5.3. Methoden zur Abbildung der Integritätsbedingungen

Datenbanksysteme modellieren einen bestimmten Teil der realen Welt. Sie sollen zum einen in sich stimmig sein und zum anderen mit dem modellierten Abschnitt der Realität inhaltlich übereinstimmen. Die Wahrung dieser Datenintegrität ist eine der wichtigsten Aufgaben eines Datenbanksystems. Bei der Integritätssicherung wird zwischen semantischer, relationaler und operationaler Integrität unterschieden.

Früher ließen sich solche Integritätsbedingungen nur im Anwendungsprogramm verifizieren. Da in der Regel aber mehrere Programme auf eine Datenbank zugreifen, ist es sinnvoller, die Verifizierung von Integritätsbedingungen nicht in jeder einzelnen Anwendung zu realisieren, sondern zentral im Datenbanksystem zu implementieren. Dadurch wird die Anwendungsentwicklung entlastet, da sie sich um die Programmierung der Integritätsbedingungen nicht kümmern muß und gleichzeitig die Qualität der Software erhöht, weil die Fehlergefahr wegfällt, die durch mehrfache Implementierung der gleichen Integritätsbedingungen entsteht.

SQL/89 (Level 2) kannte nur wenige Möglichkeiten, Integritätsbedingungen zu definieren. Dazu gehörten Schlüsselkandidaten, Fremdschlüssel und einfache Spaltenbedingungen. SQL/92 beherrscht Tabellenbedingungen, die sich auf mehrere Spalten der gleichen oder einer anderen Tabelle beziehen, sowie Assertions (nicht an Tabellen gebunden) für den Gesamtdatenbestand. Außerdem können verschiedene Aktionen bezüglich der referentiellen Integrität festgelegt und der Zeitpunkt der Prüfung von Bedingungen variiert werden.

5.3.1. Semantische Integrität

Die semantische Integrität beschäftigt sich z.B. mit der inhaltlichen Korrektheit der Daten. Diese dürfen nur die in der Realität auch möglichen Werte annehmen. Semantische Integrität wird in Integritätsbedingungen (Integrity rules) formuliert.

Beispielsweise darf das Feld „Steuerklasse" eines Mitarbeiters nur die Werte 1 bis 6 annehmen (Integritätsbedingung eines einzelnen Feldes); wenn der Mitarbeiter den Familienstand „ledig" hat, sollte der Wert im Feld Steuerklasse nicht den Wert 3 annehmen (Integritätsbedingung zwischen mehreren Feldern); die Anzahl der Mitarbeiter pro Abteilung darf einen Maximalwert nicht überschreiten (komplexe Integritätsbedingung über mehrere Datensätze einer Tabelle).

5.3.1.1. Tabellen- und Spaltenbedingungen

Unter einer Tabellenbedingung (Table Constraint) (Constraint=Einschränkung) versteht man einen bedingten Ausdruck, der in einer Tabellendefinition angegeben wird (in Access auch in der Tabellenentwurfsansicht) und sich auf eine oder mehrere Spalten der gleichen oder auch anderer Tabellen beziehen kann. Eine Spaltenbedingung (Column Constraint) dagegen bezieht sich nur auf eine Spalte der Tabelle. Sie kann unmittelbar in der Spaltendefinition oder als eigenständiges Element in der Tabellendefinition angegeben werden. Tabellen- oder Spaltenbedingungen dürfen bei ihrer Auswertung nicht den booleschen Wert FALSE ergeben.

Die zwei einfachsten Formen von Spaltenbedingungen sind NOT NULL und UNIQUE. Wird für eine Spalte NOT NULL spezifiziert, so muß das Datenbankfeld immer einen Wert enthalten. Die Bedingung UNIQUE legt für eine Spalte fest, daß zwei verschiedene Datensätze einer Tabelle nicht gleiche Werte in dieser Spalte enthalten dürfen. Die flexibelste Möglichkeit, Tabellen- oder Spaltenbedingungen zu definieren, stellt jedoch die

Access "ohne Duplicate"

CHECK-Klausel dar: sie besteht aus einem bedingten Ausdruck, der mehrere miteinander durch AND, OR oder NOT verknüpfte Prädikate verwenden darf.

Im Beispiel sind für die ersten drei Spalten der Tabelle die NOT-NULL-Bedingungen definiert. Für Geschlecht und Steuerklasse gelten Spaltenbedingungen. Zusätzlich legt eine Tabellenbedingung fest, daß der durchschnittliche Urlaubsanspruch aller Mitarbeiter 30 Tage nicht überschreiten darf.

```
PersNr NUMERIC(5) NOT NULL,
Nname CHAR(30) NOT NULL,
Vname CHAR(20) NOT NULL,
Geschlecht CHAR(8) CHECK(VALUE IN ( „männlich", „weiblich")),
Steuerkl NUMERIC(1) CHECK (VALUE BETWEEN 1 AND 6),
Urlaub NUMERIC (2),
CONTRAINT ZuVielUrlaub
CHECK((SELECT AVG (Urlaub) FROM Mitarbeiter ) <= 30)
```

Der Unterschied zwischen Tabellen- und Spaltenbedingungen spielt eine Rolle, wenn man mit DROP eine Spalte aus der Tabelle entfernen will. Bezieht sich eine Tabellenbedingung auf die zu löschende Spalte, so kann die Spalte nicht gelöscht werden. Eine Spaltenbedingung der zu löschenden Spalte wird dagegen ebenfalls gelöscht.

5.3.1.2. Wertebereiche

Im Beispiel unterliegt „Geschlecht" der Spaltenbedingung: CHECK (VALUE IN („männlich", „weiblich")). Diese Integritätsregel ist aber genauer betrachtet keine spezifische Eigenschaft der Spalte „Mitarbeiter.Geschlecht", sondern eine Regel, die generell für den Wertebereich Geschlecht - auch in anderen Tabellen - Gültigkeit besitzt.

Hier wird deutlich, wie zweckmäßig es ist, Wertebereiche unabhängig von Datenbankfeldern zu modellieren. SQL/92 führt zu diesem Zweck ein **Domain-Konzept** ein. Eine Domain besitzt einen logischen Namen und wird auf einen Datentyp abgebildet. Zusätzlich lassen sich für Domains Integritätsbedingungen und Defaultwerte definieren.

Die Domain kann bereits im CREATE-TABLE-Statement alternativ zur Angabe eines Datentyps verwendet werden (nicht in Access). Unter Verwendung des Domain-Konzeptes nimmt die DDL zur Definition unserer Beispieltabelle „Mitarbeiter" folgende Gestalt an:

```
CREATE TABLE Mitarbeiter(
...
CREATE DOMAIN Geschlechter AS CHAR(8) CHECK(VALUE IN („männlich",
„weiblich"))
Geschlecht Geschlechter,
...)
```

Nicht nur bei der Abbildung von wertebereichspezifischen Integritätsbedingungen, sondern für alle Wertebereiche, die in der Datenbank mehrfach verwendet werden, leisten Domains gute Dienste.

5.3.1.3. Bedingungen, die nicht an Tabellen gebunden sind

Neben den bisher beschriebenen Tabellen-, Spalten- und Domainbedingungen gibt es auch die Möglichkeit, Assertions zu definieren. Assertions werden nicht an eine Tabelle oder Spalte geknüpft, sondern direkt mit CREATE ASSERTION angelegt. Das Beispiel stellt sicher, dass die Anzahl der Mitarbeiter pro Abteilung einen vorgeg. Maximalwert nicht überschreitet:

```
CREATE ASSERTION Abtgroesse
CHECK (NOT EXIST (SELECT * FROM Abteilung WHERE (SELECT COUNT(*)
FROM Mitarbeiter WHERE Mitarbeiter.AbtNr =Abteilung.AbtNr) >Abtei-
lung.MaxMitarbeiter))
```

Eine Assertion erhält einen Namen, den Sie spätestens dann wieder benötigen, wenn Sie mit DROP ASSERTION die Integritätsbedingung aufheben wollen. Assertions dienen dazu, Abhängigkeiten zwischen Tabellen zu formulieren.

Die Entscheidung, ob Sie eine Integritätsbedingung einer Tabelle zuordnen oder besser als Assertion definieren, gewinnt dann an Bedeutung, wenn Tabellen gelöscht werden sollen. Ist eine Integritätsbedingung einer Tabelle zugeordnet, wird sie durch DROP TABLE gleich mit entfernt. Dagegen gilt für Tabellen, die in einer Assertion verwendet werden, eine explizite Löschsperre. In Access gibt es keine Assertions.

