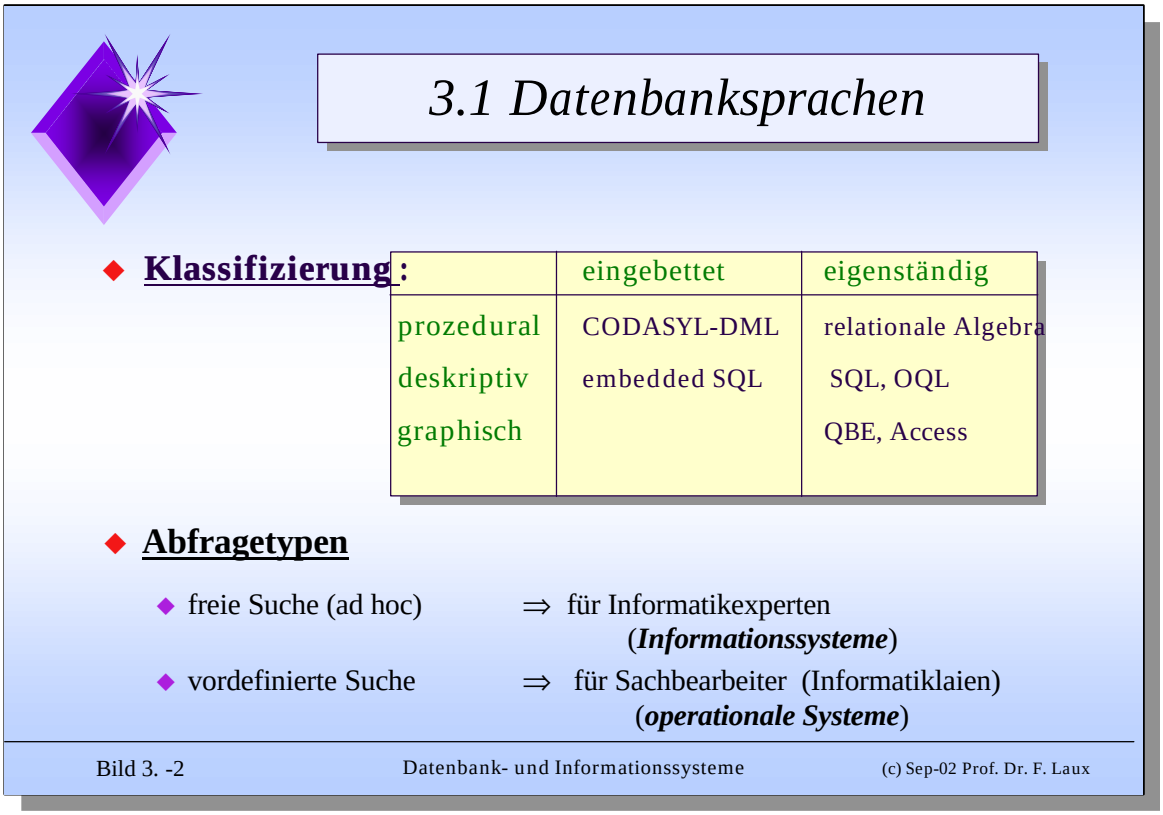
In diesem Kapitel wenden wir uns einem anderen Datenbankaspekt zu. Aus Sicht des ANSI/SPARC Modells bewegen wir uns von der konzeptionellen Ebene des vorherigen Kapitels zur internen Ebene, zu den Funktionen einer Datenbank.

Wir unterteilen die Datenbankfunktionen in vier Kategorien.

Die ersten drei werden in der Literatur intensiv behandelt, da sie für die Funktionalität einer Datenbank essentiell sind. Der letzte Punkt wird häufig als Bestandteil der Softwareentwicklung und nicht als Datenbankfunktion behandelt. Dies ist allerdings nicht ganz einsichtig, da diese Funktionalität vom Datenbankmanagementsystem erbracht werden muß und einen wesentlichen Einfluß auf die Brauchbarkeit (Usability) einer Datenbank hat. Die Unterstützung ist abhängig vom Datenbanksystem und Interface, kann also nur an konkreten Produkten detailliert untersucht werden (vgl. Stürner, Oracle System Tuning). Aus diesem Grund werden mit dem Datenbanksystem auch Modellierungs- und Tuningwerkzeuge vom Datenbankhersteller angeboten (z.B. GemBuilder, Oracle CASE, Access Wizards, etc.) Wir werden diese Produkte im Praktikum und der Softwareentwicklung kennenlernen; im Rahmen der Vorlesung wird nur ihre Funktionalität skizziert.

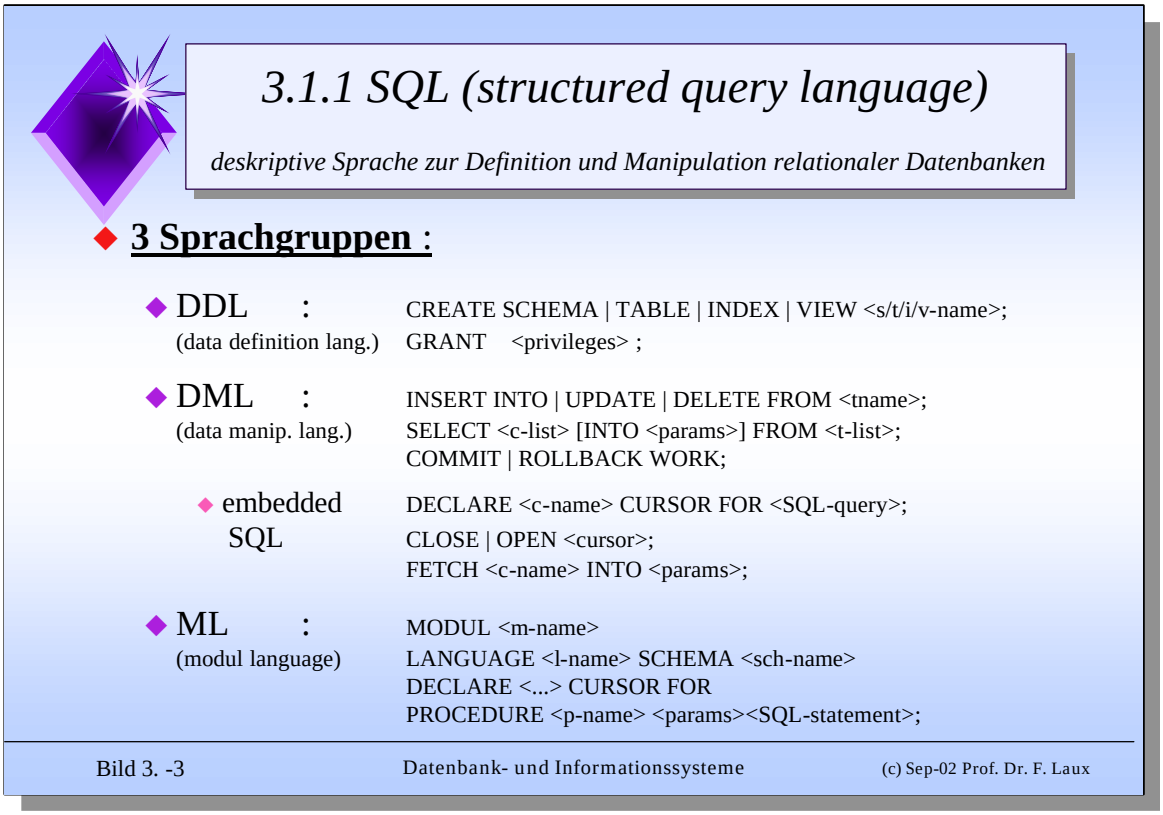
Die grundlegenden Funktionen einer Datenbank werden auf den folgenden Seiten genauer vorgestellt.



Es sind eine ganze Reihe von **Klassifikationen** für die Datenmanipulation möglich. Wir zeigen hier nur zwei in der Literatur weitverbreitete Einteilungen für Datenbanksprachen nach dem Grad der Abstraktion bzw. der Autonomie der Sprache.

Bei den **Abfragetypen** werden zwei Gruppen unterschieden:

- *Frei formulierte Abfragen* erfordern die genaue Kenntnis der Datenbankstruktur. Moderne Systeme bieten dazu Hilfestellung, indem sie Informationen des Data Dictionaries dem Benutzer in Form von Auswahllisten zur Verfügung stellen.
- *Vordefinierte Abfragen* hingegen können ohne Vorkenntnisse verwendet werden.



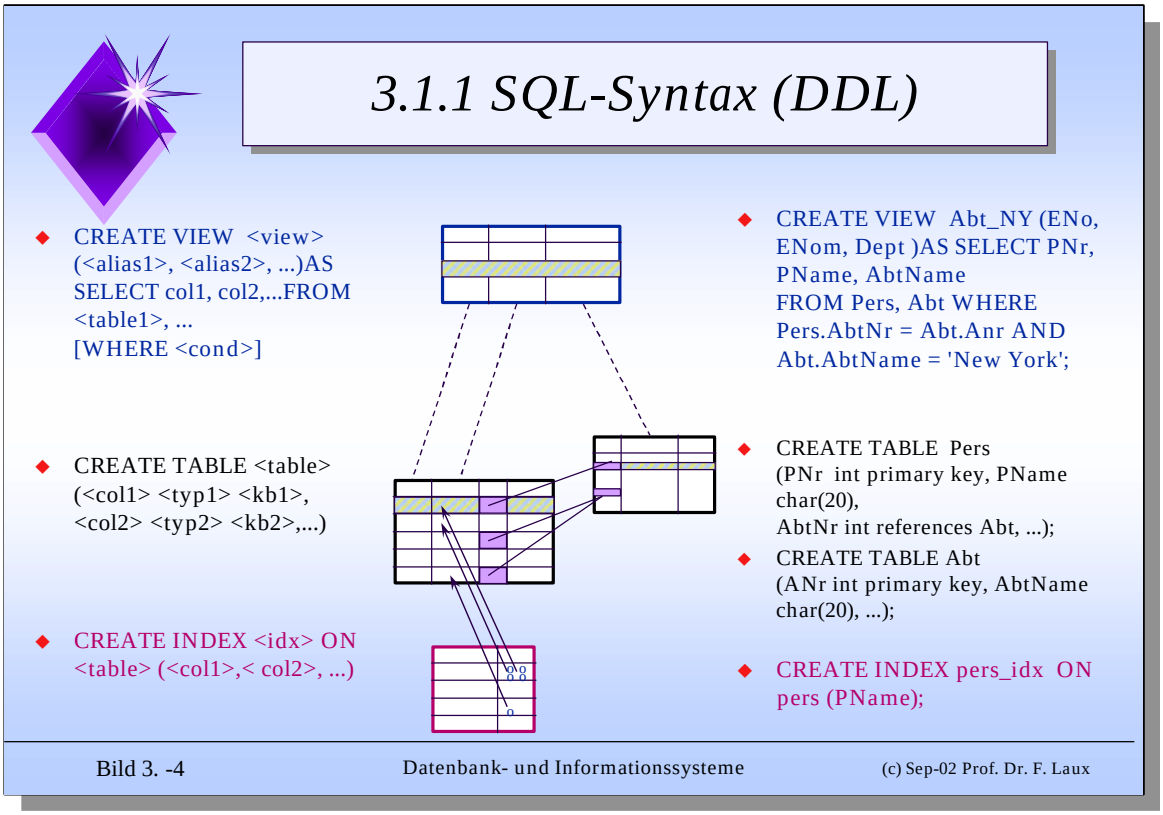
SQL (Structured Query Language, im Englischen als "Siekl" ausgesprochen) ist mit Abstand die bekannteste und wichtigste Sprache für relationale Datenbanken. Für objektorientierte Datenbanken ist OQL (Object Query Language) gerade dabei, sich als Standard der OMG (Object Management Group) zu etablieren (siehe URL: http://www.odmg.org/, Stand 1998).

SQL gliedert sich in ***drei Sprachgruppen***:

• Die ***Datendefinitionssprache DDL*** dient zur Definition von Datenbankobjekten (Tabellen, Indizes, Benutzersichten und -rechte.

• Zur ***Datenmanipulation (DML)*** (Einfügen, Ändern, Löschen und Anzeigen) werden SQL-Befehle direkt mit Hilfe eines Interpreters oder als *embedded SQL* aus einer Programmiersprache (host language) heraus ausgeführt. Zur Anpassung der SQL Ergebnismengen (Datentyp Tabelle) sind Zeiger (cursors) zu definieren, die auf einen Datensatz zeigen, der dann mit Hilfe eines FETCH-Befehls in die Variablen der Host-Language übertragen werden kann.

• Weniger bekannt ist, daß in SQL auch Prozeduren (***stored procedures***) und Module aus SQL-Statements geformt werden können (***Modulsprache***, ***ML,*** bei SQL als ***Persistent Stored Modules***, ***PSM***, bekannt). Diese können in der Datenbank gespeichert werden und über einen Modul- oder Prozedurnamen aufgerufen und ausgeführt werden.



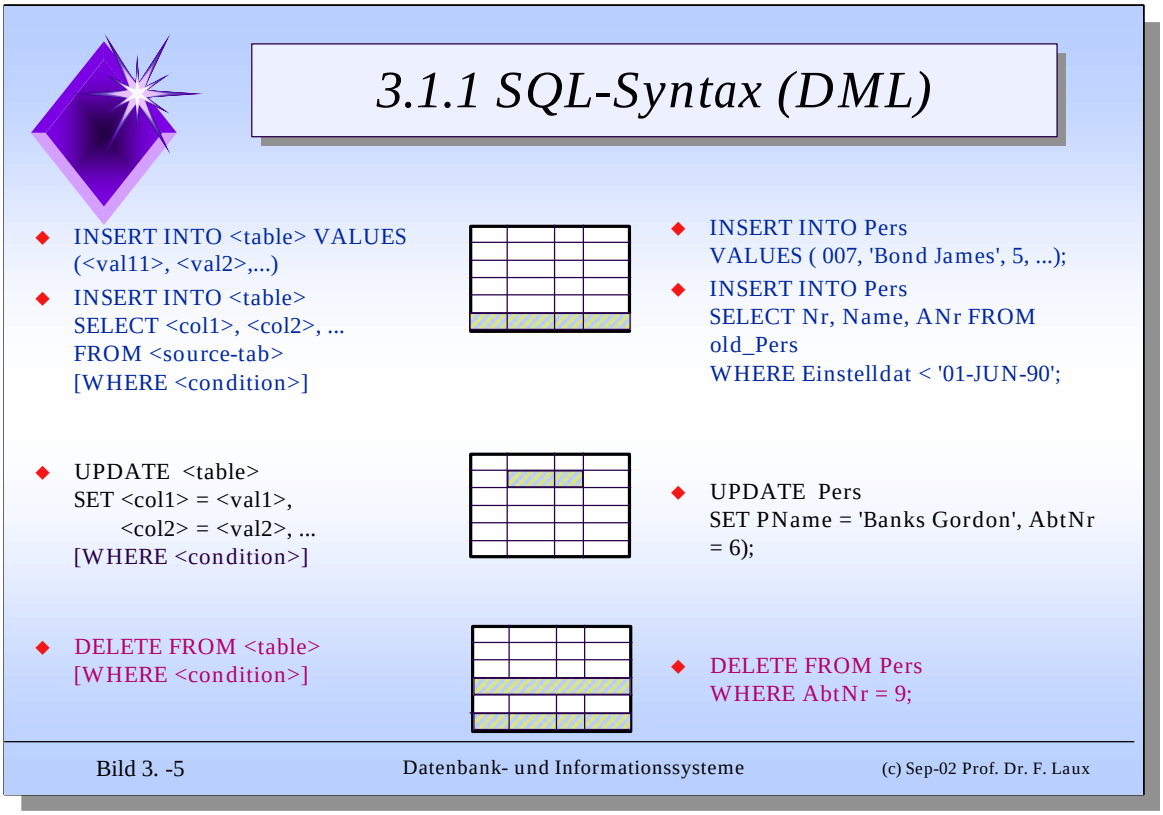
Auf dieser und den folgenden Folien wird links die Syntax vorgestellt und rechts dazu jeweils ein Beispiel angegeben.

