum Kennen lernen der Abfragesprache SQL, siehe Kapitel 6. verwendet.

2.4.4. Operationen auf relationalen Schemata

Es ist klar, dass die Definition von Schemata nicht Selbstzweck ist. Vielmehr ist sie die Voraussetzung für die Abspeicherung der Daten und ihrer Auswertung und Verarbeitung. Diese Operationen auf den Daten werden unter dem Begriff ***Datenmanipulation*** zusammengefasst. Betrachten wir die elementaren Betriebssystemoperationen (bzw. ***Dateimanipulationen***):

- ➔ Lesen (***read***) und
- ➔ Schreiben (***write***)

als so genannte System Calls mit den Operationen auf Relationen:

- ➔ Lesen (***select***),
- ➔ Hinzufügen (***insert into***),
- ➔ Verändern (***update***) und
- ➔ Löschen (***delete from***).

Diese Operationen auf den Relationen sind Bestandteil der Datenbanksprache ***SQL - S***tructured ***Q***uery ***L***anguage und werden als Datenmanipulationsbestandteil der Sprache ***DML - D***ata ***M***anipulation ***L***anguage bezeichnet.

Häufig vorkommende Operation ist das Lesen von Daten aus den Relationen, d.h., die Daten werden nicht verändert. Es werden nur unterschiedliche Sichtweisen auf die Daten vollzogen. Diese Operation wird auch als ***Retrievaloperation*** bzw. als ***Abfrage*** bezeichnet.

- ➔ Neben der Abfrage nach einzelnen, durch jeweils einen ***Primärschlüsselwert*** bestimmten Tupel, sind ***mengenorientierte Abfragen*** möglich. Anders formuliert heißt das, dass das Ergebnis einer solchen

Abfrage eine ***Menge von Tupeln***, also eine ***Relation*** im Sinne des ***relationalen Datenmodells*** sein kann.

➔ Die Formulierung der Abfragen erfolgt so, dass ***keine Navigation*** durch die Relationen vorgenommen werden muss. Die Abfragen werden im Wesentlichen auf die ***drei relationalen Operationen*** zurückgeführt:

- ➔ die Selektion,
- ➔ die Projektion und
- ➔ der Join.

2.4.4.1. Selektion

Definition 17 - Selektion:

Diese Operation wählt Tupel der Relation aus, die ein bestimmtes Prädikat erfüllen, d.h., für die eine formulierte Bedingung den logischen Wert true liefert.

An einem Beispiel soll diese Operation verdeutlicht werden:

Aufgabe:
Ermitteln Sie alle Mitarbeiter aus der Tabelle EMPLOYEES, die in der Abteilung SALES, mit der DEPARTMENT_ID = 80 arbeiten!

Ergebnis:
Daten der Selektion aus der Tabelle EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DAT	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29.01.00	SA_MAN	10500	,2	100	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11.05.96	SA_REP	11000	,3	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24.03.98	SA_REP	8600	,2	149	80

Der dazugehörige SQL-Befehl ist:

```
rem  Beispiel einer Selektion auf die Tabelle EMPLOYEES
rem  alle Mitarbeiter aus Abteilung 80

select *
from   employees
where  department_id = 80;
```

Aufgabe:

Als weiteres Beispiel ermitteln Sie alle Mitarbeiter, deren Gehalt (SALARY) größer ist als $10.000!

Ergebnis:

Daten der Selektion aus der Tabelle EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DAT	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
201	Michael	Hartstein	MHARTSTE	515.123.5555	17.02.96	MK_MAN	13000		100	20
205	Shelley	Higgins	SHIGGINS	515.123.8080	07.06.94	AC_MGR	12000		101	110
100	Steven	King	SKING	515.123.4567	17.06.87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21.09.89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP	17000		100	90
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29.01.00	SA_MAN	10500	,2	100	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11.05.96	SA_REP	11000	,3	149	80

7 Zeilen ausgewählt.

Als Spezialfall kann auch eine einzelne Zeile aus einer Relation selektiert werden:

Aufgabe:

Welcher Mitarbeiter hat die Berufsbezeichnung (JOB_ID)AC_MGR in der Abteilung 110?

Ergebnis:

Daten der Selektion aus der Tabelle EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DAT	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
205	Shelley	Higgins	SHIGGINS	515.123.8080	07.06.94	AC_MGR	12000		101	110

Der dazugehörige SQL-Befehl ist:

```
rem  Beispiel einer Selektion auf die Tabelle EMPLOYEES
rem  der Mitarbeiter mit der Berufsbezeichnung AC_MGR aus Abteilung 110

select *
from   employees
where  job_id = 'AC_MGR'
and    department_id = 110;
```

Hierbei ist zu beachten, dass die Datentypen Zeichenkette – ***string*** und Datum – ***date*** immer in einfache Anführungszeichen (’<wert>’) zu setzen sind. Die Notation des Wertes <wert> muss so erfolgen, wie die Daten in der Datenbank abgespeichert wurden.

2.4.4.2. Projektion

Definition 18 - Projektion:

Diese Operation liefert eine Ergebnisrelation, die auf einem Tupeltyp mit einer ausgewählten Attributmenge basiert.

Auch dazu sollen Beispiele diese Operation veranschaulichen:

Aufgabe:

Geben Sie alle Mitarbeiter aus der Tabelle EMPLOYEES mit der Tupelmenge (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT_ID) aus!

Ergebnis:

Daten der Projektion aus der Tabelle EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
200	Jennifer	Whalen	10
201	Michael	Hartstein	20
202	Pat	Fay	20
205	Shelley	Higgins	110
206	William	Gietz	110
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
103	Alexander	Hunold	60
104	Bruce	Ernst	60
107	Diana	Lorentz	60
124	Kevin	Mourgos	50
141	Trenna	Rajs	50
142	Curtis	Davies	50
143	Randall	Matos	50
144	Peter	Vargas	50
149	Eleni	Zlotkey	80
174	Ellen	Abel	80
176	Jonathon	Taylor	80
178	Kimberely	Grant	

20 Zeilen ausgewählt.

Der dazugehörige SQL-Befehl ist:

```
rem  Beispiel einer Projektion auf die Tabelle EMPLOYEES von allen
rem  Mitarbeiter nur (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT_ID)

select employee_id, first_name, last_name, department_id
from   employees;
```

Als Spezialfall kann auch eine einzelne Spalte aus einer Relation projiziert werden:

Aufgabe:

Geben Sie alle Abteilungsnummern der Mitarbeiter aus!

Ergebnis:

Daten der Projektion aus der Tabelle EMPLOYEES:

DEPARTMENT_ID
10
20
20
110
110
90
90
90
60
60
60
50
50
50
50
50
80
80
80

20 Zeilen ausgewählt.

Das Ergebnis ist in diesem Fall nicht sehr sinnvoll, da es mehrfach den gleichen Attributwert in dieser Projektion gibt. Besser wäre in diesem Fall folgende Projektion:

Ergebnis:

Daten der Projektion aus der Tabelle EMP:

DEPARTMENT_ID
10
20
50
60
80
90
110

8 Zeilen ausgewählt.

Der dazugehörige SQL-Befehl ist:

```
rem  Beispiel einer Projektion auf die Tabelle EMPLOYEES
rem  von allen Mitarbeiter nur die DEPARTMENT_ID, jeweils jeder
rem  Wert nur einmal

select    distinct department_id
from      employees;
```

2.4.4.3. Join

Definition 19 - Join:

> **Eine Join-Operation verbindet zwei Relationen zu einer Ergebnisrelation. Voraussetzung ist, dass es in beiden Relationen je ein Attribut gibt, deren Wertebereiche gleich sind. Anhand der Wertebelegung dieser Attribute (Joinattribute) erfolgt die Verbindung der Relation.**

Betrachten wir die Tabellen EMPLOYEES und DEPARTMENTS, so wird deutlich, dass das Attribut DEPARTMENT_ID mit gleichem Wertebereich in beiden Relationen enthalten ist, und als Verbindung beider Relationen in einem Join dient:

> **Dabei ist zu beachten, dass ein RDBMS immer bei einem Join das vollständige kartesische Produkt beider Relationen im Speicher aufbaut. Erst danach werden Spalten und Zeilen durch Projektion und Selektion verworfen.**

Aufgabe:

Geben Sie von allen Mitarbeitern die Projektion (EMPLOYEE_ID, LAST_NAME, DEPARTMENT_NAME) an!

Ergebnis:

Daten des Joins aus den Tabellen EMPLOYEES und DEPARTMENTS:

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_NAME
200	Whalen	Administration
201	Hartstein	Marketing
202	Fay	Marketing
124	Mourgos	Shipping
144	Vargas	Shipping
143	Matos	Shipping
142	Davies	Shipping
141	Rajs	Shipping
103	Hunold	IT
104	Ernst	IT
107	Lorentz	IT
149	Zlotkey	Sales
174	Abel	Sales
176	Taylor	Sales
100	King	Executive
102	De Haan	Executive
101	Kochhar	Executive
205	Higgins	Accounting
206	Gietz	Accounting

19 Zeilen ausgewählt.

Der dazugehörige SQL-Befehl ist:

```
rem  Beispiel eines Joins auf die Tabellen EMPLOYEES und DEPARTMENTS
rem  mit der Projektion (EMPLOYEE_ID, LAST_NAME, DEPARTMENT_NAME)
select  employees.employee_id, employees.last_name,
        departments.department_name
from    employees, departments
where   employees.department_id = departments.department_id;
```

Folgende Notation ist aber häufiger anzutreffen:

```
rem  Beispiel eines Joins auf die Tabellen EMPLOYEES und DEPARTMENTS
rem  mit der Projektion (EMPLOYEE_ID, LAST_NAME, DEPARTMENT_NAME)
rem  unter Verwendung von Spaltenaliasnamen

select  e.employee_id, e.last_name, d.department_name
from    employees e, departments d
where   e.department_id = d.department_id;
```

Neben den bereits aus der Mengenlehre bekannten Operationen

- ➔ Vereinigung,
- ➔ Durchschnitt und
- ➔ Differenz

sind es diese relationalen Operationen ***Selektion***, ***Projektion*** und ***Join***, auf die komplexe mengenorientierte Abfragen zurückgeführt werden können.

3. Datenbankentwurf – Entity-Relationship-Modell

Das ***Entity-Relationship-Modell*** wurde 1976 von Mister ***Peter Pin-Shan Chen*** (USA) vorgeschlagen. Der Titel seines Zeitschriftenartikels klingt programmatisch und ist treffend ***„Das Entity-Relationship-Modell: Hin zu einer vereinheitlichten Sicht der Daten“*** vorgestellt.

Seit seiner Entwicklung haben sich mehrere verschiedene Modifikationen bzw. grafische Darstellungsmöglichkeiten herausgebildet. Für die Einführung des ER-Modells soll hier die Notationsform verwendet werden, die durch Tools des relationalen Datenbankmanagementsystems **ORACLE®** direkt verarbeitet werden kann.

Für das ER-Modell gibt es folgende Anwendungsfälle:

- ➔ Im Datenbank-Entwurfsprozess zur Modellierung eines konzeptionellen Informationsschemas.
- ➔ Im Softwareentwicklungsprozess bzw. in dafür bereitgestellten Methoden und Werkzeugen als Hilfsmittel des Datenentwurfes.

3.1. Mathematische Grundlagen

Die Definitionen für die Begriffe ***Entität*** und ***Attribut*** wurden bereits im Abschnitt 2.4.2. Terminologie und Notation gegeben.

Eine Entität kann sein:

- ➔ ein Individuum (Mitarbeiter, Student, Professor, etc.)
- ➔ ein reales Objekt (Haus, Raum, Auto, Produkt, etc.)
- ➔ ein abstraktes Konzept (Vorlesung, Fahrplan, Abteilung, etc.)
- ➔ ein Ereignis (Lieferung, Klausurtermin, etc.)

Eine Entität ist somit ***„irgend etwas“***, von dem es sich lohnt, Informationen zu haben.

Definition 20 - Entitätsmenge:

> **Eine Entitätsmenge (entity set) ist eine eindeutige, benannte Menge von Entitäten gleichen Typs, d.h. von Entitäten, die durch die gleichen Merkmale (Attribute) beschrieben werden.**

Eine Entitätsmenge wird mittels einer so genannten ***„Soft Box“*** dargestellt:

Abbildung 30.: Darstellung einer Entität

Jede Entitätsmenge hat einen Namen und eine nichtleere Menge von Merkmalen.

Namenskonventionen für Entitätsmengen:

- ➔ Substantive
- ➔ Singular
- ➔ Großbuchstaben
- ➔ realitätsbezogen, üblicher Sprachgebrauch, möglichst kurz

Eine Entität wird durch Merkmale (Eigenschaften, Attribute) beschrieben.

Merkmale dienen zur:

- ➔ Identifizierung
- ➔ Beschreibung
- ➔ Klassifizierung
- ➔ Quantifizierung
- ➔ Zustandsbeschreibung

Definition 21 - Wertebereich:

Die Menge aller zulässigen Werte eines Merkmals nennt man Wertebereich (domain). Jedes Merkmal nimmt pro Entität einen konkreten Wert aus diesem Wertebereich an.

Wertebereiche werden beschrieben durch:

- ➔ Datentyp (numerisch, alphanumerisch, Datum, Geld, etc.)
- ➔ Format (Länge, Genauigkeit, Maske, etc.)
- ➔ Intervall, Menge
- ➔ kodiert, unkodiert
- ➔ obligatorisch (NOT NULL) oder optional

Unterschiedliche Attribute können den gleichen Wertebereich haben.

Wertebereiche sind die einfachste Form von Konsistenzregeln.

Beispiel für Tabelle EMPLOYEES:

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)  CONSTRAINT employees_last_name_nn  NOT NULL
    , email          VARCHAR2(25)  CONSTRAINT employees_email_nn      NOT NULL
    , phone_number   VARCHAR2(20)
    , hire_date      DATE          CONSTRAINT employees_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)  CONSTRAINT employees_job_id_nn     NOT NULL
    , salary         NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
    ,                              CONSTRAINT employees_salary_min CHECK (salary > 0)
    ,                              CONSTRAINT employees_email_uk   UNIQUE (email)
    ) ;
```

Zwischen den Entitätsmengen bestehen Beziehungen (Relationships).

Definition 22 - Relationship:

Eine Relationship (Beziehung zwischen zwei Entitätsmengen) assoziiert wechselseitig zwei (möglicherweise mehr als zwei) Entitäten jeweils aus den Entitätsmengen.

Beziehungen werden beschrieben durch:

- ➔ Namen
- ➔ Richtung, da Beziehungen zwischen den Entitätsmengen gerichtet sind
- ➔ optional bzw. obligatorisch

- ➔ Kardinalität
- ➔ Beziehungen können zur Identifizierung einer Identität gehören

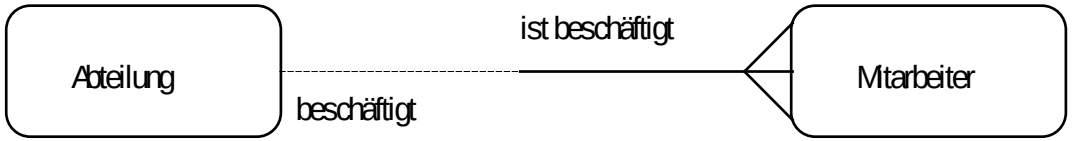


Abbildung 31.: Darstellung der Beziehung

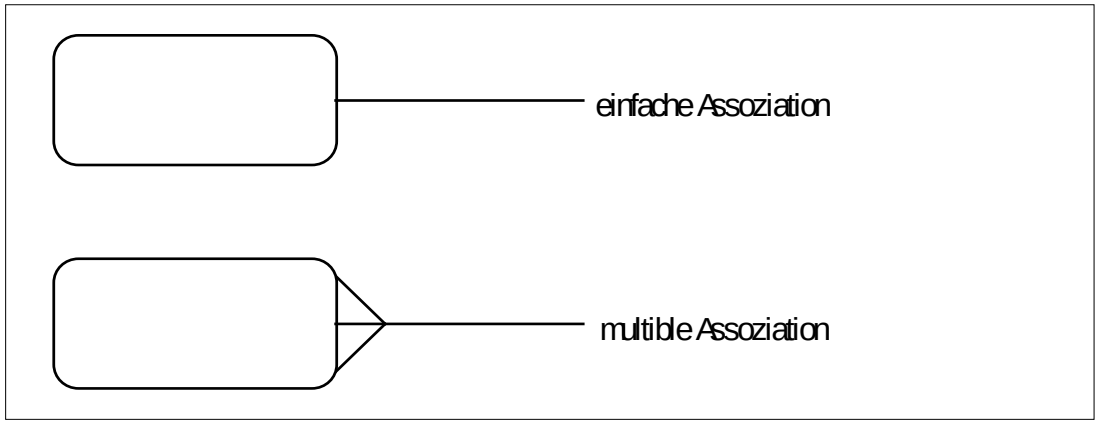


Abbildung 32.: Darstellung der Kardinalität

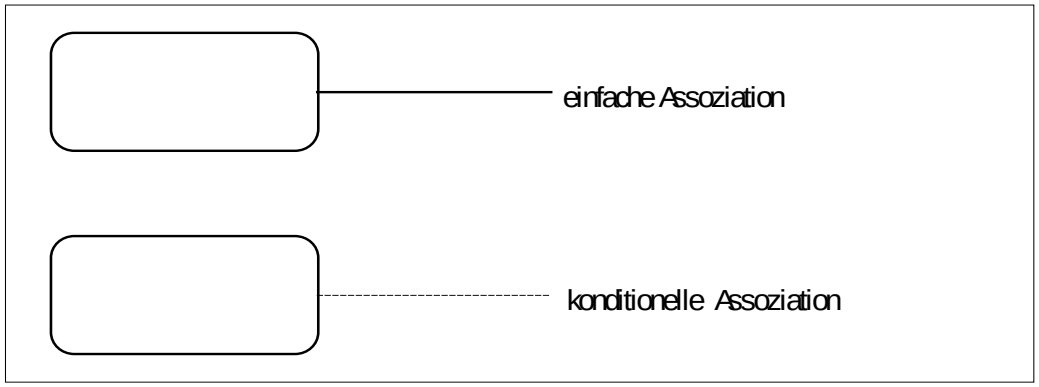


Abbildung 33.: Darstellung der Optionalität

3.2. Darstellungsmöglichkeiten im Entity-Relationship-Modell

3.2.1. Beispiele - hierarchische Beziehung

1. Zu jeder Entität der Entitätsmenge E1 gibt es genau eine Entität in der Entitätsmenge E2 und
2. zu jeder Entität der Entitätsmenge E2 gibt es genau eine Entität in der Entitätsmenge E1.

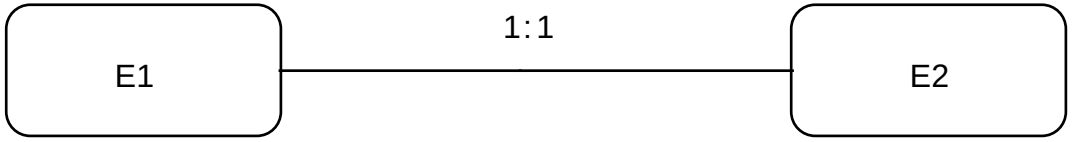


Abbildung 34.: 1:1 Beziehung

1. Zu jeder Entität der Entitätsmenge E1 gibt es keine oder eine Entität der Entitätsmenge E2 und
2. zu jeder Entität der Entitätsmenge E2 gibt es genau eine Entität in der Entitätsmenge E1.

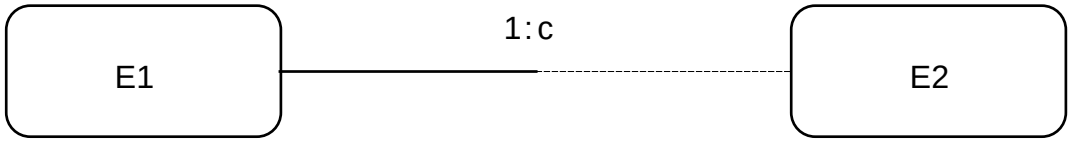


Abbildung 35.: 1:c Beziehung

1. Zu jeder Entität der Entitätsmenge E1 gibt es mindestens eine Entität der Entitätsmenge E2 und
2. zu jeder Entität der Entitätsmenge E2 gibt es genau eine Entität in der Entitätsmenge E1.

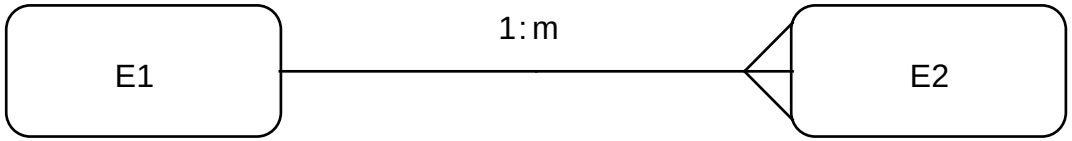


Abbildung 36.: 1:m Beziehung

3.2.2. Beispiele - konditionelle Beziehung

1. Zu jeder Entität der Entitätsmenge E1 gibt es keine oder eine Entität der Entitätsmenge E2 und
2. zu jeder Entität der Entitätsmenge E2 gibt es keine oder eine Entität in der Entitätsmenge E1.

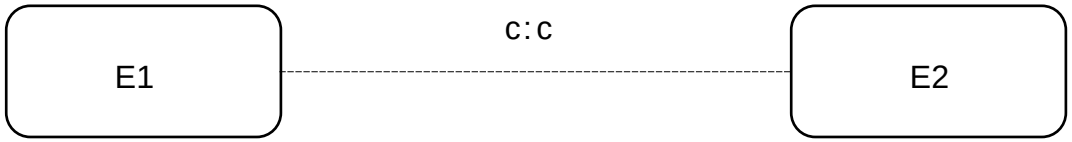


Abbildung 37.: c:c Beziehung

1. Zu jeder Entität der Entitätsmenge E1 gibt es mindestens eine Entität in der Entitätsmenge E2 und
2. zu jeder Entität der Entitätsmenge E2 gibt es keine oder eine Entität in der Entitätsmenge E1.

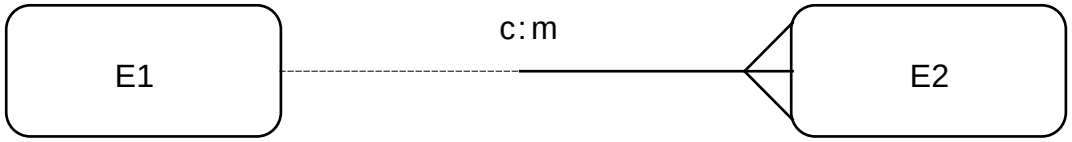


Abbildung 38.: c:m Beziehung

3.2.3. Beispiel - netzförmige Beziehung

1. Zu jeder Entität der Entitätsmenge E1 gibt es mindestens eine Entität der Entitätsmenge E2 und
2. zu jeder Entität der Entitätsmenge E2 gibt es mindestens eine Entität in der Entitätsmenge E1.

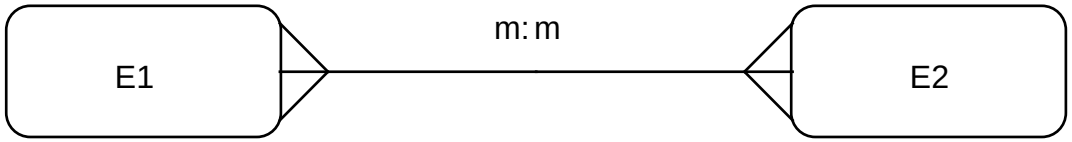


Abbildung 39.: m:m Beziehung

3.3. Entwicklung eines Beispiel-ER-Modells

Zunächst soll das ER-Modell für die **ORACLE®**-Beispieldatenbank an Hand der beiden Entitäten:

- DEPARTMENTS
- EMPLOYEES

entwickelt werden.

3.3.1. Beispiel-ER-Modell mit Entitäten und Beziehungen

Dazu werden die Entitäten einander gegenüber angeordnet:

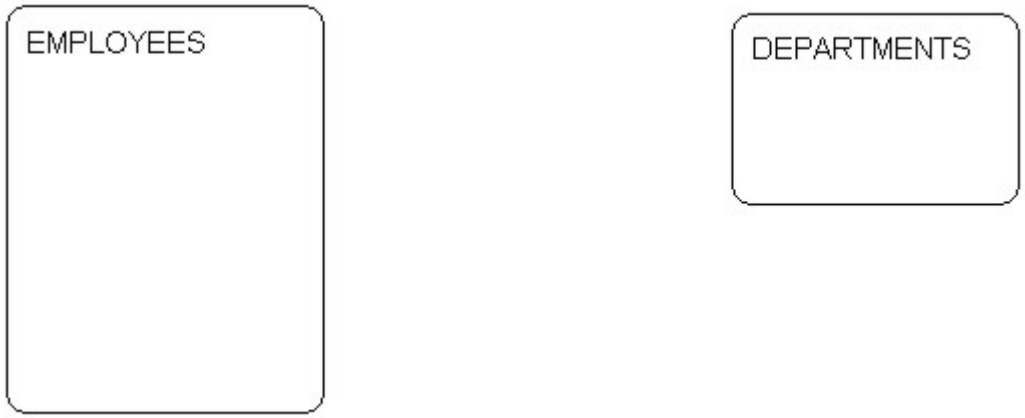


Abbildung 40.: zwei Entitäten des ER-Modells der Beispiel-Datenbank

Anschließend wird versucht, die beiden Assoziationstypen in Form von Sätzen, für jeden Assoziationstyp bzw. jede Betrachtungsrichtung einen zu formulieren und das ER-Modell zu erweitern:

Jeder Mitarbeiter (EMPLOYEE_ID) arbeitet in genau einer Abteilung (DEPARTMENT_ID).

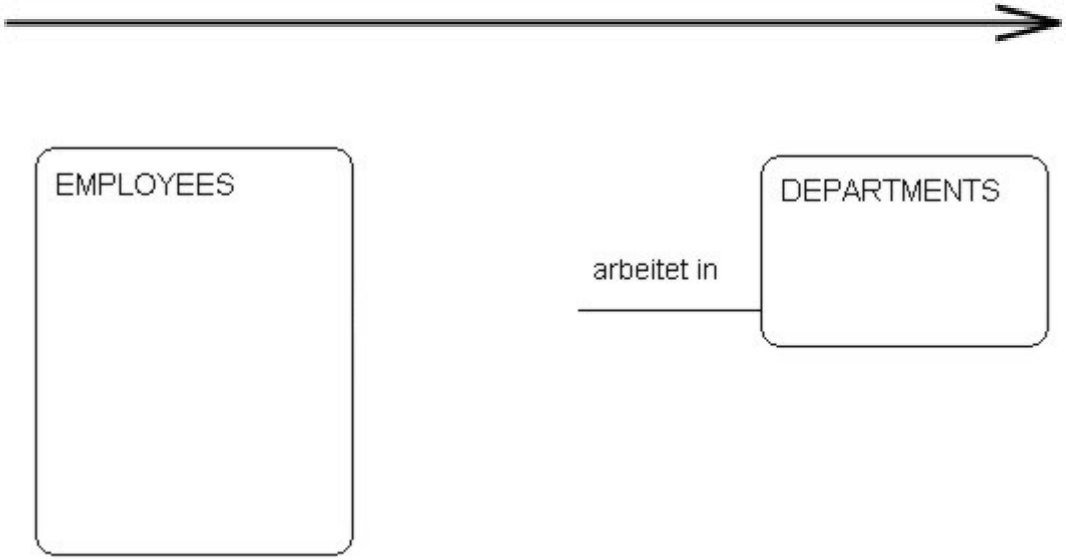


Abbildung 41.: einfache Assoziation von EMPLOYEES nach DEPARTMENTS

In jeder Abteilung (DEPARTMENT_ID) ist (sind) mindestens ein Mitarbeiter (EMPLOYEE_ID) beschäftigt.

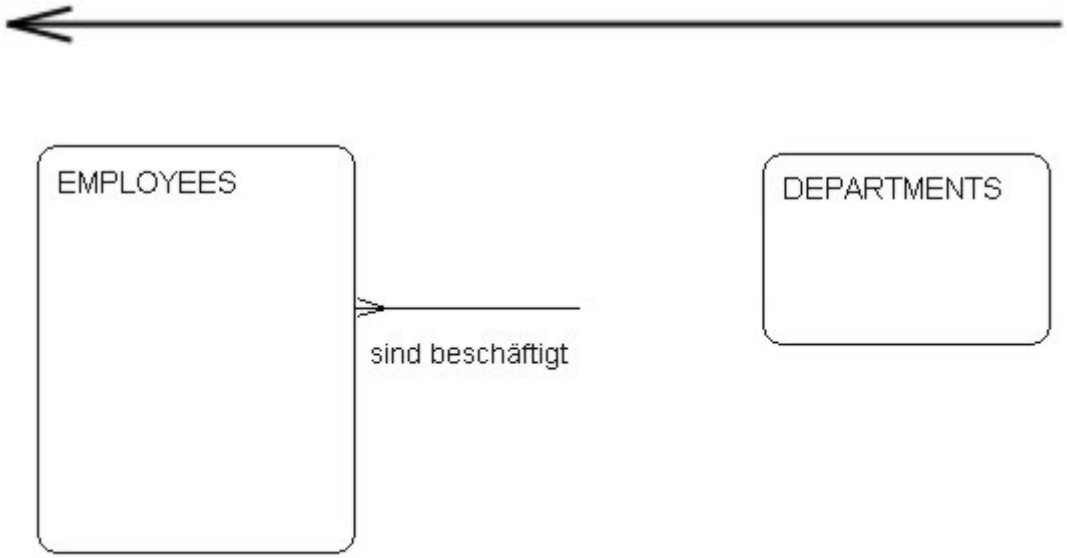


Abbildung 42.: multiple Assoziation von DEPARTMENTS nach EMPLOYEES

Ergebnis ist der erste Teil des ER-Modells für die **ORACLE®**-Beispiel-Datenbank:

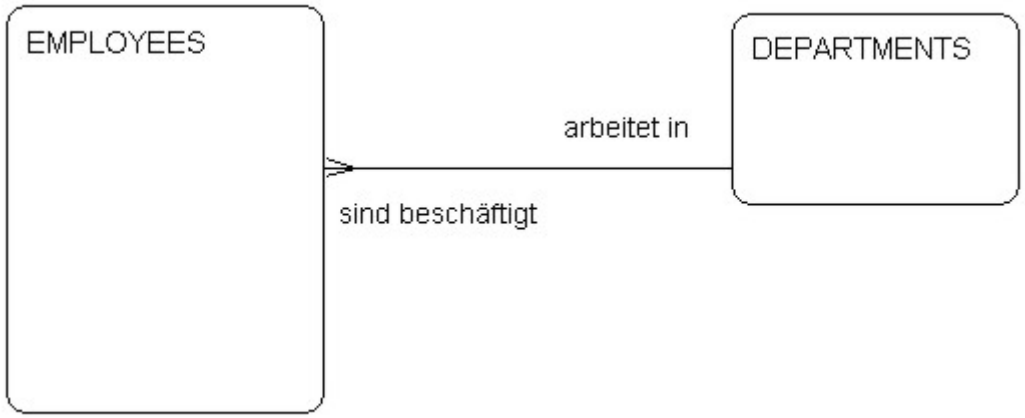


Abbildung 43.: Ausschnitt aus dem ER-Modell der Beispiel-Datenbank

Damit sind die Voraussetzungen gegeben, für die gesamte **ORACLE®**-Beispiel-Datenbank ein ER-Modell zu entwickeln:

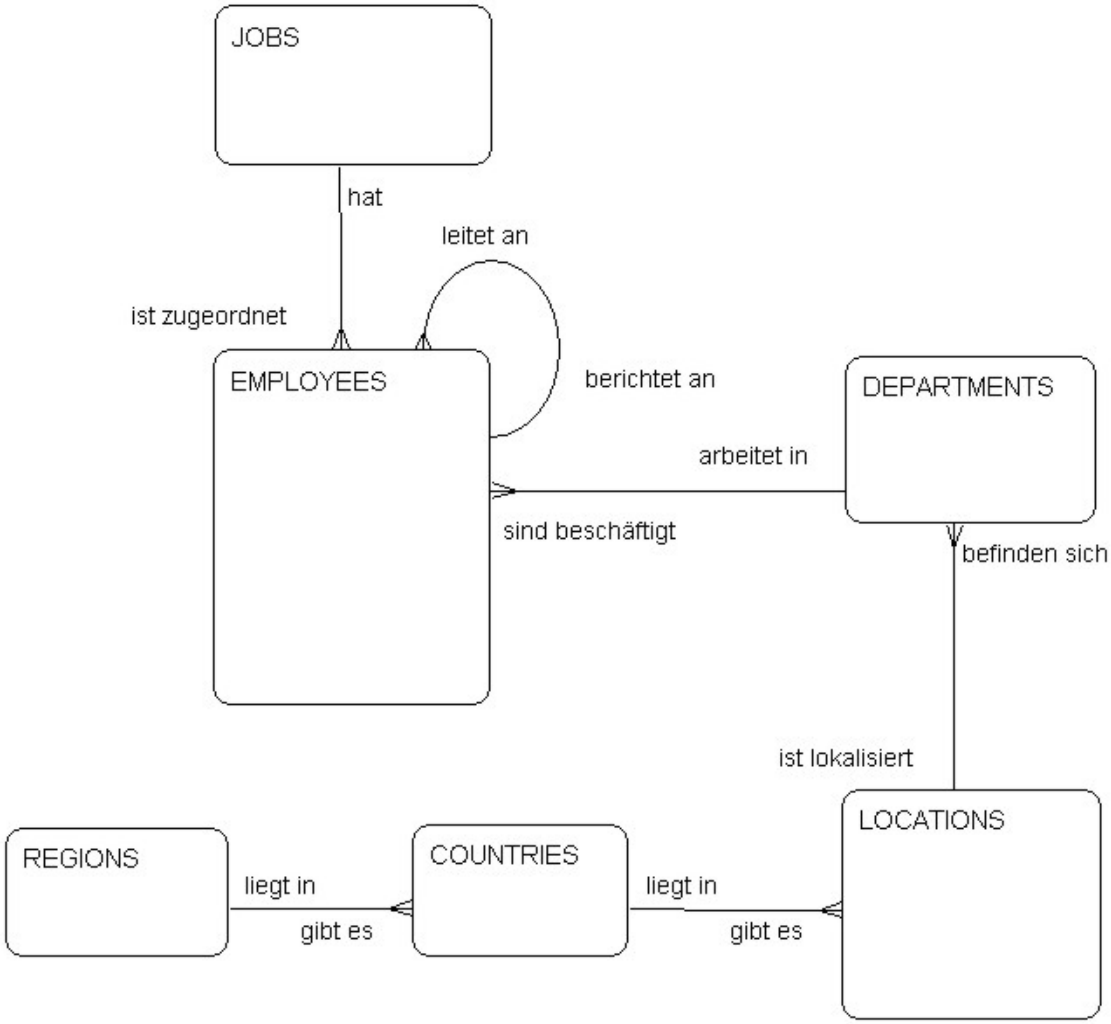


Abbildung 44.: ER-Modell der gesamten Beispiel-Datenbank

3.3.2. Beispiel-ER-Modell mit Attributen

Ein Attribut kann je nach Bedarf aus Text, Zahlen, einem Bild, einem Gefühl, einem Geruch, usw. bestehen.

Attribute müssen ebenfalls im ER-Modell dargestellt werden. Dazu werden die Attributbezeichner, die eine Entität beschreiben, in die ***Soft Box*** mit eingetragen.
Für die Relation DEPT soll die Darstellung demonstriert werden:

Attribute, die als ***Identifikationsschlüssel*** fungieren, werden dabei besonders mit (#) gekennzeichnet.

Abbildung 45.: Attributdarstellung für eine Entität des ER-Modells der Beispiel-Datenbank

Damit kann für die **ORACLE®**-Beispiel-Datenbank das vollständige Entity-Relationsship-Modell angegeben werden:

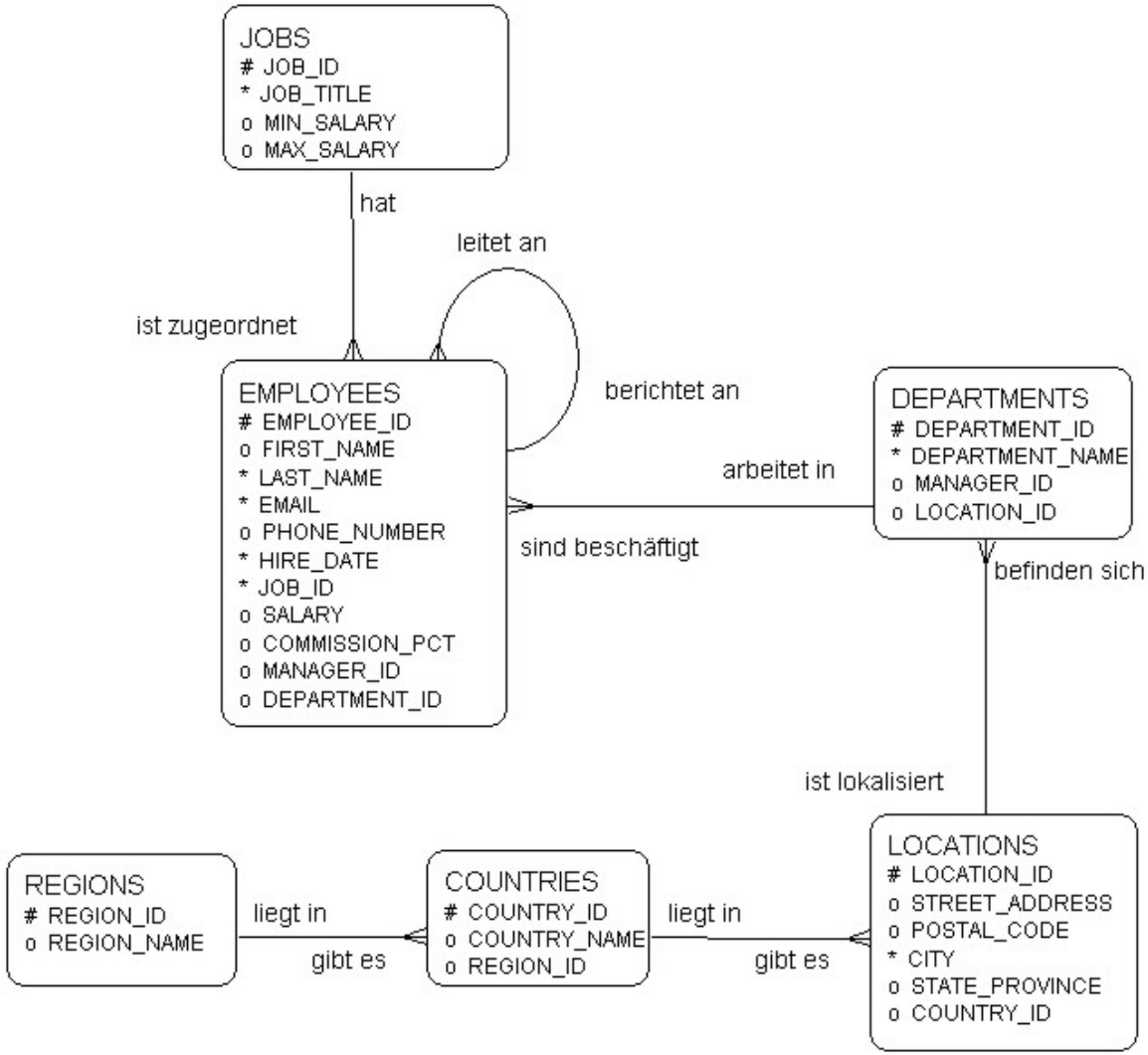


Abbildung 46.: Vollständiges ER-Modell der Beispiel-Datenbank

3.4. Case-Tools

Für die Eingabe und Generierung einer Datenbank lassen sich verschiedene Werkzeuge verwenden, die in vergleichbarer Art und Weise verwendet werden und jeweils für verschiedene Datenbankmanagementsysteme die SQL-Skripte generieren bzw. die Datenbank direkt erzeugen. Beispielhaft seien folgende Werkzeuge – **Case-Tools** (**C**omputer **A**ided **S**oftware **E**ngineering) genannt:

Hersteller	**Produkt**	**generiert in**	
Oracle Corporation	Designer 10g	ANSI Level 2	IBM DB2
		Microsoft SQL Server	Oracle
		RdB	Sybase
Sybase Inc.	PowerDesigner 11.0	ADABAS D	ANSI Level 2
		IBM DB2	Informix
		InterBase	Microsoft Access
		Microsoft SQL Server	MySQL
		Oracle	PostgreSQL
		Sybase	
Computer Associates	Erwin 4.1	Clipper	dBASE
		FoxPro	IBM DB2
		Informix	OpenIngress
		Oracle	Paradox
		Progress	RdB
		Sybase	
Oracle Corporation	SQL Developer Data Modeler	Oracle	

Tabelle 11.: Case-Tools

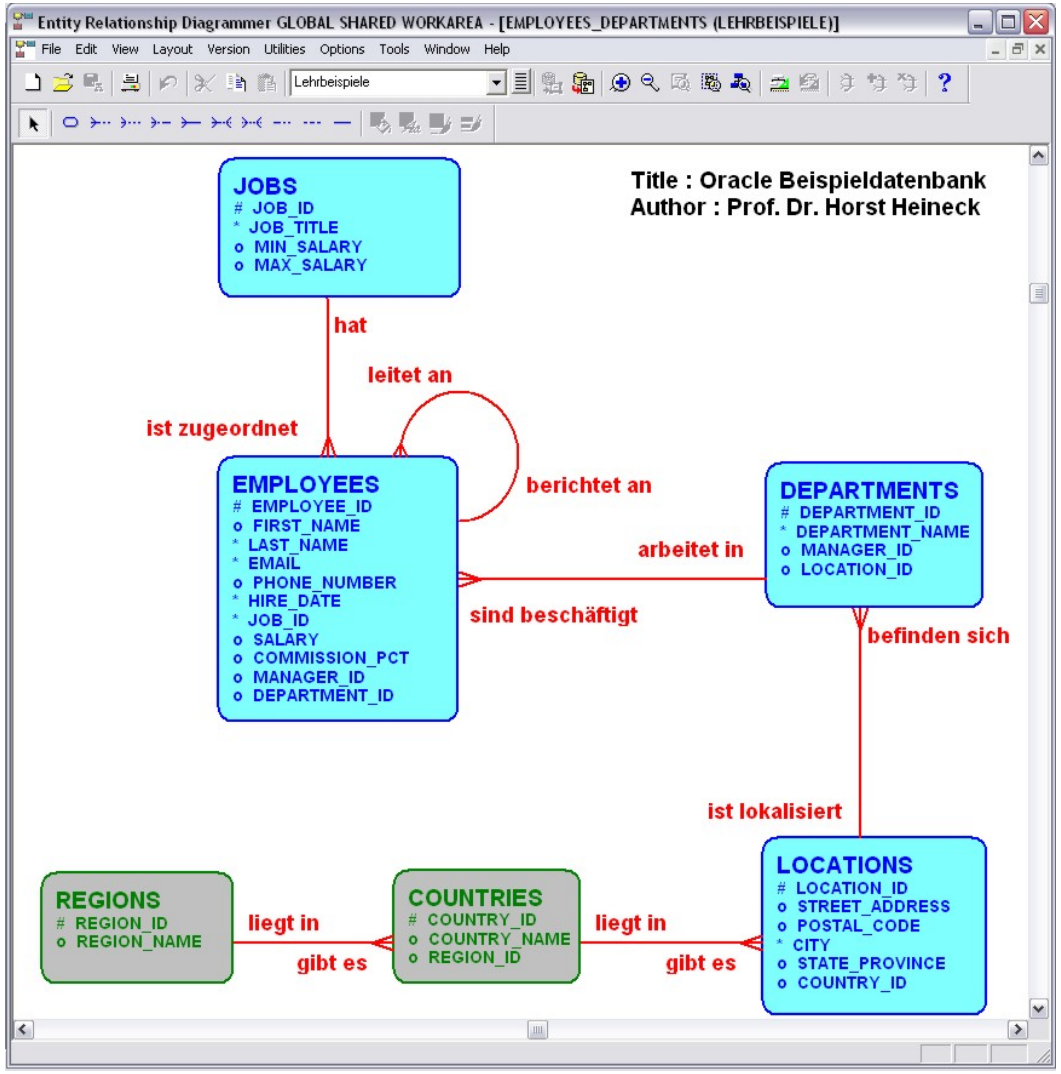


Abbildung 47.: Vollständiges ER-Modell der Beispiel-Datenbank im ORACLE® Designer

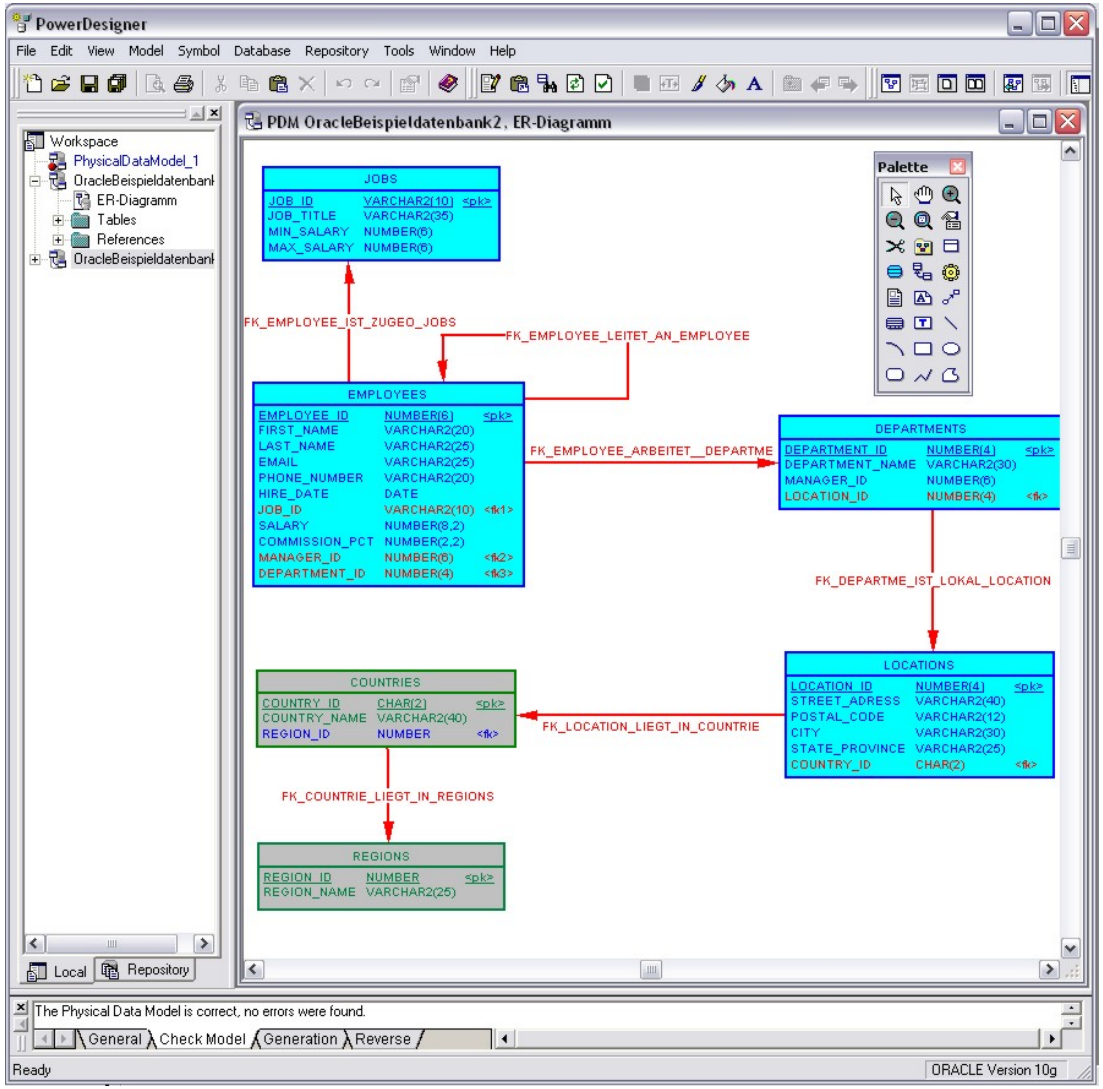


Abbildung 48.: Vollständiges ER-Modell der Beispiel-Datenbank im Sybase® PowerDesigner®

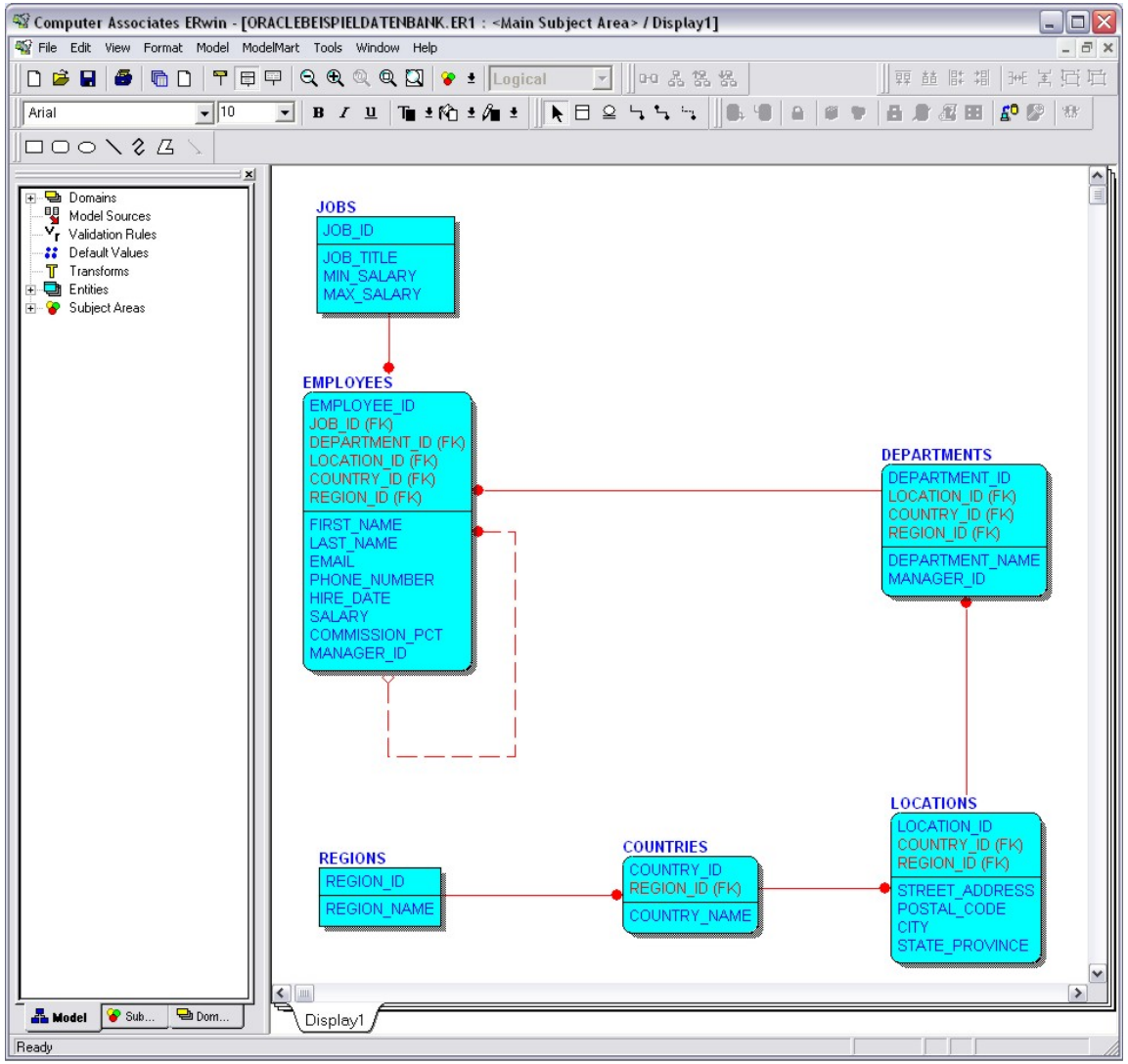


Abbildung 49.: Vollständiges ER-Modell der Beispiel-Datenbank im Computer Associates ERwin Data Modeler®

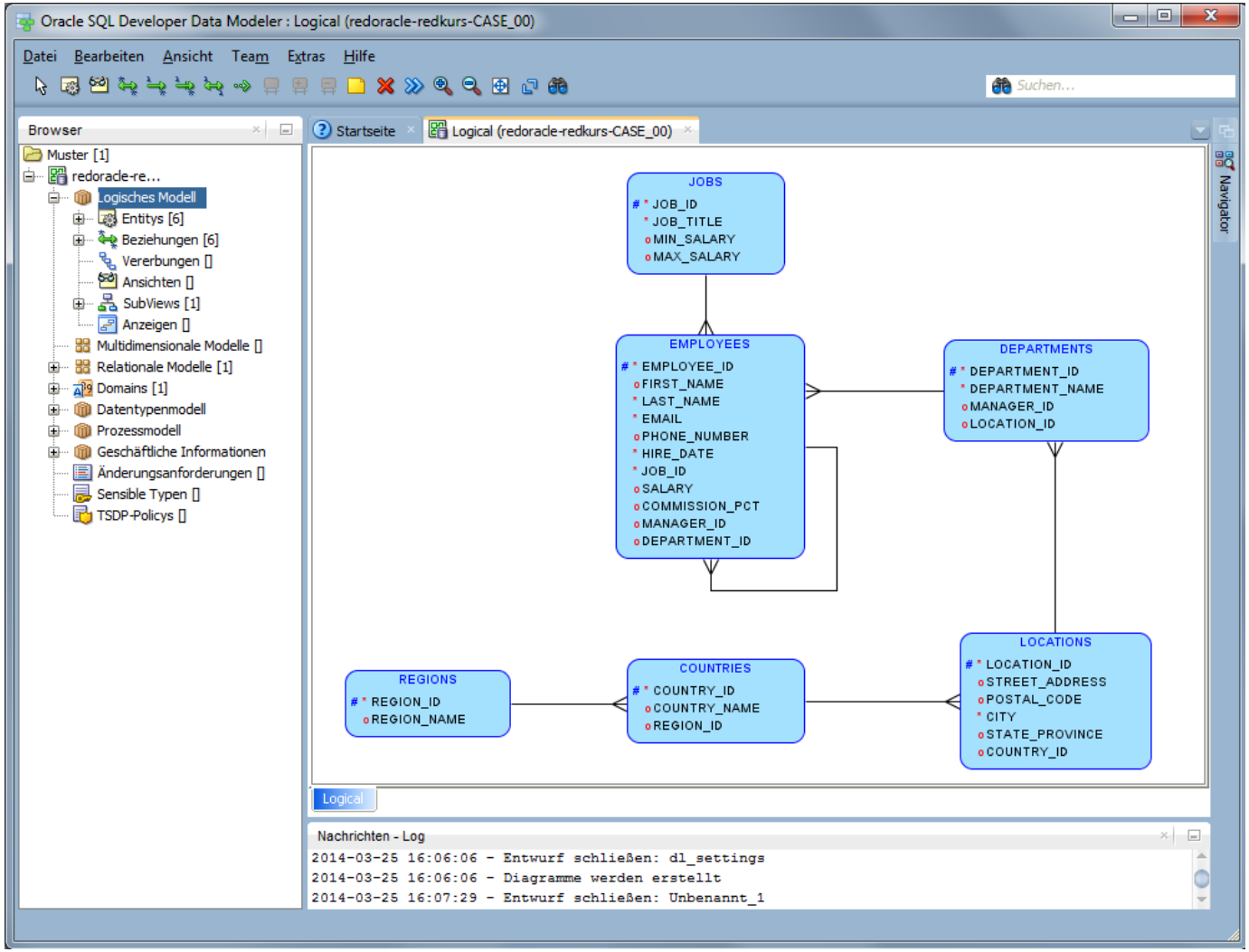


Abbildung 50.: Vollständiges ER-Modell der Beispiel-Datenbank im Oracle SQL Developer Data Modeler ®

4. Aufgabenstellung für gemeinsame Übung

Ergebnis der Gruppenarbeit soll es sein, eine Datenbank nach dem Datenmodell zu erstellen. Als Hilfsmittel kann der Oracle SQL Developer Data Modeler ® verwendet werden. Um die Korrektheit der Datenbank zu prüfen, sind in jede Relation (Tabelle) 10 Datensätze einzugeben und über geeignete SELECT-Anweisungen zu überprüfen.

Achtung Datenschutz:

Bitte beachten Sie bei Ihren Arbeiten die Prinzipien des Datenschutzes. Da sich die erzeugte Datenbank auf einem Server mit allgemeinem Zugang befindet, dürfen keine realen Daten von Personen oder Firmen als Testdaten verwendet werden. Bitte verwenden Sie immer fiktive Daten.

5. Normalisierung

Codd hat gewisse Anforderungen an das ***relationale Datenmodell*** gestellt, denen diese genügen müssen, damit die Abspeicherung der Daten möglichst redundanzfrei erfolgen kann. Damit ergeben sich wenige oder keine Widersprüche. Er hat diese Forderungen (***relationale***) ***Normalformen*** genannt. Die Notwendigkeit dieser ***Normalformen*** ergibt sich aus der Anwendung von ***Updateoperationen***. D.h., liegt ein relationales Datenmodell nicht in einer bestimmten ***Normalform*** vor, dann verursacht die Anwendung von Updateoperationen so genannte ***Updateanomalien***. Der Prozess zur Erzeugung entsprechender (normalisierter) relationaler Datenmodelle wird als Normierung bezeichnet.

5.1. Funktionale Abhängigkeit

Der Begriff der Abhängigkeit wurde bereits im Abschnitte 2.3.1. definiert. Jetzt wird dieser Begriff weiter spezifiziert:

Definition 23 - funktionale Abhängigkeit:

Sei *R* eine Relation und seinen *A* und *B* beliebige Teilmengen der Attribute von *R*. Dann heißt *B* funktional abhängig von *A* genau dann, wenn zu jeder Wertekombination der Attributwerte aus *A* genau eine Wertkombination der Attributwerte aus *B* gehört.

Dabei wird folgende Formulierung verwendet: *A* ist eine funktionale Determinante für *B* bzw. *A* determiniert *B* funktional. Man schreibt:

***A* → *B* bzw. *R.A* → *R.B*. *A* heißt auch eine Determinante.**
Falls die Menge *A* nicht mehr verkleinert werden kann, ohne dass die funktionale Abhängigkeit verloren geht, heißt *A* eine minimale Determinante.

Jedes Attribut, das nicht zum Identifikationsschlüssel (der Determinante) einer Relation gehört, ist funktional von diesem abhängig.

Bei einem zusammengesetzten Identifikationsschlüssel ist keines seiner Attribute funktional abhängig.

Relation PERSON (#Id, Name, Vorname, Strasse, Nr, Plz, Ort, Land):

- Sei **A = (Id)**. Dann besteht die funktionale Abhängigkeit A → B. Denn **Id** ist Schlüsselkandidat (Primary Key).

- Sei **A = (Name, Vorname)**. Dann besteht keine funktionale Abhängigkeit A → B, denn es kann nicht ausgeschlossen werden, dass zwei verschiedene Personen mit denselben Namen und Vornamen existieren.

- Sei **A = (Name, Vorname, Nr, Ort)** und **B = (Name, Ort)**. Dann besteht die **triviale funktionale Abhängigkeit** A → B.

- Sei **A = (Strasse, Nr, Ort, Land)** und **B = (Plz)**. Dann besteht die **nicht-triviale funktionale Abhängigkeit** A → B. A ist eine **nicht-triviale Determinante**.

5.2. Erste Normalform

Die erste Normalform beschreibt eine grundlegende Eigenschaften von Relationen, die für alle manipulativen Operationen auf den Tabellen einer Datenbank – seien es Abfragen (insbesondere mit Joins), seien es Insert-, Update- oder Deleteoperationen – unerlässlich ist. Wie bereits dargestellt, besitzen alle Attribute einer Relation einen Wertebereich. Dieser Wertebereich, der einen Datentyp für ein Attribut festlegt, ist verantwortlich für die Einhaltung der ersten Normalform (1NF).

Definition 24 - Erste Normalform:

> **Eine Relation befindet sich in der ersten Normalform, wenn zu jedem Attribut dieser Relation ein Wertebereich gehört, der nur atomare Werte enthält. Das bedeutet: Solch ein Wertebereich darf also keine Werte enthalten, die aus mehreren Einzelwerten bestehen. Genauer: Er darf kein Array und kein mengenwertiger Bereich – Set – sein.**

Eine Relation in der ersten Normalform heißt normalisiert oder auch normal. Der Hauptgrund für diese Anforderung ist, dass nur auf Relationen mit dieser Eigenschaft sinnvoll Selektionen mit WHERE-Bedingungen angewendet werden können. Ohne diesen Operator wäre z.B. kein JOIN möglich. Auch die Möglichkeiten der Sortierung nach Attributen und die Festlegung von Schlüsseln hängen von dieser Eigenschaft ab.

Damit kann für praktische Zwecke konkret festgehalten werden, dass nur folgende Datentypen in Tupeltypen anzutreffen sind:

- Datentypen für ganze Zahlen, Dezimalzahlen,
- Datentypen für logische Werte,
- Datentypen für alphanumerische Zeichen, Zeichenketten

Relation TEMPERATUR(#Ort, #Tag, Temperatur):

Ort	**Tag**	**Temperatur**				
		08	**12**	**16**	**20**	**Uhr**
Hof	25. Februar 2005	-12	-2	-6	-13	
München	25. Februar 2005	-9	3	-3	-6	

Tabelle 12.: Relation TEMPERATUR (alt)

Diese Beispielrelation verstößt gegen die erste Normalform, da die Werte des Attributs ***Temperatur*** nicht elementar, bzw. atomar sind.

Folgende Relation mit gleichem Inhalt befindet sich dagegen in der ersten Normalform:

Relation TEMPERATUR(#Ort, #Tag-Zeit, Temperatur):

ORT	TAG_ZEIT	TEMPERATUR
Hof	25. Februar 2005, 08:00 Uhr	-12
Hof	25. Februar 2005, 12:00 Uhr	-2
Hof	25. Februar 2005, 16:00 Uhr	-6
Hof	25. Februar 2005, 20:00 Uhr	-13
München	25. Februar 2005, 08:00 Uhr	-9
München	25. Februar 2005, 12:00 Uhr	3
München	25. Februar 2005, 16:00 Uhr	-3
München	25. Februar 2005, 20:00 Uhr	-6

8 Zeilen ausgewählt.

Tabelle 13.: Relation TEMPERATUR (neu)

5.3. Zweite Normalform

Eine Beispielrelation BESTELLUNG hat folgende Struktur:

Name	Null?	Typ
KUNDEID	NOT NULL	NUMBER(2)
ARTIKELID	NOT NULL	NUMBER(2)
LFDNR	NOT NULL	NUMBER(2)
ARTIKELBEZEICHNUNG		VARCHAR2(25)
MENGE		NUMBER(2)
BESTELLDATUM		DATE

Tabelle 14.: Struktur der Tabelle BESTELLUNG

Folgende Annahmen und Hinweise sind dabei zu beachten:

1. Der Primary Key für die Tabelle **BESTELLUNG** ist hier im Wesentlichen die Attributkombination aus **KundeID** und **ArtikelId**. Für den Fall, dass der gleiche Artikel von demselben Kunden mehrmals bestellt wurde, ist noch eine laufende Nummer **LfdNr** zum Primary Key hinzugefügt worden. Damit ergibt sich folgende Attributkombination für den Primary Key:

   A = (KundeID, ArtikelId, LfdNr)

2. Zusätzlich wurde „aus Peformancegründen“ das Attribut **Artikelbezeichnung** mit in die Tabelle **BESTELLUNG** aufgenommen. Der Anwender will zu einer Bestellung sehr schnell die Bezeichnung des bestellten Artikels sehen und die Anwendung, die erst auf die Tabelle **ARTIKEL** zugreifen muss, um die **Artikelbezeichnung** einzulesen, ist zu langsam.

Die Tabelle **BESTELLUNG** enthält beispielsweise die folgenden Datensätze (aufsteigend sortiert nach der **ArtikelId**):

#KundeId	#ArtikelId	#LfdNr	Artikelbezeichnung	Menge	Bestelldatum
13	2	1	Hilfsmotoren	5	12. Juli 2004
13	2	2	Hilfsmotoren	12	01. Februar 2005
12	2	1	Hilfsmotoren	1	01. Februar 2005
3	3	1	Video-Recorder	1	07. Dezember 2004
12	4	1	Steinway Flügel	9	17. März 2004
12	6	1	Akku	4	21. Januar 2005
4	7	1	Altsaxophon	2	04. Februar 2005
4	9	1	Computer	3	23. März 2004
3	10	1	Software Engineering	12	20. Oktober 2004
4	10	1	Software Engineering	20	19. November 2004

10 Zeilen ausgewählt.

Tabelle 15.: Inhalt der Tabelle BESTELLUNG

Eine solche Modellierung birgt sehr große Risiken in sich: zum Beispiel muss zum Ändern der Bezeichnung eines Artikels nicht nur ein UPDATE in der Tabelle ARTIKEL vorgenommen werden, sondern es müssen alle Datensätze in der Tabelle BESTELLUNG darauf hin untersucht werden, ob hier ebenfalls das UPDATE durchgeführt werden muss.

Offensichtlich sind Informationen redundant gespeichert, etwas was unbedingt vermieden werden sollte. Es gilt dabei zu untersuchen, ob der Zeitvorteil zum schnellen Auffinden der **Artikelbezeichnung** nicht durch den Zeitaufwand beim Ändern von Datensätzen wieder aufgebraucht wird. Auf jeden Fall ist der Verlust der Zuverlässigkeit in Größenordnungen zu beachten, der die Daten inkorrekt und inkonsistent werden lässt.

Zum Erkennen solcher Designfehler gibt es die Regel der zweiten Normalform.

Abhängig von dem Umfang der Schlüsselattributkombination lassen sich zwei Formulierungen finden, die auf die zu prüfende Tabelle anzuwenden sind:

Zweite Normalform bei genau einem Schlüsselkandidaten:

Definition 25 - Zweite Normalform:

Eine Relation liegt in der zweiten Normalform vor, wenn sie

a) die erste Normalform erfüllt und

b) jedes Attribut, das nicht zum Primary Key gehört, zwar vom gesamten Primary Key funktional abhängig ist, jedoch von keiner echten Teilmenge von Attributen des Primary Key funktional abhängig ist.

Im vorherigen Beispiel war

A = (KundeId, ArtikelId, LfdNr)

der einzige Schlüsselkandidat und daher auch der Primary Key, es bestand aber die funktionale Abhängigkeit:

(ArtikelId) → (Artikelbezeichnung)

Eine offensichtliche Verletzung der zweiten Normalform.

Im allgemeinen Fall, dass eine Tabelle genau einen oder aber mehrere Schlüsselkandidaten besitzt lautet die zweite Normalform:

Definition 26 - Zweite Normalform (allgemein):

Eine Relation liegt in der zweiten Normalform vor, wenn sie

a) die erste Normalform erfüllt und

b) jedes Attribut, das nicht zu irgendeinem Schlüsselkandidaten gehört, zwar von jedem vollständigen Schlüsselkandidaten funktional abhängig ist, jedoch von keiner echten Teilmenge von Attributen irgendeines Schlüsselkandidaten funktional abhängig ist.

Betrachtet man wieder das Beispiel der Tabelle **BESTELLUNG** und nimmt an, dass das Attribut **Artikelbezeichnung** grundsätzlich für jeden Artikel eindeutig sei.

Dann ist doch die Attributkombination

(KundeId, Artikelbezeichnung, LfdNr)

ein weiterer Schlüsselkandidat für diese Tabelle und die funktionale Anhängigkeit

(ArtikelId) → (Artikelbezeichnung)

ist eine Abhängigkeit zwischen Elementen von Schlüsselkandidaten und damit **keine** Verletzung der zweiten Normalform.

Für die Bewertung des Verstoßes gegen die zweite Normalform kann folgende Matrix einen Überblick geben:

	Zweite Normalform für **BESTELLUNG**
Artikelbezeichnung ist ***nicht*** eindeutig	***nicht*** erfüllt
Artikelbezeichnung ist eindeutig	erfüllt

Im Allgemeinen kann aber in diesem Beispiel davon ausgegangen werden, dass die **Artikelbezeichnung *nicht*** eindeutig ist und damit kommt es zur Verletzung der zweiten Normalform.

Die zweite Normalform ist nur dann für die Überprüfung einer Tabelle von Nutzen, wenn der Primary Key zusammengesetzt ist, wenn er also aus mehr als einem Attribut besteht.

5.4. Dritte Normalform

Geht man aber davon aus, dass der Primary Key aus nur einem Attribut besteht, greift die Überprüfung der zweiten Normalform nicht, sondern es muss die Regel der dritten Normalform angewendet werden.

Dazu wurde die Tabelle **BESTELLUNG** so modifiziert, dass nur noch ein Attribut **(#Id)** als Primary Key verwendet wird.

Damit ist scheinbar alles in Ordnung, der Primary Key besteht aus einem Attribut **(#Id)**, das Attribut **(#LfdNr)** wird nicht mehr benötigt und entfällt.

Die folgende Tabelle gibt die ersten zwölf Datensätze der modifizierten Tabelle **BESTELLUNG**, ebenfalls sortiert nach **ArtikelId** wieder:

#Id	KundeId	ArtikelId	Artikelbezeichnung	Menge	Bestelldatum
3	13	1	Lampenschirm	3	03. Oktober 2004
29	23	2	Hilfsmotoren	7	17. September 2004
27	13	2	Hilfsmotoren	5	12. Juli 2004
12	13	2	Hilfsmotoren	12	01. Februar 2005
18	12	2	Hilfsmotoren	1	01. Februar 2005
13	3	3	Video-Recorder	1	07. Dezember 2004
15	12	4	Steinway Flügel	9	17. März 2004
1	12	6	Akku	4	21. Januar 2005
6	4	7	Altsaxophon	2	04. Februar 2005
9	4	9	Computer	3	23. März 2004
4	3	10	Software Engineering	12	20. Oktober 2004
19	4	10	Software Engineering	20	19. November 2004

12 Zeilen ausgewählt.

Tabelle 16.: modifizierte Tabelle BESTELLUNG

Der Primary Key ist nicht mehr zusammengesetzt und damit ist die zweite Normalform automatisch erfüllt. Trotzdem scheinen die Bedenken bei der Verletzung der zweiten Normalform im vorherigen Abschnitt noch genauso.

Es ist notwendig, eine weitere Regel zu verwenden, um auch diesem Designfehler aufzudecken. Dies ist die dritte Normalform, für deren Definition es einer weiteren Abhängigkeit, der transitiven Abhängigkeit bedarf:

Definition 27 - Transitive Abhängigkeit:

Sei R eine Relation und seien A, B und C beliebige Teilmengen der Attribute von R. Dann heißt C transitiv abhängig von einem Attribut A, wenn es die folgenden funktionalen Abhängigkeiten gibt:

A → B und B → C, aber nicht A → C.

Man schreibt dafür auch: A → B → C

Falls mindestens eine der funktionalen Abhängigkeiten A → B, B → C oder A ∪ B → C trivial ist, heißt die transitive Abhängigkeit A → B → C trivial.

Damit lassen sich wiederum zwei Definitionen für die dritte Normalform finden:

Dritte Normalform bei genau einem Schlüsselkandidaten:

Definition 28 - Dritte Normalform:

Eine Relation liegt in der dritten Normalform vor, wenn sie

a) die zweite Normalform erfüllt und

b) in der Relation keine nicht-trivialen transitiven Abhängigkeiten von dem Primary Key vorliegen.

Dritte Normalform bei beliebiger Anzahl von Schlüsselkandidaten:

Definition 29 - Dritte Normalform (allgemein):

Eine Relation liegt in der dritten Normalform vor, wenn sie

a) die zweite Normalform erfüllt und

b) in der Relation keine Attribut, das nicht zu irgendeinem Schlüsselkandidaten gehört, auf nicht-trivialer Weise von einem Schlüsselkandidaten transitiven abhängig ist.

Um mehrere Schlüsselkandidaten zu erhalten wurde die Tabelle BESTELLUNG wiederum modifiziert und das Attribut LfdNr wurde wieder hinzugefügt.

Die folgende Tabelle gibt die ersten zwölf Datensätze der nochmals modifizierten Tabelle **BESTELLUNG**, ebenfalls sortiert nach **ArtikelId** wieder:

#Id	KundeId	ArtikelId	LfdNr	Artikelbezeichnung	Menge	Bestelldatum
1	13	1	1	Lampenschirm	3	03. Oktober 2004
2	23	2	1	Hilfsmotoren	7	17. September 2004
3	13	2	1	Hilfsmotoren	5	12. Juli 2004
4	13	2	2	Hilfsmotoren	12	01. Februar 2005
5	12	2	1	Hilfsmotoren	1	01. Februar 2005
6	3	3	1	Video-Recorder	1	07. Dezember 2004
7	12	4	1	Steinway Flügel	9	17. März 2004
8	12	6	1	Akku	4	21. Januar 2005
9	4	7	1	Altsaxophon	2	04. Februar 2005
10	4	9	1	Computer	3	23. März 2004
11	3	10	1	Software Engineering	12	20. Oktober 2004
12	4	10	1	Software Engineering	20	19. November 2004

12 Zeilen ausgewählt.

Tabelle 17.: erneut modifizierte Tabelle BESTELLUNG

Leicht ersichtlich ist die Verletzung der dritten Normalform, da die folgende nicht-triviale transitive Abhängigkeit von dem Schlüsselkandidaten **(#Id)** vorliegt:

(#Id) → (ArtikelId) → (Artikelbezeichnung)

Wieder wäre diese Tabelle in der dritten Normalform für den Fall, dass die **Artikelbezeichnung** eindeutig wäre, denn dann wäre die **Artikelbezeichnung** ein Teil des Schlüsselkandidaten

(KundeId, Artikelbezeichnung, LfdNr)

Und solche transitiven Abhängigkeiten erlaubt die dritte Normalform.

Folgende Matrix gibt die Sachverhalte wieder:

	Zweite Normalform für **BESTELLUNG**	Dritte Normalform für BESTELLUNG
Artikelbezeichnung ist ***nicht*** eindeutig	***nicht*** erfüllt	***nicht*** erfüllt
Artikelbezeichnung ist eindeutig	erfüllt	erfüllt

6. Die Sprache Structure Query Language

Die Sprache SQL ist die Standard-Zugriffssprache für relationale Datenbanksysteme. Sie besteht aus mehreren Teilsprachfamilien.

6.1. Die Data Manipulation Language

Entsprechend den Operationen auf dem relationalen Datenmodell gliedert sich diese Sprachfamilie in einen Befehl für das Lesen von Daten und eine Gruppe von Befehlen zum Schreiben der Daten. Diese Gruppe von Befehlen wird als ***Updateoperationen*** bezeichnet. Damit ist eine Reihe von Problemen verbunden, besonders in Bezug auf die Integrität der Daten.

Durch diese Operationen muss folgendes möglich sein:

- ➔ Einfügen eines Tupels in eine Relation, wobei garantiert sein muss, dass der Primärschlüssel des einzufügenden Tupels noch nicht in der Relation existiert.

- ➔ Löschen eines oder mehrerer Tupel, d.h. einer Tupelmenge, spezifiziert durch einen oder mehrere Werte eines oder mehrerer Attribute.

- ➔ Modifizieren eines oder mehrerer Tupel, ebenfalls ausgewählt über Primärschlüssel oder Werte von Attributen.

6.1.1. Die SELECT-Anweisung

In ihrer einfachsten Form muss eine SELECT-Anweisung eine SELECT-Klausel enthalten, in der die anzuzeigenden Spalten angegeben sind, sowie eine FROM-Klausel, in der die Tabelle angegeben ist, die die in der SELECT-Klausel aufgeführten Spalten enthält.