5.3.2. Relationale Integrität

Neben den Integritätsregeln, die die reale Welt vorschreibt, muß eine relationale Datenbank auch die Integritätsregeln einhalten, die sich aus dem relationalen Modell ergeben: das sind Primär- und Fremdschlüsselintegrität (Entity und Referential Integrity).

- **Entity Integrity** sagt aus, daß jede Tabelle ein Attribut oder eine Attributmenge besitzen muß, die die Datensätze eindeutig identifiziert. Oder anders herum: zwei Datensätze dürfen nicht denselben Primärschlüssel besitzen.
- **Referential Integrity** (RI) bezieht sich auf die Abbildung von Beziehungen durch Fremdschlüssel. Jeder Fremdschlüssel muß auf einen exisitierenden Primärschlüssel referenzieren. Primär- und Fremdschlüssel lassen sich am besten an dem klassischen Beispiel Abteilung und Mitarbeiter darstellen. Jeder Mitarbeiter kann einer Abteilung zugeordnet sein, zu einer Abteilung gehören mehrere Mitarbeiter:

```
CREATE TABLE Abteilung(
AbtNr NUMERIC(5),
Name CHAR(20),
MaxMitarbeiter INTEGER,
PRIMARY KEY ( AbtNr))

CREATE TABLE Mitarbeiter (
PersNr NUMERIC (5) NOT NULL,
...
AbtNr NUMERIC(5),
PRIMARY KEY (PersNr),
FOREIGN KEY (AbtNr) REFERENCES Abteilung [(AbtNr)])
```

Jeder Mitarbeiter verweist über den Fremdschlüssel AbtNr auf die Abteilung, in der er arbeitet. Der Wert im Feld AbtNr darf dabei nur einen der in der Tabelle Abteilung auch existierenden Primärschlüsselwerte oder den Wert Null annehmen. Alle anderen Werte würden die referentielle Integrität verletzen. In der Foreign-Key-Klausel wird die Referenz auf die Tabelle Abteilung definiert. Die Angabe des Primärschlüssels, auf den referenziert wird, ist optional, zumindest wenn in der referenzierten Tabelle ein Primärschlüssel existiert. Es besteht in SQL/92 nämlich im Widerspruch zum relationalen Modell die Möglich-

keit, eine Tabelle ohne Primärschlüssel zu definieren und mit dem Fremdschlüssel auf einen Candidate Key (Schlüsselkandidaten) zu verweisen.

Die Einhaltung der Entity Integrity und Referential Integrity (RI) bei Änderungsoperationen ist in Tabellen, in denen Primär- oder Fremdschlüssel definiert sind (Mitarbeiter), relativ einfach zu gewährleisten. Dagegen gestaltet sich die Behandlung von Primärschlüsseländerungen in den referenzierten Tabellen (hier: Abteilung) weitaus schwieriger. Wird nämlich ein Primärschlüssel geändert oder gelöscht, dann kann es eine Menge von Datensätzen in anderen Tabellen geben, die mit ihrem Fremdschlüssel gerade diesen Primärschlüsselwert referenzieren. Eine Änderung des Primärschlüssels würde in den referenzierenden Tabellen die referentielle Integritität verletzen, die Referenz ginge ins Leere.

SQL/92 bietet für solche Fälle einen eleganten Mechanismus an: mit referentiellen Aktionen (referential actions) legt man schon während der Fremdschlüsseldefinition deklarativ fest, wie die Datenbank auf das Ändern (ON UPDATE) oder Löschen (ON DELETE) eines referenzierten Primärschlüssels reagieren soll:

- CASCADES - Wird der Primärschlüssel geändert/gelöscht, so werden alle referenzierten Datensätze ebenfalls geändert/gelöscht.
- Die Aktion RESTRICT ist auf einigen SQL-Servern realisiert und auch in SQL3 vorgesehen. RESTRICT verbietet das Ändern/Löschen, solange noch Datensätze auf den Primärschlüssel referenzieren. Die Operation wird sofort verworfen, wenn noch eine Referenz besteht.
- SET NULL - Alle Fremdschlüsselfelder in den referenzierenden Datensätzen erhalten den Wert Null. Diese Aktion ist natürlich nur möglich, wenn die entsprechenden Felder nicht mit dem Column-Constraint NOT NULL belegt sind.
- SET DEFAULT - Diese Aktion ähnelt sehr der Aktion SET NULL: alle Fremdschlüsselfelder werden mit ihrem Defaultwert belegt. Ist kein Default definiert, springt der Nullwert in die Bresche.

Im Beispiel macht die Mitarbeiter-Tabelle eine Änderung der Abteilung gleich mit, beim Löschen einer Abteilung erhält der Fremdschlüssel den Nullwert, der Mitarbeiter ist dann keiner Abteilung mehr zugeordnet.

```
CREATE TABLE Mitarbeiter (PersNr NUMERIC (5) NOT NULL,
PRIMARY KEY ( PersonalNr),
FOREIGN KEY ( AbtNr) REFERENCES Abteilung [(AbtNr)]
ON UPDATE CASCADES
ON DELETE SET NULL)
```

Da die Einhaltung der referentiellen Integrität ohne Programmierung erfolgt, heißt diese Art der Sicherung auch deklarative RI.

5.3.3. Operationale Integrität

Operationale Integrität meint die Aufgabe, den Datenbestand im Mehrbenutzerbetrieb bei konkurrierendem Zugriff vor Inkonsistenzen zu schützen. Die Einhaltung der operationalen Integrität wird in einer Datenbank durch Transaktionen gewährleistet. Eine Transaktion ist dabei eine Folge von SQL-Statements (i.d.R. Änderungsoperationen), die logisch zusammengehören und deshalb als atomar anzusehen sind. Entweder werden alle Statements der Transaktion ausgeführt oder keines.

Beispielsweise führt in einem Buchhaltungssystem der Buchungssatz „Kasse an Forderungen DM 100" zur Erhöhung des Kontostandes der Kasse im Haben und der Forderungen im Soll um je 100 DM. Beide Operationen bilden zusammen eine Transaktion. Denn: Wird nur eine dieser Operationen ausgeführt, sind Daten der Datenbank inkonsistent. Neben

dem system- oder programmbedingten Abbruch einer Transaktion sind parallele Aktionen anderer Datenbankbenutzer die zweite Fehlerquelle, die ein Datenbanksystem bewältigen muß.

In SQL/92 wird eine Transaktion mit TRANEX ... END beschrieben. Eine Transaktion kann dann mit COMMIT ordnungsgemäß beendet oder mit ROLLBACK zurückgesetzt werden. Mit ROLLBACK lassen sich alle Änderungen seit Beginn der Transaktion rückgängig machen, die Datenbank befindet sich im Zustand wie vor Beginn der Transaktion.

- Falls die Anwendung ein abnormales Ende gefunden hat (Stromausfall), wird beim Hochfahren des Systems automatisch vom DBMS ein ROLLBACK ausgeführt.
- Das Rücksetzen von Transaktionen ist möglich, da in einer Protokolldatei (Log-File) über die Operationen buchgeführt wird. Protokolliert wird vor der Operation ! („write ahead-log-protocol"). Die Protokoll-Datei wird auf der Platte gehalten und zu festen Zeitpunkten (Checkpoints) ggf. auch auf Band gesichert.
- Bei interaktivem SQL ist jede SQL-Anweisung eine Transaktion
- Zu Beginn/Ende einer Transaktion wird ein Synchpoint gesetzt. Beim Setzten eines Synchpoints:
  - werden alle Änderungen seit dem vorherigen Synchpoint entweder bestätigt oder zurückgesetzt
  - Werden alle Sperren von Datensätzen aufgehoben

In allen Integritätsbedingungen, auch den relationalen, können Sie angegeben, zu welchem Zeitpunkt innerhalb einer Transaktion ihr System eine Integritätsbedingung überprüfen soll. Jede Integritätsbedingung ist zu jedem Zeitpunkt einer Transaktion in einem der zwei Modi IMMEDIATE oder DEFERRED. IMMEDIATE bedeutet, daß die Bedingung nach jeder Ausführung eines SQL-Statements überprüft wird. DEFERRED legt fest, daß die Überprüfung der Bedingung erst am Ende der Transaktion erfolgt.