Mit dem Befehl CREATE TABLE wird eine Tabelle definiert und angelegt. Für jede Spalte wird ein Attributname <colx>, ein Datentyp <typx> und optional eine Konsistenzbedingung <kbx> angegeben.

Benutzersichten (virtuelle Tabellen) werden mit dem Schlüsselwort VIEW deklariert. Sie werden nur scheinbar (virtuell) erzeugt; in Wirklichkeit wird auf einem Teil der vorhandenen Tabellen gearbeitet. Die Spalten der Views erhalten nur Namen <alias> aber keine Datentypen, da diese durch die Spalten <colx> einer realen Tabelle <table1> schon festgelegt sind.

Jedes beliebige Attribut <colx> einer Tabelle <table> kann mit dem Befehl CREATE INDEX jederzeit mit einem Index <idx> versehen werden. Indizes beeinflussen die logische Struktur einer Datenbank nicht, sie haben nur Einfluß auf die Arbeitsgeschwindigkeit der Datenbank. Je mehr Indizes definiert werden, desto länger dauert eine Einfügeoperation. Abfragen auf ein indiziertes Attribut werden jedoch wesentlich schneller durchgeführt.

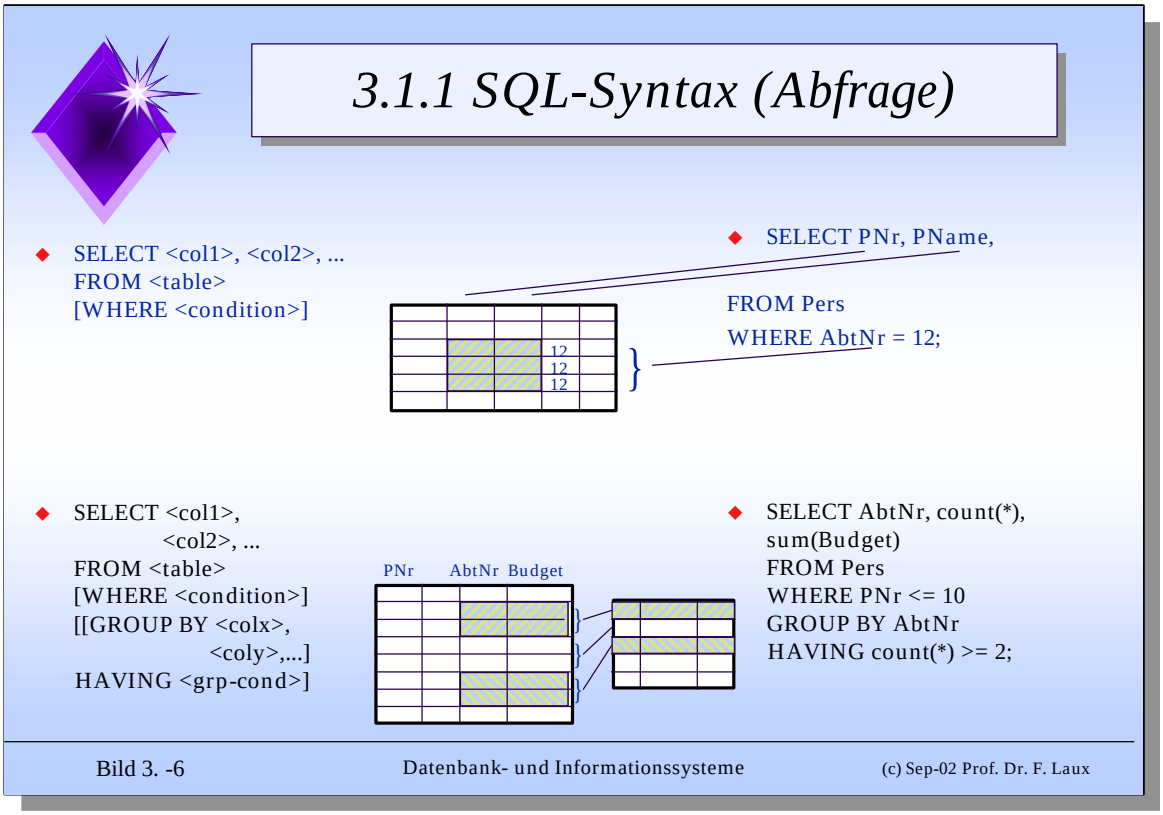
CREATE VIEW ist nicht Teil des SQL-Standards, da dieser Befehl als nicht konzeptuell sondern auf der interne Realisierungsaspekt (z.B. als Zugriffshilfe) betrachtet wird. Allerdings wird er von fast allen Produkten unterstützt, und das X/Open Gremium hat daher diesen Befehl in seine Empfehlung aufgenommen



Mit INSERT INTO wird ein Datensatz mit den Werten <valx> in die Tabelle <table> eingetragen. Wenn einige Werte frei bleiben sollen (NULL-Values), dann ist NULL oder kein Wert (,,) anzugeben. Es ist auch möglich, mit einem Befehl mehrere Sätze einzutragen, wobei die Datenquelle wiederum eine Tabelle <source-tab> sein muß.

Änderungs- und Löschoperationen wirken auf alle Datensätze, welche die WHERE-Bedingung erfüllen. Fehlt die Klausel, dann werden alle Sätze einer Tabelle bearbeitet.

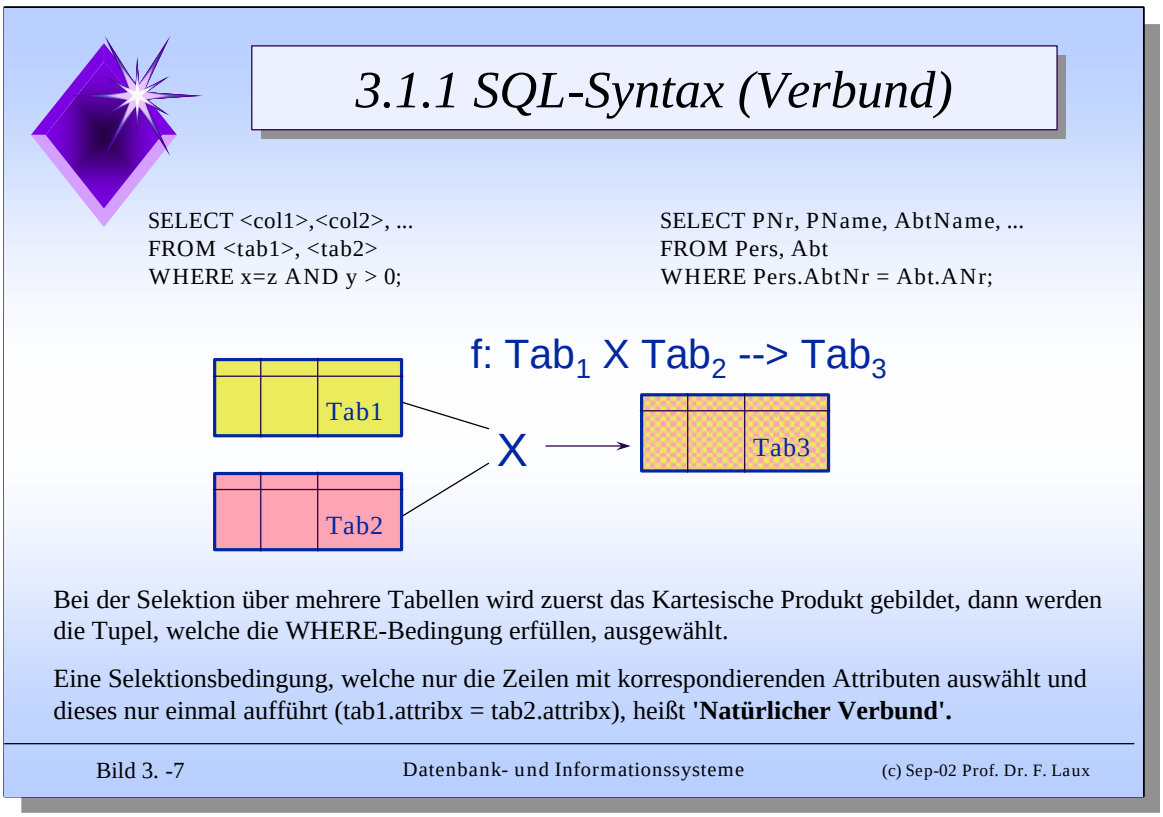
Eine WHERE-Bedingung wird als logischer Ausdruck <condition> formuliert, der für jeden Datensatz zu TRUE oder FALSE evaluiert wird. Ist die Bedingung erfüllt (TRUE), wird der Datensatz bearbeitet, andernfalls wird er nicht bearbeitet.



Die Selektion ist die komplexeste SQL-Phrase. Zunächst wird nur eine einfache Form vorgestellt.

Mit SELECT <col> werden die Spalten ausgewählt,
mit FROM <table> wird die Tabelle festgelegt und mit
WHERE werden die Sätze selektiert, welche ausgegeben werden sollen.

Mit dem SELECT-Befehl lassen sich auch Gruppierungen durchführen. Dazu wird an eine einfache Selektion ein GROUP BY <col> angefügt, um die Spalte anzugeben, nach der gruppiert werden soll. Wenn für eine Gruppierung ein Selektionskriterium gelten soll, so ist dies durch HAVING <grp-cond> anzugeben.



Die Mächtigkeit von SQL resultiert aus der ***Verbundoperation (Join)*** zweier (oder mehrerer) Tabellen. Dadurch wird es möglich, die Sätze verschiedener Tabellen miteinander zu verknüpfen und dadurch neue Ergebnistabellen zu erstellen. Diese Ergebnisse stellen immer Teilmengen des *kartesischen Produkts* zweier Tabellen dar.

Mit der Join-Operation werden Beziehungen zwischen Tabellen (temporär) erstellt. Eine spezielle Verknüpfung ist der ***natürliche Verbund***. Als Bedingung für das Zusammenfügen der Tabellen wird von allen gemeinsamen Attributen die gleichen Werte verlangt, d.h. tab1.attribx = tab2.attribx. Im Ergebnis wird nur eine der Join-Spalten aufgeführt, da ja ihre Werte identisch sind.

Besteht zwischen den beiden Tabellen eine Fremdschlüsselbeziehung, so wird diese durch den natürlichen Verbund realisiert.

Seit **SQL:92** gibt es eine neue Syntax für die Verbundoperationen:

SELECT <select-list> FROM <joined-table>

<joined-table> ::= <cross-join> | <qualified-join> | **(**<joined-table>**)**

<cross-join> ::= <table> CROSS JOIN <table>

<qualified-join> ::= <table> [NATURAL] [<join-type>] JOIN <table> [<join-spec>]

<join-type> ::= INNER | UNION | {LEFT | RIGHT | FULL} [OUTER]

<join-spec> ::= ON <search-condition> | USING (<join-column-list>)

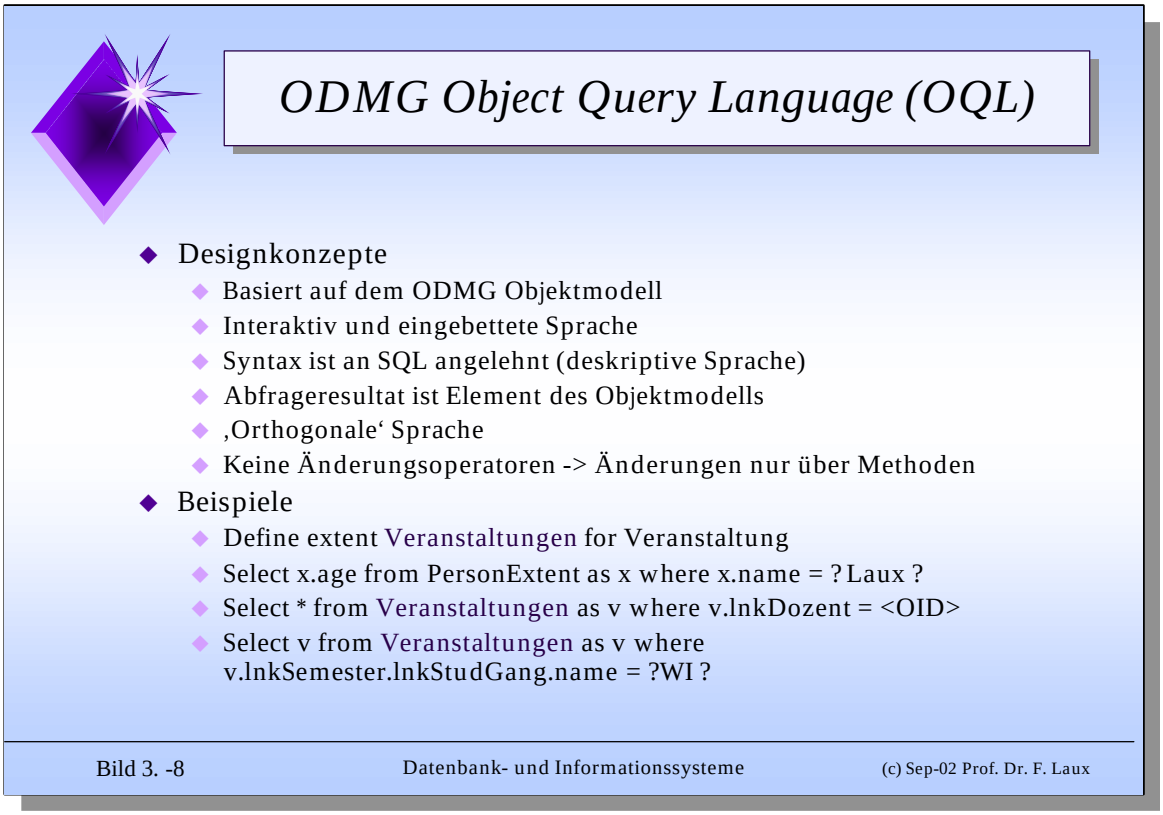
Beispiele:

select * from bestellkopf natural join bestellpos; -- listet alle Bestellungen mit Positionen

select * from bestellkopf join bestellpos using (bestellnr); -- Ergebnis wie oben

select * from bestellkopf left outer join bestellpos; -- listet auch Bestellköpfe ohne Positionen

select * from bestellkopf join kunden on (bestellkopf.kNr = kunden.nr)