```
SELECT      [DISTINCT|ALL] {*, column [alias], expr …}
FROM        table, …
[WHERE      condition(s)]
[GROUP BY   expr [, expr] …]
[HAVING     condition(s)]
[ORDER BY   {column, expr, alias} [ASC|DESC]];
```

Abbildung 51.: Syntax der SELECT-Anweisung

Syntax:

SELECT	Liste mit einer oder mehreren Spalten
DISTINCT	Schlüsselwort, um vorkommende Zeilen auszuschließen
*	Auswahl aller Spalten aller in der FROM-Klausel aufgeführten Tabellen, Views oder Snapshots
column	Auswahl der benannten Spalte
alias	Ausgewählte Spalten erhalten andere Überschriften
expr	Zeichenkette oder errechneter Ausdruck
FROM *table*	Schlüsselwort zur Angabe der Tabelle, View, oder des Snapshot mit den Spalten
WHERE	Klausel zur Einschränkung der auszuwählenden Zeilen entsprechend einer Bedingung
GROUP BY	zur Gruppierung ausgewählter Zeilen und zur Rückgabe einer zusammenfassenden Zeile

HAVING	gibt an, welche durch die GROUP-BY-Klausel definierten Zeilengruppen von der Abfrage zurückgegeben werden
ORDER BY	Reihenfolge zur Anzeige der abgerufenen Zeilen.
ASC:	aufsteigend (Standard),
DESC:	absteigend

Folgende einfache Regeln und Richtlinien müssen einhalten werden, damit gültige Anweisungen konstruiert werden, die leicht lesbar und einfach zu editieren sind:

- SQL-Anweisungen unterscheiden keine Groß- und Kleinbuchstaben.
- Klauseln stehen gewöhnlich zur leichteren Lesbarkeit und Bearbeitung in separaten Zeilen.
- Schlüsselwörter werden gewöhnlich in Großbuchstaben angegeben; alle anderen Wörter, wie z.B. Tabellennamen und Spalten, werden in Kleinbuchstaben geschrieben.
- In SQL*Plus wird eine SQL-Anweisung beim SQL-Prompt eingegeben, und die Folgezeilen werden nummeriert. Wenn Sie vor dem Beenden einer Anweisung [return] drücken, erfolgt ein Zeilenumbruch, und die Zeile erhält eine Nummer. Der Inhalt einer Anweisung steht im so genannten *SQL-Puffer*. Es kann immer nur eine Anweisung im Puffer stehen. Eine Anweisung wird entweder durch Eingabe eines Endezeichens (Semikolon oder Schrägstrich) oder durch zweimaliges Drücken von [return] beendet. Anschließend sehen Sie wieder den SQL-Prompt.

Folgendes Beispiel demonstriert anschaulich die Anwendung dieser Regeln:

```
SQL>  SELECT   last_name, salary, 12 * (salary + 100)
  2   FROM     employees;
```

Abbildung 52.: SELECT-Anweisung mit Ausdruck

Die Ausgabe in iSQL*Plus von ORACLE ist standardmäßig wie folgt formatiert:

LAST_NAME	SALARY	12*(SALARY+100)
Whalen	4400	54000
Hartstein	13000	157200
Fay	6000	73200
Higgins	12000	145200
Gietz	8300	100800
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
Lorentz	4200	51600
Mourgos	5800	70800
Rajs	3500	43200
Davies	3100	38400
Matos	2600	32400
Vargas	2500	31200
Zlotkey	10500	127200
Abel	11000	133200
Taylor	8600	104400
Grant	7000	85200

20 Zeilen ausgewählt.

Abbildung 53.: Ausgabe zur SELECT-Anweisung nach Abbildung 52

6.1.2. Die INSERT INTO Anweisung

Der Aufbau der INSERT INTO Anweisung lässt zwei Möglichkeiten des Hinzufügens von Daten in eine Tabelle zu. In der einfachsten Notationsform wird mit jedem Befehl genau ein Datensatz in eine Tabelle eingefügt.

```
INSERT INTO    table [(column [, column …])]
VALUES         (value [, value …]);
```

Abbildung 54.: Die INSERT INTO Anweisung

In der Syntax ist:

table	der Name der Tabelle
column	der Name der mit Daten zu füllenden Tabellenspalte
value	der entsprechende Wert für die Spalte

Durch diesen Befehl wird immer nur eine Zeile in die Datenbanktabelle eingefügt. Sollen Daten aus anderen Tabellen eingefügt werden so muss folgende Befehlsnotation benutzt werden:

```
INSERT INTO    table [(column [, column …])]
               SELECT   column [, column …])
               FROM     table, …;
```

Abbildung 55.: Die INSERT INTO Anweisung mit SELECT

6.1.3. Die UPDATE Anweisung

Mit Hilfe der UPDATE-Anweisung können vorhandene Zeilen bearbeiten werden.

```
UPDATE      table
SET         column = value [, column = value …])
[WHERE      condition(s)];
```

Abbildung 56.: Die UPDATE Anweisung

In der obigen Syntax ist:

table	der Name der Tabelle,
column	der Name der mit Daten zu füllenden Tabellenspalte,
value	der entsprechende Wert oder die Unterabfrage für die Spalten,
conditions	identifiziert die zu aktualisierenden Zeilen und setzt sich zusammen aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren

Anschließend sollte der Änderungsvorgang durch Abfragen der Tabelle überprüft werden, um die aktualisierten Zeilen anzuzeigen.

Generell sollten, falls vorhanden der Primärschlüssel verwendet werden, um eine einzelne Zeile zu identifizieren. Wenn Sie andere Spalten verwenden, könnte das zu unerwarteten Änderungen mehrerer Zeilen führen. Eine einzelne Zeile der Tabelle EMPLOYEES beispielsweise über den Namen zu identifizieren, könnte gefährlich sein, da mehrere Mitarbeiter den gleichen Namen haben können.

6.1.4. Die DELETE FROM Anweisung

Mit Hilfe der DELETE-Anweisung können vorhandene Zeilen gelöscht werden.

```
DELETE [FROM]  table
[WHERE         condition(s)];
```

Abbildung 57.: Die DELETE [FROM] Anweisung

In der Syntax ist:

table	der Tabellenname,
conditions	identifiziert die zu löschenden Zeilen und setzt sich zusammen aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren.

6.2. Die Data Definition Language

Diese Teilsprache von SQL dient zum Erzeugen, Ändern und Löschen von Datenbankobjekten. Aus der Vielzahl möglicher Objekte seinen hier nur die wichtigsten behandelt. Es handelt sich um die Tabellen als Ort der Abspeicherung von Daten.

Das konzeptionelle Schema mündet in eine Vielzahl, miteinander über Beziehungen verbundener Tabellen – Relationen (für Entitätsmengen und Beziehungen).

Views dienen der Steuerung des Zugriffs verschiedener Benutzergruppen auf die Daten.

6.2.1. Die CREATE TABLE Anweisung

Es werden Tabellen zum Speichern von Daten erstellt, indem die Anweisung CREATE TABLE ausführt wird. Dies ist eine Anweisung der Data Definition Language (DDL). DDL-Anweisungen sind eine Untermenge von SQL-Anweisungen zum Erstellen, Ändern oder Entfernen von Datenbankstrukturen. Diese Anweisungen wirken sich unmittelbar auf die Datenbank aus und zeichnen auch Informationen im Data Dictionary auf.

Um eine Tabelle zu erstellen, benötigt ein Benutzer das Privileg CREATE TABLE sowie einen Speicherbereich, in dem er die Objekte erstellt. Der Datenbank-Administrator verwendet zum Vergeben von Privilegien Anweisungen der Data Control Language (DCL).

```
CREATE [GLOBAL TEMPORARY] TABLE [schema.] table
      (column datatype [DEFAULT expr], …);
```

Abbildung 58.: Die CREATE TABLE Anweisung

In der Syntax ist:

GLOBAL TEMPORARY	gibt an, dass die Tabelle temporär ist und dass ihre Definition in allen Session sichtbar ist. Die Daten einer temporären Tabelle sind nur in der Session sichtbar, in welcher die Daten eingefügt wurden.
schema	entspricht dem Eigentümernamen.
table	ist der Name der Tabelle.
DEFAULT *expr*	gibt einen Standardwert an, falls die INSERT-Anweisung keinen Wert enthält.
column	ist der Name der Spalte.
datatype	ist der Datentyp und die Länge der Spalte.

Die Datentypen beschreiben den Wertebereich der Attribute und werden allgemein in Datenbanksystemen die folgenden Möglichkeiten zugeordnet:

- Zahlen,
- Geldbeträgen,
- Datum und Zeitwerten und
- Zeichenketten.

Im Besonderen ermöglichen moderne Datenbankmanagementsysteme auch den Umgang mit großen binären Datenmengen, den BLOBs.

Im Abschnitt 3.1. Mathematische Grundlagen wird ein Beispiel für eine CREATE TABLE Anweisung gegeben.

6.2.2. Die CREATE VIEW Anweisung

Eine View kann erstellen, indem eine Unterabfrage in die Anweisung CREATE VIEW einbetten wird.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
       [(alias [, alias] …)]
as subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY];
```

Abbildung 59.: Die CREATE VIEW Anweisung

In der Syntax ist:

OR REPLACE	erstellt die View neu, falls sie bereits vorhanden ist.
FORCE	erstellt die View unabhängig davon, ob die Basistabelle existiert.
NOFORCE	erstellt die View nur, wenn die Basistabelle existiert. Dies ist der Standard.
view	ist der Name der View.
alias	gibt Namen für die Ausdrücke an, die durch die View-Abfrage ausgewählt wurden. Die Anzahl der Aliasnamen muss mit der Anzahl der durch die View ausgewählten Ausdrücke übereinstimmen.
subquery	ist eine komplette SELECT-Anweisung. Für die Spalten in der SELECT-Liste können Sie Aliasnamen verwenden.
WITH CHECK OPTION	gibt an, dass nur solche Zeilen eingefügt oder aktualisiert werden können, die über die View zugänglich sind.
constraint	ist der dem Constraint CHECK OPTION zugewiesene Name.
WITH READ ONLY	stellt sicher, dass keine DML-Operationen auf diese View durchgeführt werden können.

6.3. Standardisierung der Sprache SQL

In den frühen 1970er Jahren entwickelte IBM unter der Führung des Wissenschaftlers Dr. E. F. Codd für relationale Datenmodelle ein Produkt namens **SEQUEL**, oder **S**tructured **E**nglish **Que**ry **L**anguage. Aus SEQUEL wurde dann **SQL** oder **S**tructured **Q**uery **L**anguage.

Ebenso wie andere Hersteller von relationalen Datenbankmanagementsystemen suchte IBM nach einem Standardverfahren zum Datenzugriff und zur Datenverarbeitung in einer relationalen Datenbank. Doch obwohl IBM die erste Firma war, die die Theorie der relationalen Datenbanken entwickelte, war Oracle die erste, die sie vermarktete

Die Sprache SQL hat sich seit ihrem ersten Erscheinen zu einem der wirklich wichtigen Standards in der Informationstechnik entwickelt. Als Gremien kümmern sich ISO (**ISO** – **I**nternational **O**rganization for **S**tandardization, **ANSI** – **A**merican **N**ational **S**tandards **I**nstitute) und IEC in ihrem Arbeitsausschuss „JTC 1/SC 32 Datenmanagement und Datenaustausch des Internationalen Komitees ISO/IEC JTC 1 Informationstechnik“ um die kontinuierliche Weiterentwicklung der ISO/IEC 9075- Norm, wie SQL im Standardisierungsjagon heißt.

Im Dezember 2003 erschien nach 1986, 1989, 1992 und 1999 die fünfte Version dieses Standards – SQL:2003.

6.3.1. Konformitätsebenen

Konformitätsregeln wurden erstmals von SQL92 eingeführt, und zwar in drei Kategorien: Entry, Intermediate und Full, übersetzt als Anfangsniveau, Zwischenniveau und volle Übereinstimmung. Die Hersteller müssen zumindest eine Konformität im Anfangsniveau erreichen, um eine Übereinstimmung mit ANSI SQL für sich in Anspruch nehmen zu können.

Später änderte SQL99 diese Grundebenen der Konformität. Entry, Intermediate und Full verschwanden von der Bildfläche. Mit SQL99 mussten Hersteller alle Features der untersten Konformitätsebene, nämlich Core SQL99, implementieren, um behaupten (und veröffentlichen) zu dürfen, sie seien bereit für SQL99. Core SQL99 umfasst alle Features des früheren Entry-Levels von SQL92 sowie Features von anderen SQL92-Levels und einige ganz neue Features. Ein Hersteller kann auch zusätzliche Features implementieren, die im SQL99-Standard beschrieben werden.

6.3.2. Pakete mit zusätzlichen Features im SQL2003-Standard

Der SQL2003-Standard stellt das Ideal dar, aber nur wenige Hersteller können die Anforderungen von Core SQL2003 erfüllen oder übererfüllen. Die neuen zusätzlichen Features repräsentieren verschiedene Teilmengen von Befehlen und können von den Herstellern implementiert werden oder auch nicht. Manche Features tauchen in mehreren Paketen auf, andere in keinem. Diese Pakete und ihre Features werden in der folgenden Liste beschrieben:

Teil 1 – SQL/Framework

Enthält Definitionen und Konzepte, die im gesamten Standard verwendet werden. Legt fest, wie der Standard strukturiert ist und wie seine Einzelheiten zusammenhängen. Beschreibt die Konformitätsforderungen, die das Standard-Komitee erlassen haben.

Teil 2 – SQL/Foundation

Dies ist der Kern (Core): Er ist eine Erweiterung des SQL99-Core und der umfangreichste und wichtigste Teil des Standards.

Teil 3 – SQL/CLI (Call-Level-Interface)

Legt fest, über welche Aufrufschnittstelle (Call-Level-Interface) externe Anwendungen dynamisch SQL-Anweisungen aufrufen können. Im SQL/CLI-Teil finden sich außerdem mehr als 60 Spezifikationen für Routinen, die die Entwicklung einer wirklich portablen Software erleichtern.

Teil 4 – SQL/PSM (Persistent Stored Modules)

Standardisiert prozentualer Sprachkonstrukte, wie man sie in proprietären SQL-Dialekten wie PL/SQL und Transact-SQL findet.

Teil 9 – SQL/MED (Management of External Data)

Legt fest, wie Daten außerhalb der Datenbankpattform mit Hilfe von Datalinks und einem Wrapper-Interface verwaltet werde.

Teil 10 – SQL/OBJ (Object Language Binding)

Beschreibt, wie SQL-Anweisungen in Java-Programmen eingebettet werden können. Hängt eng mit **JDBC** (**J**ava **D**ata**b**ase **C**onnection) zusammen, bietet aber auch Vorteile gegenüber JDBC. Ist außerdem sehr verschieden von der traditionellen Hostsprachen-Bindung, die in frühen Versionen des Standards möglich waren.

Teil 11 – SQL/Schemata

Definiert mehr als 85 Views (drei mehr als in SQL99), die die Metadaten jeder Datenbank beschreiben und in einem Schema namens INFORMATION_SCHEMA gespeichert werden. Einige Views, die in SQL99 bereits existierten, wurden aktualisiert.

Teil 12 – SQL/JRT (Java Routines and Types)

Definiert eine Reihe von SQL-Routinen und –Typen in der Programmiersprache Java. Java-Features wie statische Methoden und Java-Klassen werden nunmehr unterstützt.

Teil 14 – SQL/XML
Fügt einen neuen Typ namens XML, vier neue Operatoren (XMLPARSE, XMLSERIALIZE, XMLROOT und XMLCONCAT), mehrere neue Funktionen und das neue Prädikat IS DOCUMENT hinzu. Enthält überdies Regeln für die Zuordnung von SQL-Elementen (wie Identifiern, Schemata und Objekten) zu XML-Elementen.

Zu beachten ist dabei, dass die Teile 5, 6, 7 und 8 nicht entwurfsbedingt sind.

6.3.3. SQL2003-Anweisungsklassen

Vergleicht man die Anweisungsklassen, so wird die Trennung zwischen SQL2003 und SQL92 noch klarer. In SQL92 werden SQL-Anweisungen in drei große Gruppen eingeteilt:

Data Manipulation Language (DML)
Spezielle Datenmanipulationsbefehle wie SELECT, INSERT, UPDATE und DELETE.

Data Definition Language (DDL)
Befehle, welche die Zugänglichkeit und Bearbeitung von Datenbankobjekten beeinflussen, darunter CREATE und DROP.

Data Control Language (DCL)
Befehle zur Steuerung von Berechtigungen, darunter GRANT und REVOKE.

Dagegen nennt SQL2003 sieben Kern-Kategorien, inzwischen als Klassen bezeichnet, die einen allgemeinen Rahmen für die in SQL vorhandenen Befehlstypen bilden, siehe Tabelle 18:

Klasse	Beschreibung	Befehlsbeispiele
SQL-Verbindungsanweisungen (Connection Statements)	Aufbau und Beendigung einer Clientverbindung	CONNECT, DISCONNECT
SQL-Steuerungsanweisungen (Control Statements)	Steuern die Ausführung einer Reihe von SQL-Anweisungen	CALL, RETURN
SQL-Datenanweisungen (Data Statements)	Anweisungen mit einer persitenten Wirkung auf Daten	SELECT, INSERT, UPDATE, DELETE
SQL-Diagnostikanweisungen (Diagnistic Statements)	Anweisungen, die diagnostische Informationen bereitstellen und Exceptions und Fehler auslösen	GET DIAGNOSTICS
SQL-Schemaanweisungen (Schema Statements)	Anweisungen mit einer persitenten Wirkung auf Datenbankschema und die Objekte darin	ALTER, CREATE, DROP
SQL-Session-Anweisungen (Session Statements)	Steuern das Default-Verhalten und andere Sessions-Parameter	SET-Anweisungen, wie SET CONSTRAINT
SQL-Transaktionsanweisungen (Transaction Statements)	Bestimmen den Anfang und das Ende einer Transaktion	COMMIT, ROLLBACK

Tabelle 18.: SQL2003-Anwendungsklassen

Wer regelmäßig mit SQL arbeitet, sollte sich sowohl die alten (SQL92) als auch die neuen (SQL2003) Anweisungsklassen anschauen, da viele Programmierer und Entwickler noch die alten Bezeichnungen verwenden, auch für die neuen Features und Anweisungen.