Die Option einer verzögerten Integritätsprüfung ist sinnvoll, da komplexe Integritätsbedingungen oft erst nach Ausführung mehrerer SQL-Statements erfüllt sind. SET CONSTRAINT IMMEDIATE/DEFERRED legt den Modus fest. Bereits bei der Definition einer Integritätsbedingung können Sie festlegen, in welchem Modus sie arbeiten soll (INITIALLY IMMEDIATE / INITIALLY DEFERRED) beziehungsweise ob sie überhaupt verzögert werden kann (DEFERRABLE / NOT DEFERRABLE). Ohne zusätzliche Angaben gilt defaultmäßig NOT DEFERRABLE.

Zur Synchronisation des Mehrbenutzerbetriebs muß sich der SQL-Programmierer auch nicht um das Sperren von Datensätzen oder von Tabellen kümmern. Deklarativ bietet SQL/92 über SET TRANSACTION vier abgestufte Isolationsebenen (Isolation Level) an.

Jedes Isolation Level schließt bestimmte Phänomene, die im Mehrbenutzerbetrieb auftreten können, aus oder nimmt sie in Kauf. Dirty Read läßt zu, daß die eigene Transaktion Daten lesen kann, die andere Transaktionen geändert haben, obwohl die anderen Transaktionen noch nicht mit COMMIT beendet worden sind. Dieser READ UNCOMMITTED birgt die Gefahr, daß die anderen Transaktionen mit Rollback zurückgesetzt werden und die gelesenen Daten in der Datenbank gar nicht in Erscheinung treten.

Das Isolation Level READ COMMITTED schließt genau diesen Fall aus, läßt aber den „Nonrepeatable Read" zu. READ COMMITTED erlaubt es, daß Datensätze, die die eigene Transaktion liest, vor dem Ende der eigenen Transaktion durch andere Transaktionen verändert werden dürfen. Damit kann wiederholtes Lesen desselben Datensatzes innerhalb einer Transaktion unterschiedliche Ergebnisse liefern. Das Isolation Level REPEATABLE READ verhindert diesen Effekt.

Das Phantomproblem tritt auf, wenn die eigene Transaktion eine Menge von Datensätzen liest, die eine bestimmte Bedingung erfüllen. Fügt eine andere Transaktion einen neuen Datensatz ein, der ebenfalls diese Bedingung erfüllt, dann führt bei einer Wiederholung dieselbe Abfrage innerhalb einer Transaktion zu unterschiedlichen Ergebnissen. Der Isolation Level SERIALIZABLE verhindert dieses Phantomproblem.

Isolationsebene	Dirty Read Phänomen	Nonrepeatable Read-Phänomen	Phantom Phänomen
READ UNCOMMITTED	erlaubt	erlaubt	erlaubt
READ COMMITTED	nicht erlaubt	erlaubt	erlaubt
REPEATABLE READ	nicht erlaubt	nicht erlaubt	erlaubt
SERIALIZABLE	nicht erlaubt	nicht erlaubt	nicht erlaubt

5.4. Einbindung von SQL in 3GL-Sprachen

Hat man eine so mächtige und elegante Sprache für den Zugriff auf relationale Datenbanken wie SQL, dann möchte man diese nicht nur für interaktive Ad-hoc-Abfragen nutzen (obwohl gerade das der Wunsch der Urväter von SQL war, dem Manager eine einfache Sprache anzubieten, mit der er seine relationale Datenbank abfragen kann). Um SQL als Datenbankprogrammiersprache einzusetzen, fehlen aber Variablenkonzepte und die elementaren Konstrukte der prozeduralen Programmierung, wie Wiederholungen (z. B. REPEAT ... UNTIL) und Bedingungen (IF ... THEN ... ELSE, CASE). Zweckmäßiger ist es, SQL in Programmiersprachen einzubinden.

Ein Problem bei der Verwendung von SQL in Programmiersprachen der 3. Generation sind die Ergebnistabellen der SQL-Mengenoperationen (Select, Join, Union, etc.). Die lassen sich nicht unmittelbar weiterverarbeiten. Zwar bietet SQL mit UPDATE, INSERT und DELETE die Möglichkeit, eine Menge von Datensätzen zu verändern, umgekehrt kann man aber auf die Ergebnistabellen von SQL-Mengenoperationen mit Befehlen von 3GL-Sprachen nicht zugreifen. Abhilfe schafft das Cursorkonzept von SQL.

5.4.1. Cursorkonzept und Embedded SQL

Ein mit dem DECLARE-Befehl angelegter Cursor deklariert eine SQL-Mengenoperation. Nach dem Öffnen des Cursors (OPEN CURSOR) kann man satzorientiert auf die Ergebnismenge des Cursors zugreifen. Der satzorientierte Zugriff erfolgt dabei mit Hilfe des FETCH-Befehls.

Ausgehend vom aktuellen Datensatz des Cursors kann der Programmierer über Navigationsfunktionen den Datensatzzeiger absolut oder relativ positionieren (FETCH NEXT | PRIOR | FIRST | LAST | ABSOLUTE number | RELATIVE number) und mit der Deklaration des Cursors bereits Zugriffsrechte (nur Lesen, Ändern etc.) definieren. Dies ist notwendig, da die SQL-Datenbank in Abhängigkeit vom gewählten Isolation Level alle Sperren selbständig ausführt (siehe operationale Integrität). Mit SQLCA (SQL-Communication-Area) können Statusmeldungen von SQL, etc. abgefragt werden.

Mit dem Cursorkonzept ist die Möglichkeit einer satzweisen Verarbeitung von Tabellen in 3GL-Sprachen geschaffen. Notwendig sind jetzt noch Regeln, nach denen SQL in 3GL-Sprachen verwendet wird. Der SQL/92-Standard bietet hier verschiedene Konzepte an.

Embedded SQL ermöglicht die Verwendung von SQL-Statements in einer Wirtssprache (host language). SQL/92 unterstützt als Wirtssprachen Ada, C, Cobol, Fortran, Pascal und PL/I.

SQL-Deklarationen und -Statements gehen direkt in den Quelltext der Wirtssprache ein. Embedded SQL wird in der Hostsprache durch ein vorangestelltes EXEC SQL identifiziert. Ein Precompiler setzt dann SQL-Statements in Funktionsaufrufe um.

Beispiel zu eingebettetem SQL:

```
/* Zur Vereinfachung wird auf die Behandlung von Fehlern verzichtet.
   Wir definieren einen Cursor für die Lieferanten und einen
   Cursor für die von einem Lieferanten belieferten Projekte */

EXEC SQL  DECLARE ASupplier CURSOR FOR
          SELECT    SNo, Name, Status, City
          FROM      Supplier
          ORDER BY  SNo;

EXEC SQL  DECLARE AProject CURSOR FOR
          SELECT    JNo, Name, City
          FROM      Project
          WHERE     JNo IN
                    (SELECT Jno
                     FROM SPP
                     WHERE SNo = :Sup_No)
          ORDER BY JNo;

/* Nun laufen wir mit den Cursors durch die Lieferanten
   und bei jedem Lieferanten durch die belieferten Projekte */

EXEC SQL OPEN ASupplier;
  Stop? := 0;      /* lokale Variable innerhalb dieses Blocks.
                      Schleife abbrechen, falls kein Lieferant
                      mehr gefunden */
  DO WHILE Stop? <> 100
    EXEC SQL  FETCH ASupplier
              INTO :Sup_No, :Sup_Name, :Sup_Status, :Sup_City;
    Stop? = SQLCA.SQLCODE
    IF Stop? <> 100 THEN DO
      BEGIN
      PRINT Sup_No, Sup_Name, Sup_Status, Sup_City;
      EXEC SQL OPEN AProject;
        Stop? = 0;  /* lokale Variable innerhalb dieses Blocks.
                       Schleife abbrechen, falls kein Projekt
                       mehr gefunden */
        DO WHILE Stop? <> 100
          EXEC SQL  FETCH Aproject
                    INTO :Pro_No, :Pro_Name, :Pro_City;
          Stop? = SQLCA.SQLCODE
          IF Stop? <> 100 THEN DO
            PRINT Pro_No, Pro_Name, Pro_City;
        ENDDO;
        EXEC SQL CLOSE AProject;
      ENDDO
  ENDDO
EXEC SQL CLOSE ASupplier;
```

Das Programm liefert die Daten aller Lieferanten, aufsteigend nach den Lieferantennummern. Bei jedem Lieferanten-Datensatz wird unmittelbar ein Listing mit den Daten aller Projekte ausgegeben, für die der Lieferant ein Teil geliefert hat, geordnet nach der Projektnummer. In der Tabelle SPP ist die Zuordnung von Lieferanten zu Projekten festgehalten.