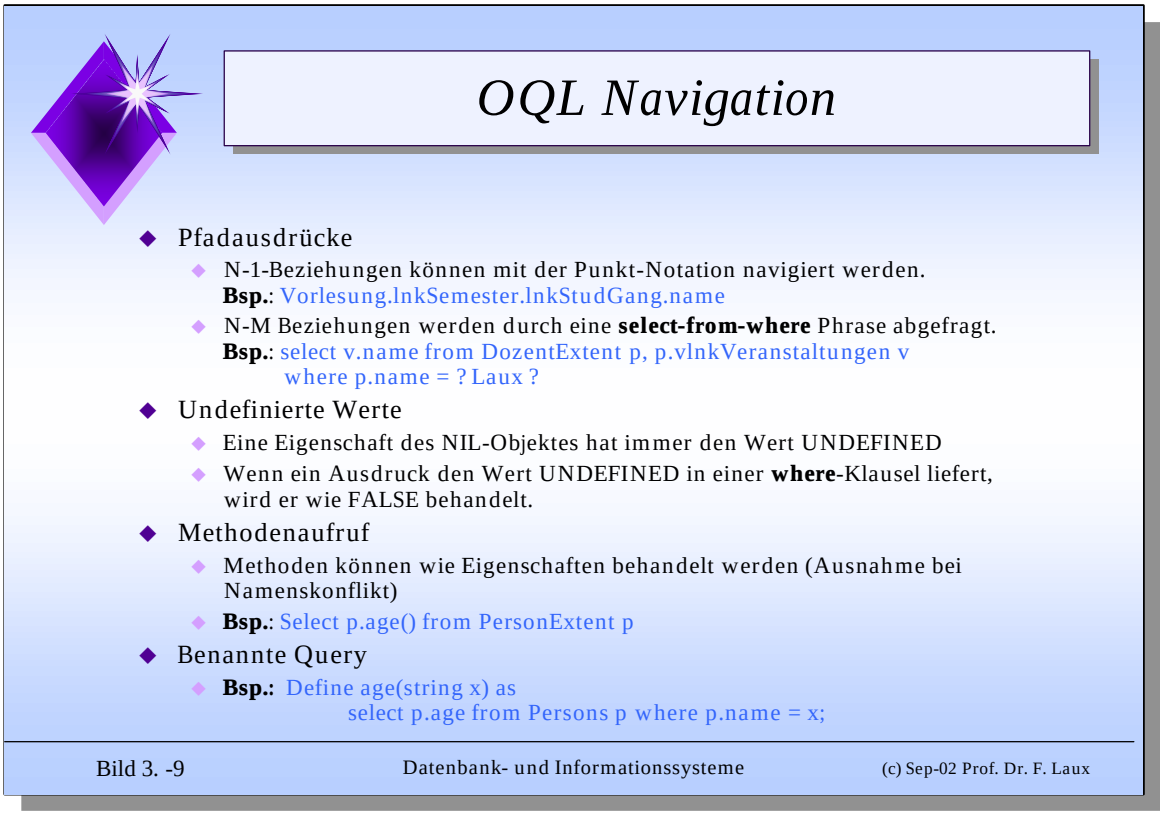


Die **Object Query Language (OQL)** ist die Abfragesprache für objektorientierte Datenbanken nach dem Objektmodell der ODMG. Sie kann interaktiv benutzt oder in einer Programmiersprache (eingebettet) verwendet werden.

Die Syntax ist der von SQL ähnlich, d.h. sie ist **deskriptiv** (Sprache der 4. Generation). Das Ergebnis einer Abfrage kann wiederum abgefragt werden, da es Element des Objektmodells ist. Diese Eigenschaft wird als „*Abgeschlossenheit*“ bezeichnet. An jeder Stelle einer Abfrage ist ein beliebiger Ausdruck erlaubt, sofern sein Wert vom Typ her zulässig ist. Z.B. kann anstelle eines Extents (Menge von Objekten des gleichen Typs) wieder eine Abfrage stehen, da diese ebenfalls eine Objektmenge eines bestimmten Objekttyps liefert. Diese schöne und sehr flexible Eigenschaft nennt man „*Orthogonalität*“ weil ein Query-Ausdruck nicht von einem anderen abhängig ist. Das ist bei SQL nicht so, deshalb ist deren Definition auch viel umfangreicher - ca. 2000 Seiten gegenüber 40 Seiten bei OQL, wobei die formale Definition in EBNF nur 6 Seiten umfasst.

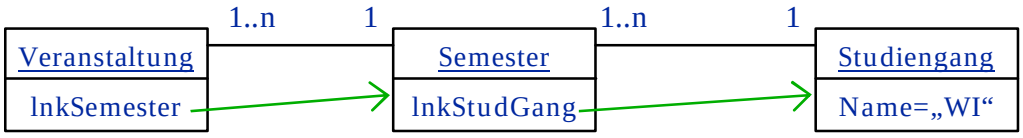
OQL ist im Gegensatz zu SQL eine reine Abfragesprache. Änderungsoperationen können nur indirekt über Methoden durchgeführt werden. Dies hat den Vorteil, dass die Sprache einfach bleibt und die Konsistenzbedingungen in dem Methoden und nicht in der Sprachdefinition berücksichtigt werden müssen.

Wenn die **Objektidentität (OID)** eines Objektes bekannt ist, kann Sie zur Selektion genau dieses Objektes verwendet werden. Die externe Darstellung dieser OID ist allerdings implementierungsabhängig. Für die Poet Datenbanken hat diese die externe Form **(n:m-x#y, z)**, wobei **n** die Datenbanknummer darstellt, **m-x** ist die Identifikation (2x32bit), **y** ist die Blockadresse in der Datenbank und **z** die Identifikationsnummer für die Klasse zu der das Objekt gehört. Die Identifikation **m-x** eines Objektes ändert sich nie und wird auch nach dem Löschen des Objektes nicht wieder verwendet.



Ein wichtiger Unterschied zum Relationalen Modell sind die in der Datenbank gespeicherten **Beziehungen**. Damit kann man ohne Abfrage, quasi durch „Verfolgen“ eines Links von einem Objekt zum andern gelangen. Diese **Navigation** erfolgt in Form einer **Punkt-Notation**, wenn es sich um „zu-1“-Verbindungen handelt. Die Darstellung <Object>.<Object_ref> bedeutet, dass man ausgehend vom aktuellen <Object> sich über den Link <Object_ref> zum in Beziehung stehenden Objekt bewegt. Das ist möglich, da es sich um genau ein Objekt (evtl. das NIL-Objekt) handelt.

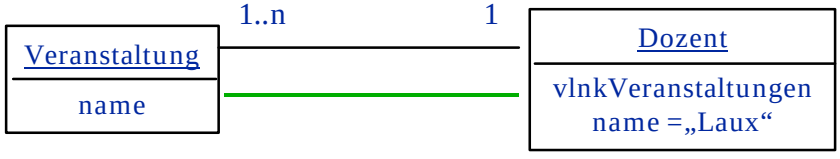
Beispiel:



select v from VeranstaltungExtent v where v.lnkSemester.lnkStudgang.name = ? WI?

Komplexe (N:M) Beziehungen können so nicht aufgelöst werden, da eine Auswahl aus mehren möglichen Verbindungen vorzunehmen ist. Die Beziehungsvariable kann wie ein Extent verwendet werden und die Auswahl erfolgt durch eine WHERE-Bedingung in der Abfrage. Damit können ein oder mehrere Objekte ausgewählt werden.

Beispiel:

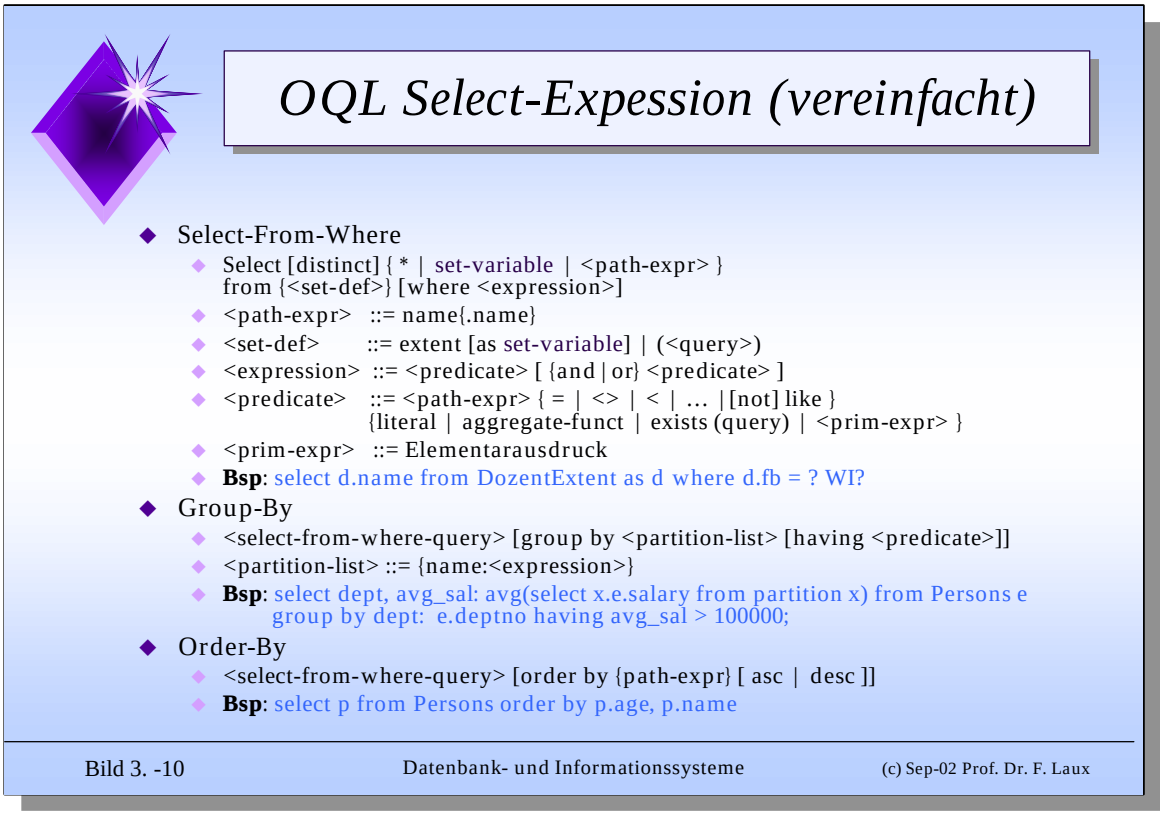


Select v.name from DozentExtent p, p.vlnkVeranstaltungen v where p.name = ? Laux?

Parameterlose Methoden können syntaktisch wie Eigenschaften behandelt werden. Bei Namensgleichheit sind die Parameterklammern hinzuzufügen.

Beispiele: select p.oldest_child.address.street from Persons p where p.lives_in(„Paris“);

Hier sei oldest_child eine Methode ohne Parameter und lives_in eine Methode mit einem Parameter. Für jede Person p, welche in Paris wohnt, wird zum ältesten Kind navigiert und dessen Straße aus dem Adressfeld ermittelt. Falls die Objekte der Klasse Person ein Attribut namens oldest_child hätten, müsste die Methode als oldest_child() geschrieben werden, um eindeutig zu sein.



Eine OQL-Query ist optisch einer SQL-Query ähnlich. Eine Query hat daher die Form einer **select-from-where** Phrase, wobei die **where**-Klausel optional ist. Ebenfalls optional ist das Sortierkriterium. Die Semantik der OQL-Phrase deckt sich weitgehend mit SQL. Allerdings sind Join-Bedingungen für <set-def> mit Beziehungen zwischen den Extents überflüssig (siehe Beispiel auf der vorherigen Seite).

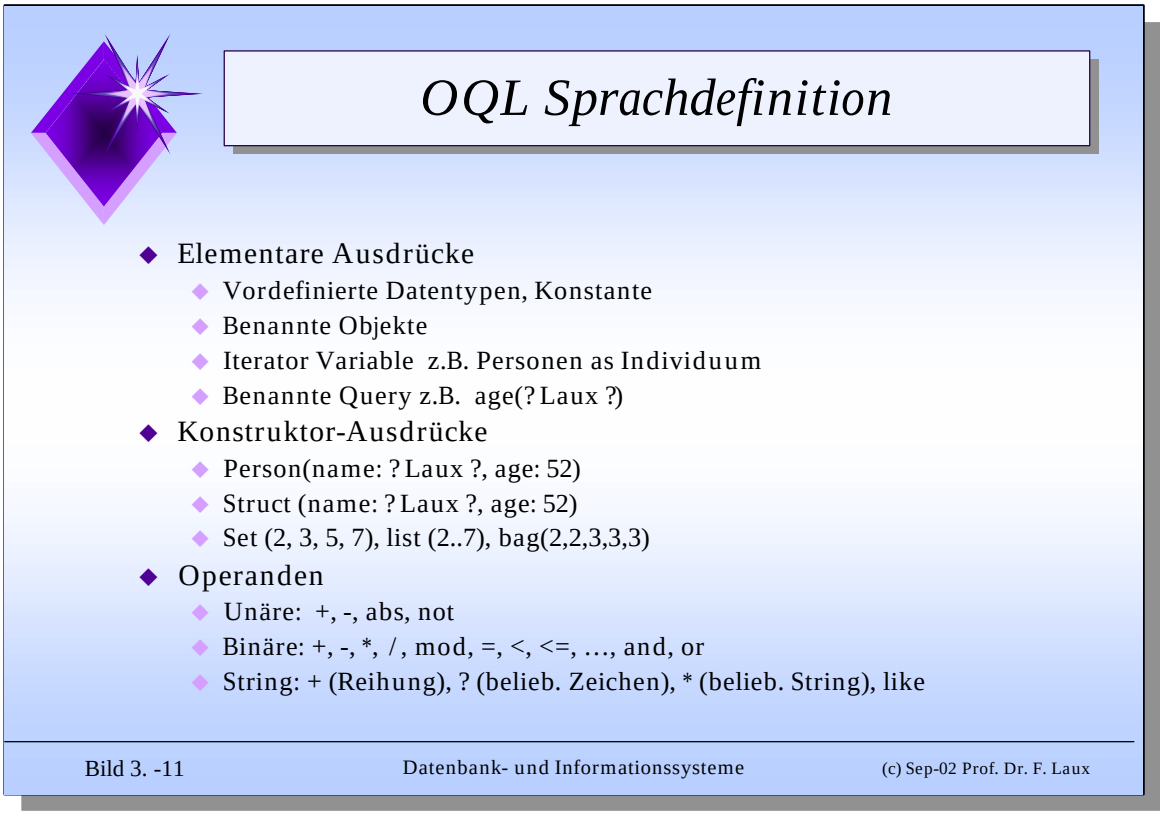
Wenn Aggregate wie Mittelwert (*avg()*), Summe (*sum()*) oder Maximum (*max()*) gebildet werden, ist ein Gruppierungskriterium durch eine **group-by**-Liste anzugeben. Das Ergebnis einer Gruppierung kann mit einer **having**-Klausel weiter eingeschränkt werden.