7. Schnittstellen zu Datenbanksystemen

In vielen Bereichen, gerade bei der Entwicklung von Anwendungen im Datenbankumfeld, ist man aus verschiedenen Gründen gezwungen, die Welt eines gewählten Datenbankanbieters zu verlassen oder Anwendungen in einem heterogenen Datenbankumfeld universell einsetzbar zu gestalten. Es erwies sich als notwendig, Schnittstellen zu definieren, die aus verschiedenen höheren Programmiersprachen heraus den Zugriff auf das Relationale Datenbanksystem zu definieren. Diese Schnittstellen erfordern logischerweise eine Standardisierung und die Realisierung für verschiedenen (alle) Datenbankmanagementsysteme um diesen universellen Einsatz über Datenbanksystemgrenzen hinaus einsetzbar zu gestalten.

Historisch haben sich zwei solcher Schnittstellen herauskristallisiert, die sich für den standardisierten Zugriff auf die unterschiedlichsten Datenbanksysteme anbieten:

7.1. ODBC - Open DataBase Connectivity

ODBC steht für **O**pen **D**ata**B**ase **C**onnectivity (Offene Datenbank-Verbindungsfähigkeit) und ist ein Datenbanktreiber, bietet also eine Programmierschnittstelle (API), die es einem Programmierer erlaubt, seine Anwendung unabhängig vom verwendeten Datenbanksystem (der Datenbank) zu entwickeln. Die Technik ermöglicht es, Daten aus beliebigen Anwendungen in ein Datenbanksystem einzubinden. Diese Schnittstelle wurde ursprünglich für Microsoft Access entwickelt, ist aber inzwischen auch von anderen Softwareherstellern übernommen worden. In vielen Bereichen ist ODBC mittlerweile als Standard etabliert.

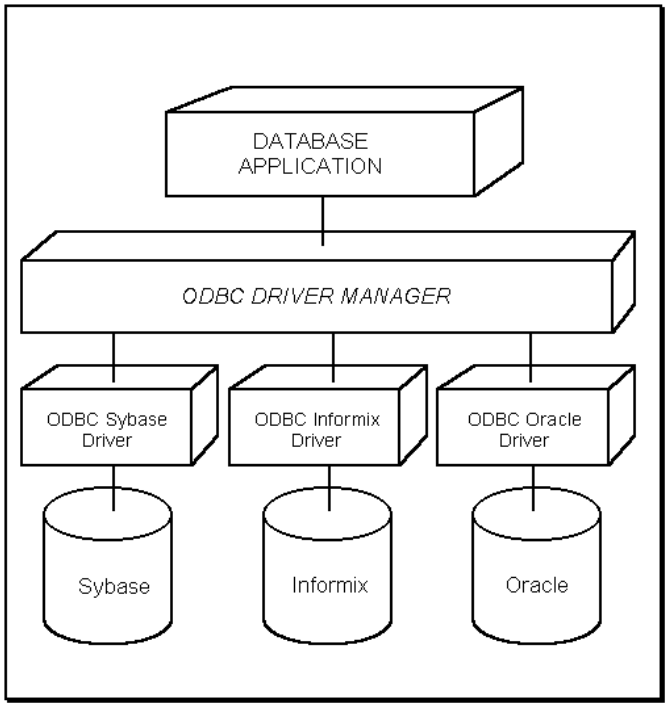
Der Aufbau und die Funktion von ODBC soll im Folgenden dargestellt werden. Dabei soll zuerst in einem Codebeispiel eine Datenbankoperation in ODBC einer Datenbankoperation in Embedded SQL gegenübergestellt werden.

```
1 /* Deklaration der Hostvariablen. Über sie erfolgt der Austausch */
2 /* der Daten zwischen Datenbankoperation und C-Programm*/
3 EXEC SQL BEGIN DECLARE SECTION;
4 char ENAME[10] ;
5 short DEPTNO;
6 EXEC SQL END DECLARE SECTION;
7
8 main()
9 {
10 /* Abfrage der Zeilen, die mit einer best. Abt.-Nr. übereinstimmen */
11 /* C1 stellt einen Zeiger auf die aus der Abfrage resultierenden   */
12 /* Zeilen dar. Er wird in einer while-Schleife zeilenweise ausgelesen */
13 EXEC SQL DECLARE C1 CURSOR FOR
14  SELECT ENAME, DEPTNO FROM EMP WHERE DEPTNO = 20;
15 EXEC SQL OPEN C1
16
17 while (SQLCODE == 0)
18  {
19  EXEC SQL FETCH C1 INTO :ENAME, DEPTNO;
20  }
21 EXEC SQL CLOSE C1
22 }
```

Diese Embedded-SQL Anwendung wird nun durch einen Precompiler übersetzt, bevor sie vom Compiler der Host-Programmiersprache (Programmiersprache, in welche die SQL-Anweisungen eingebettet sind) übersetzt werden kann. Dieser Precompiler ersetzt die eingebetteten SQL-Anweisungen (EXEC SQL) durch Funktionsaufrufe der DBMS-Laufzeitbibliothek. Damit muss für jedes zu verwendende DBMS ein spezifischer Precompiler eingesetzt werden.

Der gleiche Anwendungsfall wie im vorhergehenden Beispiel soll nun mit Hilfe der ODBC-API in C realisiert werden. Auch bei diesem Codebeispiel wird auf die Darstellung des Aufbaus und Abbaus der Verbindung zum DBMS verzichtet.

```
1 #include <sql.h>      /* enthält alle ODBC-Funktionen und Konstanten*/
2
3                       /* Deklaration der Hostvariablen. Über sie erfolgt der Austausch */
4                       /* der Daten zwischen Datenbankoperation und C-Programm*/
5 RETCODE rc;           /* Rückgabewert für ODBC-Funktionen  */
6 HSTMT hstmt;          /* Anweisungs-Handle, muss einer bestimmten  */
7                       /* Verbindung zum DBMS zugeordnet werden (hier */
8                       /* nicht weiter dargestellt)*/
9 char ENAME[10] ;
10 short DEPTNO;
11
12 main()
13 {
14                      /* Abfrage der Zeilen, die mit einer best. Abt.-Nr. übereinstimmen */
15                      /* rc stellt einen Zeiger auf die aus der Abfrage resultierenden   */
16                      /* Zeilen dar. Er wird in einer while-Schleife zeilenweise ausgelesen */
17 SQLExecuteDirect(hstmt, "select ename, deptno from emp where deptno=20");
18 for (rc = SQLFetch(hstmt); rc == SQL_SUCCESS; rc = SQLFetch(hstmt))
19  {
20  SQLGetData(hstmt, 1, ENAME);
21  SQLGetData(hstmt, 2, DEPTNO);
22  }
23 }
```



ODBC bildet eine Anwendungsschnittstelle, die eine Menge von Funktionen (z.B. *SQLConnect, SQLExecuteDirect, ...)* zur Verfügung stellt. Da diese Funktionen in ODBC standardisiert sind und damit nicht vom DBMS abhängen, braucht die Anwendung nur einmal übersetzt werden und kann dann auf verschiedene DBMS zugreifen.

Eine Datenbank-Anwendung, welche über ODBC auf eine Datenquelle zugreift besteht aus vier Komponenten: Anwendung, ODBC-Treiber-Manager, ODBC-Treiber und Datenquelle, die Architektur der ODBC-Schnittstelle ist in Abbildung 60 dargestellt.

Abbildung 60.: ODBC-Architektur

7.1.1. Datenquelle

Die Datenquelle enthält die für die Anwendung relevanten Daten. Dabei kann die Datenquelle aus einer Datenbankdatei (Textdatei, Exceldatei) oder einer Datenbankdatei und DBMS für die Zugriffsverwaltung bestehen (in diesem Fall nennt man Datenbankdatei und DBMS zusammen Datenbanksystem).

7.1.2. Anwendung

Eine Anwendung stößt durch einen Befehl einen Verbindungsaufbau zu einer ODBC-Datenquelle an. In diesem Befehl ist die adressierte ODBC-Datenquelle in Form des so genannten *Data Source Name* (DSN, Datenquellenname) enthalten. Unter dem DSN ist eine bestimmte Datenquelle im ODBC-Treiber-Manager registriert. Der Befehl zum Verbindungsaufbau kann weitere zum Verbindungsaufbau notwendige Parameter wie z.B. Benutzer-Kennung und Passwort enthalten. Ist die Verbindung aufgebaut, so kann die Anwendung SQL-Anweisungen auf die Datenquelle ausführen.

7.1.3. ODBC-Treiber-Manager

Beim Treiber-Manager handelt es sich um eine DLL (ODBC32.DLL), die von Microsoft als Teil der ODBC--Installation auf dem Rechner der Datenbank-Anwendung bereitgestellt wird. Die Aufgabe des Treiber-Managers ist es, die jeweils benötigten ODBC-Treiber zu laden. Der Treiber-Manager verwaltet die Liste der auf dem Rechner vorhandenen DSN in einer Initialisierungsdatei (ODBCINST.INI) oder der Registry. Die DSN können dabei u.a. entweder als benutzerspezifische (d.h. nur für den jeweiligen Benutzer verfügbare) DSN oder systemweite (d.h. einem Computer zugewiesene DSN, ab ODBC 2.5) angelegt werden.

7.1.4. ODBC-Treiber

Auch beim ODBC-Treiber handelt es sich um eine DLL, z.B. für Oracle SQORA*.DLL. Die für den jeweiligen Treiber notwendigen Parameter zum Verbindungsaufbau mit einer Datenquelle werden ebenfalls in einer Initialisierungsdatei (ODBC.INI) bzw. in der Registry gespeichert. Ein ODBC-Treiber enthält Funktionen zum Verbindungsaufbau mit einer Datenquelle, er interpretiert die Abfragen der Anwendung und sendet sie an die Datenquelle. Außerdem übernimmt er Formatkonvertierungen (z.B. Zeichensätze) zwischen Anwendung und Datenquelle.

Nachdem die Ergebnisse aus der Datenquelle ermittelt sind, gibt der Treiber sie an die Anwendung zurück. Fehlermeldungen aus der Datenquelle werden vom ODBC-Treiber interpretiert und in Form eines Standard-Fehlercodes an die Anwendung zurückgegeben.

Es existieren zwei Arten von ODBC-Treibern:

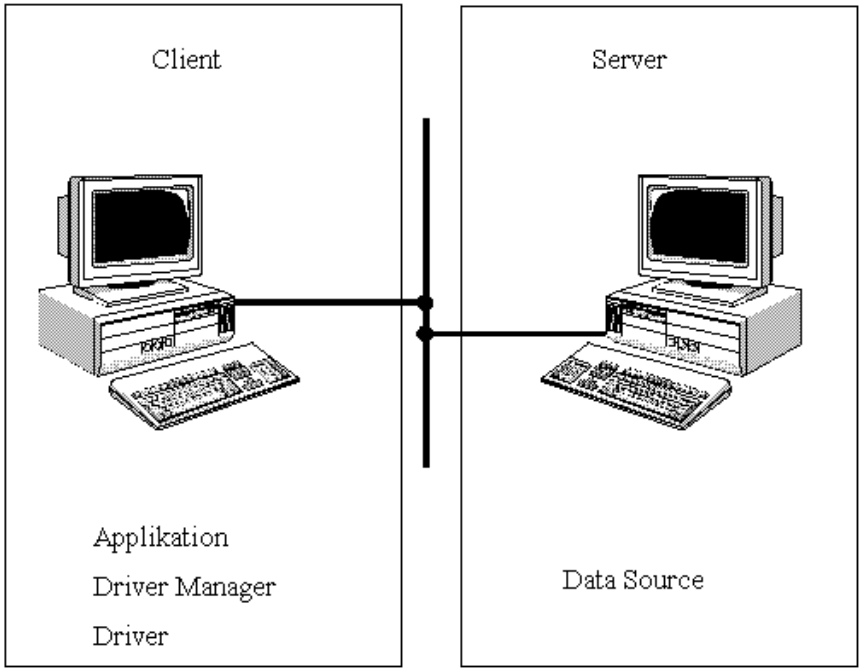
a) Single-Tier-Treiber stellen zum einen eine Verbindung zu einer Datenquelle her. Zum anderen enthalten sie die komplette Funktionalität, um direkt eine Datenbankdatei anzusprechen. Dies ist zum Beispiel notwendig, um Textdateien oder Exceldateien als Datenquelle in SQL-Anweisungen benutzen zu können. Dabei müssen diese Dateien ein bestimmtes Format aufweisen (z.B. bei Textdateien: Verzeichnis stellt Datenquelle dar, Datei stellt eine Tabelle dar innerhalb derer die Werte der Spalten z.B. durch Komma getrennt stehen; bei Excel: Datei stellt Datenquelle dar, Excel-Tabelle stellt Datenbank-Tabelle dar.). Der Single-Tier-Treiber setzt Datenbankabfragen (SQL-Anweisungen) aus der Anwendung in Abfragen an die Datenquelle um und agiert damit quasi als DBMS, indem er die Datenbankabfragen selbst verarbeitet. Dabei benutzt er z.T. Bestandteile (DLLs) der Datenquellen-Anwendung (z.B. Excel).

b) Multi-Tier-Treiber senden Datenbankabfragen an einen Datenbank-Server, der diese verarbeitet. Eventuell werden die Datenbankabfragen dabei in ein für den Server verständliches Format umgewandelt. Dabei kann sich der Server entweder auf dem lokalen Rechner oder auf einem anderen Rechner in einem Netzwerk befinden.

Ein ODBC-Treiber kann somit Folgendes beinhalten:
Eine Bibliothek mit Funktionsaufrufen zum Verbindungsaufbau (inkl. Anmeldung beim DBMS), zum Ausführen von SQL-Anweisungen und zur Ermittlung der Ergebnisse. Zum Verbindungsaufbau kann z.B. entweder die Funktion SQLConnect, SQLDriverConnect oder SQLBrowse Connect verwendet werden. Bei SQLConnect liefert die Anwendung die Verbindungsdaten (Namen der Datenquelle, die User-ID und Passwort) mit.

Bei SQLDriverConnect können die Verbindungsdaten entweder durch die Anwendung mitgeliefert oder durch Dialogfenster des ODBC-Treiber-Managers (Auswahl der Datenquelle) und des ODBC-Treibers (User-ID, Passwort) abgefragt werden. SQLBrowse Connect ermittelt die Verbindungsattribute, welche der ODBC-Treiber zum Verbindungsaufbau benötigt, sodass die Anwendung darauf aufbauend ein eigenes Dialogfenster zum Verbindungsaufbau erzeugen kann.

Eine standardisierte SQL-Syntax (eventuell in verschiedenen Ausbaustufen, Konformitätsebenen). Es kann eine SQL-Umwandlung vorgenommen werden, wenn die SQL-Syntax des DBMS nicht der standardisierten SQL-Syntax entspricht (z.B. Standard-Ungleichsoperator != aus der Anwendung wird bei Microsoft SQL Server durch DBMS-spezifischen Ungleichsoperator <> ersetzt).



Eine Standardmenge an Fehlercodes und Fehlerfunktionen (z.B. zur Rückgabe des Fehlercodes und der Fehlermeldung des DBMS an die Anwendung).

Eine Standardmethode zur Konvertierung von Datentypen und Zeichensätzen.
Katalogfunktionen: standardisierte Informationen über den Aufbau der Datenbank (Systemtabellen)

Eine mögliche Anwendung von ODBC in verteilten Systemen zeigt Abbildung 61.

Abbildung 61.: ODBC in verteilten Systemen

7.2. JDBC - Java DataBase Connectivity

Der Begriff JDBC – **J**ava **D**ata**B**ase **C**onnectivity) bezeichnet ein von JavaSoft entwickeltes und im März 1996 vorgestelltes Paket, das auf den Java-Basisklassen aufsetzt und über die Bereitstellung von relationalen Datenbankobjekten sowie der entsprechenden Methoden den Zugriff aus Java-Applikationen auf beliebige Datenbanken ermöglicht. JDBC setzt auf dem X/OPEN SQL-Call-Level-Interface (CLI) auf und besitzt damit die gleiche Basis wie die ODBC-Schnittstelle.

In der letzten Zeit ist eine immer stärker werdende Nutzung des Internets, sowohl im privaten als auch im kommerziellen Bereich zu verzeichnen. Des Weiteren setzen viele Firmen auf firmeninterne systemunabhängige Netze auf TCP/IP-Basis (sog. Intranets), welche die Möglichkeit bieten, von jedem Arbeitsplatz der Firma aus (unabhängig vom verwendeten System) mit einem Browser auf Firmendaten zugreifen zu können.

Hier zeichnet sich schon ab, dass der herkömmliche Weg, statische Daten auf HTML-Seiten auszugeben, bei weitem nicht mehr ausreicht, um die vielfältigen Bedürfnisse der Anwender zu befriedigen. In nahezu jeder kommerziellen Anwendung, die heutzutage bei einer Firma im Einsatz ist, befinden sich Datenbanken im Hintergrund, welche die Anwendungsprogramme mit den entsprechenden Informationen versorgen (z.B. Produktdatenbanken, Kundendatenbanken, etc.).

7.2.1. Serverseitiger Zugriff auf die Datenbank

7.2.1.1. Anbindung über das Common-Gateway-Interface (CGI)

Hier sendet der Client über den Web-Browser eine Anfrage an den Web-Server. Dieser leitet die Anfrage über das Common-Gateway-Interface an den Application-Server weiter, welcher den Kontakt zum Datenbanksystem herstellt und die Ergebnisdaten wieder über das CGI an den Web-Server zurückliefert. Dort wird das Ergebnis in Form statischer Webseiten (HTML) wieder an den Client zurückgesandt.

Der Nachteil dieser Methode besteht darin, dass für jede Datenbankanfrage ein eigener CGI-Prozess gestartet werden muss, was eine speicher- und zeitaufwendige Ausführung zur Folge hat.

7.2.1.2. Anbindung über eine proprietäre Web-Server-API

Diese Methode ähnelt der vorhergehenden, mit dem einzigen Unterschied, dass anstelle des standardisierten CGI eine proprietäre API verwendet wird, um den direkten Kontakt zwischen dem Web-Server und dem Application-Server (der die DB-Abfrage ausführt) herzustellen. Die Übersendung der Ergebnisdaten an den Client geschieht auch hier in Form statischer HTML-Seiten.

Der Vorteil dieser Methode gegenüber der vorhergehenden besteht in der höheren Performance, sowie der Aufrechterhaltung eines begonnenen Prozesses für folgende DB-Anfragen.

Demgegenüber besteht der wesentliche Nachteil in der komplexen Programmierung der Abfrageroutinen, die zudem aufgrund der verwendeten API system- und programmiersprachenabhängig sind, was zu erheblichen Problemen bei Änderungen der verwendeten Software führen wird.

7.2.2. Clientseitiger Zugriff auf die Datenbank

7.2.2.1. Anbindung über Java-Applets (mit JDBC)

Diese Methode (die Grundlage der vorliegenden Abhandlung ist), stellt die direkte Verbindung zwischen dem Client und dem Datenbanksystem her. Die Verbindung erfolgt auf der Grundlage der von Sunsoft entwickelte JDBC-API (Java-Database-Connectivity), welche Java-Applikationen (oder Applets) den direkten Zugriff auf Datenbanksysteme ermöglicht.

Die Vorteile dieser Lösung liegen auf der Hand: Der Web-Server wird von intensiven Datenbankabfragen entlastet, die Programmierung über CGI oder eine spezifische API entfällt, und die Portabilität ist durch die Verwendung der Java-Schnittstelle JDBC gewährleistet.

Die Nachteile dieses Verfahrens bestehen in der längeren Ladezeit (für das Applet), in der aufwendigeren Programmierung der Benutzerschnittstelle (die Oberfläche besteht nicht mehr aus HTML-Code sondern muss in Java programmiert werden) sowie in den Sicherheitsrestriktionen für das Ausführen von Java-Applets, auf die im Abschnitt genauer eingegangen wird.

7.2.3. Architektur von JDBC-Anwendungen

Ein Java-Applet, das mit einem Browser vom Server auf den Client geladen wird, kann aufgrund der strengen Sicherheitsvorschriften für Applets nur mit dem Rechner in Verbindung treten, von dem es geladen wurde. Wenn das Applet direkt mit dem Datenbankserver kommunizieren soll, ist es daher notwendig, dass Datenbank und Web-Server auf der gleichen Maschine installiert werden. Man spricht in diesem Fall von einer "Two-Tier-Architektur". Diese Architektur, die im Wesentlichen der klassischen Client-Server-Architektur entspricht, kann auf zwei Arten realisiert werden (siehe Abbildung 62 und Abbildung 63):

Der Client lädt ein Java-Applet vom Server, welches sich über ein DBMS-spezifisches Protokoll (z.B. JDBC) mit der Datenbank verbindet, die auf der gleichen Maschine liegt wie der Web-Server.

Abbildung 62.: Two-Tier-Architektur mit Java-DB-Client

Der Client lädt ein "trusted" Java-Applet vom Server, welches sich über einen nativen DBMS-Treiber (z.B. die JDBC-ODBC-Bridge) mit der Datenbank verbindet, die auf der gleichen Maschine liegt wie der Web-Server.

Abbildung 63.: Two-Tier-Architektur mit nativem DB-Client

Zu dieser Architektur ist folgendes anzumerken:

a) Die Verbindung des Web-Servers sowie des Datenbankservers auf einer Maschine setzen nicht nur eine sehr leistungsfähige Maschine voraus, sondern sind auch in sicherheitsrelevanter Sicht fraglich.

b) Das Laden des "pure java"-Treibers vom Web-Server (siehe Abbildung 63) verlängert die Ladezeit für das Applet erheblich (vereinfacht dafür Softwareverteilung und Portabilität).

c) Das Laden des nativen Treibers vom lokalen Rechner (siehe Abbildung 64) setzt ein "trusted applet" voraus und erschwert die Softwareverteilung und schränkt die Portabilität ein (verkürzt dafür die Ladezeiten).

Eine andere Möglichkeit, bei welcher der Web-Server und der Datenbankserver auf getrennten Maschinen installiert werden können ist in Abbildung 64 dargestellt. Bei dieser so genannten "Three-Tier-Architektur" wird ein Applet vom Web-Server geladen, das sich über ein DBMS-unabhängiges Protokoll mit einem Gateway (auf dem Web-Server) verbindet, welches dann als Datenbank-Client die Abfrage für das Applet übernimmt und die Ergebnisse anschließend an das Applet zurücksendet (ein Beispiel für eine solche Three-Tier-Architektur ist die Verbindung zweier Java-Applets über die Remote Method Invocation (RMI).

Abbildung 64.: Three-Tier-Architektur

Der Client lädt ein Java-Applet vom Server, welches sich indirekt über ein Gateway mit der Datenbank verbindet, die auf einem getrennten Rechner installiert ist.

Zu dieser Architektur ist folgendes anzumerken:

Die Trennung des Web-Servers und des Datenbankservers entlasten den Web-Server, und führen zu erhöhter Sicherheit für den DB-Server (Zugriff nur über das Gateway möglich), dafür ist die Programmierung des Gateways (Umsetzung der DBMS-unabhängigen Anweisungen in DBMS-spezifische Befehle) erheblich komplexer.

7.2.4. Allgemeine Funktionsweise von JDBC

Wie bereits erwähnt, definiert JDBC Objekte und Methoden, mittels derer es dem Programmierer ermöglicht wird, in der Applikation den Zugriff auf darunterliegende Datenbanken zu realisieren. Der Zugriff erfolgt in der folgenden Weise:

a) Herstellung einer Verbindung zur Datenbank über den entsprechenden JDBC-Treiber für das verwendete DBMS.

b) Erstellung eines Statement-Objektes

c) Weitergabe des auszuführenden Statements über das Statement-Objekt an das darunterliegende DBMS

d) Abholung der Ergebnisse über die Ergebnisdatensätze

e) 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 (siehe Abbildung 65) übernommen. Dieser kann mehrere Verbindungen zu unterschiedlichen Datenbanken gleichzeitig verwalten und ermöglicht somit auch den Zugriff auf verteilte Datenbanken.

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

Abbildung 65.: JDBC-Treiber-Manager

7.2.5. Verschiedene Arten von JDBC-Treibern

Wie bereits erwähnt, verlangt die Anwendung von JDBC den Einsatz eines JDBC-Treibers für die verwendete Datenbank, der über den so genannten JDBC-Treiber-Manager verwaltet wird.

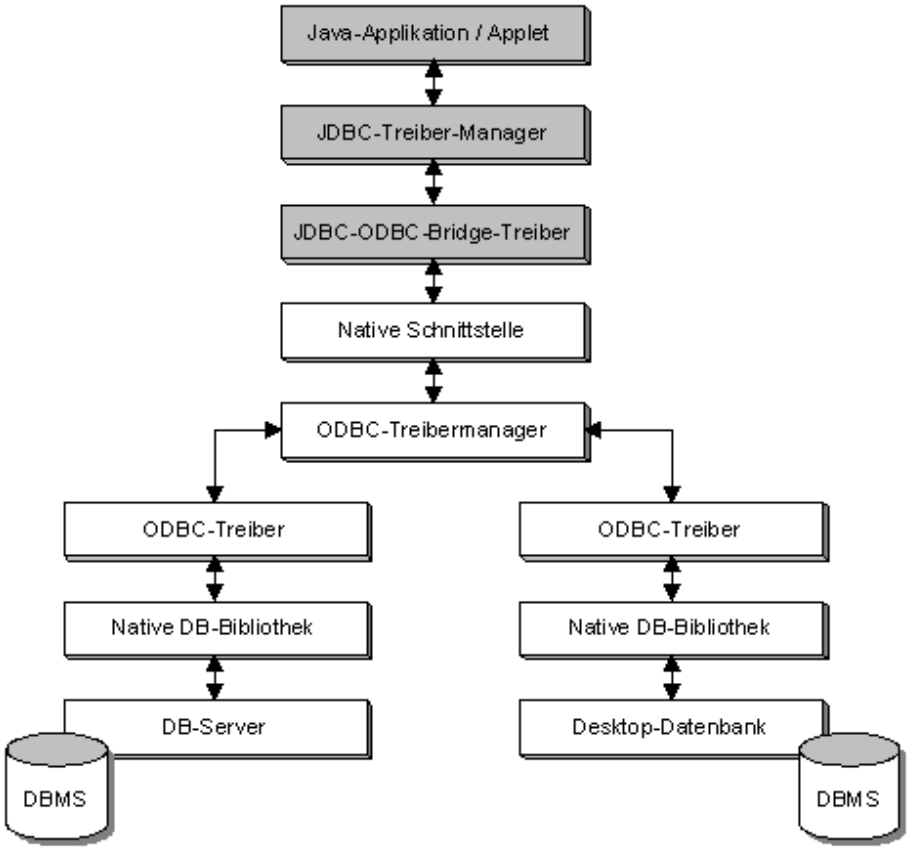


Abbildung 66.: JDBC-Treiber Typ-1

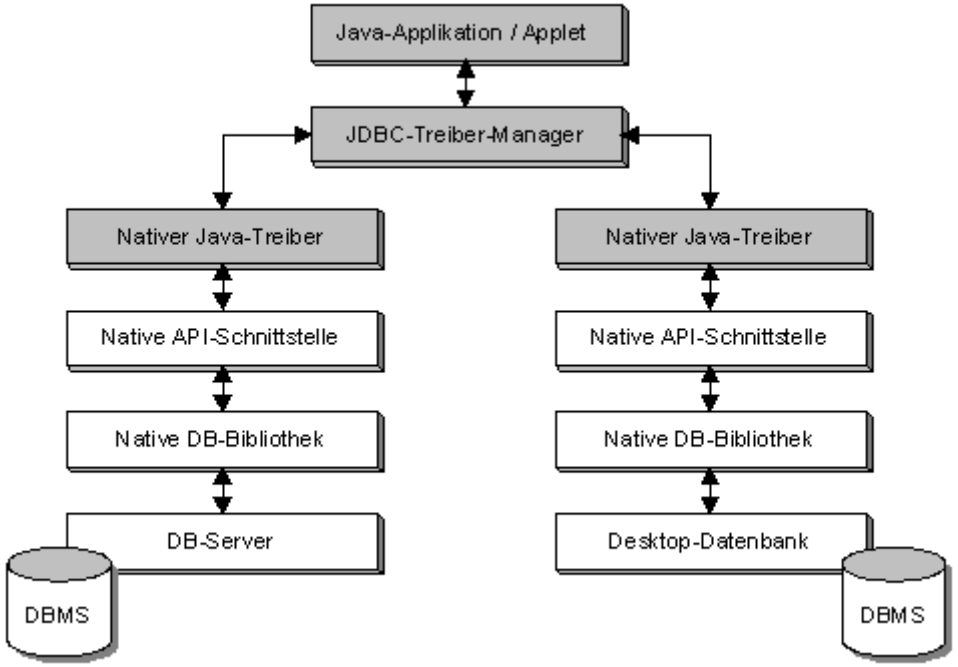
Die JDBC-Treiber lassen sich in vier verschiedene Kategorien unterteilen (Typ-1, Typ-2, Typ-3, Typ-4). Die Unterschiede in den Architekturen der einzelnen Treiber werden im nachfolgenden erläutern und anhand von Abbildungen veranschaulichen.

7.2.5.1. Typ-1

Bei dieser Variante wird für den Datenbankzugriff ein nativer ODBC-Treiber verwendet, der lokal beim Client installiert werden muss und über die "JDBC-ODBC-Bridge" von Sun angesteuert wird.

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-Teiber-Manager in Verbindung mit einem lokal installierten ODBC-Treiber zur Datenbankanbindung anspricht.

7.2.5.2. 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.

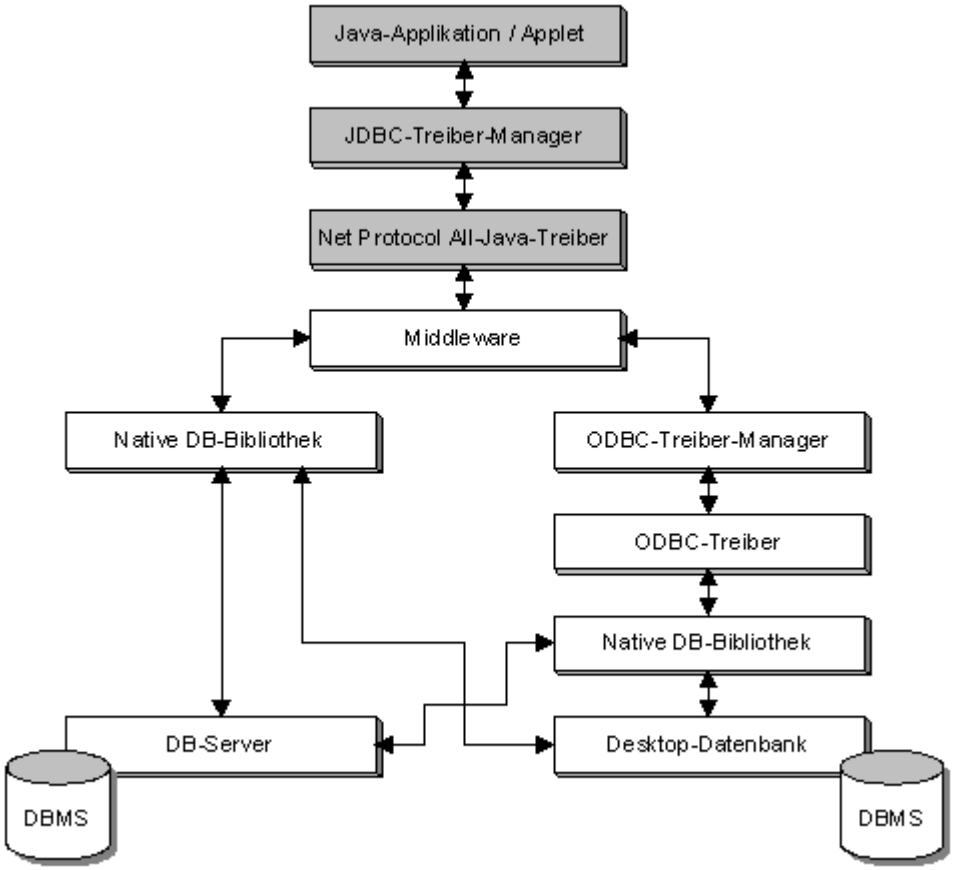
Die Java-Applikation bzw. das Applet binden über den JDBC-Treiber-Manager einen nativen JDBC-Treiber ein, welcher über native API-Schnittstellen (z.B. Windows DLLs) die Verbindung zur Datenbank herstellt.