5.4.2. Dynamic SQL

Dynamisches SQL ist lediglich eine konzeptionelle Erweiterung von Embedded SQL. SQL-Statements werden erst zur Laufzeit generiert und ausgeführt, Datenbankoperationen müssen deshalb nicht im voraus feststehen. Dadurch verschlechtert sich allerdings auch die Performance, da der Syntax-Check erst zur Laufzeit erfolgt.

Unabhängig vom ANSI-Standard haben sich auf dem Markt alternative Lösungen zur Datenbankprogrammierung entwickelt. Beispielsweise wurden traditionelle Programmiersprachen der dritten Generation wie Pascal um eine Datenmanipulationssprache (DML) erweitert. Daraus entstanden eine Reihe von xBase-Dialekten wie dBase, FoxPro und Clipper mit recht bescheidenem SQL-Umfang. Diese Sprachen zeichneten sich in erster Linie durch eine performante, satzorientierte Tabellenbearbeitung aus. Konstrukte zur Definition relationaler und sematischer Integritätsbedingungen fehlten allerdings, auch die operationale Integrität mußte der Programmierer durch Satz- und Dateisperren realisieren.

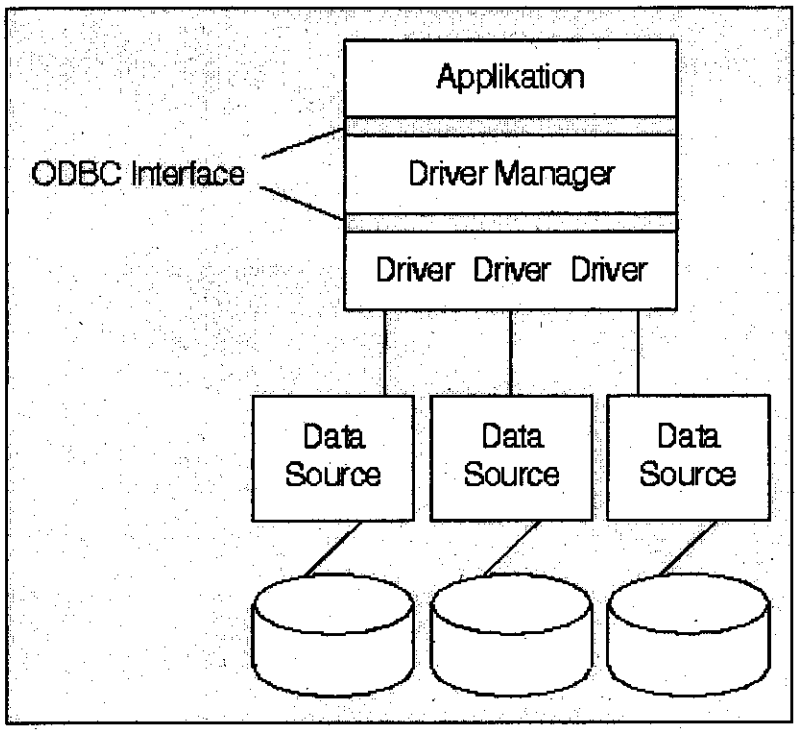
Die Erweiterung von SQL selbst zu einer Datenbankprogrammiersprache bieten heute nahezu alle SQL-Server an (z.B. PLSQL in Oracle). Damit sind jedoch nicht nur Vorteile verbunden. Eine auf einen bestimmten Server abgestimmte 4GL-Sprache (z.B. Visual Basic in Access) garantiert zwar eine effektive Nutzung der Datenbankmaschine und hohe Produktivität bei der Entwicklung, gleichzeitig wächst jedoch die Abhängigkeit vom Hersteller. Die Portierung auf andere 4GL-Umgebungen gelingt, wenn überhaupt, nur mit unverhältnismäßig hohem Aufwand, zumal in der Regel auch herstellerspezifische Masken-, Menü- und Reportgeneratoren Verwendung finden.

5.4.3. Call-Schnittstelle

Die modernste und zukunftsträchtigste Möglichkeit, SQL in 3GL-Sprachen zu verwenden, bieten prozedurale Schnittstellen. SQL-Statements werden in Funktionsaufrufe als String-Parameter übergeben und zur Laufzeit vom DBMS interpretiert. Diese spezielle Form des dynamischen SQL wird im Rahmen des SQL3-Standards als SQL/CLI (Call Level Interface) ebenfalls normiert.

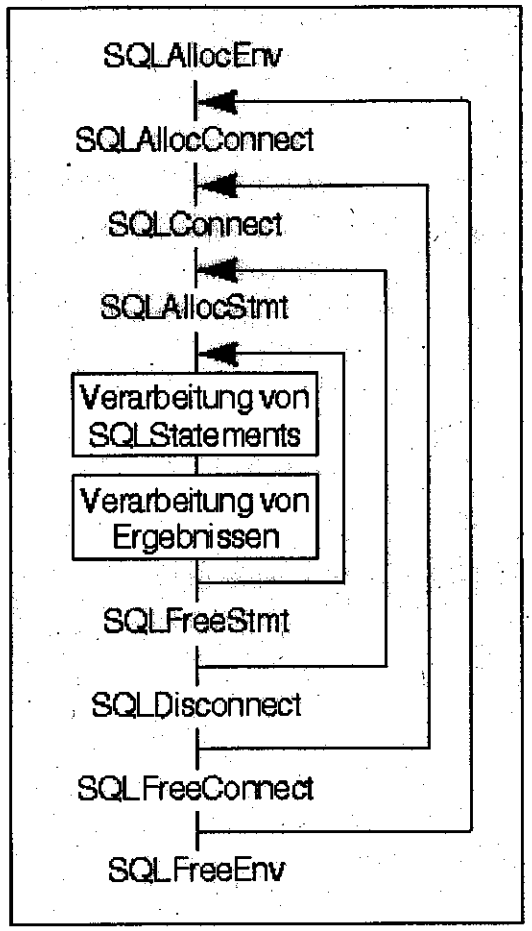
Die verbreitetste Implementierung einer Call-Schnittstelle ist wohl zur Zeit ODBC unter Microsoft Windows, das im wesentlichen auf einem CLI-Standard von X/Open und der SQL Access Group (SAG) basiert. Diese Spezifikation wurde bei der ISO als Erweiterung von SQL vorgeschlagen und hat gute Chancen, weitestgehend als SQL/CLI-Norm übernommen zu werden.

Eine ODBC-Schnittstelle ist dabei eine Funktionsbibliothek, aus der sich ein Anwendungsprogramm bedienen kann. Sie beinhaltet Funktionen, die zum Beispiel die Verbindung zu einer Datenbank aufbauen, SQL-Statements ausführen oder die Ergebnisse einer Datenbankabfrage zur Verfügung stellen.



Um überhaupt Befehle an einen SQL-Server schicken zu können, müssen Sie zunächst eine Verbindung zur betreffenden Datenbank über einen ODBC-Treiber aufbauen. Dies geschieht dreistufig über ein Environment-Handle, Connection-Handles und Statement-Handles (Handles sind Pointer auf spezielle Datenstrukturen).

Jede Applikation benötigt für den Datenbankzugriff über ODBC-Treiber ein Environment Handle, das die Verbindung zum jeweiligen Treiber verwaltet. Für jede Verbindung zu einer Datenquelle - sprich SQL-Server - benötigt man außerdem ein Connection Handle. Mit ODBC ist es problemlos möglich, aus einem Anwendungsprogramm auf mehrere SQL-Server zuzugreifen. Zur Ausführung eines SQL-Befehls muß zusätzlich ein Statement Handle alloziert werden. Das Statement Handle identifiziert einen abgesetzten SQL-Befehl und dient unter anderem dazu, einem SQL-Befehl dynamische Parameter zu übergeben beziehungsweise Speicherbereiche für diese bereitzustellen oder Cursornavigations-operationen für ein bestimmtes Select-Statement auszuführen (z.B. **SQLFetch()**).