Beispiele:

```
Select count(*) from VeranstaltungsExtent v
where v.lnkDozent = PtRef (?(0:0-1601#4184, 101) ?);  //Poet‘s <OID>
```

```
Select d.age, count(*) from DozentExtent d
group by v.age
order by v.age;
```

Die auf der Folie wiedergegebene Syntax ist vereinfacht und entspricht nahezu der OQL-Implementierung von Poet.



Untersuchen wir nun die Sprachdefinition von OQL etwas systematischer: Ein Query wird durch **Ausdrücke** (Pfad-, Mengen- und Prädikatenausdrücke siehe vorherige Seite) formuliert, welche die gesuchten Objekte beschreiben. Die Ausdrücke können rekursiv mit **Operanden** und **Operatoren** gebildet werden. OQL ist eine sogenannte „getypte“Sprache (*typed language*), d.h. die Ausdrücke und alle (Zwischen)Ergebnisse haben einen Typ.

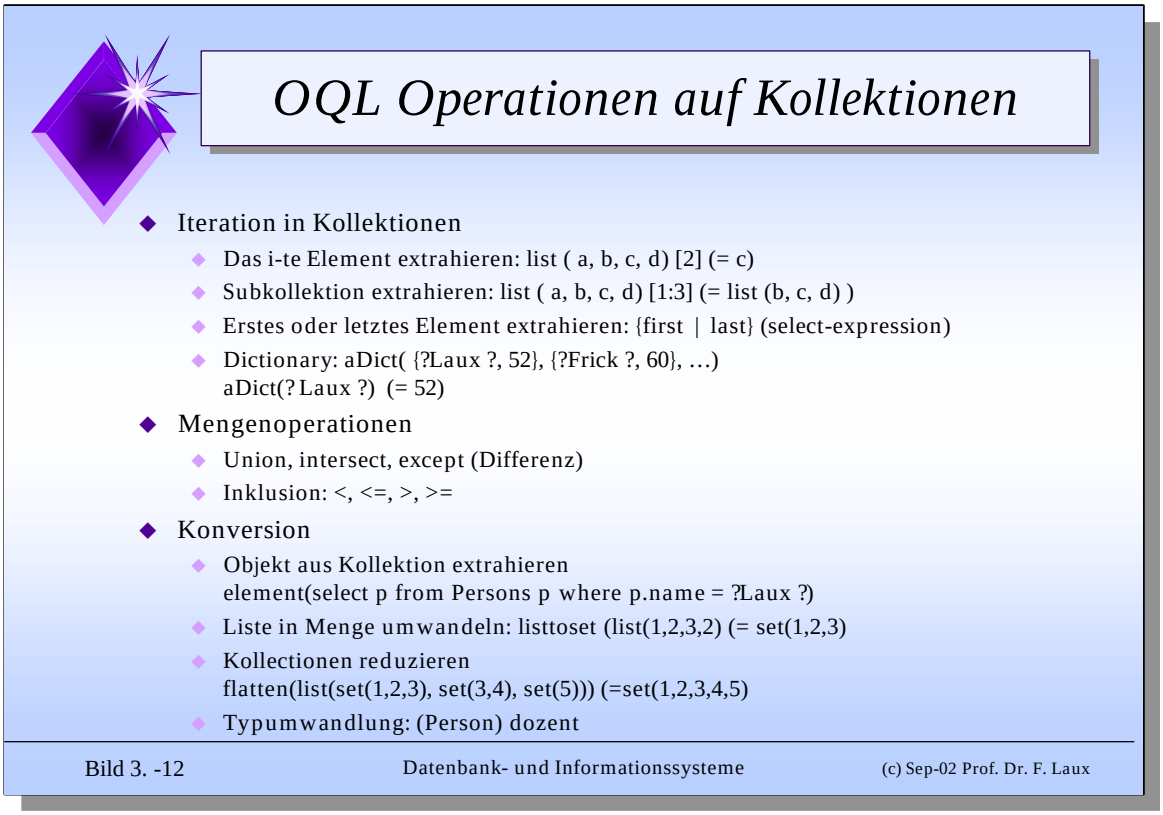
<operand> ::= [<unary_operator>] <operand> | <operand> <binary_operator> <operand> | <string_expression>

<operand> ::= <elementary_expression> | <constructor_expression> //siehe Folie

<string_expression> ::= <string_expression> {?|*|+} <string_expression> | <string>

Elementare Ausdücke, aus denen Operanden gebildet werden können, sind Konstante, vordefinierte Datentypen, benannte Objekte oder Queries und Iteratorvariablen.

Für die **where**-Klausel sind Konstruktor-Ausdrücke nützlich, um Vergleichsobjekte aufzubauen. Beispiel: select d from DozentExtent d where d = Dozent(age:50);



Die Ergebnisse einer Query sind Kollektionen von Objekten. Extents sind ebenfalls Kollektionen. Daneben gibt es Konstruktoren für Kollektionen.

Um diese Kollektion zu durchlaufen oder einzelne Objekte darin zu lokalisieren, werden Iteratoren verwendet.

Beispiele:

first (select v from VeranstaltungExtent v where v.name =?Mathe?);
liefert die *erste* Veranstaltung mit dem Namen „Mathe“ im Extent *Veranstaltung*

element(select v from VeranstaltungExtent v where v.name = ?Mathe III?).noten[0];
liefert die erste Note (Index 0) für *die* Veranstaltung mit Namen „Mathe III“ (Singelton), wenn es genau eine gibt.

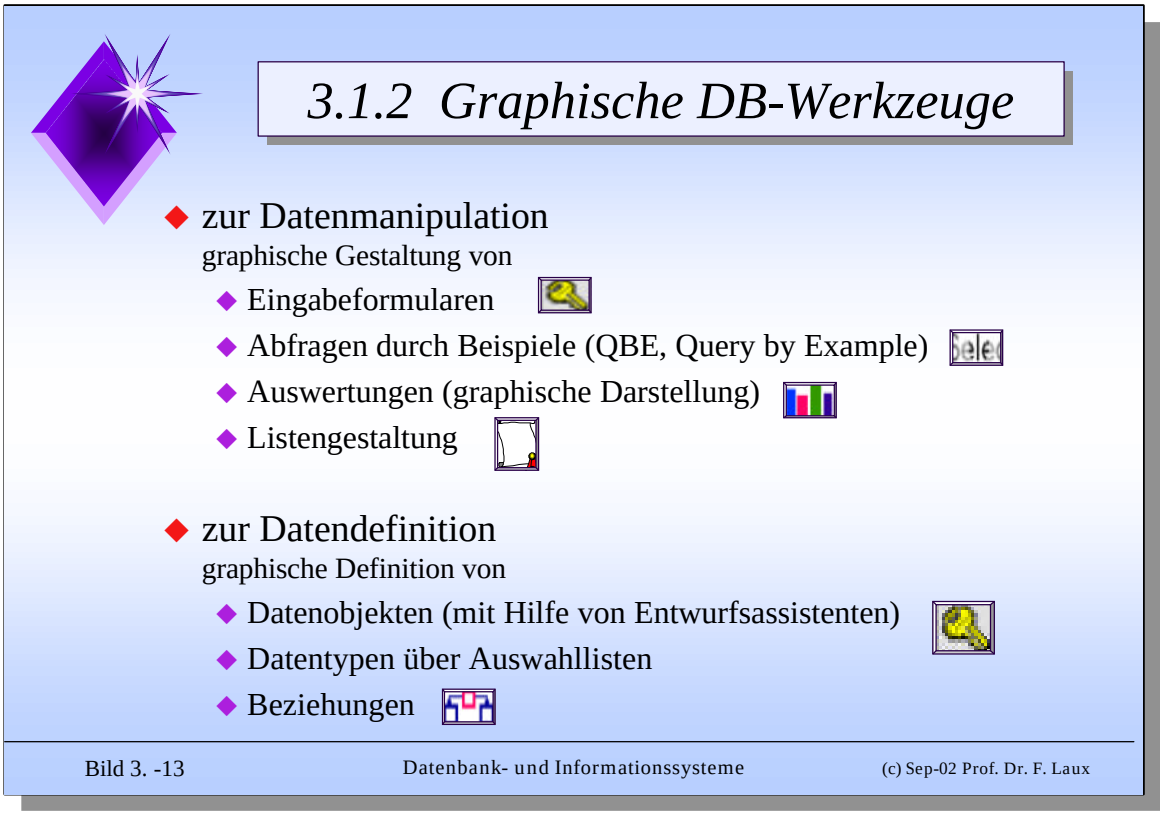
Sei vorlesDict({?WI124?,?Mathe I?}, {?WI456?,?Datenbanken?}, …) ein Dictionary mit Vorlesungsnummer-Vorlesungsnamen-Paaren.

select v from VeranstaltungExtent v where v.name = vorlesDict[?WI124?];
liefert eine Kollektion von Veranstaltungen mit dem Namen „Mathe I“

Define praktika as select * from VeranstaltungExtent p where p.typ = ?Praktikum?;

list (select * from (VeranstaltungExtent except praktika) as v)[0:9];
liefert die ersten zehn Veranstaltungen, die keine Praktika sind.

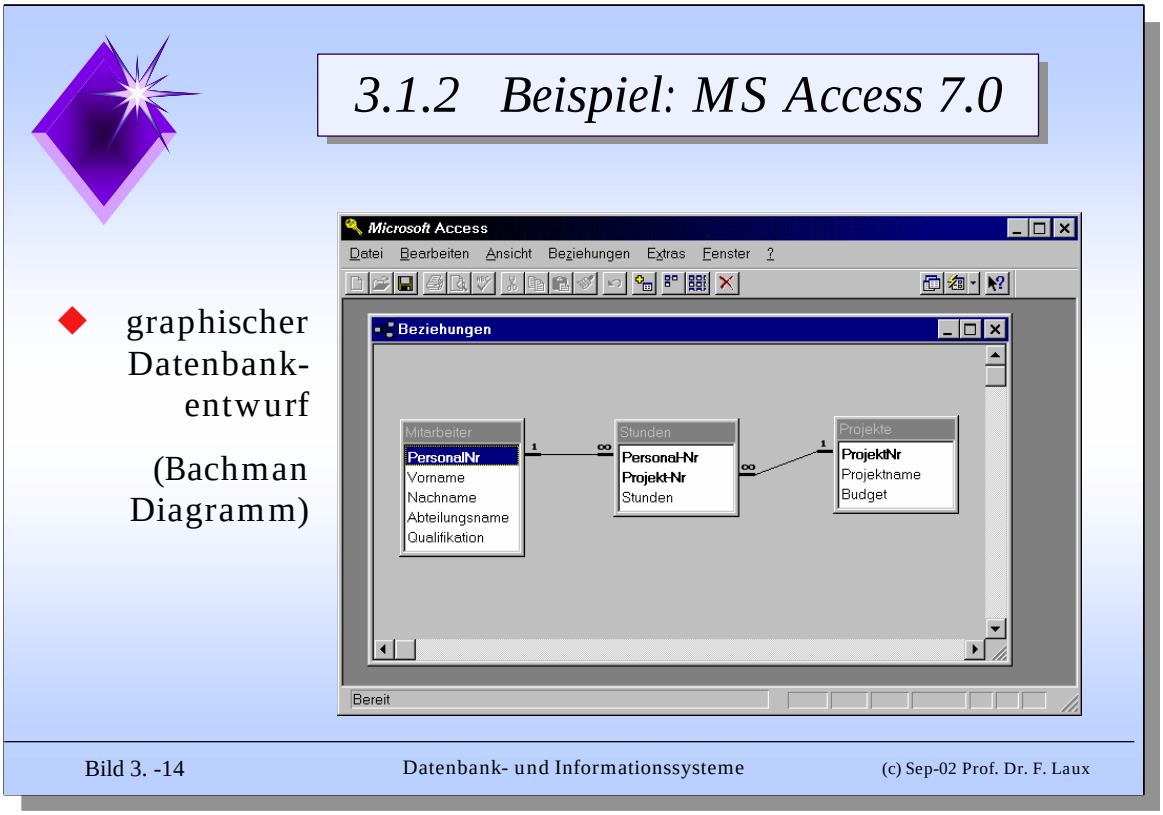
Select (Person)* from DozentExtent d where d.age > 50;
lieferst alle Dozenten, die älter als 50 sind als Personenobjekte.