Abbildung 67.: JDBC-Treiber Typ-2

7.2.5.3. 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.

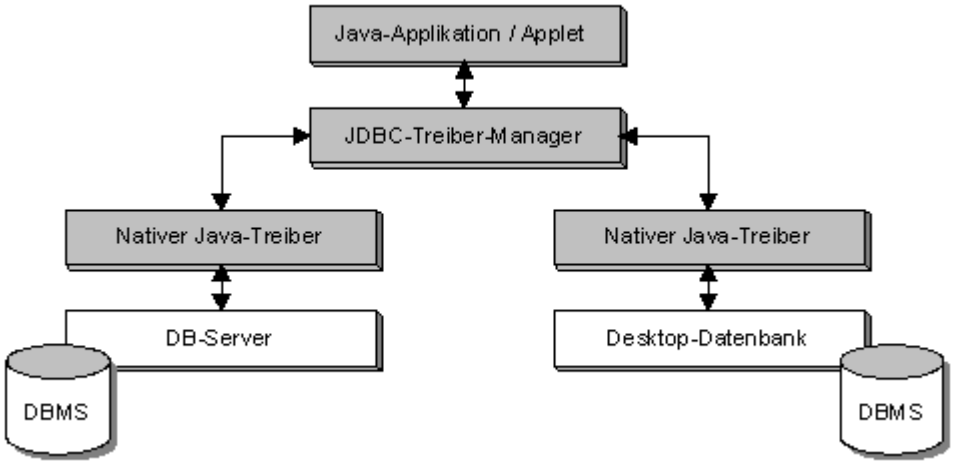


Die Java-Applikation bzw. das Applet binden über den JDBC-Treiber-Manager einen universellen Java-JDBC-Treiber ein, welcher die Aufrufe in ein DBMS-unabhängiges Protokoll übersetzt und anschließend an eine auf dem Server installierte Middleware-Applikation sendet. Diese wandelt die Aufrufe entsprechend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

Abbildung 68.: JDBC-Treiber Typ-3

7.2.5.4. 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.

Abbildung 69.: JDBC-Treiber Typ-4

7.2.6. Programmieren mit JDBC

7.2.6.1. Importieren der notwendigen Klassen

Die für die Ausführung von JDBC notwendigen Klassen liegen allesamt im Package **`java.sql`** vor, das (wie eingangs erwähnt) Bestandteil des von SUN ausgelieferten JDK 1.1.4 ist. Diese Klassen müssen vor der Anwendung innerhalb eines Java-Programms (zu Beginn des Programms) mit der Anweisung **`import Package.Klasse`** importiert werden. Durch den Platzhalter (*) kann man erreichen, dass alle Klassen des angegebenen Packages importiert werden.

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

```
import java.sql.*;
```

7.2.6.2. 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 die JDBC-ODBC-Bridge sein, die in Verbindung mit einem lokal installierten und eingerichteten ODBC-Treiber jede ODBC-Datenbank ansprechen kann, oder ein nativer 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.

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

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

7.2.6.3. Connect zur Datenbank

Hier kommt der bereits mehrfach erwähnte JDBC-Driver-Manager "ins Spiel". Es handelt sich auch hierbei um eine Klasse, die sich (erwartungsgemäß) im **`java.sql`**-Package befindet.

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

a) String IN » URL der Datenbank, zu der die Verbindung aufgenommen werden soll

b) String IN » Anmeldename auf der Datenbank

c) String IN » Passwort für die Datenbankanmeldung

Die beiden letzten Parameter sind optional und können ggf. auch als Liste übergeben werden, auf diesen Spezialfall wird hier jedoch nicht eingegangen.

Bei fehlerfreier Ausführung wird eine geöffnete Datenbankverbindung (vom Typ **`connection`**) zurückgeliefert:

d) Connection OUT » Offene DB-Verbindung

Der Aufruf im Programm hat demzufolge die folgende Syntax:

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

7.2.6.4. Erzeugen eines Statements

Ein JDBC-Statement besteht aus einem Objekt der Klasse Statement, welches mit der Methode **`createStatement()`** der Klasse **`connection`** erzeugt wird. Zu beachten ist, dass diese Methode für eine bestehende Datenbankverbindung erfolgen muss.

Die Syntax hierfür lautet also:

```
Statement my_stmt = my_con.createStatement();
```

7.2.6.5. Ausführen eines Statements

Ein 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 MUSTERTABELLE");
```

oder

```
int lv_result = my_stmt.executeUpdate("UPDATE MUSTERTABELLE SET ...");
```

7.2.6.6. Auswerten des Ergebnisses

Wie bereits erwähnt wird das Ergebnis einer Abfrage (executeQuery) in Form einer Ergebnistabelle vom Typ **`ResultSet`** zurückgeliefert. Da der Inhalt und der Aufbau dieser Ergebnistabelle von der zugrunde liegenden Datenbanktabelle (auf der die Abfrage ausgeführt wurde) abhangen, 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 deptno, ename FROM DEPT;
```

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

```
int lv_deptno = my_result.getInt("deptno");
String lv_ename = my_result.getString("ename");
```

Das Interface ResultSet bietet für jeden Datentyp eine entsprechende Methode der Art **`getTyp()`** an, mittels derer die entsprechenden Ergebnisspalten ausgelesen werden können.

Ist der Aufbau der zugrunde liegenden Datenbanktabelle nicht bekannt (Beispiel: "SELECT * FROM DEPT"), 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.

7.2.6.7. 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. Anwendungsentwicklung mit Oracle-Tools

In Erweiterung der Betrachtungen in Abschnitt 3. Datenbankentwurf – Entity-Relationship-Modell soll es im Folgenden darum gehen, aus dem entwickelten Entity-Relationship-Modell eine fertige Anwendung zu erzeugen. Weiterhin soll dazu unterstellt werden, dass nicht die klassische Variante, *»man nehme eine höhere Programmiersprache und entwickle die gesamte Anwendung damit«*, verwendet wird. Dieser Weg sollte nur beschritten werden, wenn

a) entsprechende Vorgaben des Auftraggebers (Firma oder Kunde) dies erzwingen oder
b) eine Toolunterstützung nicht vorhanden ist.

Auf jeden Fall kann man in der Regel davon ausgehen werden, dass die hier vorgeschlagene Vorgehensweise bei gleichem Kenntnisstand der Entwickler, im Vergleich zur höheren Programmiersprache, eher zu funktionierenden Lösungen führen wird.

Da sowohl das Entity-Relationship-Modell als auch die Datenbankstruktur in einem Datenbankmanagement-system vorhanden sind, kann man für die Entwicklung der Anwendung darauf aufbauen. Entsprechende Entwicklungstools der Anbieter der Datenbankmanagementsysteme nutzen diese Vorgaben aus und reduzieren dadurch die Fehlerhäufigkeit und damit den Entwicklungszeitraum, siehe Abschnitt 8.2.2. Generierung einer Java-Anwendung.

Im Fall der Verwendung des Datenbankmanagementsystems der Firma Oracle hat man dazu verschiedene Möglichkeiten der Toolunterstützung, die sich sowohl in der Zielsetzung für die Anwendungsentwicklung als auch in der Komplexität der Möglichkeiten und damit der Erlernbarkeit unterscheiden.

In der gegenwärtigen Zeit der Durchdringung des Informationsflusses innerhalb eines Unternehmens durch das Internet bzw. das Intranet, vollzieht sich ein Übergang weg von der klassischen Client/Server-Architektur hin zur webbasierten Anwendungsstruktur. Im Abschnitt 7.2. JDBC - Java DataBase Connectivity wurden diese unterschiedlichen Möglichkeiten der Gestaltung von Anwendungen bereits kurz beleuchtet.

8.1. Oracle Developer®

Als klassische Variante, um im Oracleumfeld Datenbankanwendungen zu erzeugen, ist der Oracle Developer bzw. neu, die Oracle Developer Suite zu nennen. Die enthaltenen Tools haben bereits alle Stufen der Anwendungsentwicklung von den rein Alphanumerischen über die GUI-Oberflächen bis hin zur webbasierten Anwendungsstruktur in Verbindung mit dem Oracle Application Server durchlaufen.
Kennzeichnend für die klassischen Aufgaben einer Datenbankanwendung sind die Tools Oracle Forms bzw. Oracle Forms Builder für die Generierung von Eingabemasken, Oracle Report bzw. Oracle Report Builder für die Generierung von verschiedenen Reportingvarianten und Oracle Graphics für das Erstellen irgendwelcher grafischer Übersichten um die in der Datenbank gespeicherter Daten darzustellen.

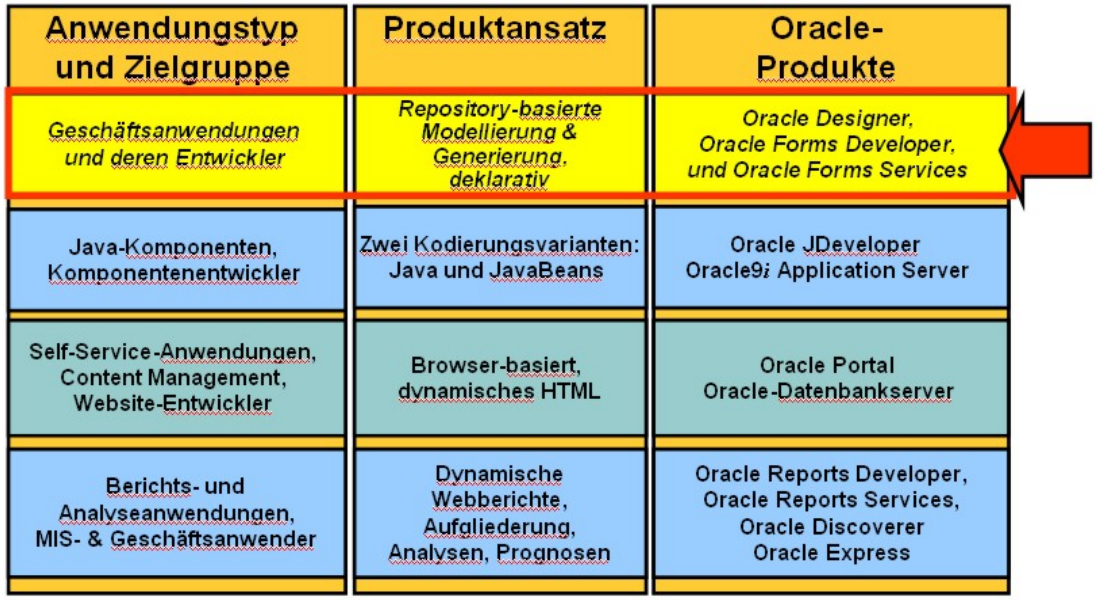


Abbildung 70: Einordnung der Oracle Tools

Das zuletzt genannte Tool, Oracle Graphics ist in der gegenwärtig ausgelieferten Version der Oracle Developer Suite nicht mehr enthalten.

8.1.1. Oracle Forms Builder

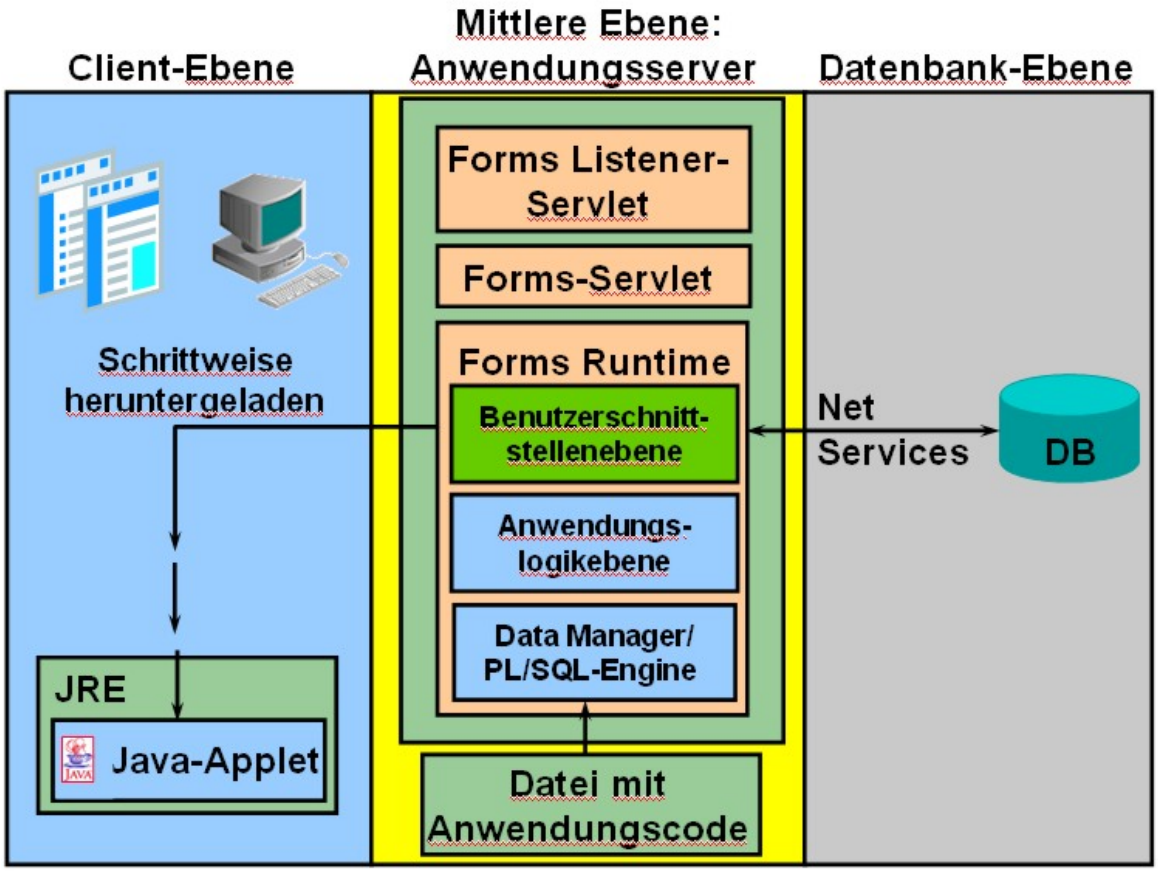
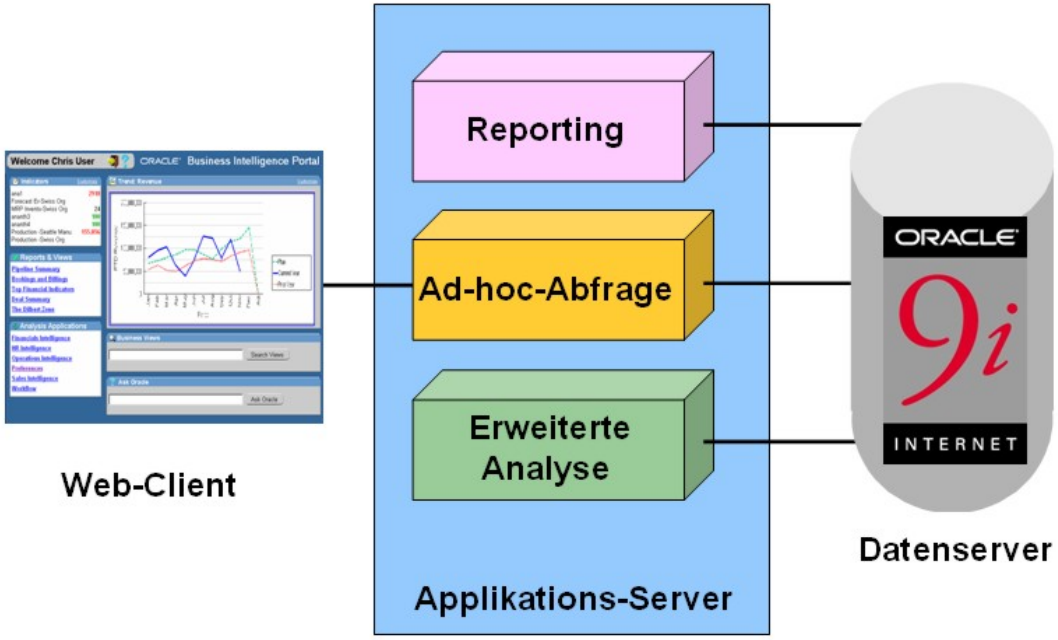


Abbildung 71: Forms Services Architektur

Sobald Anwendungen webbasiert entwickelt werden, wird die klassischen Client/Server-Architektur verlassen und es erfolgt der Übergang zur sogenannten Drei-Ebenen-Architektur, wie sie am Beispiel einer Oracle Forms Applikation in Abbildung 71 zu sehen ist. Selbstverständlich können sich die drei Ebenen auch auf drei verschiedenen Rechnersystemen oder alle Ebenen auf demselben Rechner befinden.

- Die Client-Ebene enthält den Web-Browser, in dem die Anwendung angezeigt und verwendet wird.
- Die mittlere Ebene enthält den Anwendungsserver, auf dem sich die Anwendungslogik und die Server-Software befinden.
- Die Datenbankebene besteht aus dem Datenbank-Server, auf dem die Unternehmensdaten gespeichert sind.

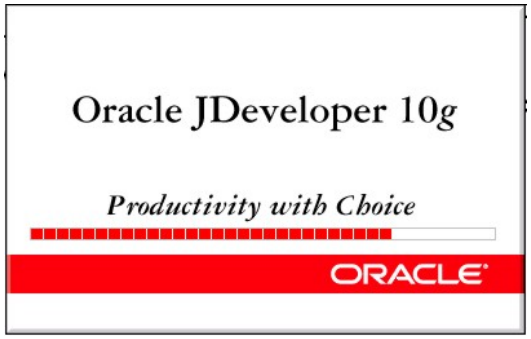
8.1.2. Oracle Report Builder



In Zeiten von Internet und Intranet versteht sich Unternehmensreporting nicht ausschließlich in der Ausgabe der Daten auf Papier. Verschiedenste Möglichkeiten der Ausgabe in unterschiedlichen Formaten, inklusive der Ausgabe in einem Webportal sind darunter zu verstehen. Dem Rechnung tragend wurde auch die Funktionalität des Oracle Report Builders erweitert, wie in Abbildung 72 verdeutlicht wird.

Abbildung 72: Unternehmensreporting

8.2. Oracle JDeveloper®



Der Oracle JDeveloper ist eine integrierte Entwicklungsumgebung **IDE** (**I**ntegrated **D**evelopment **E**nvironment) für die Java Programmierung. Er stellt Funktionen für das Design, die Entwicklung, das Debugging und das Deploying von Java- und anderen Dateien zur Verfügung, die Teile der Java 2 Platform, Enterprise Edition (J2EE) Strategie sind. Der JDeveloper ist ein Entwicklungs-Framework das eine Reihe von Assistenten (wizards) und Code Generatoren beinhaltet, um komplexe Funktionalitäten sehr einfach in Java abzubilden, um Business Probleme zu lösen.

Die Entwicklung des JDeveloper begann im Jahr 1997, als die Firma Oracle die Lizenzierung des JBuilder Java-Entwicklungstools der Firma Borland International erwarb, um dieses Werkzeug in die Datenbankentwicklung einzubinden.

Funktionen des Oracle JDeveloper 10g:

a) Der JDeveloper ist ein vollständiges J2EE Entwicklungswerkzeug, mit dessen Hilfe Anwendungen, vergleichbar mit denen des Oracle Forms Developer implementiert werden können. Damit werden Java und Web-Entwickler voll in allen Entwicklungsphasen durch die Möglichkeit des Generierens von Code, grafische WYSIWYG Designerwerkzeuge u.v.m. in ihrer Arbeit unterstützt. Die Funktionalität ist vergleichbar mit Tools für so genannten 4GL wie Delphi. Es können komplexe Client-/Server-Anwendungen in Java implementiert werden. Die Einbindung andere Sprachen wie PL/SQL ist möglich und vorgesehen.

b) Der JDeveloper bietet die Funktionalität einer Design- und Entwicklungsumgebung. Es ist möglich, UML-und Entity-Relationship-Diagramme, HTML- und XML-Dokumente zu generieren. Somit bietet der Oracle JDeveloper ähnlich Möglichkeiten wie der Oracle Designer.

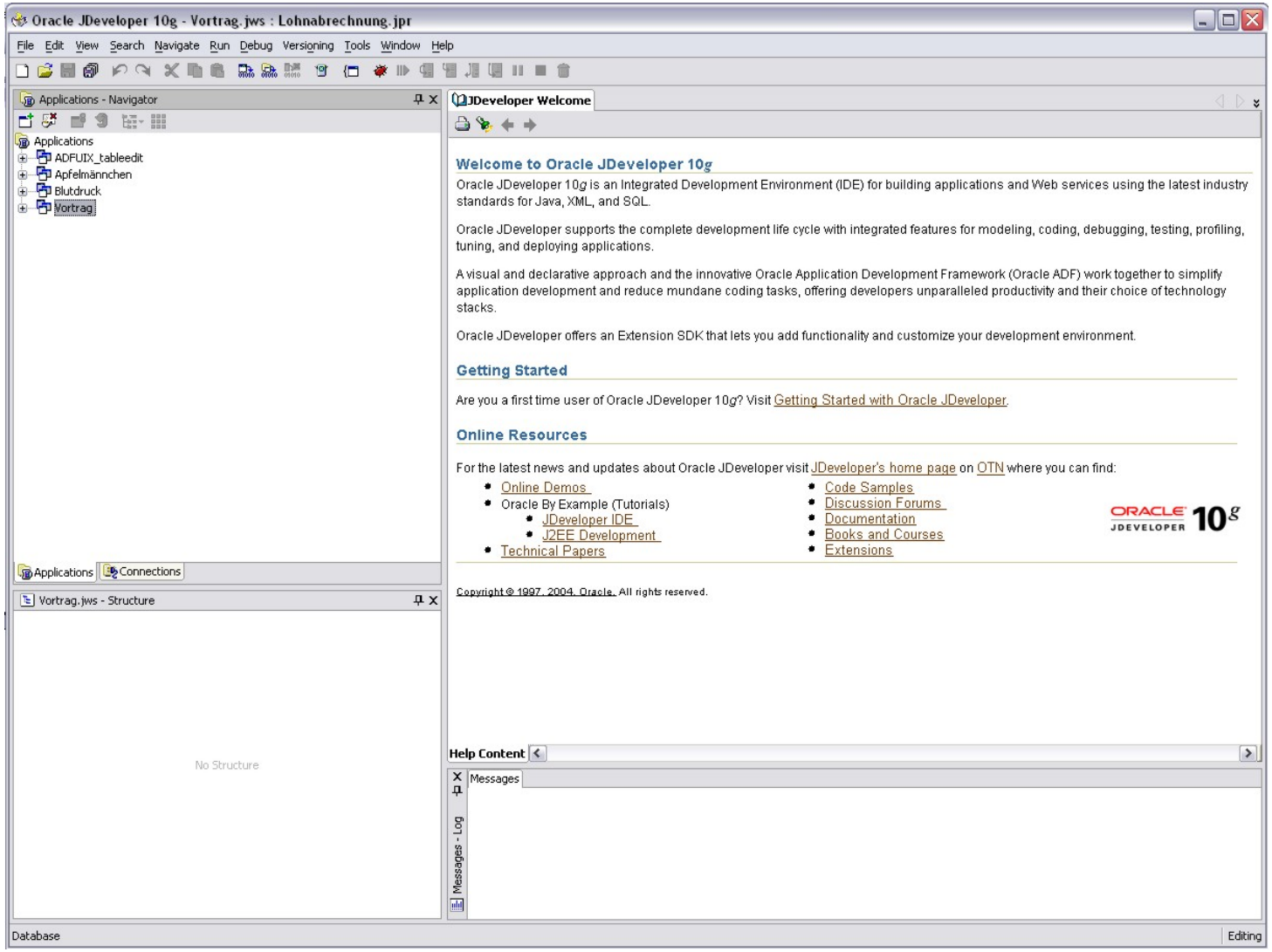


Abbildung 73: Startbild des JDeveloper 10g

8.2.1. Generieren von Datenbankverbindungen

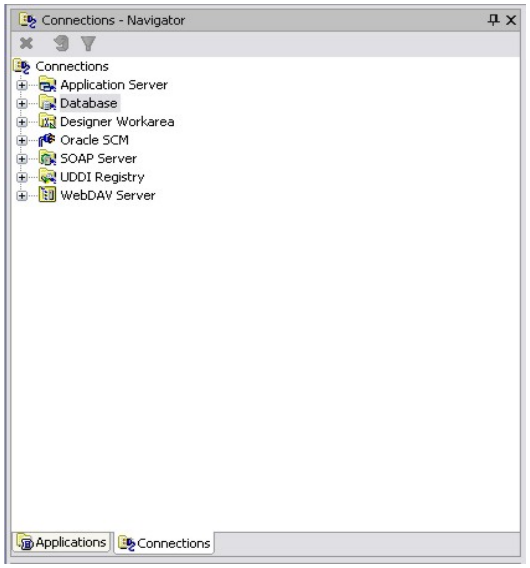


Abbildung 74.: Connections - Navigator

Die vornehmliche Aufgabe, die sich mit dem JDeveloper realisieren lässt, sind Anwendungen im Datenbankumfeld. Basieren auf der Programmiersprache Java werden diese Verbindungen auf Basis der in Abschnitt 7.2. JDBC - Java DataBase Connectivity vorgestellten Datenbankschnittstelle sehr einfach generiert. Dazu wird im Navigatorfenster der Reiter Connections gewählt, siehe Abbildung 75. Nach dem Betätigen der rechten Maustaste auf dem Knoten **Database** wird anschließend **New Database Connection…** ausgewählt.

In einem neuen Fenster wird der Assistent zum Generieren der Datenbankverbindung geöffnet, siehe Abbildung 75.

Im 1. Schritt kann die neue Datenbankverbindung benannt und der passende Treiber ausgewählt werden.

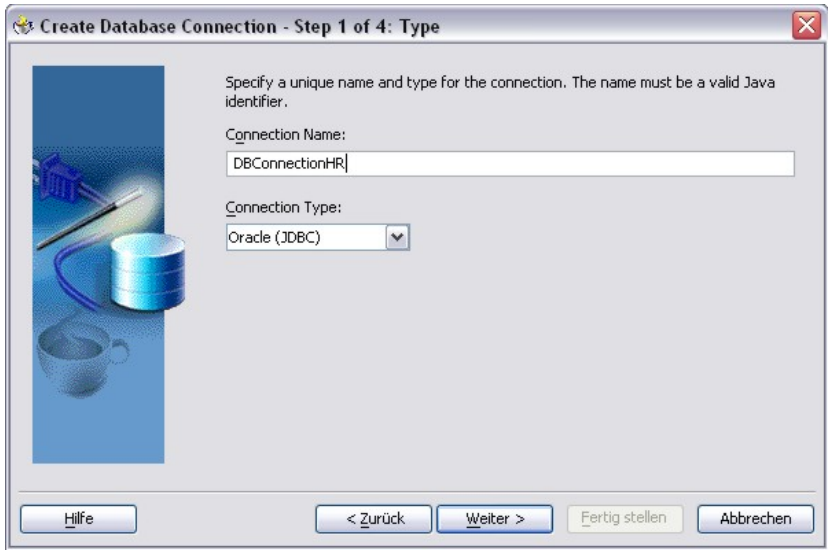


Abbildung 75.: Assistent zur Herstellung der Datenbankverbindung – Schritt 1

Im anschließenden Schritt 2 werden die Benutzerdaten für die neue Datenbankverbindung (Benutzername/Passwort) eingegeben, siehe Abbildung 76.

Abbildung 76.: Assistent zur Herstellung der Datenbankverbindung – Schritt 2

Für die Eingabe des Treiber- und Hostnames, sowie Portadresse und Name der Instanz wird der Dialog im Schritt 3 verwendet, siehe Abbildung 77.