Der wesentliche Vorteil der Call-Schnittstelle gegenüber Embedded SQL besteht darin, daß alle SQL-Befehle über C-Funktionsaufrufe realisiert werden. Damit funktioniert der Datenbankzugriff über ODBC sofort in allen Programmiersprachen, die C-Funktionsaufrufe unterstützen. Im Gegensatz dazu ist man bei Embedded und Dynamic SQL auf einen Precompiler angewiesen, der für jede Programmiersprache anders ausfällt. Es ist damit zu rechnen, daß das CLI zunehmend Embedded SQL verdrängen wird.

-> Übung: SQL-Abfragen in Access anlegen

5.5. SQL 3

Mit der Verabschiedung der SQL/92-Norm fiel gleichzeitig der Startschuß für weitergehendere Standardisierungsbemühungen. Im geplanten SQL3 (genaueres unter http://www.jcc.com/sql_stnd.html) soll zum einen der Leistungsumfang der relationalen Befehle verbessert werden, zum anderen richtet sich das Interesse auf die Einführung objektorientierter Konzepte. Die Anzahl der Embedded SQL unterstützenden Hostsprachen vergrößert sich, außerdem wird die oben beschriebene CLI-Spezifikation mit in den Standard aufgenommen.

SQL3 im Überblick:

- **Neue Datentypen:** Der neue SQL3-Standard unterstützt den Datentyp BOOLEAN, Aufzählungstypen (Enumerated Data Types) wie zum Beispiel Wochentage sowie benutzerdefinierte Datentypen mit komplexer interner Struktur (Arrays, Listen, Sets).
- **Sub- und Supertabellen:** Zukünftig unterstützen Tabellen die Definition von Vererbungshierarchien in Form von Sub- und Supertabellen ähnlich der strukturellen Objektorientierung. Attribute werden von Super- auf Subtabellen vererbt und können in

der Subtabelle verändert (überschrieben) werden. Allerdings ist nur die Einfachvererbung vorgesehen - keine Mehrfachvererbung (zur Modellierung siehe unter „Erweiterung des ER-Modells").

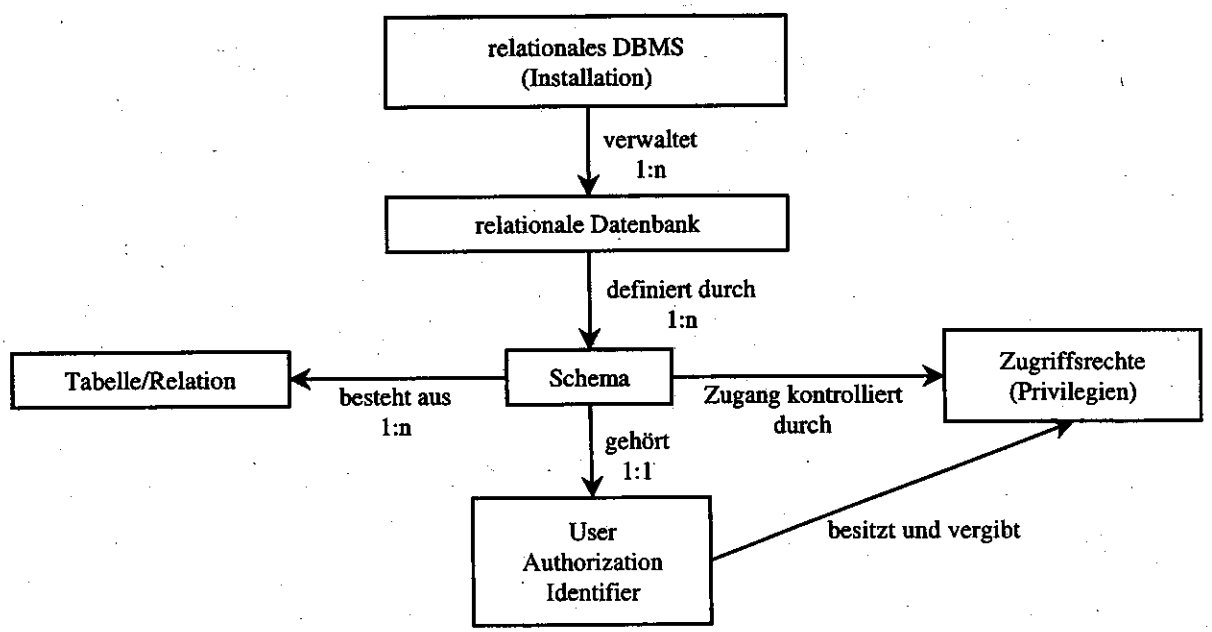
- **Trigger:** Trigger bewirken, daß SQL-Statements automatisch vor oder nach bestimmten Ereignissen ausgeführt werden. Solche Ereignisse sind INSERT oder DELETE von Datensätzen einer Tabelle oder das UPDATE bestimmter Spalten. Außerdem läßt sich einstellen, ob der Trigger vor oder nach (BEFORE / AFTER) dem bewußten Ereignis, einmal pro Operation oder für jeden betroffenen Datensatz (FOR EACH ROW) ausgelöst wird. Es können auch Assertions auf die gleiche Art getriggert werden: diese werden dann nur noch bei Eintritt des definierten Ereignisses überprüft.
- **Sicherungspunkte:** Innerhalb einer Transaktion kann der Benutzer Sicherungspunkte (Transaction Savepoints) definieren. Diese Sicherungspunkte ermöglichen dem Benutzer dann ein partielles Rollback einer Transaktion über den Befehl ROLLBACK TO <savepoint>. Alle Änderungen der Transaktion vor dem Sicherungspunkt bleiben erhalten, alle nach dem Sicherungspunkt werden zurückgesetzt

5.6. Der Systemkatalog

Regeln von Codd, die jedes DBMS erfüllen sollte:

- Der Systemkatalog muß aus ganz normalen Tabellen bestehen
  - In DB2: SYSTABLES = name + columns (Anz. der Spalten) + create (wer hat sie erstellt) + ...
    In dieser Tabelle wird für jede Tabelle der Datenbank ein Datensatz abgelegt.
  - SYSCOLOMNS = name + Tbname (Name der zugeh. Tabelle) + coltyp (Datentyp) + ...
    Fremd Tbname Ref Tabelle
    In dieser Tabelle wird für jedes Feld (Spalten aller Tabellen) ein Datensatz abgelegt.
- Die Tabellen des Systemkataloges können mit normalen SQL-Statements gelesen werden

Aufbau von Datenbanken und Zugriffsrechten:



6. Weitere Datenmodelle

Jedem der Datenbankmanagementsysteme liegt ein spezielles Datenmodell zugrunde. Neben den gängigen „relationalen" Datenbanken werden sogenannte „objektorientierte" und „objektrelationale" immer populärer. Darüber hinaus treten „multidimensionale" gegen „relationale" „OLAP"-Datenbanken für „Data Warehouses" an. Hier soll ein grundsätzlicher Überblick (Einordnung) ohne Anspruch auf Vollständigkeit geschaffen werden.

Die klassischen Datenbankanwendungen verwenden überwiegend relativ einfache Datentypen wie Zahlen und Buchstaben oder kurze Texte. Um auch komplexere Daten wie lange Texte, Grafiken und Musikstücke aufnehmen zu können, haben viele Hersteller von relationalen DBMS den Datentyp BLOB („binary large object") eingeführt. BLOBs sind großvolumige Binärdatenobjekte, die in einem relationalen DBMS auf spezielle Weise gespeichert werden, im relationalen Modell jedoch gar nicht vorkommen und in einer SQL-Anfrage deswegen auch nicht angesprochen werden können.

Aufgrund des Internet-Multimedia-Booms arbeiten die großen Hersteller relationaler Datenbanken mittlerweile an Erweiterungen für ihre Produkte, die für die Verwaltung von HTML-Dokumenten und anderen komplexen Datentypen spezielle Funktionen anbieten. Jedoch sind diese Erweiterungen proprietär, und ihre systematische Integration in die Datenmanipulations- und Abfragesprache SQL ist nicht gewährleistet.