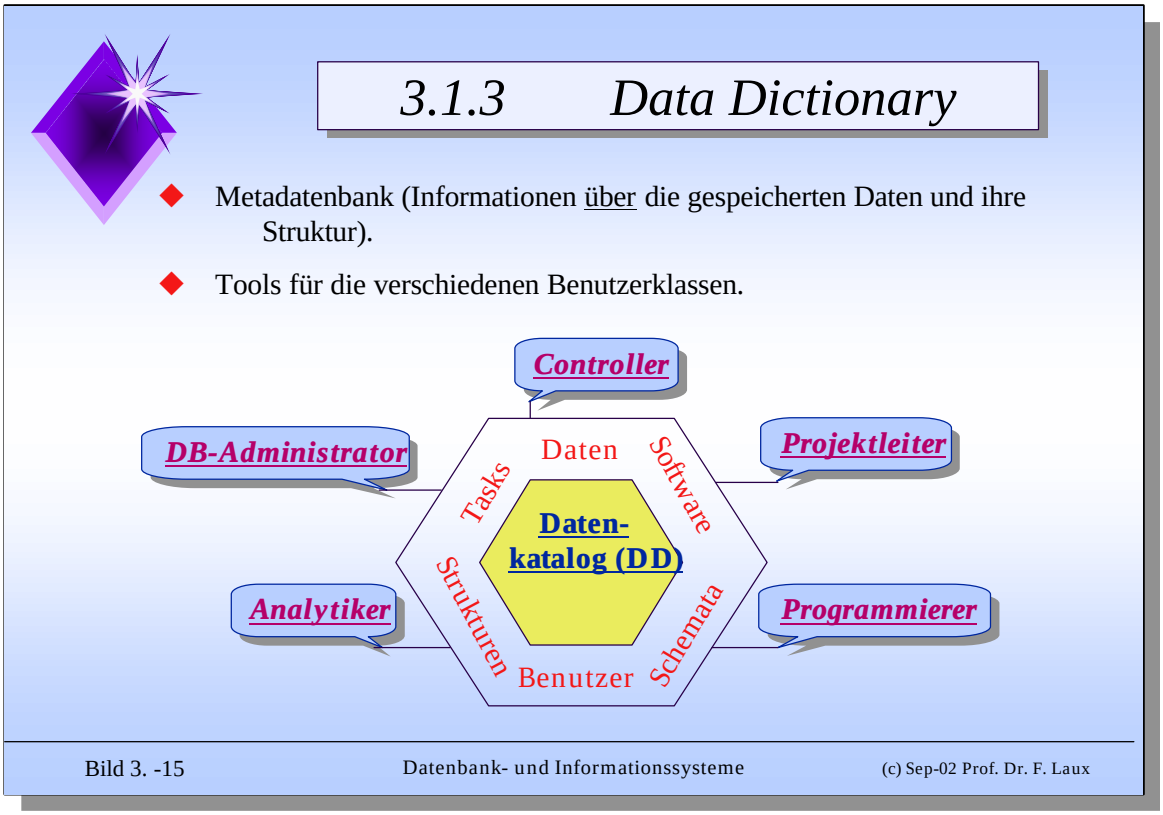
Besonders beliebt sind Anwenderwerkzeuge (end user tools) zur Gestaltung von einfachen Eingabemasken und zur Durchführung von Abfragen. Bei diesen Werkzeugen wird dem Benutzer eine Auswahl von Tabellen angeboten, für die er eine Maske gestalten oder eine Abfrage generieren kann. Numerische Ergebnisse können für die Anzeige graphisch aufbereitet werden.

Für die Definition von Datenbankobjekten bieten sogenannte Assistenten Standardvorschläge an, die der Benutzer für seine Bedürfnisse anpassen kann.

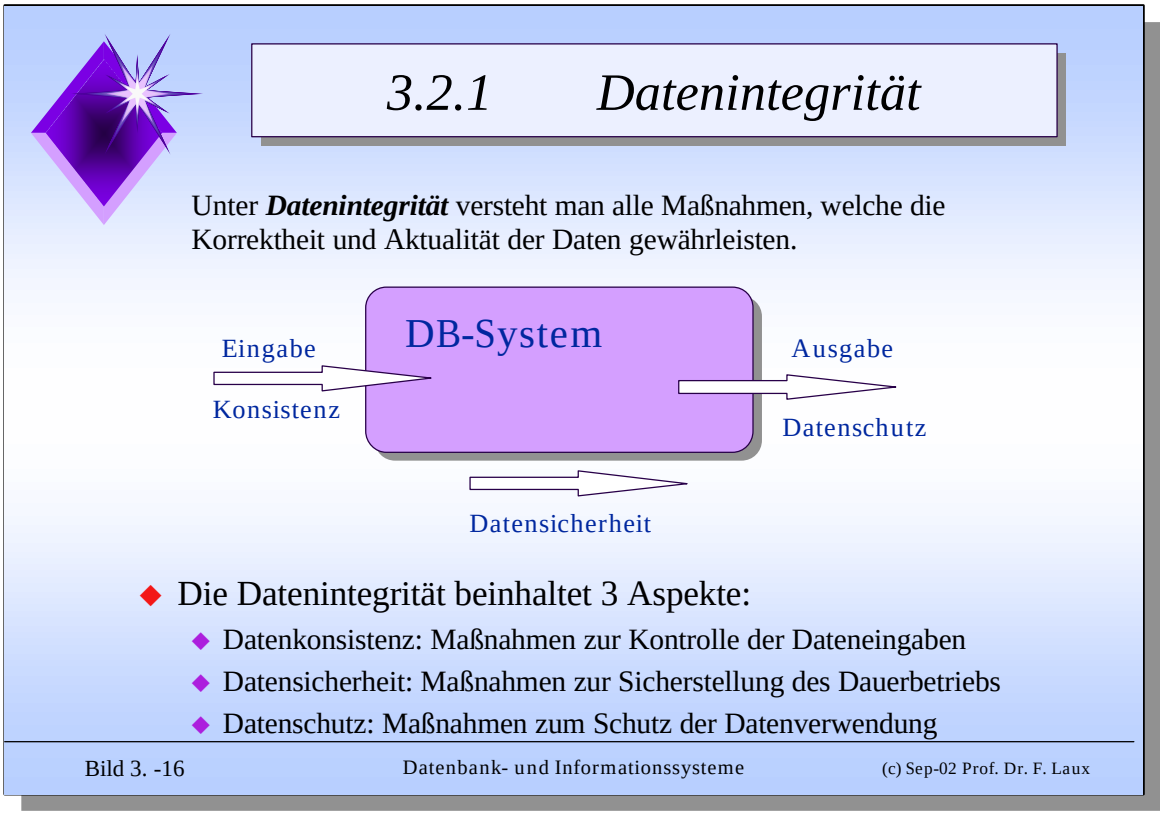
Andere Produkte listen alle verfügbaren Datentypen auf. Die Datenbankstruktur lässt sich dann aus diesen mittels Drag-and-Drop zusammensetzen. Die Beziehungen zwischen den Objekten können mit der Maus "gezeichnet" werden. Die folgende Folie zeigt das Ergebnis einer solchen graphischen Datenbankdefinition.



Dieses Relationendiagramm (Bachman Diagramm) zeigt die Datenbankstruktur für unser Beispiel *Projektverwaltung* aus Kap. 2.4. Zu erkennen sind an der linken und rechten Seite die Tabellen für die *Mitarbeiter* und die *Projekte*. In der Tabelle in der Mitte werden die *Stunden* erfaßt. Über die Fremdschlüssel *PersNr* und *ProjNr* der Tabelle Stunden werden die Beziehungen zu *Mitarbeiter* und *Projekte* hergestellt. Die Beziehungen zeigen, daß ein Stundeneintrag sich auf genau einen Mitarbeiter und ein Projekt bezieht. Ohne Eintrag in die Tabelle *Stunden* ist keine Projektzuordung von Mitarbeitern möglich. Wir können in diesem Modell eine Zuordnung hilfsweise so darstellen, daß ein Eintrag mit 0 Stunden gemacht wird.



Ein ***Data Dictionary* (*Datenkatalog*)** beschreibt Informationen über Objekte (Daten, Datenbankprogramme, Benutzer, Jobs) eines Datenbanksystems. Es enthält also *Metainformationen*, d.h. 'Daten über Daten'. Aus Sicht eines Datenbanksystems stellt der Datenkatalog (D.) eine Metadatenbank dar, in der die Schemata des Datenbanksystems verwaltet werden und diese Information den Benutzern zur Verfügung gestellt wird. Dient er nur der Beschreibung oder Dokumentation von Objekten, wird er als '**passiv**' bezeichnet. Wenn daraus auch Dateien, Datenstrukturen, Programme (z.B. Abfragen) oder Jobs generiert werden, spricht man von einem '**aktiven**' D.
Ein Datenkatalog kann als eigenständiges Softwaresystem realisiert sein. In diesem Fall enthält er häufig Informationen über mehrere Datenbanksysteme, Dateien, Programme und Hardwarekomponenten eines Computernetzwerks. Die Begriffe *Repository*, *Directory System* (Verzeichnissystem) werden verwendet, wenn der D. eine Komponente besitzt, in welchem die Speicherorte und Zugriffspfade der Objekte (Daten, Programme, Benutzer, Jobs) abgelegt sind. Der D. stellt ein nützliches, zentral verwaltetes Hilfsmittel für Datenbankadministratoren, Projektleiter, Systemanalytiker, Programmierer, EDV-Controller und andere Benutzer dar.



Unter Datenintegrität versteht man die ständige Aktualität, Korrektheit, Verfügbarkeit und kontrollierte Verwendung des Datenbestandes eines Datenbanksystems.

Sie beinhaltet alle Maßnahmen zur Sicherstellung von ***Datenkonsistenz***, ***Datensicherheit*** und ***Datenschutz***. Diese Aufgaben werden durch das Datenbankverwaltungssystem wahrgenommen.

Die *Konsistenz der Daten* wird mit Hilfe von Konsistenzbedingungen (KB) definiert. Sie werden als logische Aussagen formuliert, die in bestimmten Situationen erfüllt sein müssen (z.B. zum Transaktionsende). Die Festlegung, was sachlich und logisch richtig ist, wird bei der Definition einer Datenbank von Datenbankverwalter festgelegt. Wir unterscheiden zwei Arten von Konsistenzbedingungen:

• **primäre KB** sind bei jeder Elementaroperation (Einfügen, Löschen, Ändern) einzuhalten

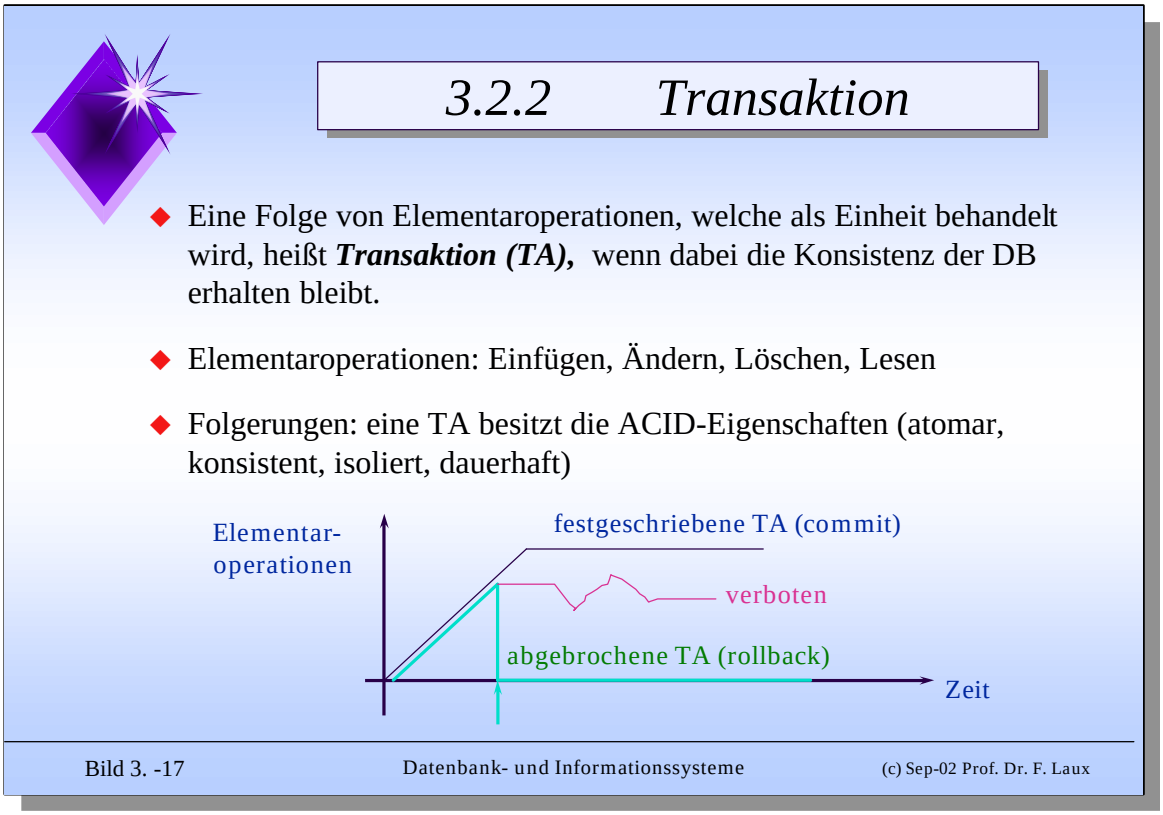
• **sekundäre KB** müssen zum Abschluß einer Transaktion (siehe 3.2.2) erfüllt sein

Beispiele :

primäre KB:	die Matrikelnummer identifiziert einen Studenten eindeutig, das Gehalt muß größer null (> 0) sein, der Status einer Heizung kann 'in Betrieb', 'ausgeschaltet' oder 'Störung' sein
sekundäre KB:	die Summe der Beträge einer Buchhaltungstransaktion ist null (= 0), ein Auftrag umfaßt mindestens eine Position, der Bestellwert muß mindestens 100 DM betragen

Datensicherheit ist das Ziel aller Maßnahmen gegen den Verlust oder die Verfälschung von Daten. Datenbanksysteme enthalten deshalb Komponenten zur Datensicherung und Wiederherstellung (Recovery) von Daten. Wird die Datensicherheit verletzt, so zieht dies neben dem realen Schaden (Verlust der Daten) in der Regel weiteren wirtschaftlichen Schaden (Kosten, Ertragsausfall) nach sich.

Die *Datenschutz*gesetze regeln die Verwendung und Weitergabe von personenbezogenen Daten. Sie werden durch das Datenbanksystem auf technischer Ebene durch eine Identifizierung der Benutzer, durch Zugriffsbeschränkungen und durch Zugriffsprotokolle unterstützt.



Eine TA wird durch 4 Transaktionsregeln (*ACID = atomic, consistent, isolated, durable*) hinreichend beschrieben: sie muß

1. als eine Einheit durchgeführt werden, d.h. *unteilbar* (*atomic*) sein (Alles-oder-nichts Prinzip),
2. alle Konsistenzbedingungen einhalten (*consistent*),
3. während der Durchführung gegenüber anderen Transaktionen/Benutzern *abgeschirmt* (*isolated*) werden, damit keine Seiteneffekte auftreten (d.h. serialisierbar sein (3.3.2))

und

4. sie muß *dauerhaft* (*durable*) gespeichert werden, auch im Falle eines Fehlers.

Aus den Transaktionseigenschaften folgt: Eine TA überführt eine vorher konsistente DB in einen neuen konsistenten Zustand.