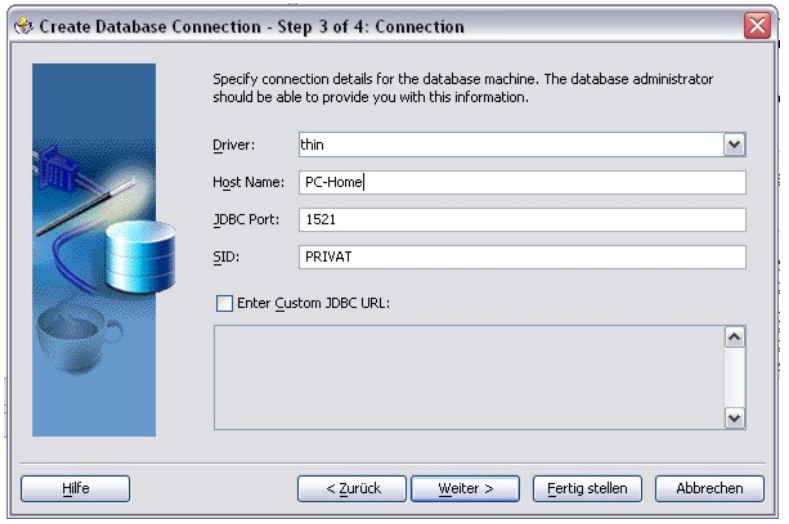


Abbildung 77.: Assistent zur Herstellung der Datenbankverbindung – Schritt 3

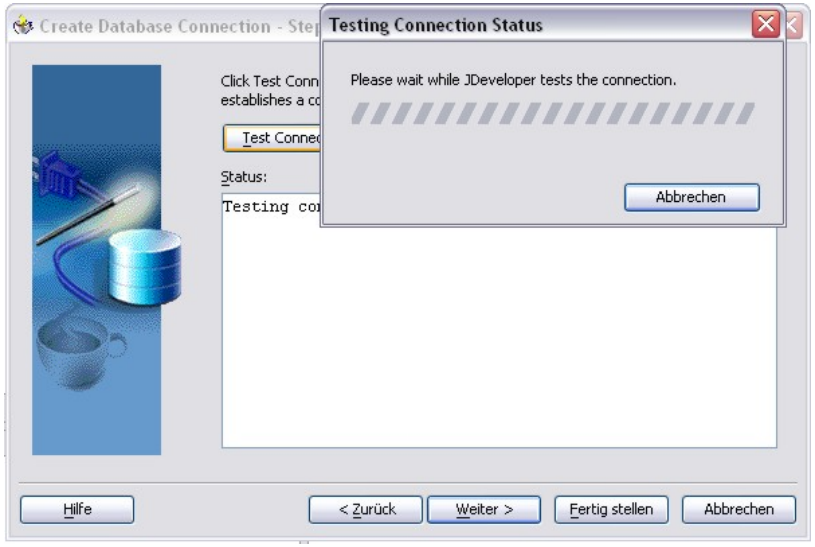
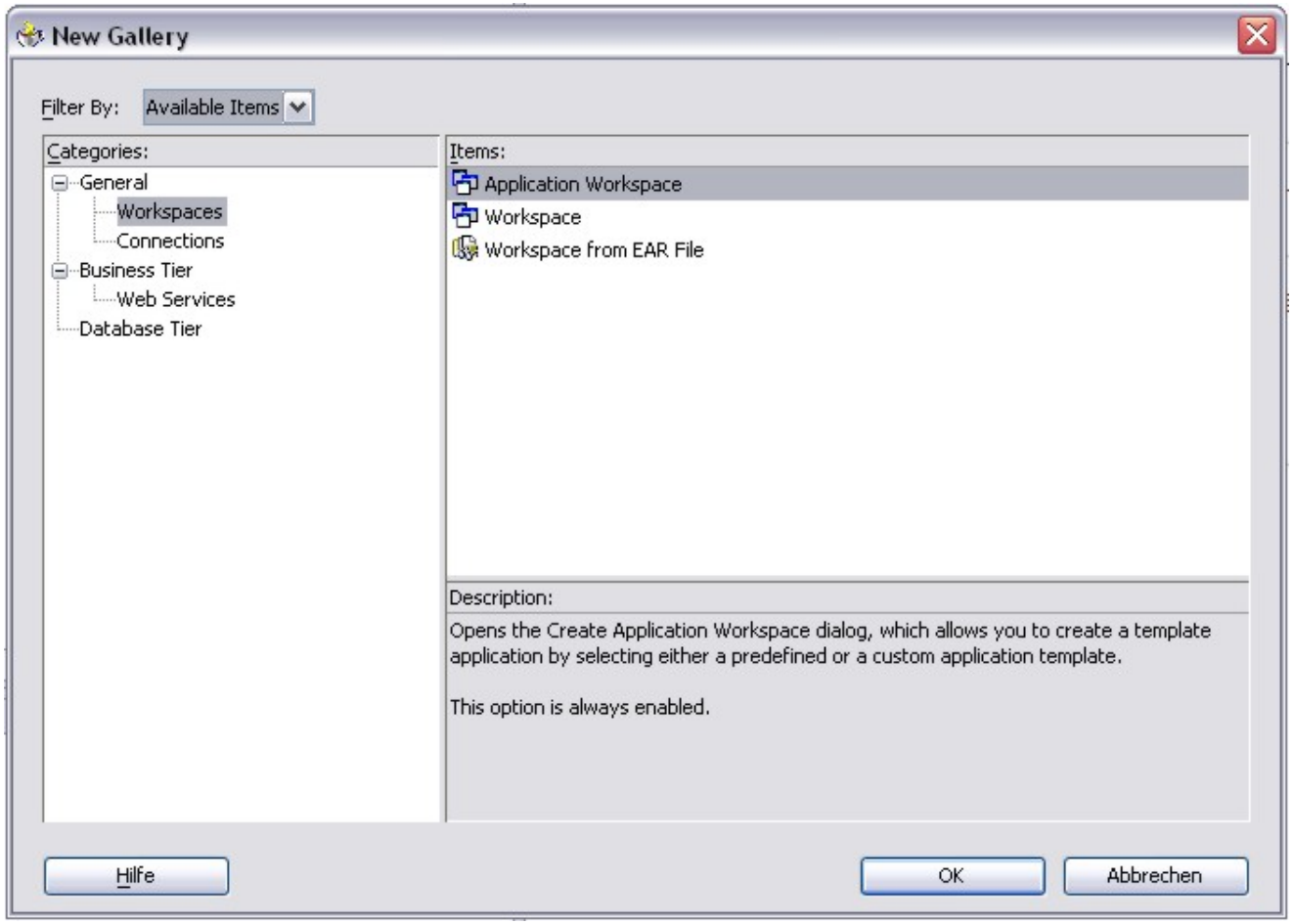


Abbildung 78.: Assistent zur Herstellung der Datenbankverbindung – Schritt 4

Der abschließende Funktionstest im Schritt 4 und das Klicken auf **Fertig stellen** beenden den Assistenten zur Herstellung einer neuen Datenbankverbidung.

8.2.2. Generierung einer Java-Anwendung



Auch diese Tätigkeit wird weitestgehend durch einen Assistenten übernommen und dadurch wird der Applikationsprogrammierer in seiner Tätigkeit unterstützt.
Zunächst muss im **Application – Navigatorfenster** eine neue Applikation erstellt werden. Die geschieht durch Betätigung der rechten Maustaste auf den obersten Knoten **Applications**, siehe Abbildung 73.

Es öffnet sich ein Dialigbild, wie es in Abbildung 79 dargestellt ist. Es wird aus **Categories General** → **Workspaces** der **Application Workspace** ausgewählt.

Abbildung 79.: Erzeugen einer neuen Applikation

Im nächsten Dialogschritt gibt man der neuen Applikation einen Namen und legt das Verzeichnis zur Ablage der Dateien für die neue Applikation und den Typ der Vorlagen, die verwendet werden sollen, fest. Die nötigen Angaben sind in Abbildung 80 dargestellt.

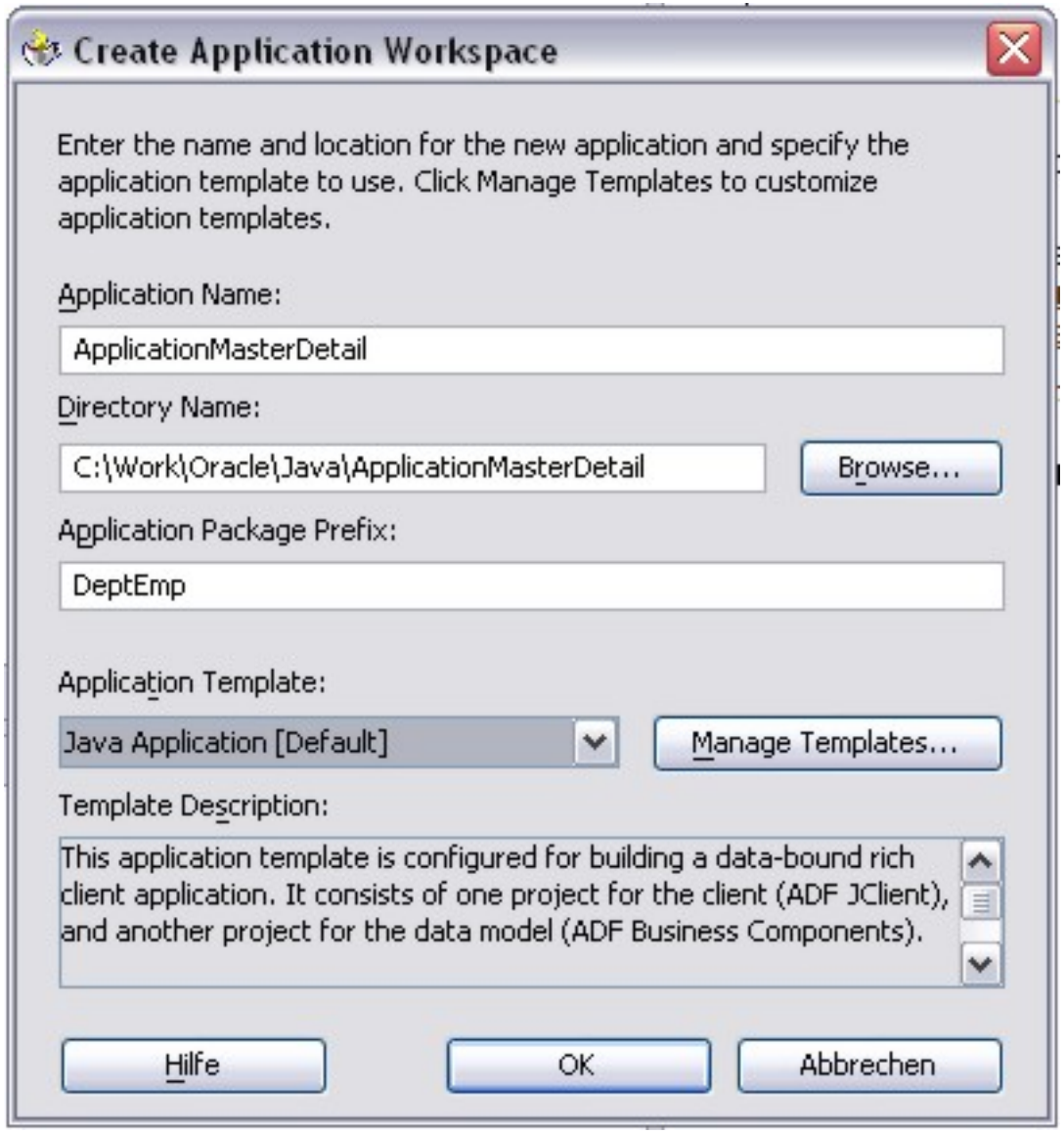


Abbildung 80.: Festlegen der Eigenschaften des neuen Application Workspaces

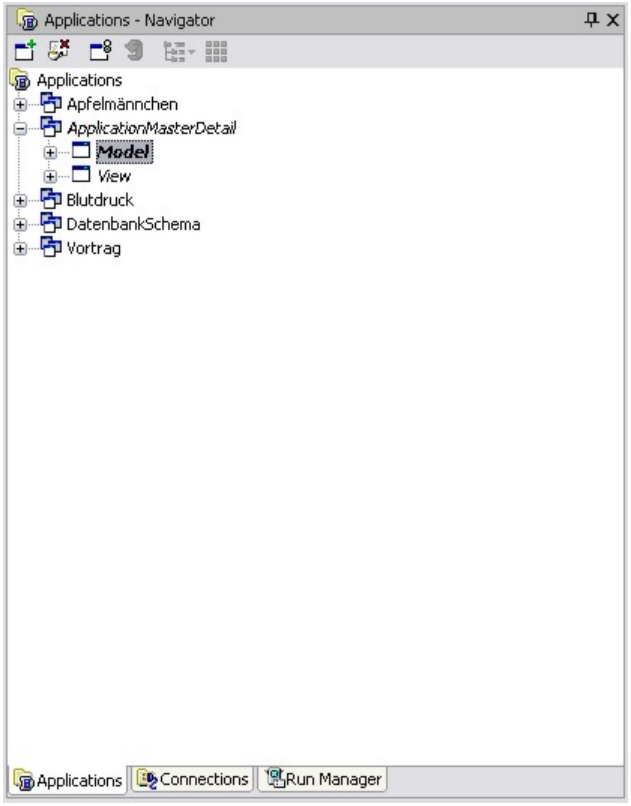


Abbildung 81.: Neuer **Application Workspaces - AppicationMasterDetail**

Im **Applications – Navigator** sind nun die neu erstellten Angaben sichtbar. Es wurde ein **Application Workspace *ApplicationMasterDetail*** erzeugt. Entsprechend der gewählten Vorlage wurde ein **Model** für die Anbindung von Datenbanktabellen und **View** für die Darstellung der neuen Applikation generiert, siehe Abbildung 81.

8.2.2.1. Generieren der Business Logik

Wiederum wird durch das Betätigen der rechten Maustaste auf dem Navigationsknoten **Model** die Möglichkeit eröffnet, die Business Logik für die neue Applikation zu generieren.

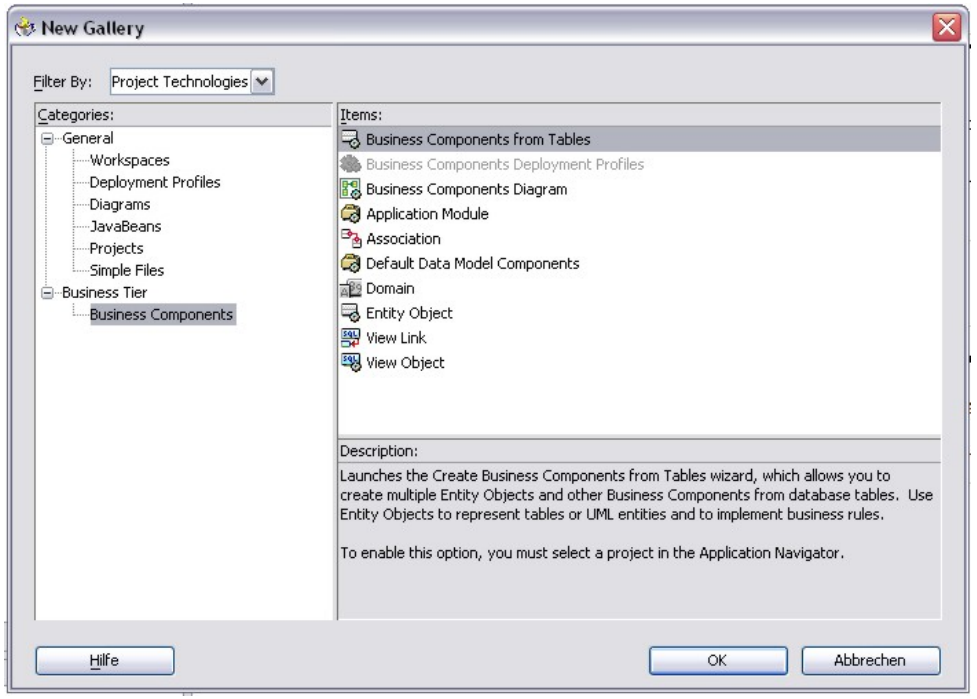


Abbildung 82.: Festlegung der Business Logik

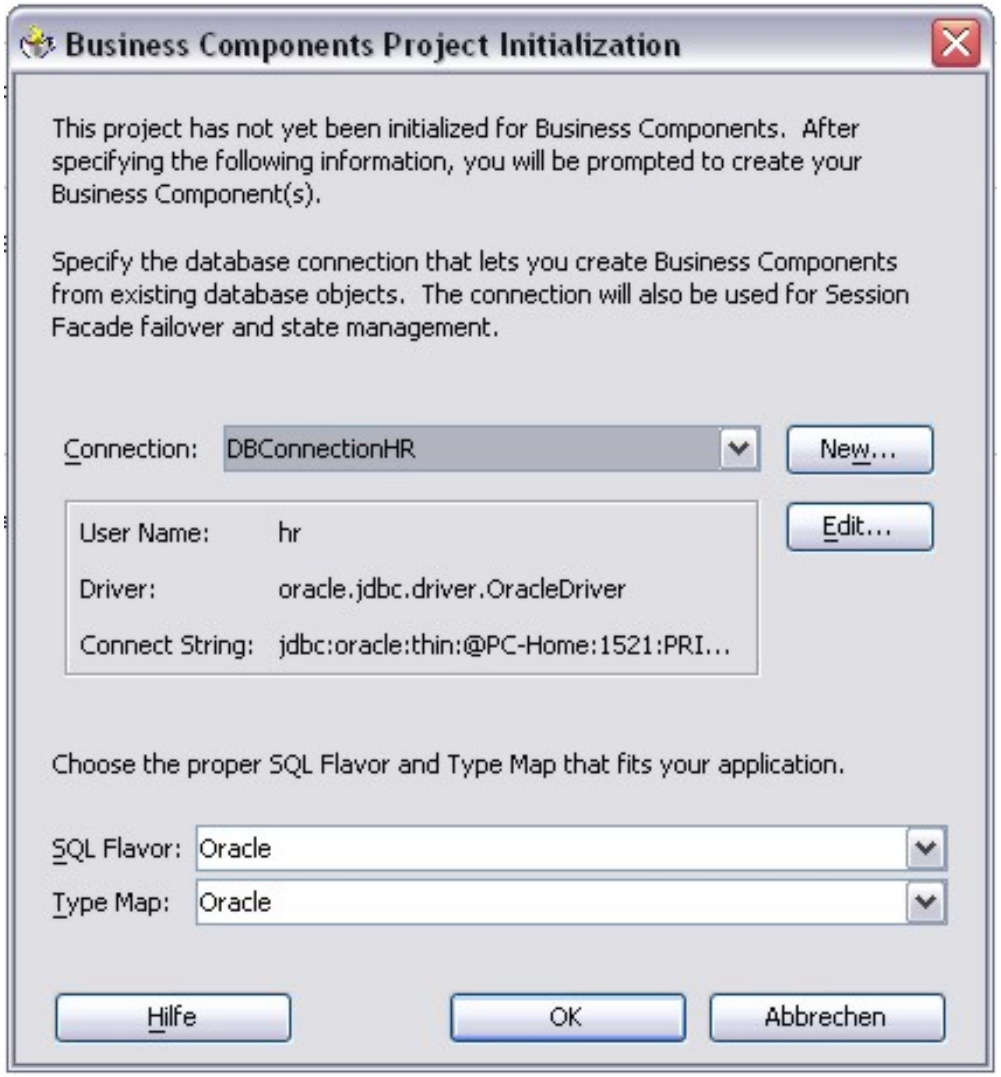


Abbildung 83.: Auswahl der Datenbankverbindung

Anschließend wird die Verbindung zur Datenbank ausgewählt, die für die neue Applikation verwendet werden soll. Dazu wurde im Abschnitt 8.2.1. Generieren von Datenbankverbindungen die Datenbankverbindung **DBConnectionHR** erzeugt, die für diese Applikation verwendet werden soll.

Wie in Abbildung 83 dargestellt, wird die vorhandene Verbindung zur Datenbank ausgewählt. Es kann aber an dieser Stelle auch eine neue Verbindung angelegt werden. Der Dialog, wie er im Abschnitt 8.2.1. Generieren von Datenbankverbindungen beschrieben wurde wird dann aufgerufen.

In einem weiteren Dialogschritt wird das Passwort für die Verbindung zur Datenbank eingegeben.

Im Anschluß erfolgt der Zugriff auf die Datenbank und es werden die Tabellen dieser Datenbankverbindung zur Auswahl dargestellt, siehe Abbildung 84.

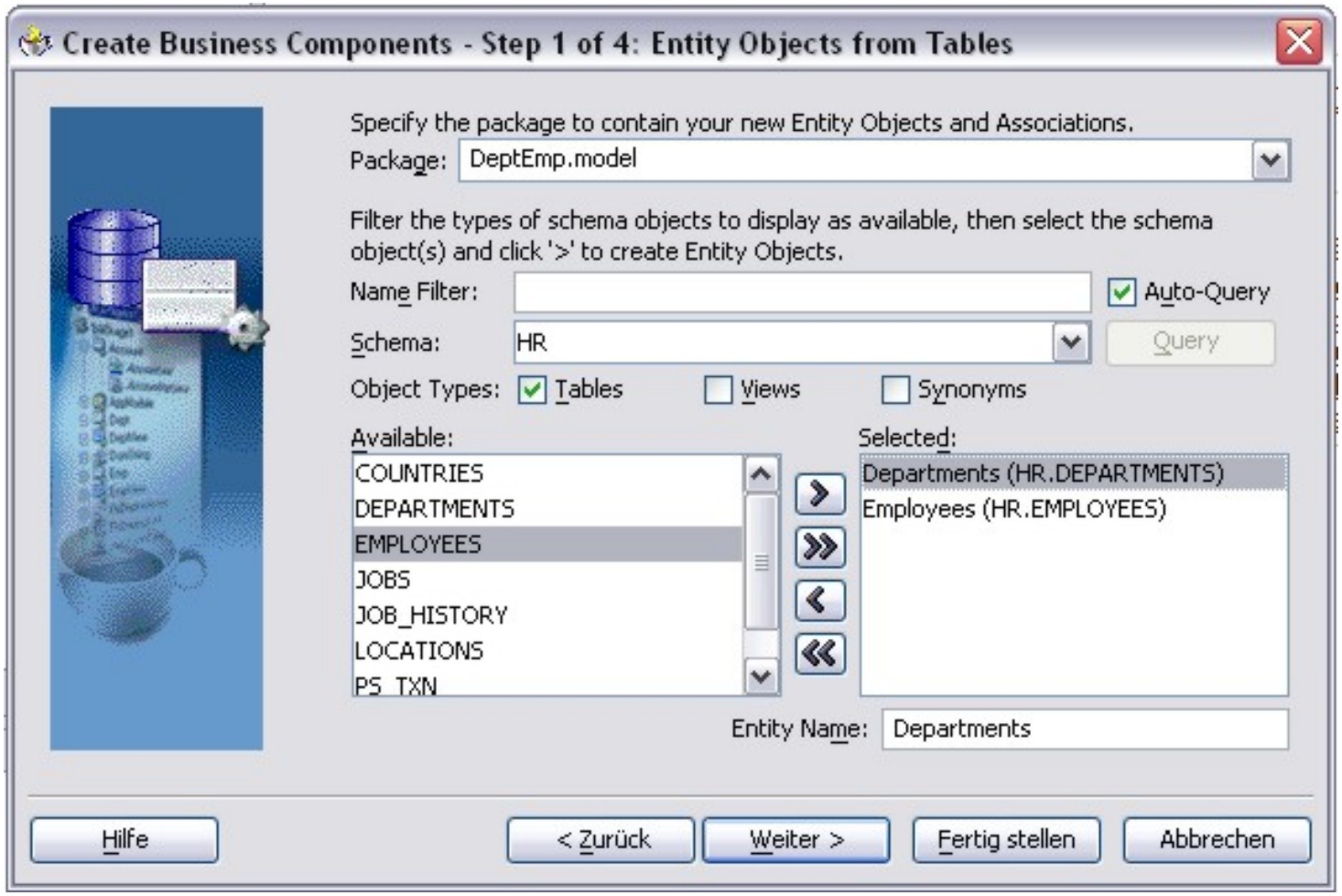


Abbildung 84.: Auswahl der Tabellen für die neue Applikation

Für die neu zu generierende Applikationen werden die Tabellen **DEPARTMENTS**, als Mastertabelle und die Tabelle **EMPLOYEES** als Detailtabelle ausgewählt.

Über einen weiteren Dialogschritt wird der Name des Application Modul festgelegt, so wie es in Abbildung 85 dargestellt ist.

Abbildung 85.: Benennung des Applications Modul

Nachdem im letzten Dialogschritt alle Komponenten, die neu zu generieren sind, angezeigt werden, siehe Abbildung 86, werden diese nach der Betätigung des **Fertig stellen**- Buttons durch das System generiertn.

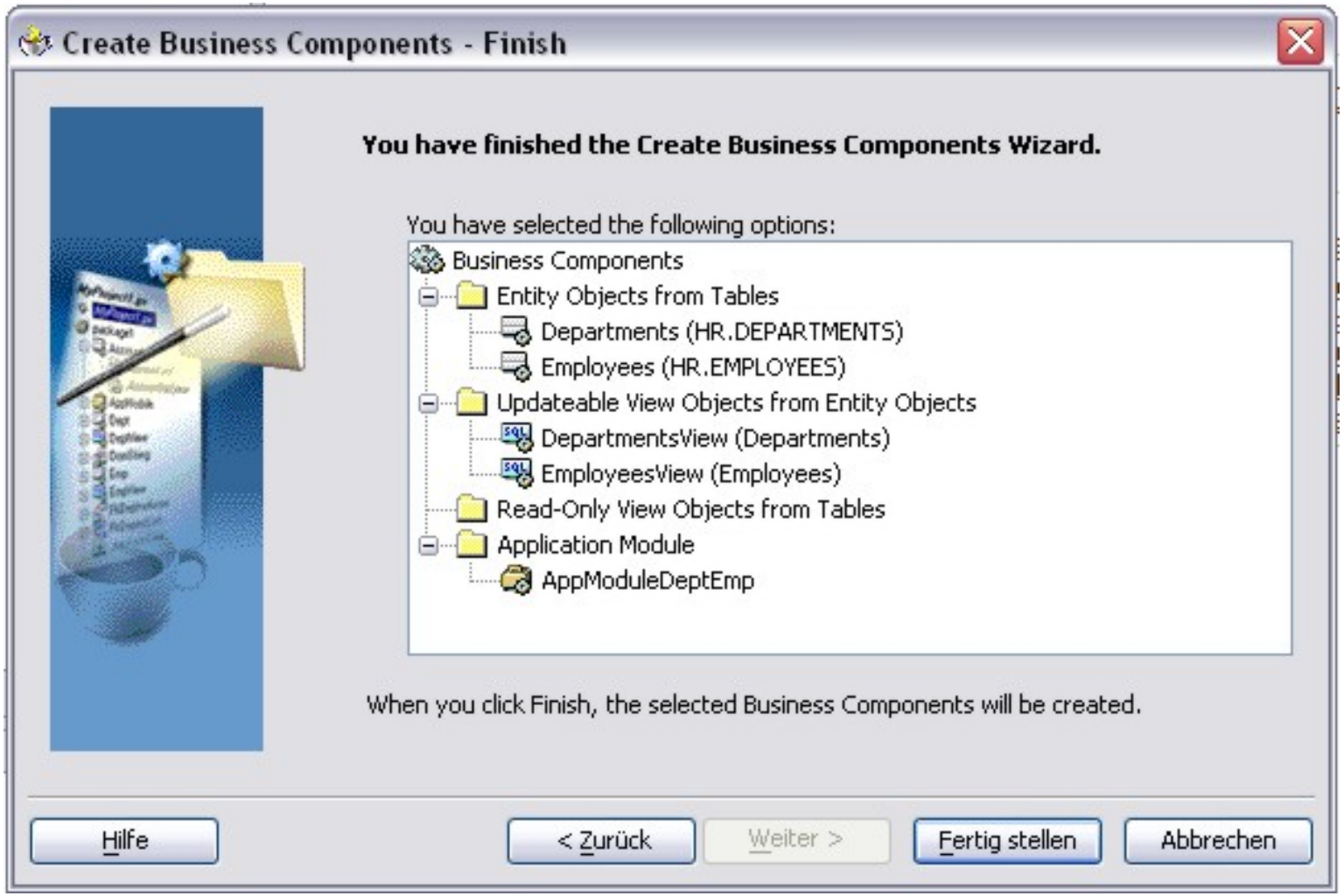


Abbildung 86.: Anzeige aller zu generierenden Komponenten

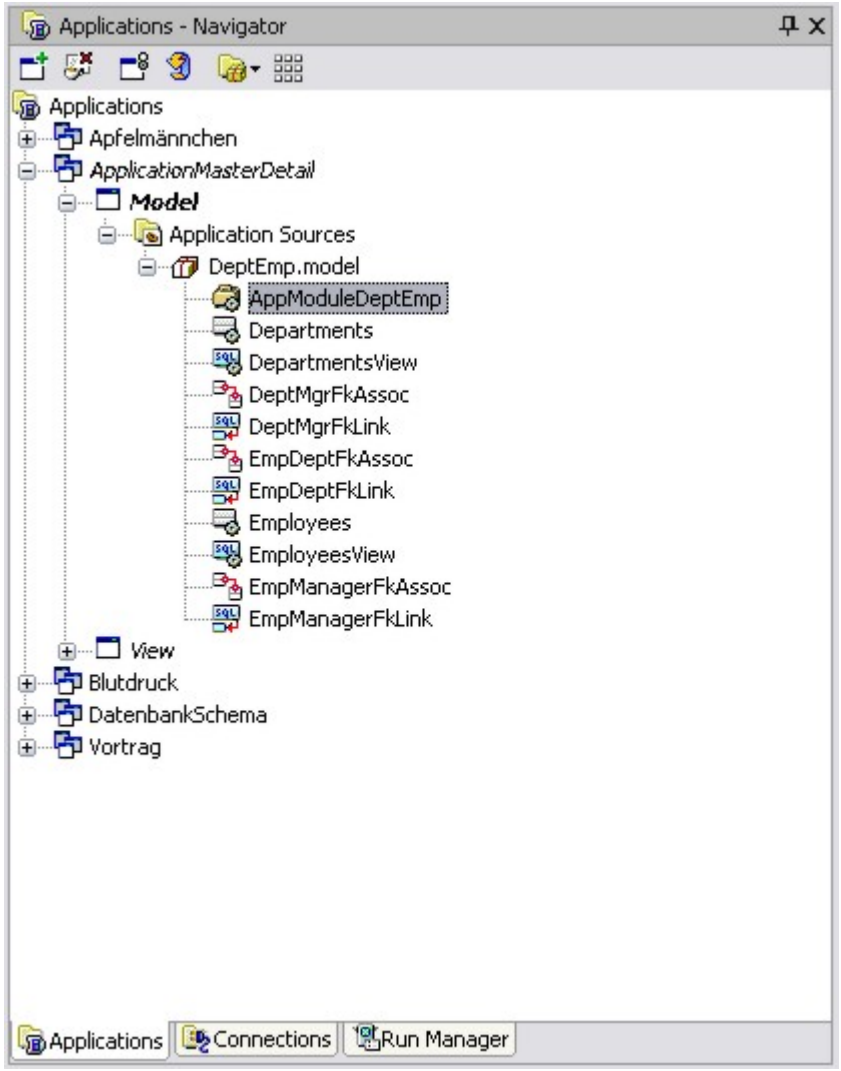


Abbildung 87.: Alle neu generierten Komponenten

Nach Abschluss des Generierungsvorganges im Anschluss an die Beendigung des aufgerufenen Assistenten sind alle neu generierten Komponenten im **Applications – Navigator** sichtbar, siehe Abbildung 87.

Die Art und Weise der Verknüpfungen innerhalb der Business-Logik wird aus der Datenbankverbindung, anhand der dort vorhandenen Constraints und Data Dictionary-Einträge ermittelt.

Der Applikationsprogrammierer ist somit von der Eingabe dieser Verknüpfungen befreit. Allerdings wird damit klar, welchen hohen Stellenwert damit der Datenbankentwurf und dessen Exaktheit gewinnen.

Die soeben erzeugten Komponenten können im Anschluss sofort getestet werden, siehe Abbildung 88. Dazu wird eine Standardoberfläche zur Verfügung gestellt.