Neuartige Anwendungen wie Dokumentenmanagement und Multimedia-Informationssysteme erfordern jedoch eine einheitliche und vollständig integrierte Unterstützung komplexer Datentypen. Vor allem wäre es sehr wünschenswert, in einer Anfrage auf komplexe Objekte wie Landkarten oder Dokumente auf dieselbe Weise Bezug nehmen zu können wie auf andere Attribute. Diese Möglichkeit überfordert die „herkömmlichen" relationalen Systeme. Darüber hinaus haben sich durch die Erfahrungen mit dem relationalen Modell auch dessen grundsätzliche Schwächen herauskristallisiert.

Hauptpunkte der Kritik sind:

- das Problem großvolumiger Datenobjekte: In modernen Informationssystemen wird es immer wichtiger, auch Text- oder Multimedia-Dokumente zu erfassen und zu manipulieren. Der zu diesem Zweck in heutigen relationalen Systemen eingeführte Datentyp „BLOB" ist nur eine halbherzige Lösung und wird durch SQL nicht unterstützt.
- das Problem der Vererbung: Im konzeptionellen Datenbankentwurf erhält man oft hierarchische Subsumptionsbeziehungen zwischen Entitätstypen: Ein Manager ist etwa ein Mitarbeiter mit besonderen Attributen (eigene Sekretärin, Firmenwagen). Diese können im relationalen Modell nicht berücksichtigt werden.
- Diskrepanz zwischen SQL und den normalerweise benutzten imperativen Programmiersprachen (Anweisungs-/Blockorientiert - Programm/Unterprogramm - strukturiert) wie Cobol oder C. Während SQL eine Antwortmenge als Ergebnis einer Abfrage liefert und auf der Mengenverarbeitung basiert, sind Programmiersprachen wie C nur in der Lage, einzelne Zeilen der Ergebnistabelle zu verarbeiten. Dieser „Impedance Mismatch" erfordert somit ein umständliches Hantieren bei der Weiterverarbeitung von Abfrageergebnissen.

6.1. Objektorientierte Datenbanken

Neben der Kritik am relationalen Modell besitzt das Objektdatenbankparadigma einen zweiten Vater: die objektorientierte (OO-)Programmierung. Die ersten Objektdatenbanksysteme orientierten sich mehr an OO-Programmiersprachen als an Prinzipien von Datenbanksystemen. Sie bieten Verwaltungsfunktionen für beständige (persistente) Objektkollektionen auf der Ebene des Dateisystems und ergänzen damit die OO-

Programmiersysteme, die Objekte nur im Arbeitsspeicher, also nicht persistent, verwalten können. Man könnte sie daher auch treffender als „persistente Objektmanagementsysteme" bezeichnen.

Objektdatenbanken repräsentieren Informationen in der Form von Objekten als Instanzen einer Klasse. Wie bei hierarchischen und Netzwerkdatenbanken muß man als Benutzer eines Objekt-DBMS (ODBMS) wissen, wie man durch eine Datenbank navigiert, um Informationen wiederzugewinnen. Es gibt oft keine Schemaverwaltung (also auch keine benutzerspezifischen Zugriffsrechte), keine deklarative Abfragesprache (also auch keine Abfrageoptimierung) und keine Integritätsbedingungen. Viele dieser im Rahmen des relationalen Modells entwickelten Datenbankkonzepte sind aber so fundamental, daß man sie heute zu den Grundfunktionen eines DBMS zählt, welche von jedem neuen ernstzunehmenden Datenbankparadigma unterstützt werden müssen.

Von Anfang an war die mangelnde konzeptionelle Klarheit und das Fehlen von Standards ein großes Problem für Objekt-DBMS. In der Datenbankforschung gab es daher mehrere Versuche, die Grundfunktionen eines Objekt-DBMS zu definieren. Das „Objektorientierte-Datenbanksystem-Manifest" von 1989 rechnet dazu:

- Abstrakte Datentypen erlauben die Kapselung von Objekten und auf sie zugeschnittener Operationen („Methoden"). Sowohl die Objektdaten als auch die speziell für sie definierten Operationen werden in der Datenbank verwaltet.
- Objektidentität: Objekte haben eine von ihrem Zustand unabhängige Identität, das heißt, es kann unterschiedliche Objekte geben, die in allen Eigenschaften übereinstimmen.
- Klassen fassen alle gleichartigen Objekte zu einer Kollektion zusammen.
- Vererbung: Klassen vererben ihre Attribute und Operationen an alle Unterklassen.

Um einheitliche Standards zu schaffen, haben sich 1991 einige Hersteller von Objekt-DBMS zur **Object Data Management Group (ODMG)** zusammengeschlossen (genaueres unter http://www.odmg.org). Ihre ODMG-93-Spezifikation umfaßt die Objektdefinitionssprache ODL, die deklarative Objektabfragesprache OQL und Vorgaben zur Objektmanipulation in den OO-Programmiersprachen Smalltalk und C++. Im Gegensatz zu SQL2 gibt es in der ODMG-93-Spezifikation weder Integritätsbedingungen noch Trigger. Es ist jedoch erklärtes Ziel der ODMG, daß OQL kompatibel mit SQL2-Anfragen werden soll, das heißt, daß SELECT-Ausdrücke in OQL die gleiche Semantik wie in SQL2 haben sollen.

Auch wenn Objekt-DBMS der ersten Generation nicht SQL-kompatibel sind und nicht alle Funktionen unterstützen, die bei ihren relationalen Gegenstücken zum Grundinventar gehören, haben sie durchaus ihre Daseinsberechtigung: Insbesondere für Aufgaben, die mehr Objektflexibilität fordern, als sie RDBMS bieten können, empfehlen sich die Objektverwalter. Solche Einsatzgebiete können zum Beispiel Dokumentenverwaltungssysteme, Netzmanagementsysteme und geografische Informationssysteme sein. Gegen eine weitere Verbreitung spricht allerdings die im Vergleich zu SQL-Datenbanken oft größere Kompliziertheit und geringere Benutzerfreundlichkeit von Objekt-DBMS der ersten Generation.

6.2. *Objektrelationale Datenbanken*

Objektrelationale DBMS erweitern das relationale Datenbankmodell um die Kernfunktionen des OO-Paradigmas. Das heißt vor allem, daß sie die Entkopplung von logischer und physikalischer Struktur und das Konzept einer deklarativen Abfragesprache beibehalten und darüber hinaus die klassische SQL-Funktionalität um objektorientierte Konzepte er-

gänzen. Die Kompatibilität zu SQL-Datenbanken soll die Akzeptanz der neuen Technologie gewährleisten.

Objektrelationale DBMS lösen die zuvor aufgeführten Probleme des relationalen Modells:

- das Problem großvolumiger Datenobjekte: „Large Objects" (LOBs) können mitsamt der nötigen Operationen (speziellen Methoden) definiert werden.
- das Problem der Vererbung: Neue Klassen können als Unterklassen von bereits definierten Klassen eingerichtet werden. Attribute und Operationen werden auf alle Unterklassen vererbt.