Beispiel:

Betrachten wir die Transaktion *Auftrag erfassen* einer relationalen Datenbank mit den Tabellen *Auftr_kopf, Auftr_Pos, Artikel* und *Kunde*. Folgende KB sind dabei einzuhalten:

1. Für den Preis eines Artikels in einem Auftrag kann max. 10% Nachlaß gegenüber dem Listenpreis in Tabelle *Artikel* gewährt werden.
2. Die Auftragsmenge einer Positionen muß > 0 sein.
3. Eine Auftragsnummer ist eindeutig, d.h. identifiziert einen Auftrag.
4. Es können nur Artikel aus dem Artikelstamm (Tabelle *Artikel*) bestellt werden.
5. Der Kunde eines Auftrags muß im Kundenstamm (Tabelle *Kunde*) existieren.
6. Ein Auftrag umfaßt mindestens eine Auftragsposition.

ÜA: Welche KB in o.g. Beispiel sind primär und welche sekundär ?

3.2.3 Der Transaktionsmechanismus

- Um das TA-Konzept zu verwirklichen, muß
  - eine explizite Bestätigung (commit) oder
  - eine vollständige Annulation (rollback)

  durchgeführt werden
- Die Mechanismen, welche im Fehlerfall angewandt werden, sind:
  - REDO : Operation wiederholen
  - UNDO : Operation rückgängig machen
- Beide Funktionen sind idempotent, d.h.

  REDO(x) = REDO(REDO(x)) bzw. UNDO(x) = UNDO(UNDO(x))

Bild 3. -18 Datenbank- und Informationssysteme (c) Sep-02 Prof. Dr. F. Laux

Der Transaktionsmechanismus hat die Aufgabe, die *Transaktionsregeln* (*ACID*) in jeder Situation zu erfüllen. Dadurch wird sichergestellt, daß die Datenbasis nur konsistente Zustände annimmt. Damit eine TA als Einheit behandelt werden kann, muß die Anwendung oder der Benutzer das Ende der TA explizit markieren. Dies geschieht durch eine Bestätigung (*commit*) mit der die Änderungen festgeschrieben werden. Im Fehlerfall wird eine vollständige Rücknahme aller Änderungen durchgeführt (*rollback*). Dies kann auf Grund einer Konsistenzverletzung durch das DB-Management selbst erfolgen (*backout*) oder durch den Anwender absichtlich herbeigeführt werden (*abort*).Wenn ein Systemfehler auftritt, ist nicht sicher, welche Operationen noch durchgeführt werden konnten und welche nicht. Um diese Unsicherheit zu beheben, sind zwei Grundfunktionen (primitives) notwendig: UNDO und REDO. Damit kann ohne Seiteneffekt eine Operation wiederholt oder rückgängig gemacht werden.

Hinweis: UNDO und REDO weichen in ihrer Funktionsweise von den entsprechenden Funktionen eines Texteditors ab.

Das Transaktionsmanagement hat über den Transaktionsmechanismus hinaus insgesamt folgende Aufgaben:

1. die Verwaltung der Arbeits- und Massenspeicher,
2. die Protokollierung (Transaktionslog),
3. die Synchronisation paralleler Transaktionen (siehe 3.3),
4. die Behandlung von Verklemmungen,
5. die Überprüfung der Konsistenzbedingungen (KB) und
6. gegebenenfalls eine Fehlerbehandlung vorzunehmen.

3.2.4 Zwei-Phasen Commitment (2PC)

- Für ein verteiltes Datenbanksystem oder ein System mit mehreren Prozessen ist es notwendig, daß alle Prozesse die gleiche Entscheidung über die TA treffen. Das 2PC-Protokoll stellt eine ***einheitliche*** Entscheidung sicher.

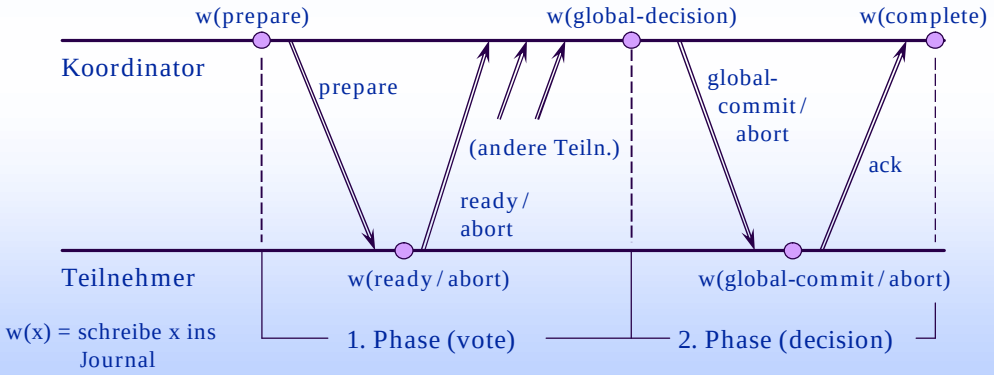
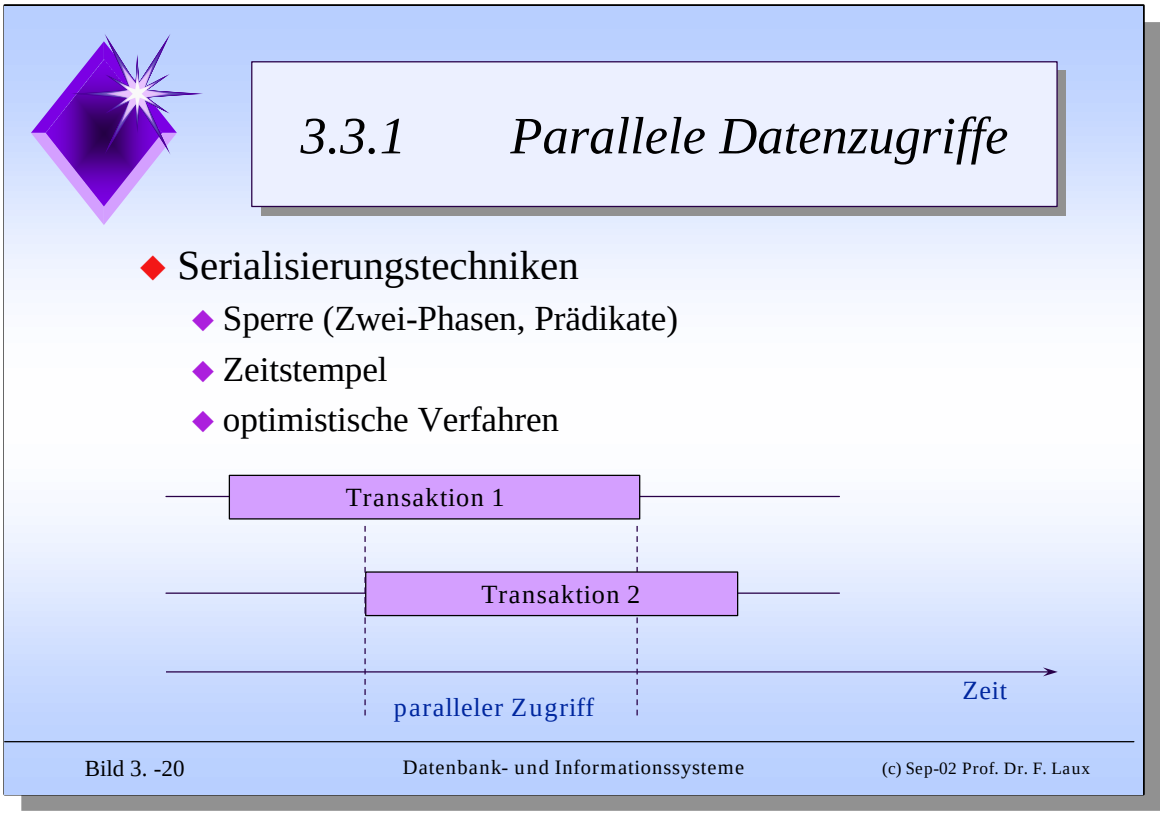


Bild 3. -19 Datenbank- und Informationssysteme (c) Sep-02 Prof. Dr. F. Laux

Das erfolgreiche Ende einer Transaktion wird durch eine Festschreibephase (Commitment = Verpflichtung) markiert. In dieser Phase wird eine bestimmte Folge von Nachrichten an die beteiligten Prozesse verschickt (Protokoll), um sicherzustellen, daß alle die gleiche Entscheidung treffen, damit die Transaktion entweder vollständig oder überhaupt nicht durchgeführt wird. Ein bekanntes Protokoll ist das Zwei-Phasen-Freigabeprotokoll (2-Phase-Commitment Protocol, 2PC). In der *ersten Phase (Abstimmung)* werden alle beteiligten Prozesse abgefragt, ob sie die Transaktion erfolgreich abschließen können. Wenn dies für alle Prozesse der Fall ist, wird in der *zweiten Phase (Entscheidung)* die Transaktion festgeschrieben.

Der Ablauf des 2PC wird protokolliert, so daß im Fehlerfall der Ablauf rekonstruiert und nach der Behebung des Fehlers die TA zum Abschluß (commit oder abort) geführt werden kann.



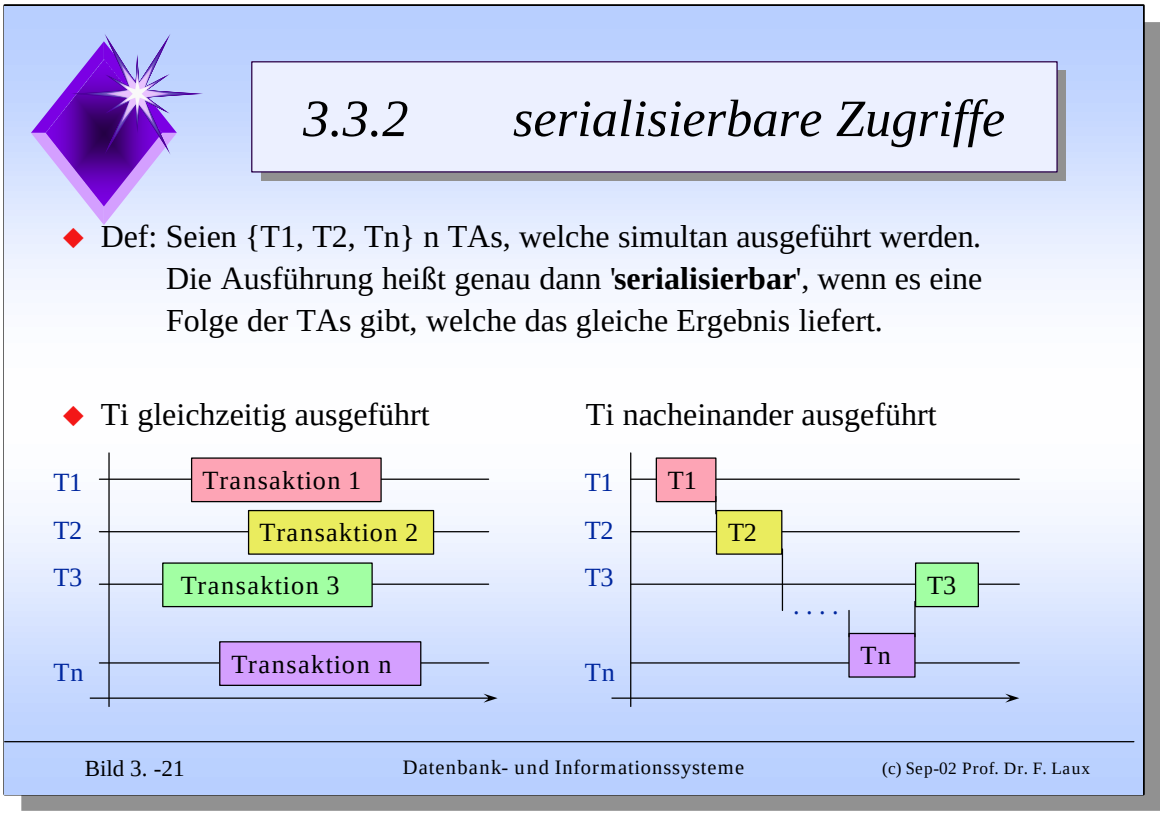
Die Steuerung *paralleler Datenzugriffe* ist ein wichtiger Aspekt eines Datenbanksystems, damit viele Benutzer gleichzeitig und verzögerungsfrei mit den Daten arbeiten können. Parallelität ist dann möglich, wenn die Transaktionen *serialisierbar* (Def. siehe 3.3.2) sind.

Sperrverfahren werden bei zentralen DBS am häufigsten eingesetzt. Dabei werden konfliktbehaftete TA nacheinander ausgeführt. Dadurch wird vermieden, daß inkonsistente Zwischenzustände von anderen Transaktionen gesehen werden oder nicht serialisierbare Ergebnisse erzeugt werden.

Beispiel: Zwei Transaktionen, T1 und T2 wollen den Preis eines Artikels um 10% bzw. 20% erhöhen. Dies ist nicht gleichzeitig möglich, da beide TAs den gleichen Ausgangspreis lesen. Ohne Serialisierung geht eine Transaktion verloren. (welche?)

Ein geeignetes Sperrverfahren (z.B. 2PL, Kap. 3.3.3) stellt sicher, daß die Transaktionen serialisierbar sind. Der Nachteil dabei ist, daß Verklemmungen (Deadlocks, vgl. Kap. 4.3.4) auftreten können, die zum Rollback einer der beteiligten TA führen muß, obwohl sonst keine Fehlersituation vorliegt.

Zeitstempel und *optimistische Verfahren* vermeiden Verklemmungen und sind deshalb besonders für verteilte Datenbanken geeignet (vgl. Kap. 4.3.5). Die geringe Verbreitung dieser Verfahren ist darauf zurückzuführen, daß die Behandlung von Konfliktsituationen schwieriger ist und das System bei zu vielen Konflikten zur Instabilität neigt.



Zur Illustration der Definition seien folgende Transaktionen gegeben:

T1: liest den Preis von Artikel Nr. 12345.

T2: ändert den Preis von Artikel Nr. 12345.

T3: Summiert die Preise aller Artikel.