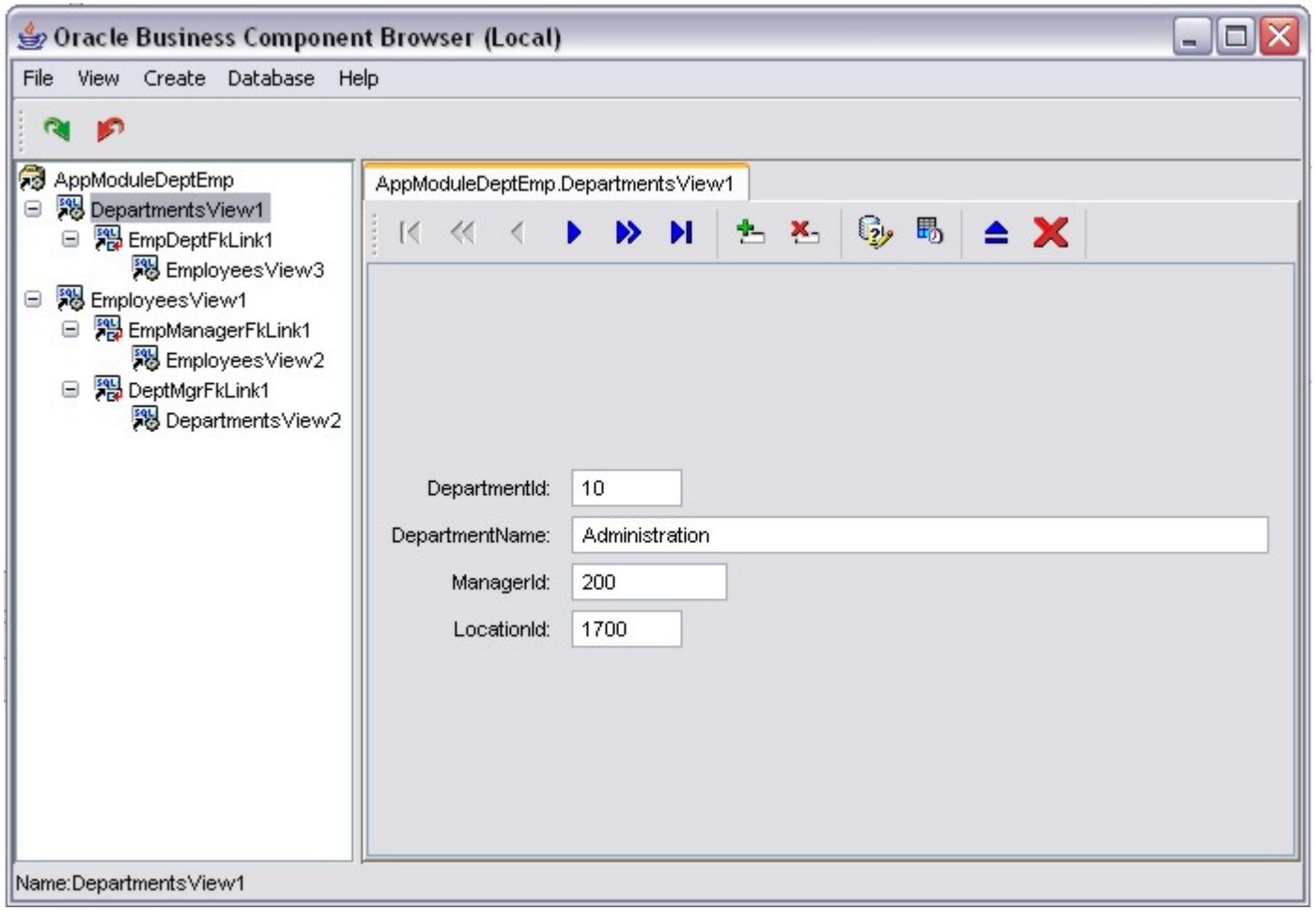


Abbildung 88.: Test der generierten Business Logik

8.2.2.2. Generieren der Applikationsoberfläche

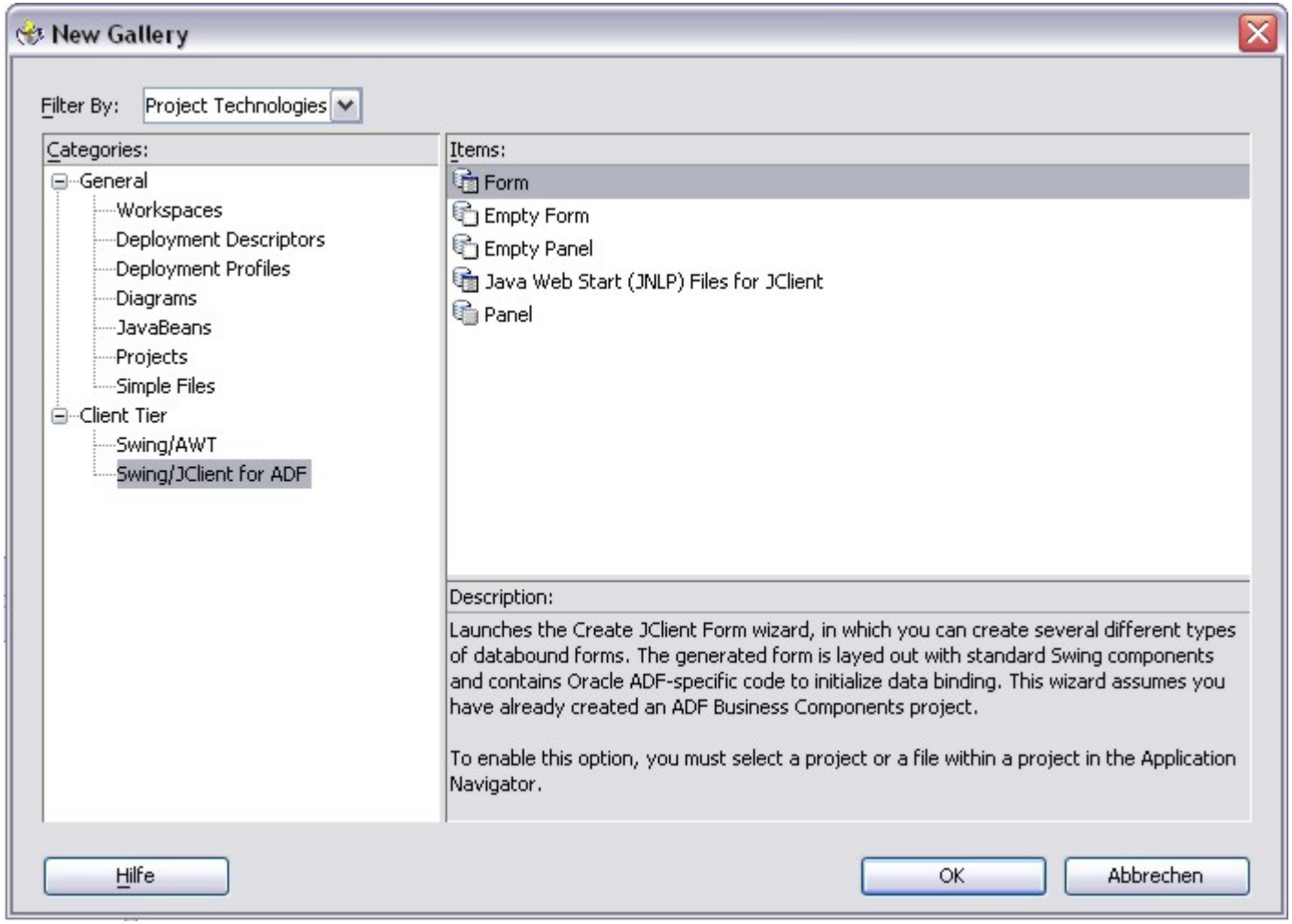


Abbildung 89.: Auswahl der Applikationsoberfläche

Nach dem Erzeugen der Business Logik wird ebenfalls durch einen Assistenten gesteuert, die Applikationsoberfläche generiert. Nach der Auswahl der Art der Anwendung, hier als **Swing/JClient for ADF** mit **Item Form**, siehe Abbildung 89, wird in den weiteren Dialogschritten der Aufbau und die Darstellung für die Applikation festgelegt.

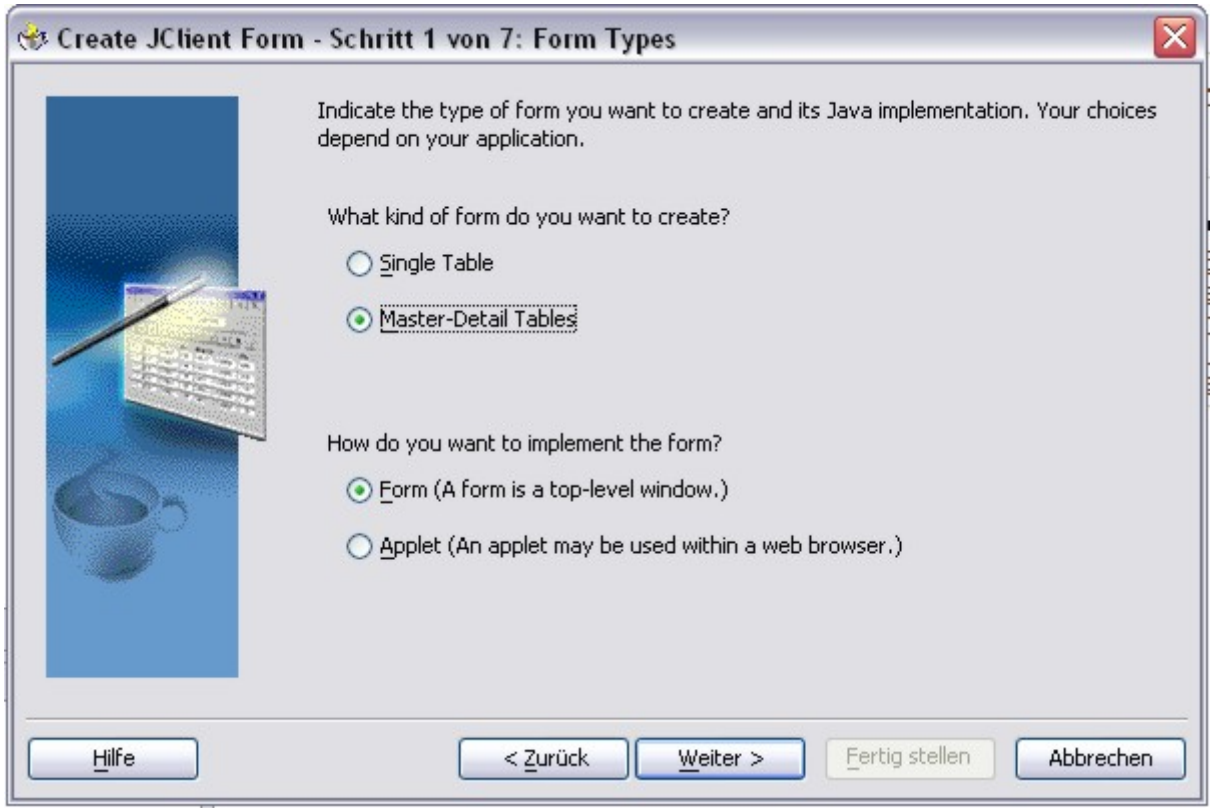


Abbildung 90.: Auswahl des Formtyps

Für die neue Applikation sollte es eine Master-Detail Anwendung für die beiden Tabellen **DEPARTMENTS** und **EMPLOYEES** sein, die bereits für die Business Logik ausgewählt wurden.

Natürlich sind gewisse Angaben in den beiden Assistenten übereinstimmend anzugeben, siehe Abbildung 90.

Die Gestaltung und Darstellung der Mastertabelle **DEPARTMENTS** wird im nächsten Dialogschritt ausgewählt, siehe Abbildung 91.

Die weiteren Dialogschritte zur Fertigstellung der Applikation sind in Abbildung 92 bis Abbildung 95 zu sehen.

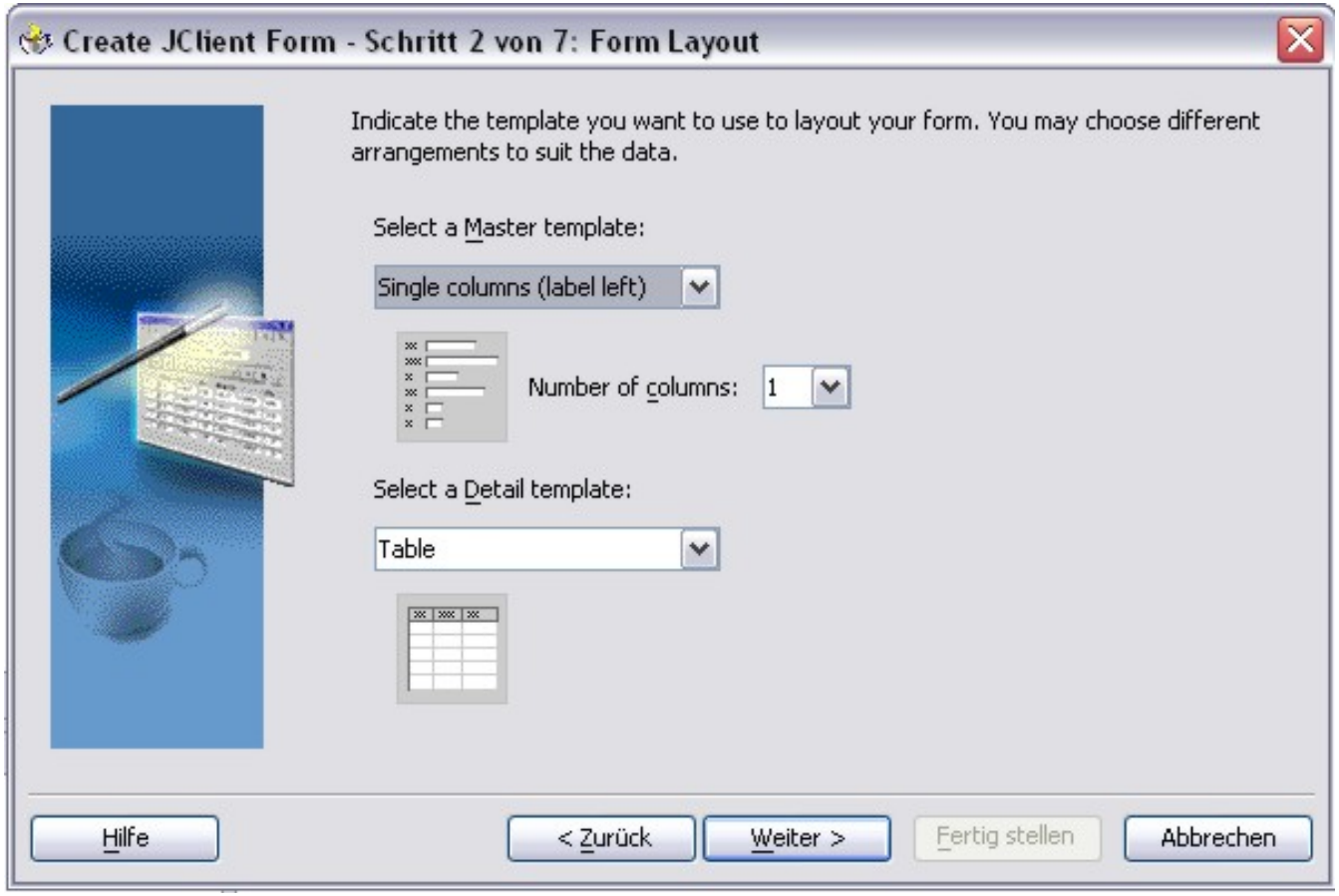


Abbildung 91.: Gestaltung der Darstellungsart beiden Tabellen

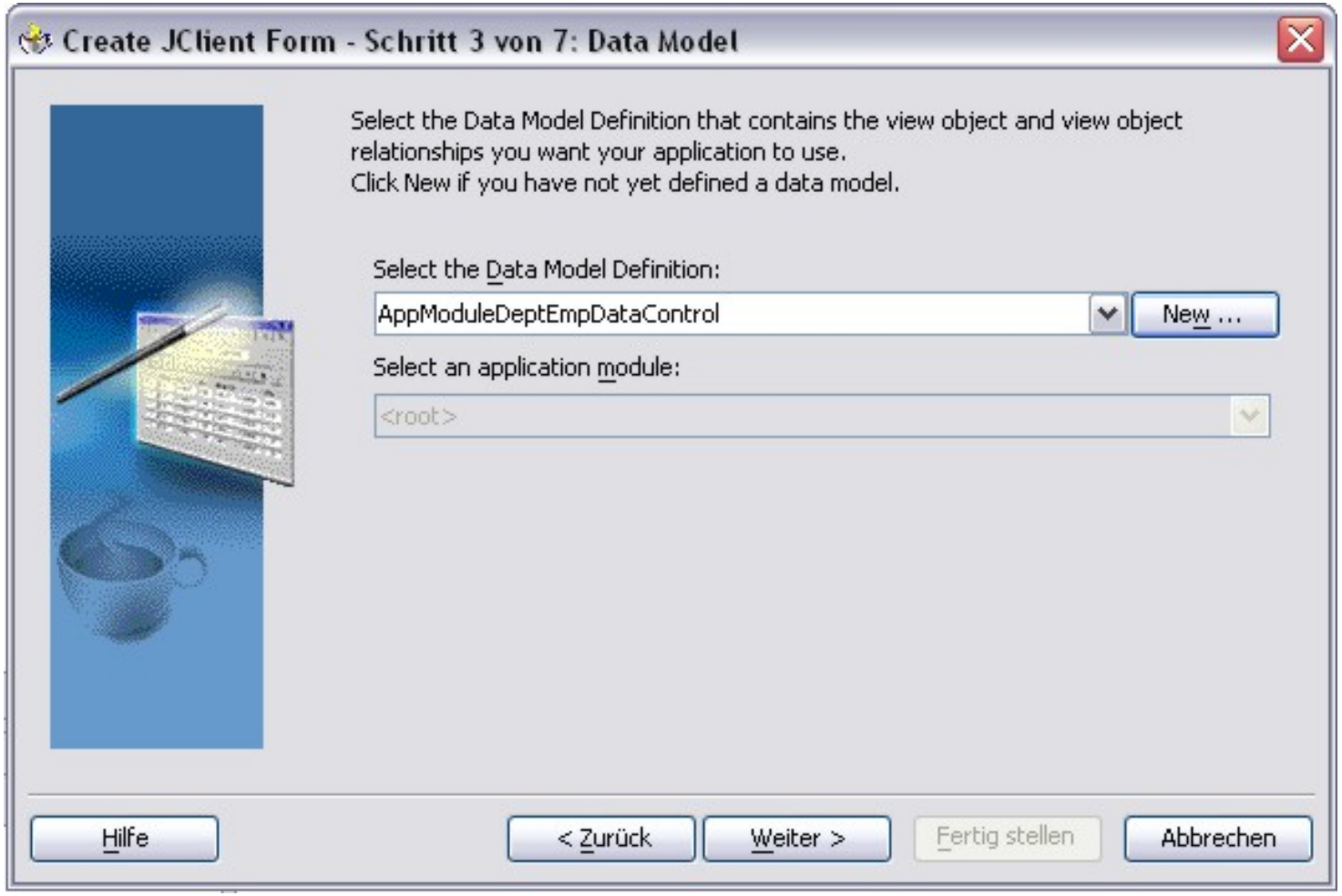


Abbildung 92.: Benennung des **Data Model**

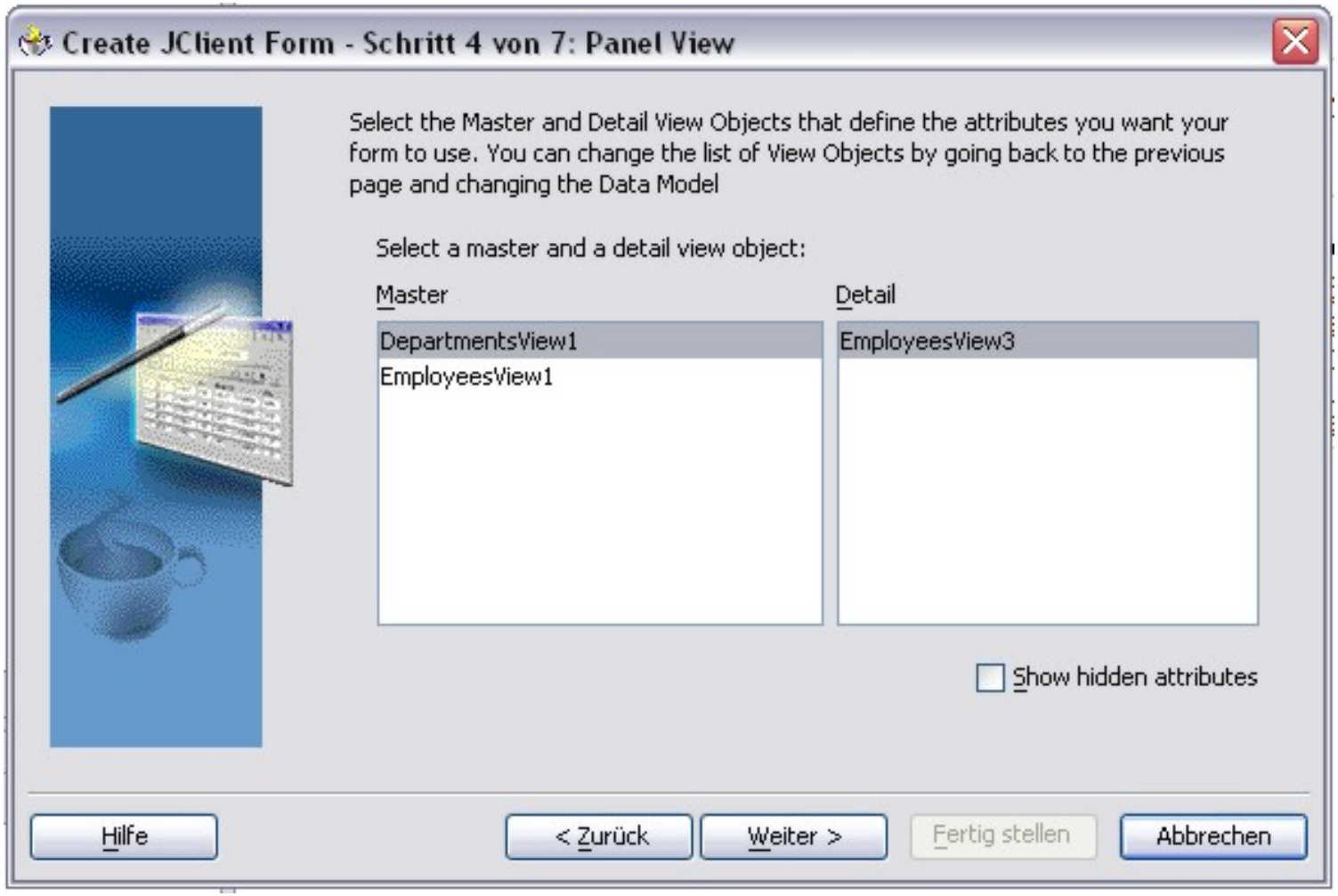


Abbildung 93.: Zuordnung der Komponenten der Business-Logik

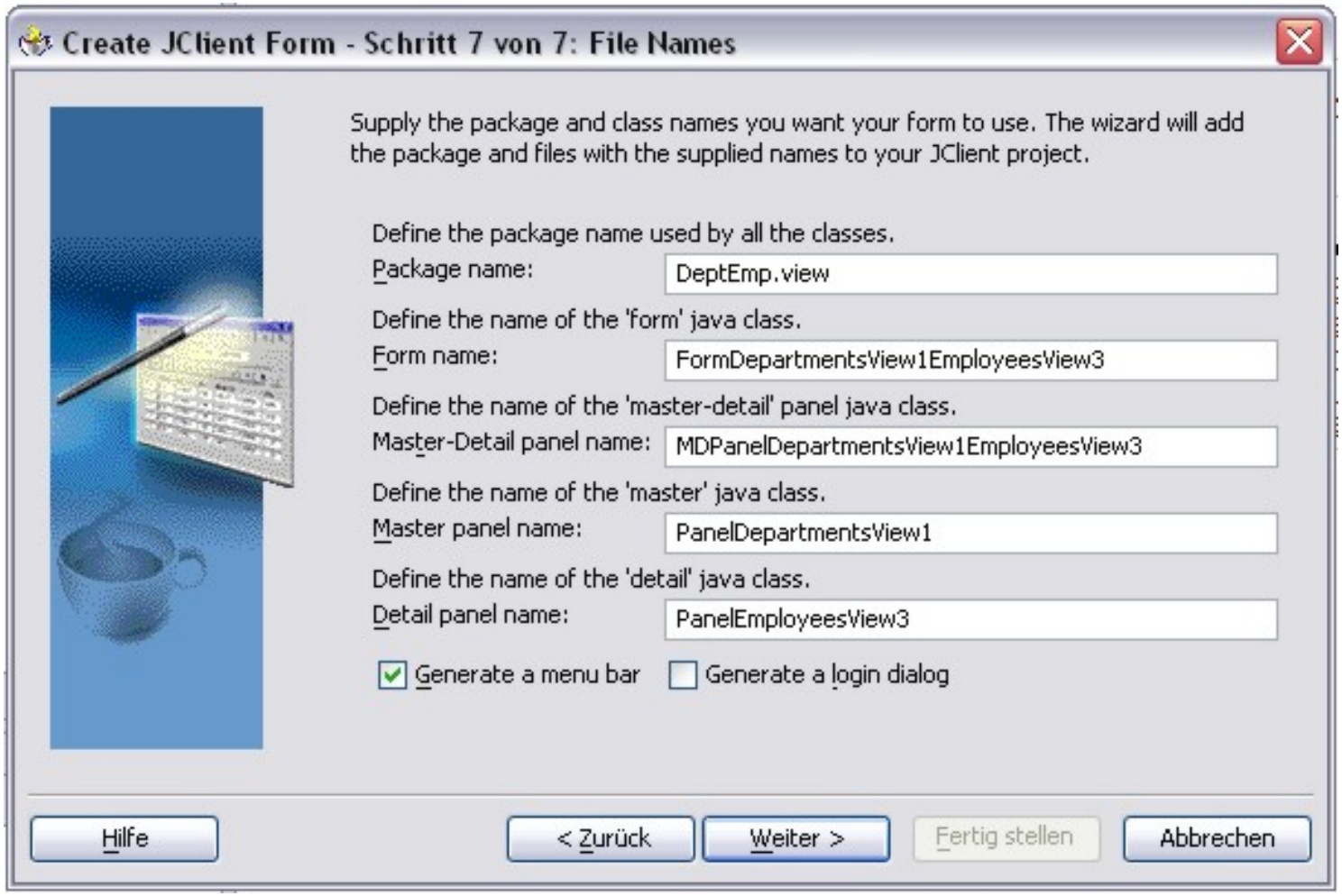


Abbildung 94.: Anzeige der zu generierenden Komponenten

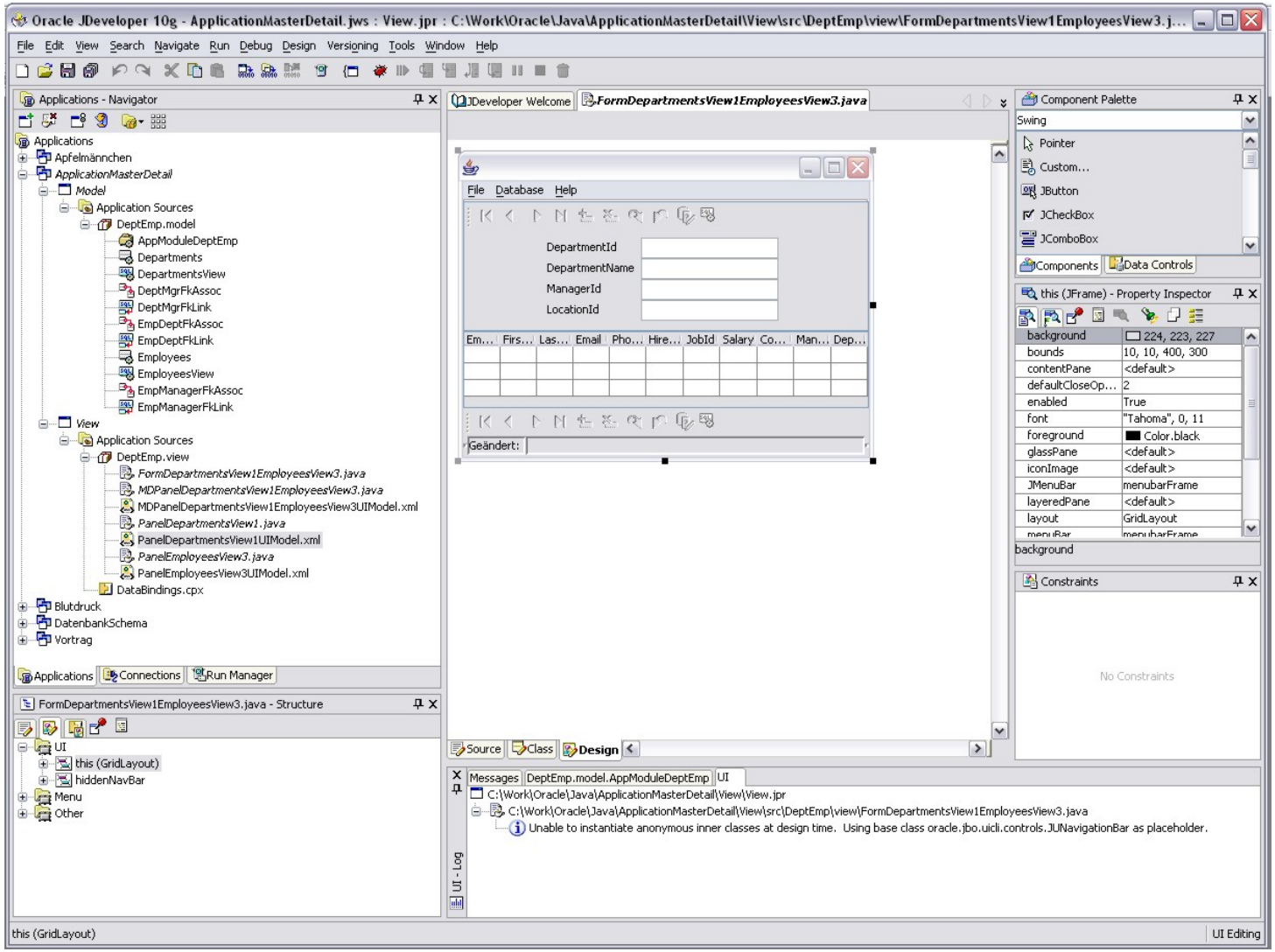


Abbildung 95.: Zusammenfassung aller erzeugten Komponenten inklusive der GUI

Abschließend kann natürlich die neue Applikation sofort getestet werden, wie in Abbildung 96 dargestellt wurde.

Mit diesem Abschnitt ist nur ein kleiner Überblick über die Möglichkeiten des JDeveloper 10g (in der Version 10.1.2.) vorgestellt worden. Eine Vielzahl weiterer Generierungsmöglichkeiten ist mit diesem Tool möglich.