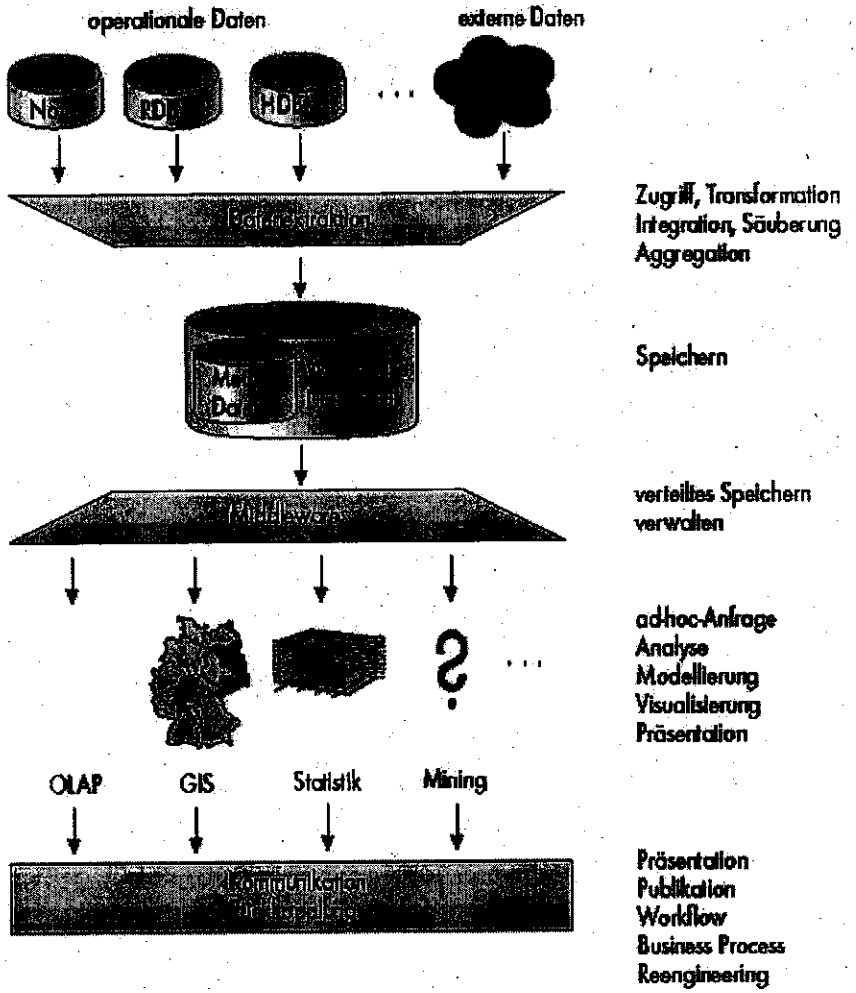
Bei der Überführung eines ER-Modells in ein objektrelationales Datenbankschema werden Entitätstypen als Klassen repräsentiert und können abgeleitet werden. Manager lassen sich etwa durch eine aus der Mitarbeiterklasse abgeleitete Unterklasse repräsentieren:

```
CREATE CLASS Manager (Managerattribute) SUBCLASS OF Mitarbeiter
```

Sowohl das SQL3-Standardisierungskomitee als auch die ODMG bemühen sich inzwischen um eine Harmonisierung von SQL3 und OQL. Idealerweise soll OQL eine Obermenge des Abfrageteils von SQL2 und SQL3 eine Obermenge von OQL werden. Dabei gibt es jedoch zahlreiche Probleme. So muß es in SQL3 möglich werden, als Abfrageresultat nicht nur eine Tabelle, sondern auch eine Kollektion von möglicherweise ungleichartigen Objekten zu erhalten.

6.3. Multidimensionale und OLAP-Datenbanken für „Data-Warehouses"

Als zentrales Datenlager sammelt ein Data-Warehouse in regelmäßigen Intervallen Informationen aus den operativen Systemen, konsolidiert sie, filtert Unwichtiges heraus, ordnet die Daten nach Themen und zeigt mit Hilfe von Analysewerkzeugen Relevantes an. Die vielseitigen Aufgaben eines Data Warehouse erledigt nicht etwa eine monolithische Anwendung, sondern ein komplexes System, bei dem etliche Anwendungen kooperieren.



Kernelement eines solchen Infolagers ist eine eigene zentrale Datenbank. Es liegt nicht nur an Ressourcenbeschränkungen, die die Verwendung der operativen Datenbanken als Datenmanager für Data-Warehouse-Anwendungen unmöglich macht. Der Zugriff auf die Informationen erfordert auch eine ganz andere Organisation.

So ist es die Aufgabe der operativen Datenbankmanagementsysteme, das Tagesgeschäft zu bedienen. Die Struktur der Datenbanken orientiert sich an den zugrundeliegenden Anwendungen beziehungsweise Geschäftsobjekten. Die Datenbankzugriffe spiegeln die wesentlichen betrieblichen Vorgänge wider. Insgesamt entspricht der Datenbestand also immer einem Schnappschuß des Unternehmens.

Typischerweise greift eine große Zahl von Benutzern auf die operativen Datenbanken zu, die sehr häufig kurze Transaktionen durchführen. Typ und Frequenz dieser Transaktionen lassen sich gut prognostizieren. Datenbanktechnisch bestehen sie aus relativ einfachen Operationen, die jedoch viele Änderungen im Datenbestand vornehmen.

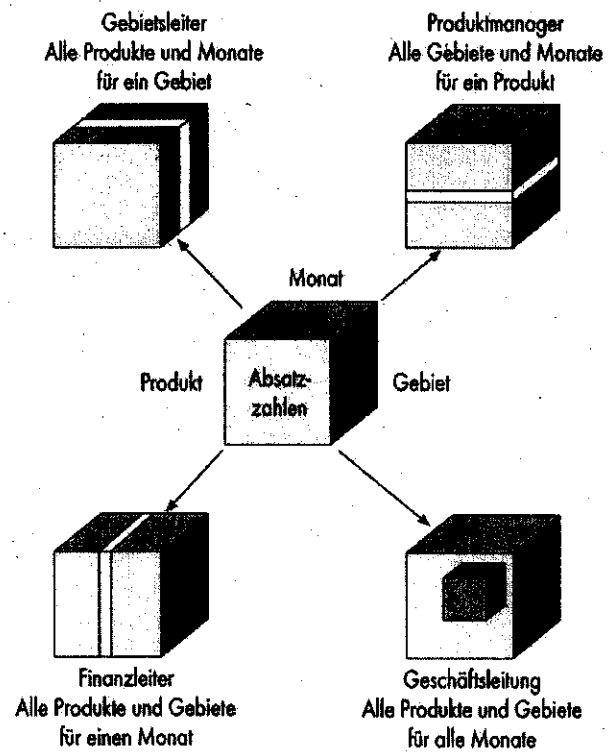
Da operative Datenbanken normalerweise im Online-Betrieb gefahren werden, was sehr hohe Anforderungen an ihre Verfügbarkeit stellt, spricht man auch vom `Online Transaction Processing (OLTP)´.

Als unternehmensweites Instrument soll ein Data Warehouse sämtliche Entscheider flexibel unterstützen. Diese sollen auf möglichst intuitive Weise zu ihren Informationen finden können. Über `Produkt´, `Zeit´ und `Gebiet´ hinaus existieren viele **Dimensionen** im `Vektorraum Unternehmen´, insbesondere `Zeit´, aber auch `Kunden´, `Filialen´, `Produktionsstandorte´, etc.

Darüber hinaus muß sich der Maßstab jeder Dimension skalieren lassen. So will der Produktmanager nicht an Monate als Betrachtungszeiträume gebunden sein, sondern bei Bedarf auch die kumulierten Werte pro Jahr betrachten können oder den Umsatzverlauf von Woche zu Woche unter die Lupe nehmen. Entsprechend flexibel sollten auch die anderen Dimensionen einteilbar sein.

operativ	**entscheidungsunterstützend**
detaillierte Daten	verdichtete, aufbereitete Daten
aktuelle Daten	historische, aktuelle und projizierte Daten
ständig aktualisierter Bestand	nur sporadisch aktualisierter Bestand
Systemlast vorhersehbar	Systemlast kann stark schwanken
transaktionsgesteuerte Verarbeitung	analysegesteuerte Verarbeitung
hohe Anforderung an Verfügbarkeit	geringe Anforderung an Verfügbarkeit
statische Struktur	flexible Struktur
kleine Datenmengen pro Prozeß	große Datenmengen pro Prozeß

Dies erfordert eine andere Sichtweise auf die Daten, ein anderes Datenmodell, als es relationalen Datenbanken zugrunde liegt. Das sogenannte multidimensionale Datenmodell wird der Vielschichtigkeit gerecht. Es repräsentiert die Unternehmensdaten in einem mehrdimensionalen Datenraum mit einer Dimension pro relevantem Unternehmensmerkmal (von denen man sich der besseren Anschauung halber nur drei vorstelle).



Da die flexible analytische Arbeit mit dem multidimensionalen Datenmodell eines Data Warehouse auch im Online-Betrieb ablaufen soll spricht man auch von Online Analytical Processing (OLAP).

OLAP-Server benötigen auf den ersten Blick keine hohe Rechenpower. So bedienen Data Warehouses im Vergleich zu OLTP-Systemen weniger gleichzeitige Benutzer - schließlich beschränkt sich deren Klientel auf administratives Personal. Manager führen auch viel seltener Analysen durch, als etwa eine Sachbearbeiterin Bestellungen eingibt. Darüber hinaus

ändern die Anwender von Data Warehouses nie den Datenbestand, sondern fragen nur vorhandene Daten ab.