Beispiel einer **serialisierbaren** Ausführung (Schedule) :

Die zeitgleiche Ausführung der Transaktionen ist serialisierbar, wenn alle TAs erfolgreich sind und sie eines der folgenden Resultate liefern:

(Notation: T1,T2,T3 = auf T1 folgt T2 und dann T3)

T1, T3, T2, oder

T2, T1, T3, oder

T1, T2, T3, oder

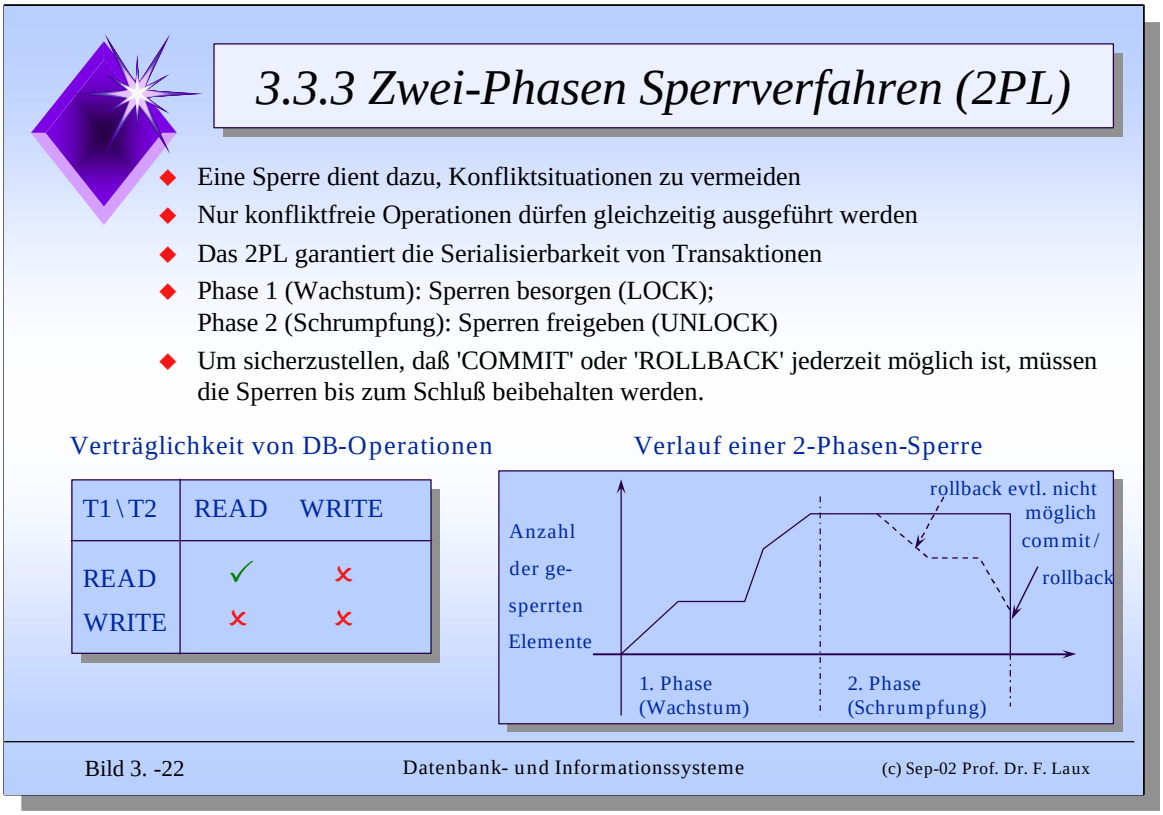
T3, T2, T1.

Die restlichen 2 Möglichkeiten sind mit einer der vorgenannten äquivalent.

Beispiel einer **nicht serialisierbaren** Ausführung (Schedule) :

Wenn T1 den durch T2 geänderten Artikel 12345 liest und T2 danach annulliert wird. T1 liefert in diesem Fall ein falsches Ergebnis (das nie existiert hat).

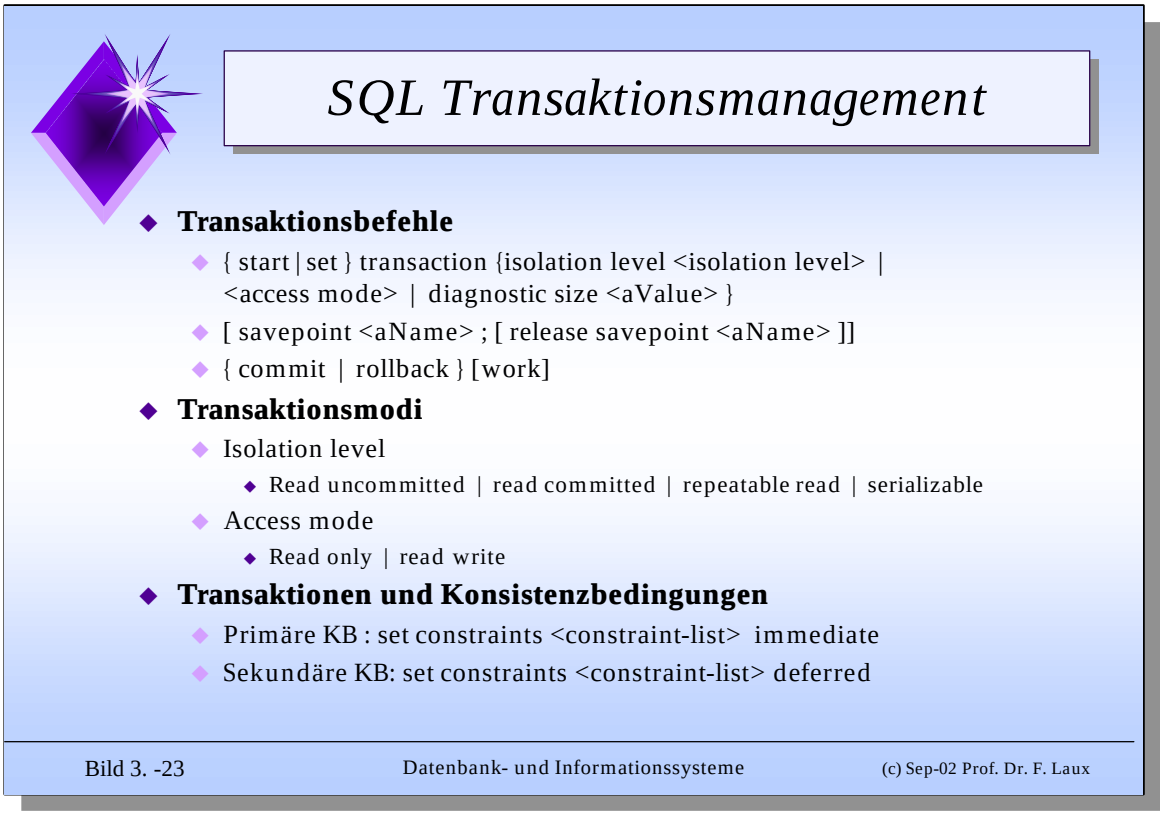
ÜA: Ist die folgende Situation serialisierbar? T3 liest den Preis, bevor T2 ihn ändert, beendet aber erst nach Abschluß von T2 die Summation.



Eine Möglichkeit zur *Synchronisation paralleler Transaktionen* sind *Sperrverfahren*. Die *Serialisierbarkeit* wird durch das **Zwei-Phasen-Sperrprotokoll** (**two phase locking, 2PL**) gewährleistet, welches besagt, daß innerhalb einer Transaktion (TA) nach Freigabe einer Sperre keine neue mehr angefordert werden darf. Werden alle Sperren bis zum Abschluß gehalten, so kann die TA jederzeit zurückgesetzt werden. Wenn alle Sperren schon zu Beginn besorgt werden (**preclaiming**), kann es keine Verklemmung geben. Allerdings ist zu Transaktionsbeginn oft nicht klar, welche Sätze gesperrt werden müssen. Sperrbereiche können durch die Angabe einer Adresse, des Identschlüssels, einer Tabelle, etc. oder durch **Prädikate** (logische Aussage mit Parametern) festgelegt werden, z.B. alle Sätze, für die das Prädikat *Kunde = "Maier"* den Wert TRUE hat.
Durch die Sperre eines Datenbereiches werden andere TA davon abgehalten, diesen zu benutzen oder gar zu verändern. Je kleiner der gesperrte Bereich (**Sperrgranularität**) ist, desto größer ist die Wahrscheinlichkeit, daß andere TA unbehindert durchgeführt werden können. Übliche Sperreinheiten sind Datensätze, Tupel, Dateien, Relationen oder die ganze Datenbasis. Im letzteren Fall hat nur eine TA Zugriff auf die Datenbank (z.B. für die Wiederherstellung (Kap. 3.4.3) oder zur Restrukturierung der Datenbank). Datenbereiche können *exklusiv* (Lese- und **Schreibsperre**) oder nur zum Lesen (*shared*, **Lesesperre**) gesperrt werden. Nur Lesesperren sind miteinander verträglich. Sperren müssen explizit vom Datenbankmanagementsystem angefordert werden, da der Sperrtyp von der beabsichtigten Folge von DB-Operationen abhängt.

Es gibt sogar Fälle bei denen Daten gesperrt werden müssen, die von der Transaktion weder gelesen noch geschrieben werden.

Beispiel: Während einer Auftragserfassung muß sichergestellt werden, daß der Artikelstamm sich nicht ändert. Insbesondere dürfen keine Artikel gelöscht werden.



Die Konsistenzstufen sind bei SQL durch bestimmte unerwünschte Erscheinungen definiert.

Ein ***‚dirty read‘*** bezeichnet eine Situation, bei der eine TA1 einen Wert liest, der von einer TA2 geschrieben, aber nicht festgeschrieben (committed) wurde. Das bedeutet, dass evtl. ein inkonsistenter Wert gelesen und verwendet wird.

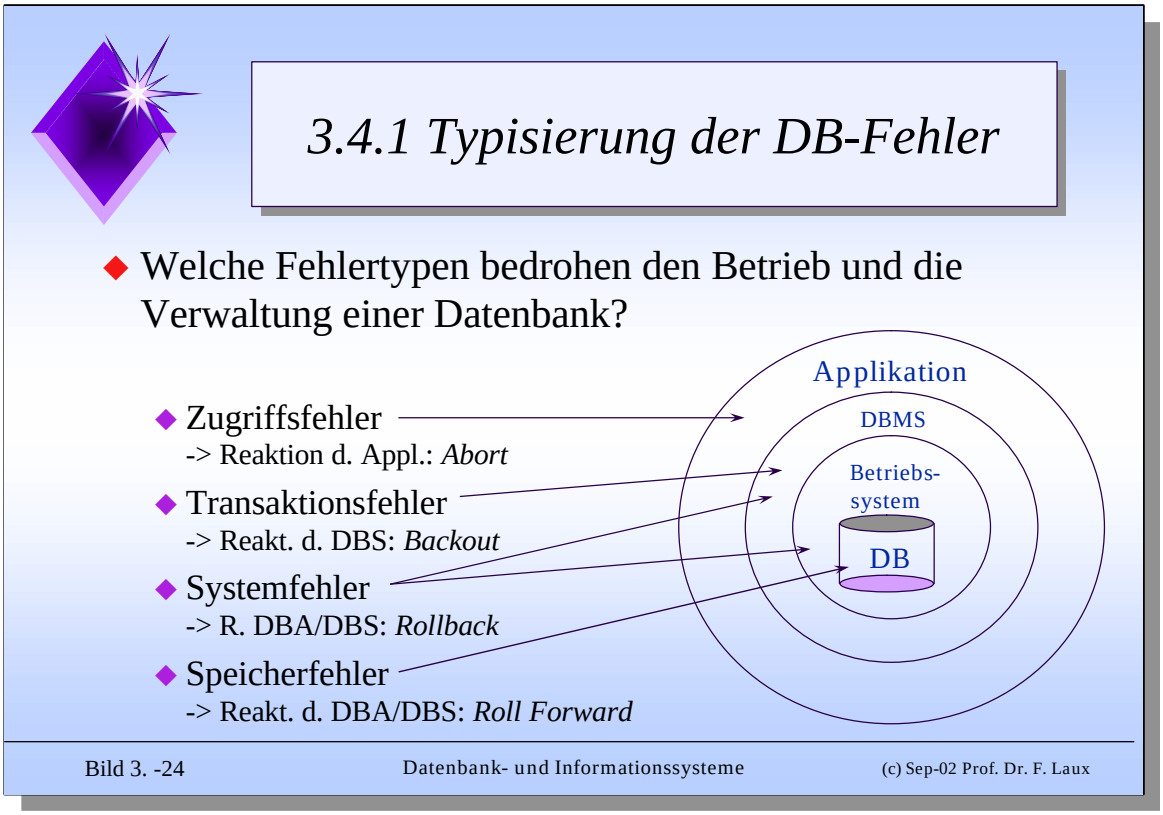
Als ***‚nonrepeatable read‘*** bezeichnet eine Situation, in der eine TA1 einen Wert liest, der während des TA1-Verlaufes geändert wird, so dass bei einem wiederholten Lesen, der Wert nicht mehr derselbe ist.

Ein ***‚phantom read‘*** Phänomen tritt auf, wenn eine TA1 eine Query ausführt der bei seiner Wiederholung andere Zeilen und damit eine andere Wertemenge liefert.

Die folgende Tabelle zeigt die Konsistenzstufen (Isolation levels) und die sie definierenden Phänomene:

Level	**Dirty read**	**Nonrepeatable read**	**phantom**
Read uncommitted	Möglich	Möglich	Möglich
Read committed	Unmöglich	Möglich	Möglich
Repeatable read	Unmöglich	Unmöglich	Möglich
serializable	Unmöglich	Unmöglich	Unmöglich

Hinweis: Die verwendeten Schlüsselwörter für die Konsistenzstufen sind teilweise irreführend, da sie die Situation nicht korrekt charakterisieren. Z.B. repeatable read heisst nicht, dass eine Query innerhalb einer TA immer das selbe Ergebnis liefern muss. Wenn neue Sätze eingefügt oder Daten so geändert werden, dass sie das Selektionskriterium nicht mehr erfüllen, kann eine Wiederholung der Query ein anderes Ergebnis liefern. Der von der Query gelesene Wert einer Zeile bleibt jedoch unverändert.



Beim Betrieb eines Datenbanksystems können Fehler auftreten, welche die Datenkonsistenz und die Dauerhaftigkeit von Transaktionen verletzen.