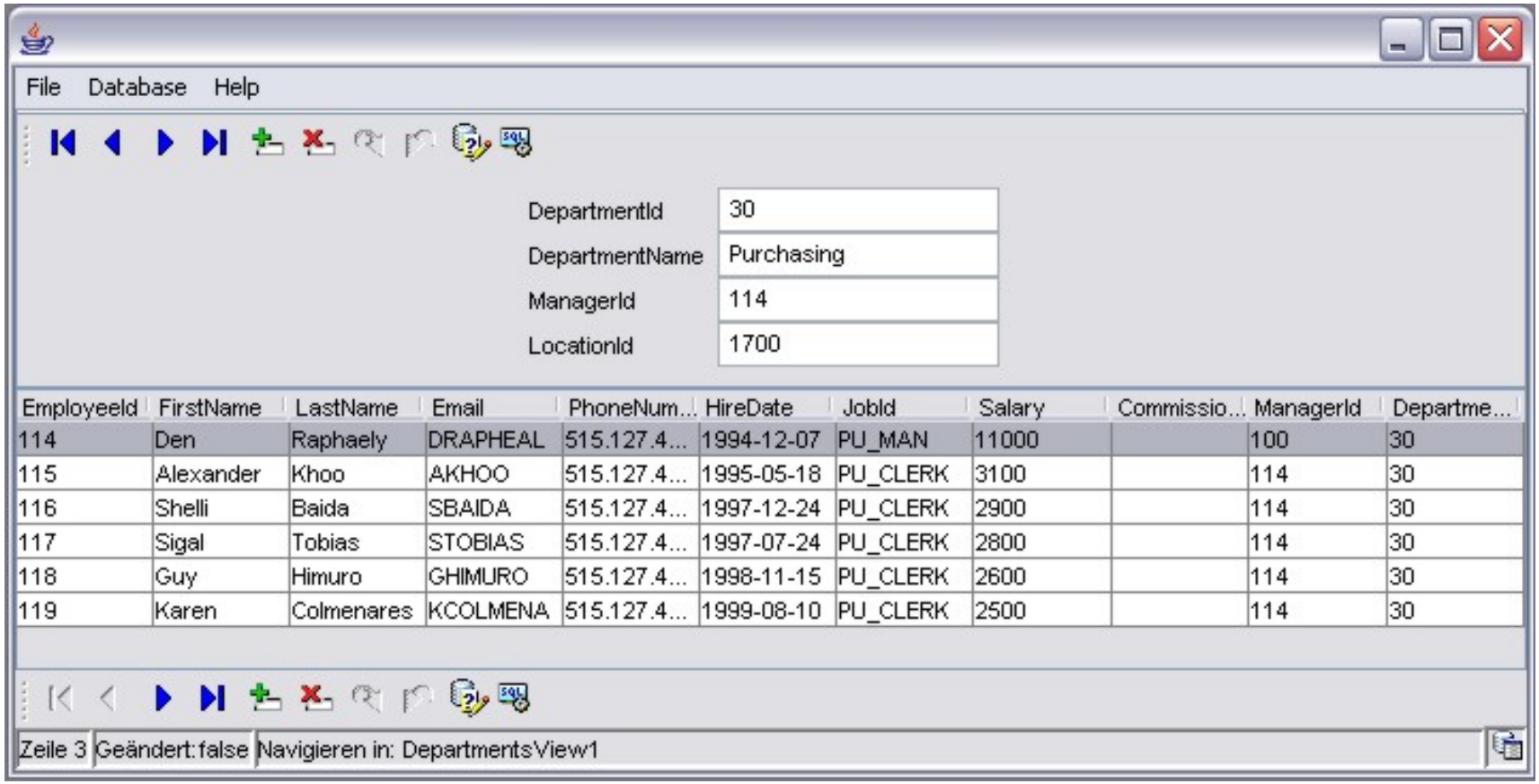


Abbildung 96.: Test der fertigen Applikation

8.3. Oracle Application Express®

In fast allen Unternehmen und Behörden sind Informationsquellen wie Arbeitsblätter, Desktop-Datenbanken oder selbst geschriebene Anwendungen im Einsatz. Dies hat eine Vielzahl von Informationsinseln auf den unterschiedlichsten Plattformen zur Folge. Diese Lösungen sind häufig historisch gewachsen, nutzen eigene Sicherheitsmechanismen, müssen separat gesichert und mit hohem Aufwand aktuell gehalten werden. Oft sind diese Lösungen an eine bestimmte Plattform gebunden, schlecht in Unternehmenslösungen integriert und wenig bis gar nicht skalierbar.

An dieser Stelle setzt die Oracle Application Express an. Mit der Application Express können datenbankgestützte Anwendungen schnell und einfach entwickelt und im Intranet oder Internet verfügbar gemacht werden. Für Entwicklung und Nutzung der Anwendungen reicht ein normaler Web-Browser aus; Software-Verteilung ist nicht mehr nötig.

Die Anwendung und deren Daten liegen in einer zentralen Oracle-Datenbank. So kombiniert die Application Express die einfache Bedienung und hohe Produktivität in der Entwicklung mit der Skalierbarkeit, Verfügbarkeit und Sicherheit einer zentralen Oracle-Datenbank.

8.3.1. Architektur

Anwendungen werden mit der Oracle Application Express deklarativ erstellt. Mit Hilfe von Assistenten werden die Inhalte und Merkmale der Anwendungen definiert. Diese Definitionen werden im Application Express-Repository gespeichert. Es wird dabei kein Code für die Anwendung generiert. Vielmehr werden die Seiten der Anwendung anhand der Informationen im Repository zur Laufzeit generiert.

Die Architektur der Application Express ist schlank gehalten. Sie besteht aus einer Oracle-Datenbank und dem Oracle HTTP Server (Apache).

Der Oracle HTTP Server (Apache) ist mit einem zusätzlichen Modul (mod_plsql) ausgestattet, welches die Application Express-Umgebung für den Browser zugänglich macht. Die Application Express-Entwicklungsumgebung läuft wie die damit erstellten Anwendungen vollständig innerhalb der Datenbank ab. So stehen alle Möglichkeiten der Datenbank auch für Application Express-Anwendungen zur Verfügung.

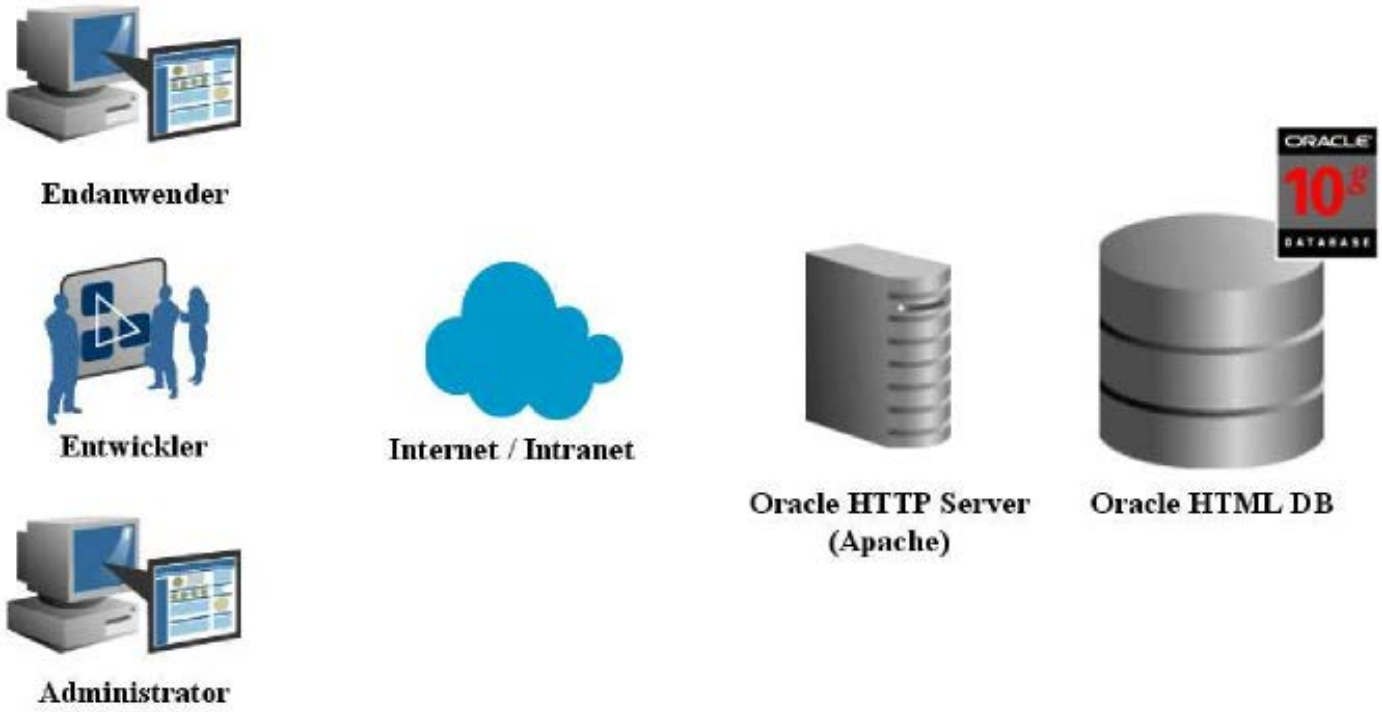
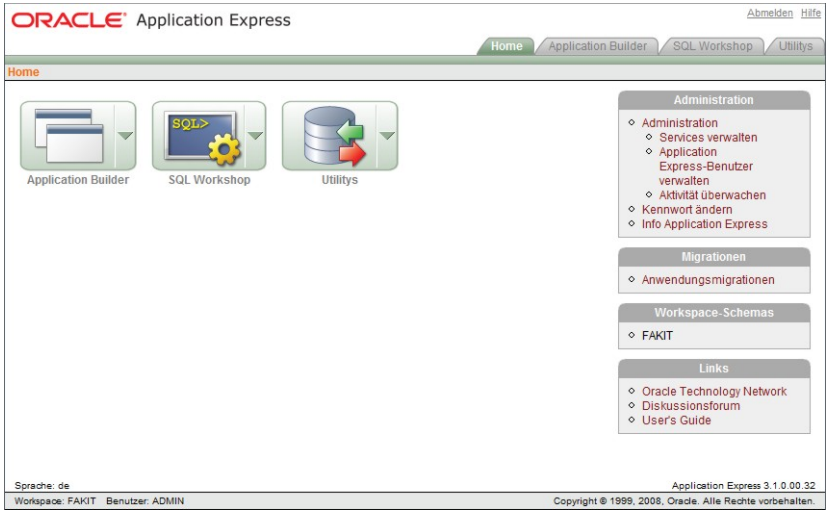


Abbildung 97.: Architektur der Oracle Application Express

8.3.2. Application Express Konzepte

8.3.2.1. Workspace



Workspaces garantieren ein wichtiges Designziel der Application Express, die Flexibilität bei der Entwicklung von Anwendungen. Ein Workspace ist ein autonomer Bereich innerhalb der Application Express mit eigener Benutzerverwaltung. Workspaces werden vom Application Express-Administrator auf Antrag bereitgestellt. Danach kann der Eigentümer seinen Workspace selbstständig administrieren, also neue Benutzerkonten einrichten und Privilegien innerhalb dieses Workspace vergeben. Der Application Express-Administrator muss nur dann aktiv werden, wenn Workspaces erweitert oder gelöscht werden sollen.

Abbildung 98.: Application Express-Konzept

Ein Workspace wird mit einem Datenbankschema, welches normalerweise bei Erstellung des Workspace angelegt wird, verknüpft. Entwickler können ihre Tabellen oder andere Datenbankobjekte in diesem Schema anlegen. Die Anwendungen im Workspace greifen dann auf diese Objekte zu. Das Datenbankschema muss jedoch nicht zwingend neu angelegt werden, die Verknüpfung mit einem bestehenden Datenbankschema ist ebenso möglich wie die Zuordnung mehrerer Datenbankschemas zu einem Workspace. So können bei Bedarf mehrere Workspaces auf den gleichen Daten arbeiten.

8.3.2.2. Anwendungen

Ein Application Express-Workspace kann mehrere Anwendungen beinhalten. Eine Anwendung wird vom Eigentümer oder von einem anderen Nutzer mit Entwicklerprivileg erstellt. Eine Anwendung besteht aus mehreren Seiten; diese enthalten die einzelnen Komponenten. Typische Anwendungskomponenten sind Berichte, Baumstrukturen, Formulare oder Grafiken.

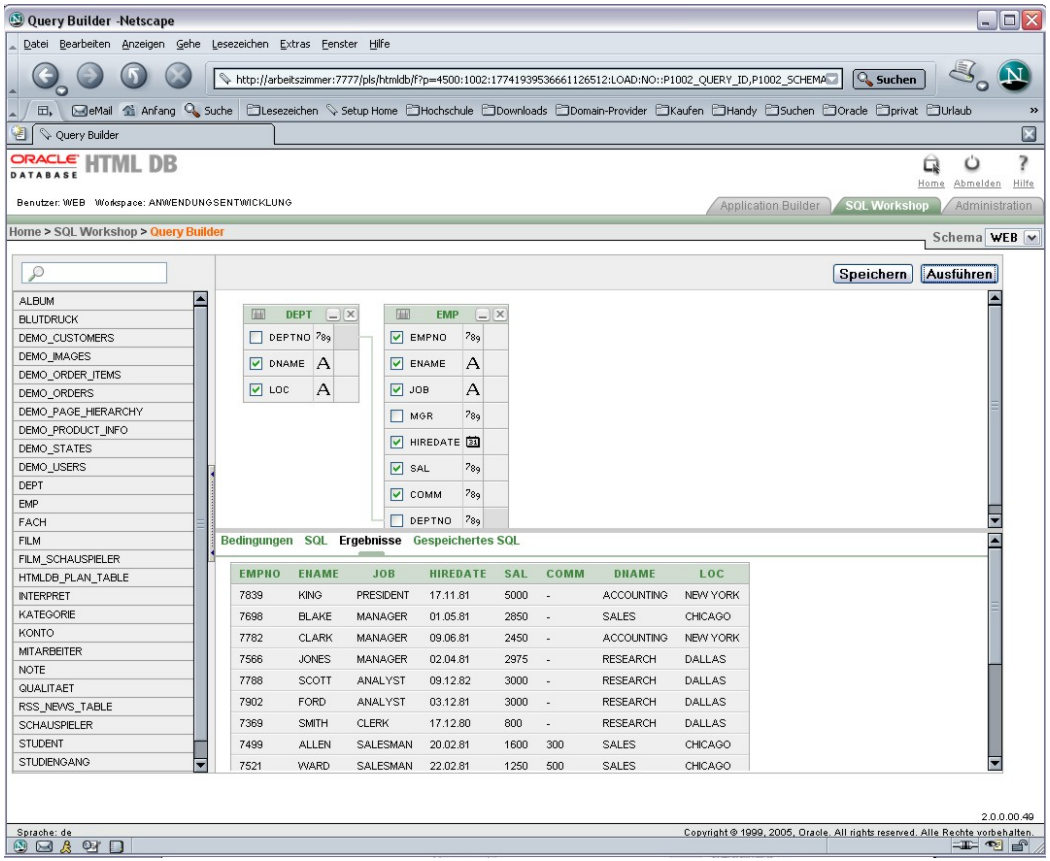


Abbildung 99.: Application Express Query Builder

8.3.2.3. Arbeiten mit Datenbankobjekten: SQL Workshop

Datenbankobjekte wie Tabellen, Prozeduren, Trigger oder andere können mit dem Objekt Browser im SQL Workshop erstellt, betrachtet und bearbeitet werden.

Zum Bearbeiten von PL/SQL steht ein in die Application Express integrierter Editor zur Verfügung. Funktionen wie Find & Replace, Undo oder Kompilieren des PL/SQLCodes stehen hier zur Verfügung – direkt im Browser. Auch für diesen Editor wird keine zusätzliche Client-Software benötigt.

Für Abfragen sind in der Application Express in der Regel keine SQL-Kenntnisse erforderlich. Mit einem integrierten Query Builder lassen sich die Abfragen sehr einfach zusammenstellen siehe Abbildung 99.

8.3.2.4. Datenimport in die Application Express

Die Entwicklung einer Anwendung beginnt normalerweise mit dem Laden eines Datenbestandes, der meist als kommaseparierte Datei oder als XML-Dokument vorliegt.

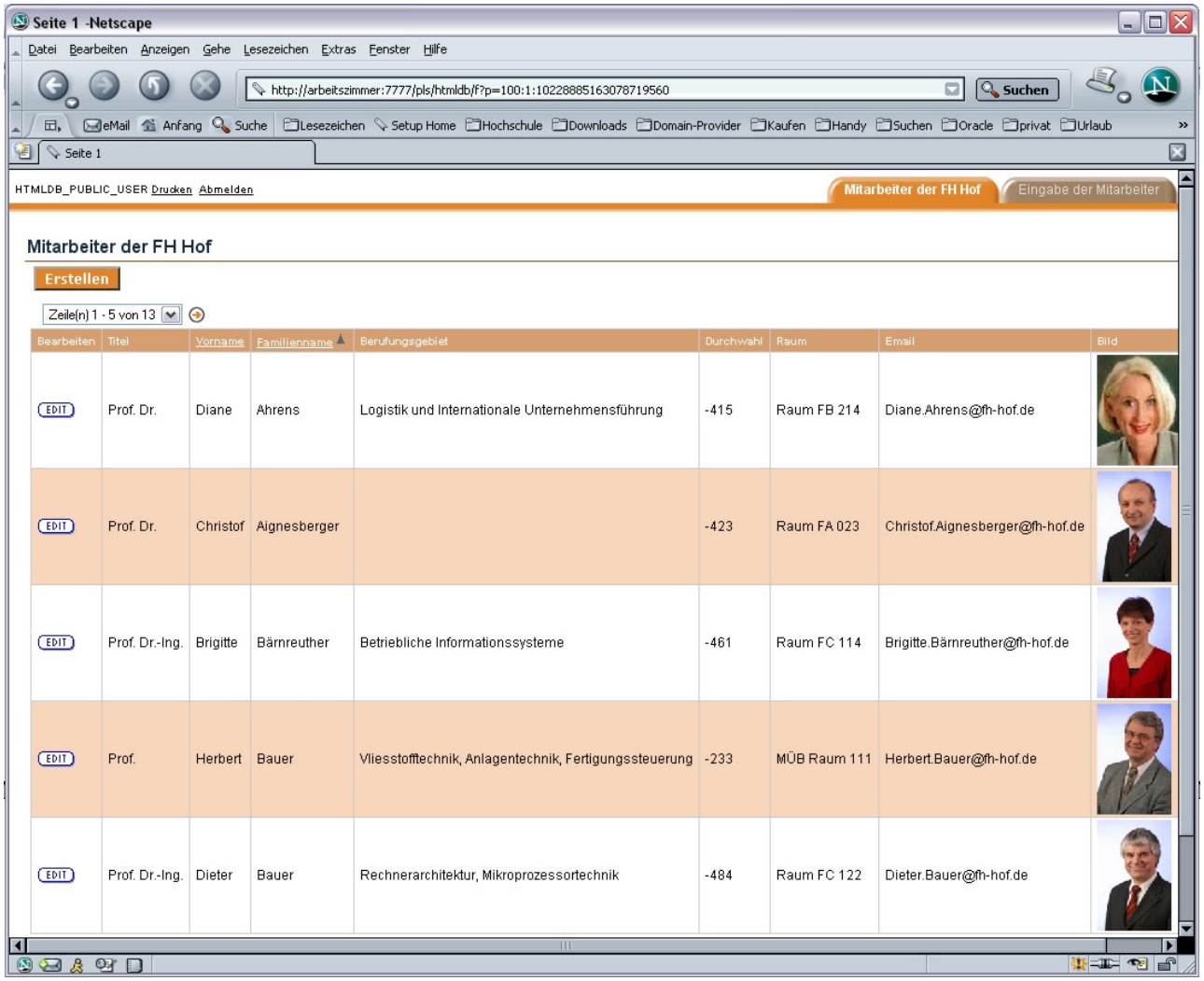
Kleine Datenmengen können per Copy & Paste in die Application Express übernommen werden, für größere Dateien steht ein Datei-Upload bereit. Wenn die Datenmengen extrem groß werden, sollte mit dem Administrator der Datenbank ein geeigneter Weg (SQL*Loader) gefunden werden.

Die hochgeladenen Daten werden auf eine Tabellenstruktur abgebildet. Der Entwickler hat nun die Möglichkeit, den Datenimport zu konfigurieren. So können die Namen und Datentypen der Tabellenspalten geändert oder Formatmasken für den Datenimport hinterlegt werden.

Die Verwaltung des für die Tabelle wichtigen Primärschlüssels übernimmt die Application Express automatisch – der Entwickler legt nur fest, ob eine neue oder eine bestehende Spalte als Schlüsselspalte dienen soll. Zum Generieren der eindeutigen Schlüssel werden die in der Oracle-Datenbank eigens für diesen Zweck vorhandenen Sequences genutzt.

8.3.2.5. Erstellen einer Application Express Anwendung

Bei der Erstellung von Application Express-Anwendungen wird der Entwickler von Assistenten unterstützt. Einmal gemachte Angaben können als Design Model gespeichert werden – mit der Application Express kann so auch komponentenorientiert entwickelt werden.



Die Application Express bietet viele vorgefertigte Anwendungskomponenten an. Dem erfahrenen Datenbank-entwickler erlaubt die Application Express auch die Hinterlegung von eigenem Java- oder PL/SQL-Code und so die Erfüllung auch ausgefallenster Anforderungen.

Abbildung 100.: Application Express Anwendung

Berichte zeigen die Inhalte von Datenbanktabellen auf einer Seite der Anwendung an. Funktionen wie Blättern im Bericht oder automatisches Sortieren bei Klick auf eine Spaltenüberschrift bietet die Application Express standardmäßig an. Der Bericht kann mit Hilfe des Query Builder bequem grafisch zusammengestellt werden.

Die gebräuchlichste Form eines Formulars ist das tabellenbasierte Formular. Für jede Tabellenspalte wird ein Eingabefeld erstellt.

Neben einfachen Textfeldern bietet die Application Express eine große Auswahl an Formularelementen an, darunter Auswahllisten, Mini-Kalender für die Datumsauswahl, siehe Abbildung 101, spezielle Eingabefenster für HTML-Dokumente und vieles mehr.

Abbildung 101.: Formularelement Kalender

Das spezielle Master-Detail Formular basiert auf zwei Tabellen, wird vollautomatisch erzeugt und erlaubt die gleichzeitige Navigation durch Master und Detail, siehe Abbildung 102.

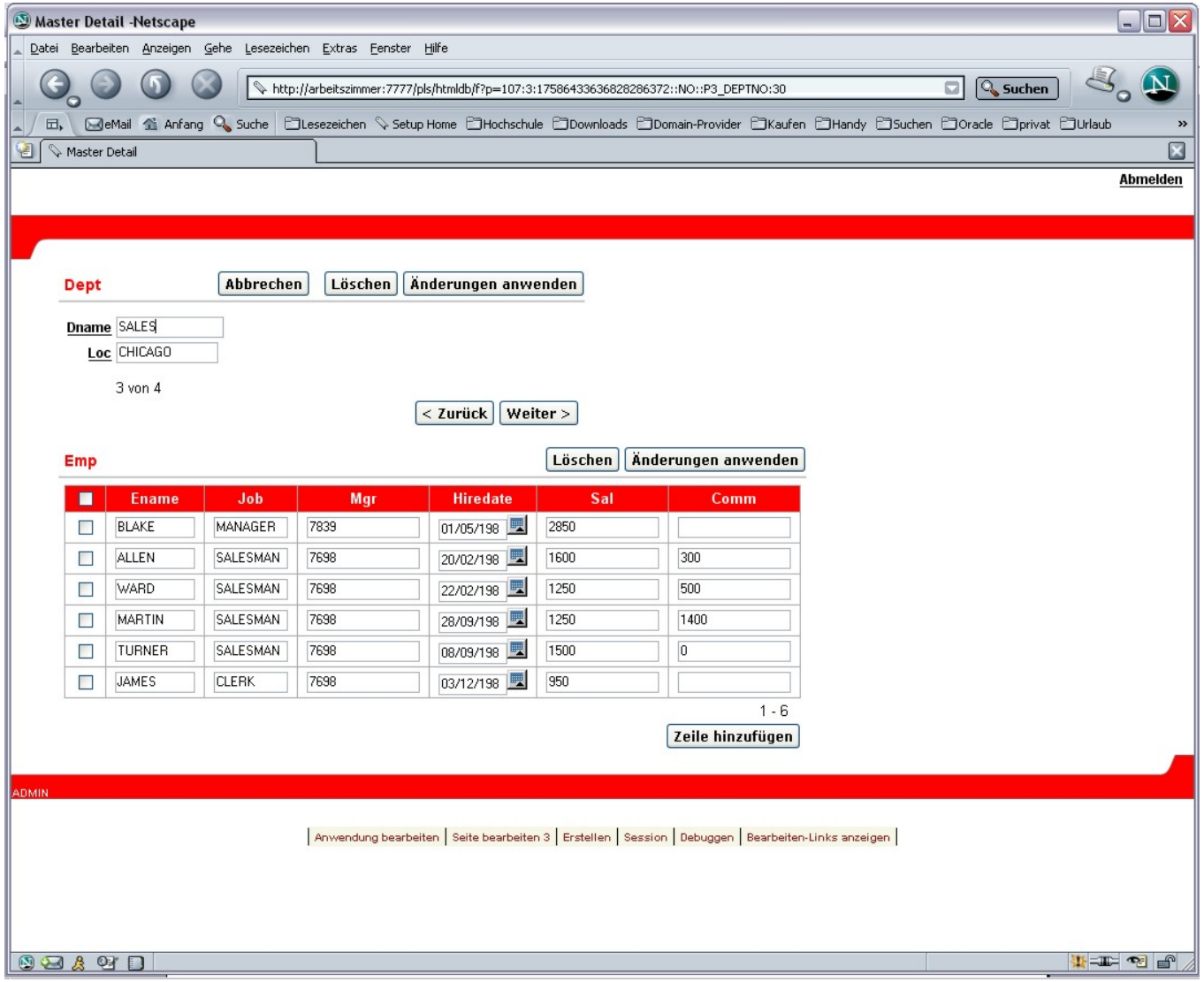


Abbildung 102.: Application Express Master Detail Anwendung

Jedes Eingabefeld kann mit Gültigkeitsbedingungen versehen werden. Besonders nützlich erweisen sich hier die mit der Oracle Database 10g eingeführten regulären Ausdrücke. Diese werden von der Application Express bereits unterstützt; insofern können auch komplexere Prüfungen elegant und ohne großen Codier-Aufwand gelöst werden.

Mit der Application Express können Diagramme auf sehr einfachem Weg in die Anwendung integriert werden. Grundlage ist dabei lediglich eine SQL-Abfrage; zur Erstellung dieser Abfrage steht ein eigener Assistent oder der weiter oben beschriebene Query Builder zur Verfügung.

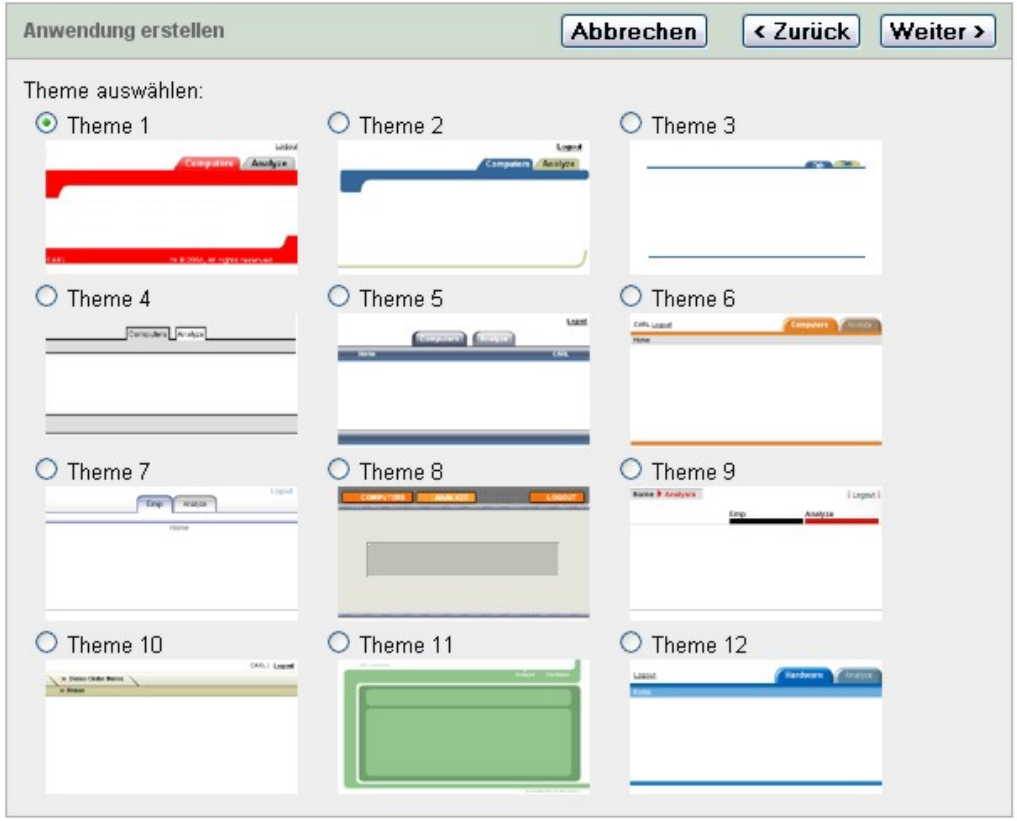
Ein Diagramm kann als SVG- (Scalable Vector Graphics) oder als einfaches HTML-Diagramm erstellt werden. SVG-Diagramme erfordern ein Plug-In im Browser, sie erlauben im Gegenzug jedoch eine sehr große Auswahl an Diagrammtypen. Neben einfachen Balkendiagrammen sind auch „gestapelte“ Balkendiagramme, Kreis- oder Liniendiagramme möglich.

Der Aufwand zur Implementierung von Diagrammen wird so erheblich reduziert; die Aufgabe des Entwicklers beschränkt sich auf die Bereitstellung der Abfrage und die Festlegung von Details wie Farben, Größe oder Legende des Diagramms.

8.3.2.6. Layoutgestaltung mit Themes und Templates

Die Darstellung der Application Express-Anwendungen ist vollständig durch Themes und Templates getrieben. Das Aussehen einer Seite wird im Template hinterlegt. So ermöglicht die Application Express die Trennung zwischen Definition und Layout der Komponenten.

Zusammengehörende Templates bilden ein Themes. Durch Austauschen des Themes wird das Look & Feel der Anwendung „auf einen Klick“ vollständig gewechselt.



Allein der Entwickler eines Templates entscheidet, wo Elemente wie Menüs oder Reiterkarten in der Anwendung erscheinen. So können Reiterkarten durchaus auch senkrecht angeordnet oder links erscheinen. Die mitgelieferten Themes und Templates dienen als Ausgangspunkt zur Erstellung von Anwendungen und zur Definition eigener Templates.

Abbildung 103.: Application Express mitgelieferte Themes

8.3.2.7. Zusammenfassung

Mit der Oracle Application Express können datenbankgestützte Anwendungen auf einfachem Wege erstellt und im Unternehmen verfügbar gemacht werden. Da die Anwendungen in einer zentralen Oracle-Datenbank liegen,

treten die in Bereich der Individualentwicklung typischen Probleme wie fehlendes Backup und Recovery, fehlende Mehrbenutzerfähigkeit oder mangelnde Integration in die IT-Landschaft nicht auf.

Durch die Assistentenunterstützung werden die Anwendungen sehr schnell erstellt. Die Application Express-Workspaces geben dem Entwickler die Flexibilität, die im täglichen Leben bei der Erstellung von Kleinanwendungen benötigt wird. Steht der Workspace einmal zur Verfügung, kann eine neue Anwendung einfach hinzugefügt werden. Publiziert wird sie durch Bekanntgabe einer URL. Langwierige und kostspielige Prozesse sind nicht mehr erforderlich.

Die Application Express bietet dem Entwickler damit sehr hohe Produktivität und Flexibilität bei der Entwicklung von Anwendungen, der IT-Betrieb hat darüber hinaus die Gewähr, dass alle Daten in einer zentralen Datenbank zur Verfügung stehen und zentral administrierbar bleiben. Somit sind die Vorteile der zentralen Datenbank mit der Flexibilität dezentraler Systeme kombiniert und durch die Web-Technologie überall im Unternehmen sofort verfügbar.

Tabellenverzeichnis

Abbildungsverzeichnis

Definitionsverzeichnis

Index