Daß eine Data-Warehouse-Datenbank trotzdem noch eine Nummer leistungsfähiger sein sollte als die OLTP-Systeme, hat andere Gründe: Zum einen fällt durch die zusätzliche historische Dimension sehr viel mehr Material an, als für den täglichen Betrieb vorgehalten werden muß; ein Data Warehouse umfaßt nicht selten um den Faktor 100 bis 1000 mal mehr Daten als die operativen Systeme.

Ein weiterer Grund besteht in der großen Variabilität analytischer Anfragen. Die Queries der Anwender lassen sich schlecht vorhersagen. Außerdem sind die Suchaufträge von Warehouse-Benutzern in der Regel komplexer als OLTP-Transaktionen.

Im Gegensatz zu den auf ihren Fachbereich spezialisierten operativen Datenbanken soll ein Data Warehouse eine globale Sicht ermöglichen. Hierzu muß es so viel Informationen anbieten können wie möglich. Neben den aktuellen Daten beinhaltet es daher auch die alten Werte. Neue Informationen integriert es periodisch. Hierbei sammelt das Informationslager die Angaben sämtlicher verfügbaren Quellen.

Zu diesen kann alles gehören, was die Geschichte der EDV an Datenverwaltern hervorgebracht hat. Neben den modernen relationalen Datenbankmanagementsystemen gehören hierzu etwa hierarchische Datenbanken, Dokumente, die irgenwo im Unternehmensnetz verstreut sind, Workflow-orientierte Systeme wie Lotus Notes, Standardanwendungen, zum Beispiel SAP R/3, moderne objektorientiere Datenbanken und externe Quellen, etwa Wirtschaftsdienste im Internet.

Die Programme, die die Daten für ein Data Warehouse einsammeln, müssen also viele Sprachen sprechen: Nicht nur die diversen Datenbankidiome sind sie gezwungen zu beherrschen, sondern auch die unterschiedlichen Netzwerkprotokolle der Betriebssysteme und die Datenstrukturen der einzelnen Anwendungen. Aus diesen extrahieren sie die einzelnen Daten heraus, validieren sie (`Hat der Speisequark in der Verkaufsdatenbank dieselbe ID wie in der Buchhaltung?´) und korrigieren mitunter fehlerhafte Datensätze.

Integrator-Programme sammeln die Daten aus den verschiedenen Quellen zu einer gemeinsamen Sicht. In einem weiteren Schritt werden sie auf die benötigte Informationsmenge reduziert. Besonders intelligente Datensammler beziehungsweise -mischer gehen noch einen Schritt weiter. Wenn sie interessante Werte finden (Absatzzahlen besonders hoch oder niedrig), gehen sie selbständig auf die Suche nach den Ursachen und liefern diese gleich mit.

Als letzter Arbeitsgang werden sämtliche gefundenen Informationen in die eigentliche Warehouse-Datenbank geladen. Der gesamte Vorgang der Datengewinnung läuft dabei nach einem genau vorgegebenen Zeitplan ab.

Das Zeitschema gibt die Metadatenbank vor. Sie ist die Schnittstelle zwischen den Infosammlern und der zentralen Datenbank. In ihr verwaltet das Gesamtsystem außerdem Informationen über Quellen, die anzuwendenden Filter und Aggregationsverfahren.

6.3.1. Kategorien der Werkzeuge der Endanwender

Report- und Abfragewerkzeuge gehören zu den gängigsten Hilfsmitteln, insbesondere für den Zugriff auf relationale Datenbanken. Die weite Verbreitung von Standards für diese Datenmanager (SQL, ODBC) hat hierzu sicherlich beigetragen

Statistikprogramme: Berechnung von arithmetischem Mittel, Standardabweichung, etc.

Zu den Endbenutzer-*OLAP*-Werkzeugen zählen zum einen Abfrage-Tools für die multidimensionalen Datenbank-Server, zum anderen Desktop-MDBMS mit integrierten Report- und Query-Möglichkeiten. Vom Funktionsumfang ähneln OLAP-Programme Tabellenkalkulationen, bieten aber intuitivere Möglichkeiten, durch die Daten zu navigieren.

Data-Mining-Werkzeuge sind darauf spezialisiert, noch unbekannte Zusammenhänge innerhalb von Unternehmensdaten zu finden. Im Gegensatz zu den klassischen Abfragewerkzeugen muß ihr Anwender nicht von vornherein wissen, wonach er sucht. Vielmehr führen sie den Anwender von sich aus zu den (vermeintlich) interessanten Informationen.

Sie finden abhängige Daten, Abweichler, klassifizieren und machen Vorhersagen. Hiefür benutzen sie Techniken aus den Bereichen Statistik und maschinelles Lernen, also etwa neuronale Netze, genetische Algorithmen.

Document-Retrieval-Systeme bändigen die Flut elektronischer Dokumente. Denn außer in den für Informationsverwaltung spezialisierten Datenbanken fallen täglich unzählige mit Textverarbeitungen, Tabellenkalkulationen etc. erstellte Dateien an.

Werden diese nicht mit einem leistungsfähigen Werkzeug verwaltet, gehen sie über kurz oder lang im Irrgarten `Unternehmensnetz´ verloren. Document-Retrieval-Systeme speichern sämtliche Dokumente eines Unternehmens daher in einer eigenen, zentralen Datenbank. Dort kann der Anwender etwa nach allen Dateien suchen, die einen bestimmten Schlüsselbegriff enthalten, von einem bestimmten Dateityp sind, in einem vorgegebenen Zeitraum oder von einem bestimmten Mitarbeiter erstellt wurden.

Aktive Informationsfilter durchsuchen selbsttätig die Datenbestände nach Informationen. Dazu spezifiziert der Anwender bestimmte Themen oder Ereignisse, über die er in der Folgezeit informiert werden möchte - etwa daß die Verkaufserträge einer Region unter einem bestimmten Wert liegen.

Die Filterprogramme schicken sogenannte Agenten auf den Weg. Diese Anwendungen durchkämmen in regelmäßigen Zeitabständen die Datenbestände nach den gesuchten Objekten. Ihre Ergebnisse liefern sie regelmäßig aus; auf Wunsch schlagen sie auch sofort Alarm, etwa wenn Gefahr droht.

Prozeßmodellierung erlaubt dem Manager, am Rechner verschiedene Szenarien durchzuspielen, ohne das reale Unternehmen zu gefährden. So kann er sich etwa am Rechner eine Fertigungsstraße nachbauen und umstricken, um zu versuchen, ihren Ausstoß zu optimieren.

Geographische Informationssysteme (kurz GIS) stellen Daten in ihrer geographischen Dimension dar. In sogenannten thematischen Karten färben diese etwa in Abhängigkeit vom Datenmaterial bestimmte Bereiche unterschiedlich ein. Damit läßt sich etwa der Umsatz eines Produkts nach Postleitzahlgebieten sehr anschaulich aufbereiten.

Führungsinformationssysteme: Viele der bislang besprochenen Werkzeuge nötigen dem Anwender Eingewöhnungszeit ab. Der vielbeschäftigte Chef wird aber meist nicht die Zeit haben, um sich mit der Funktionsweise der komplizierten Werkzeuge vertraut zu machen. Er wird es im Zweifelsfall an seinen Assistenten delegieren, eine statistische oder geographische Analyse durchzuführen - und wieder von einem Zuarbeiter abhängig sein. Führungsinformationssysteme bieten extrem einfache Methoden, Informationen abzufragen oder zu erzeugen. So können über vorgefertigte Analyseschablonen bestimmte Leistungswerte abgefragt werden.

Abteilungsspezifische Tools sind für die Anforderungen bestimmter Unternehmensteile zugeschnitten

Industriespezifische Tools orientieren sich an den Bedürfnissen bestimmter Wirtschaftszweige.

6.3.2. Fazit

Data Warehouses gibt es nicht als schlüsselfertige Standardlösung zu kaufen. Konzeption und Erstellung eines Datenkaufhauses kann sich - abhängig von der Unternehmensgröße und den eingesetzten Systemen - zu einem langwierigen und teuren Prozeß auswachsen.

Dabei sind Fragen wie die folgenden zu klären:

- Welche Daten benötigen die Anwender?
- Welche Bedeutung haben die Daten für welche Zwecke?
- In welcher Tiefe und Breite müssen die Informationen vorliegen?
- In welchen Formaten und auf welchen Quellen sind die Daten verfügbar?
- Wie greift man auf diese Daten zu?