Geben wir für jeden Fehlertyp ein Beispiel:

Zugriffsfehler: fehlende Daten, fehlerhafte Operation

Lösung: fehlende Daten eingeben, Operation wiederholen oder TA durch die Applikation zurücksetzen (*abort*)

Transaktionsfehler: Programmierfehler, Verklemmung (*Deadlock*)

Lösung: TA durch DBMS zurücksetzen (*backout*)

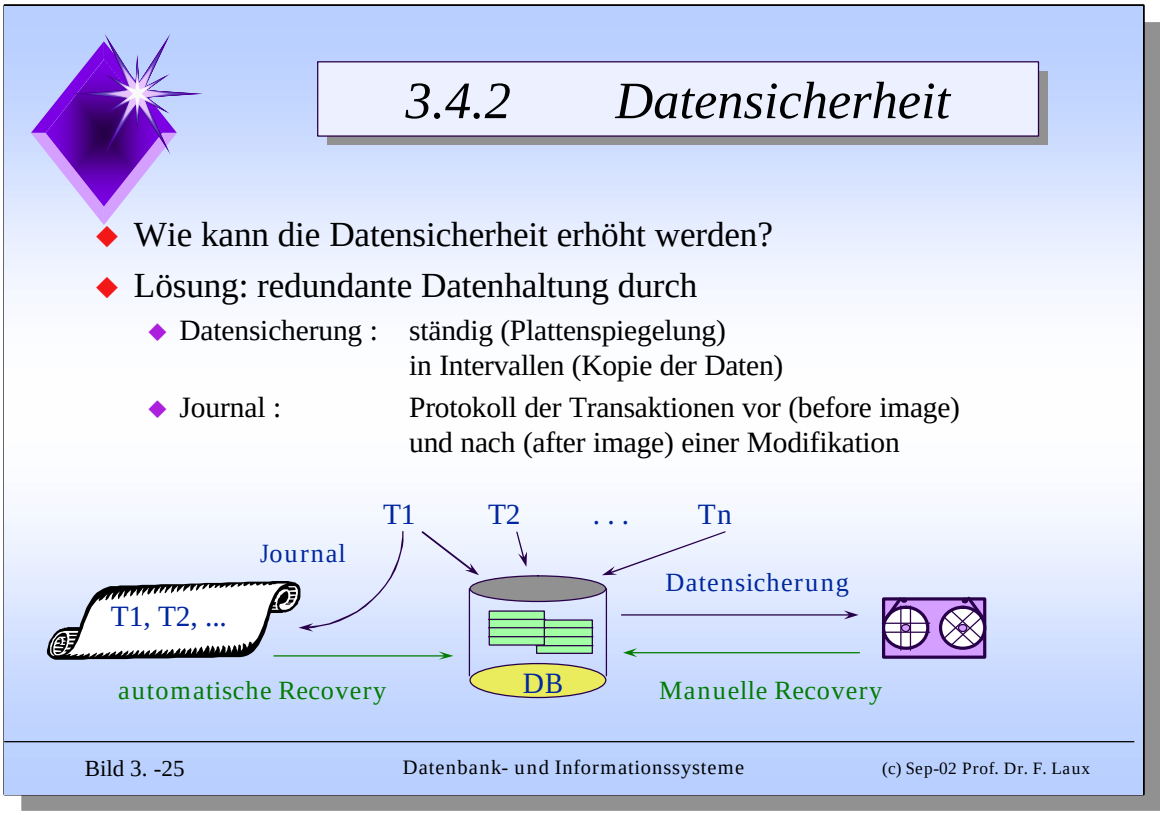
Systemfehler: Datenbank- oder Betriebssystemfehler, Stromausfall (ohne Speicherfehler)

Lösung: Neustart des Systems mit Zurücksetzen (*rollback*) aller nicht beendeten TAs (automat. Wiederherstellung durch das DBMS)

Speicherfehler: Systemfehler mit Schreibfehler auf dem Massenspeicher, Beschädigung der Festplatte (z.B. head crash)

Lösung: Speicher reparieren, konsistenten Datenbestand restaurieren (mit Hilfe einer Datensicherung), Wiederherstellung aller TAs seit der letzten Datensicherung mit Hilfe des TA-Protokolls (*roll forward*)

Alle Transaktionen, die zum Fehlerzeitpunkt nicht festgeschrieben (commited) waren, müssen neu eingegeben werden.



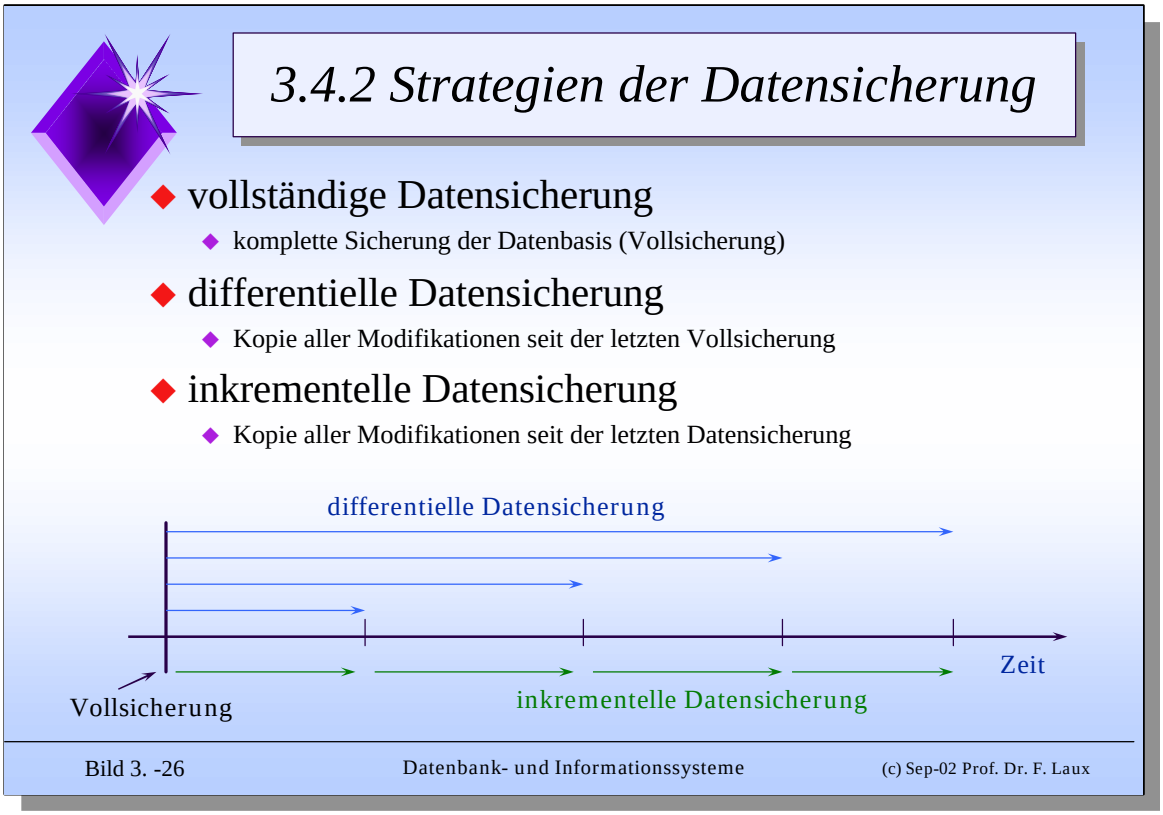
Datensicherheit (D.) ist das Ziel aller Maßnahmen gegen den Verlust oder die Verfälschung von Daten. Datenbanksysteme enthalten deshalb Komponenten zur *Wiederherstellung* (*Recovery*) von Daten. Wird die D. verletzt, so zieht dies neben dem realen Schaden (Verlust der Daten) in der Regel weitere wirtschaftliche Konsequenzen (Kosten, Ertragsausfall) nach sich.
Ein Datenverlust entsteht als Ergebnis einer Beschädigung des Datenträgers (Hardwarefehler) oder eines ungewollten bzw. unberechtigten Löschvorgangs. Datenverfälschungen können durch unberechtigte Verarbeitung, konkurrierende Zugriffe oder Programm- und Systemfehler zustande kommen.

Da reale Systeme nicht absolut zuverlässig sind, versucht man durch

1. Datenredundanz,
2. organisatorische Maßnahmen und
3. gebäude- und gerätetechnische Maßnahmen

die Wahrscheinlichkeit von Datenverlusten oder Verfälschungen zu reduzieren.

Datenredundanz kann permanent oder zu bestimmten Zeitpunkten (*Datensicherung*, siehe Bild 3-20) hergestellt werden. Ständige Redundanz ergibt sich z.B. durch die spiegelbildliche Speicherung der Daten auf zwei unabhängigen Datenträger (*Plattenspiegelung*, shadowing, mirroring) oder durch *Protokollierung* der Transaktionen (*Transaktionslog, Journal*). Gebäudetechnische Maßnahmen, wie geographische Verteilung der Anlagen, Zutrittskontrolle etc. sind dazu geeignet, sich gegen Sabotage und Naturkatastrophen abzusichern. Fehlertolerante Geräte (z.B. Hardware mit Prüfziffern zur *Fehlererkennung* und Korrektur, unterbrechungsfreie Stromversorgung) können die Verfügbarkeit von Informationssystemen wesentlich erhöhen.



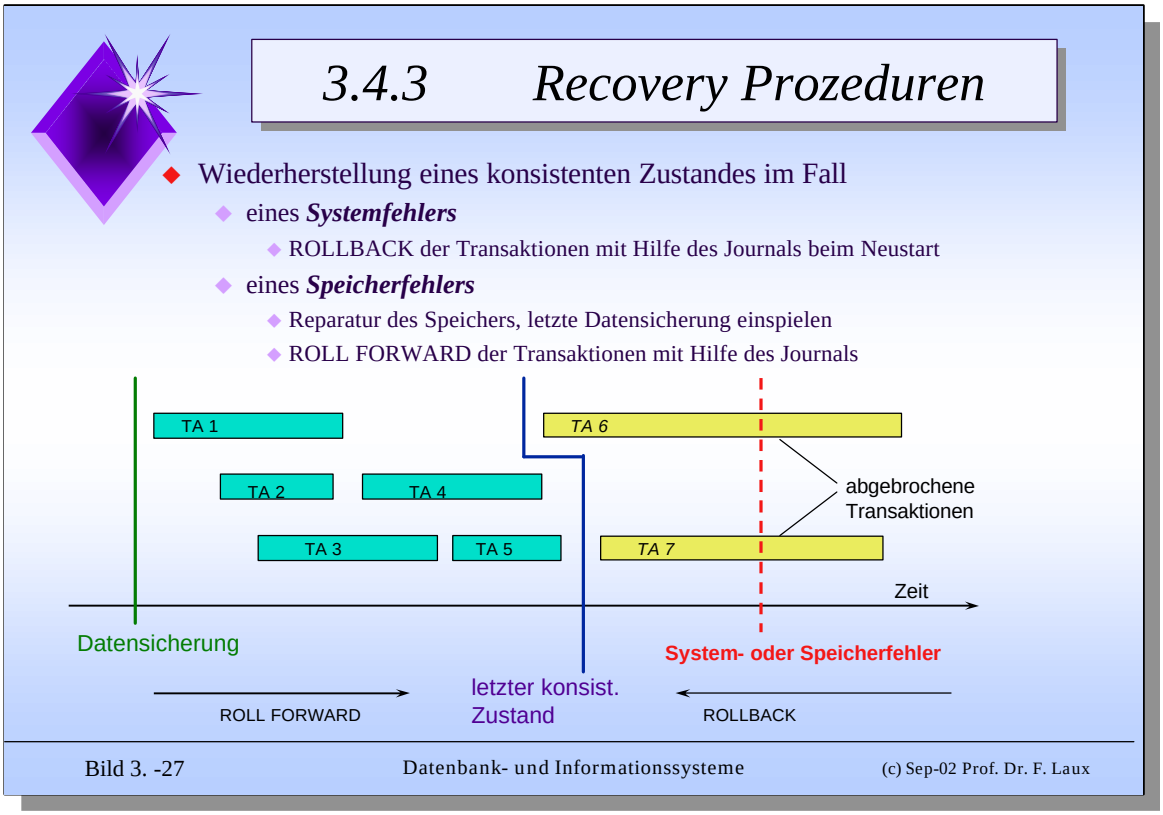
Die wichtigste und häufigste vorbeugende Maßnahme zur Datensicherheit ist die ***Datensicherung***. Eine Systematisierung kann nach folgenden technischen Gesichtspunkten erfolgen:

1. ***physische Kopie*** (struktur- und mediumsabhängig, engl. *dump*)
2. ***logische Kopie*** (struktur- und mediumsunabhängig, engl. *export*)

Eine physische Kopie erlaubt die Wiederherstellung nur auf einem gleichartigen Datenträger, während die logische Kopie auch zur Umstrukturierung und Übertragung auf andere Datenbanksysteme verwendet werden kann.

Die Strategie zur Erstellung von Sicherungskopien ist abhängig von der *Sicherungstiefe* (Anzahl Versionen), vom Aufwand für die *Sicherungsmaßnahme* und für die *Wiederherstellung*. Für die Datensicherung sind geeignete Verfahrensabläufe (wer, wann, welche Daten sichert) festzulegen, durch Kontrollen zu überprüfen und vorsorgliche Regelungen zur Wiederherstellung zu treffen.

Zur Verringerung des Aufwandes und Speicherbedarfs werden Teilsicherungen eingesetzt. Ausgehend von der letzten **Vollsicherung** werden alle seit diesem Zeitpunkt geänderten Daten gesichert (**differentielle Sicherung**) oder es werden nur die seit der jeweils letzten Teilsicherung geänderten Daten gesichert (**inkrementelle Sicherung**). Für die Wiederherstellung der Daten sind bei der differentiellen Sicherung die Vollsicherung und die letzte Teilsicherung erforderlich; bei der inkrementellen Sicherung sind hingegen alle Teilsicherungen seit der letzten Vollsicherung erforderlich.



In Abhängigkeit von der Fehlersituation kann die Wiederherstellung durch eine Vorwärtsrekonstruktion (*roll forward*) oder ein Zurücksetzen (*rollback*) erfolgen.

Rollback:

Wenn die *physische Konsistenz* der Daten durch einen Fehler *nicht verletzt* wurde, kann die logische Konsistenz wiederhergestellt werden, indem schrittweise die betroffenen Transaktionen rückgängig gemacht werden. Dies kann dadurch geschehen, daß der Zustand vor Beginn der letzten Transaktion (**before image**) aus dem Transaktionslog ermittelt wird und in die Datenbank geschrieben wird.

Roll Forward:

Ist die *physische Konsistenz* der Datenbank *verletzt* oder der *Datenträger unbrauchbar* geworden, so muß nach der Reparatur des Datenträgers die Datenbank neu angelegt werden und die Datensicherung eingespielt werden. Damit ist wieder ein logisch konsistenter, aber noch kein aktueller Zustand der Datenbank, erreicht. Durch ein Nachfahren aller erfolgreichen Transaktionen seit der letzten Datensicherung erhält man wieder den aktuellen Zustand der Datenbank unmittelbar vor dem Fehlerzeitpunkt. Die dazu notwendigen Informationen über die Endzustände der einzelnen Transaktionen (**after image**) können ebenfalls aus dem Transaktionslog entnommen werden.

Literatur zu Kapitel 3

- Korth, Silberschatz : Database System Concepts, Mc Graw Hill 1991
- C. J. Date : An Intro. to Database Systems, Vols 1 & 2, Addison-W. 1995
- Gray, J., Reuter, A.: Transaction processing, Morgan Kaufmann, 1993

Alle genannten Bücher sind in unserer Bibliothek vorhanden.