

# 12. Objektorientierte und Objekt-Relationale Datenbankkonzepte

- **DBS-Markt**
- **Beschränkungen klassischer Datenmodelle**
  - Beispiel CAD-Systeme
- **Grundkonzepte der Objektorientierung**
  - OODBS-Manifesto
  - Überblick über einige Konzepte
- **Objekt-relationale DBS – Vision**
  - verschiedene Systemarchitekturen
  - objekt-relationale DB-Technologie, Erweiterbarkeitsinfrastruktur
- **Standardisierung von SQL – Überblick<sup>1</sup>**
- **SQL:1999 – Neue Funktionalität**
- **Unterstützung für LOBs mit Einschränkungen**
  - Auswertung von Prädikaten
  - Verarbeitung und Indexierung
- **Lokator-Konzept**
  - „Verweis“ auf in DB gespeicherten LOB
  - Kapselung des Zugriffs
- **Erhöhung der Ausdrucksmächtigkeit**
  - Allgemeine Tabellenausdrücke
  - Rekursion
  - Rekursion mit Berechnungen

---

1. Information Technology – Database Language SQL - Part 1 and Part 2: Framework (for SQL:1999) and Foundation (SQL:1999), International Standard ([www.jtc1sc32.org](http://www.jtc1sc32.org))  
Information Technology – Database Language SQL - Technical Corrigendum xxx for SQL:1999,  
... [2006-02-13: >1320 Dokumente](#)

## DBS-Markt

<b>Daten- strukturen</b>	<b>komplex</b>	OO $n \cdot 10^8$ \$	OR ?
	<b>einfach</b>	Dateisysteme Video-Server	RM (SQL) $k \cdot 10^{10}$ \$
		<b>einfach</b>	<b>komplex</b>
<b>Anfragen</b>			

- **Einfache Daten, einfache Anfragen**

- Datenstruktur ist dem System nicht bekannt
- Künftig werden solche Systeme wahrscheinlich mit Anfragemöglichkeiten (z. B. SQL) ausgestattet

- **Einfache Daten, komplexe Anfragen**

- RDBS: skalierbar, robust, Zugriff über Struktur und Inhalt
- Begrenzte Unterstützung für komplexe, als BLOBs gespeicherte Daten
- RDBS können diese BLOBs nicht indexieren, manipulieren oder über ihren Inhalt suchen

- **Komplexe Daten, einfache Anfragen**

- Persistente komplexe Objekte, die durch Java, C++, Smalltalk, ... manipuliert werden
- Begrenzte Skalierbarkeit in Bezug auf große Datenvolumina und große Anzahlen von Benutzer

- **Komplexe Daten, komplexe Anfragen**

- OR-Server können komplexe Daten als Objekte handhaben
- Benutzerdefinierte Funktionen lassen sich zur Manipulation der Daten im Server heranziehen
- Erweiterbarkeit ist für Datentypen und Funktionen möglich

# Beschränkungen der klassischen Datenmodelle

- **einfach strukturierte Datenobjekte**
  - satzorientiert, festes Format
  - nur einfache Datentypen
- **geringe semantische Ausdrucksfähigkeit**
  - fehlende Abstraktionskonzepte
  - begrenzte Auswahlmächtigkeit der Anfragesprachen
- **nur einfache Integritätsbedingungen**
- **umständliche Einbettung in Programmiersprachen**
- **auf kurze Transaktionen zugeschnitten (ACID)**
- **keine Unterstützung**
  - von Zeit und Versionen
  - von räumlichen Beziehungen
- **mangelnde Effizienz und Effektivität**  
bei anspruchsvollen Anwendungen
- ...

➔ **Wie wirkt sich das bei komplexen Anwendungen aus?**

# Beispiel: CAD-Systeme

- **Eigenschaften der Daten**

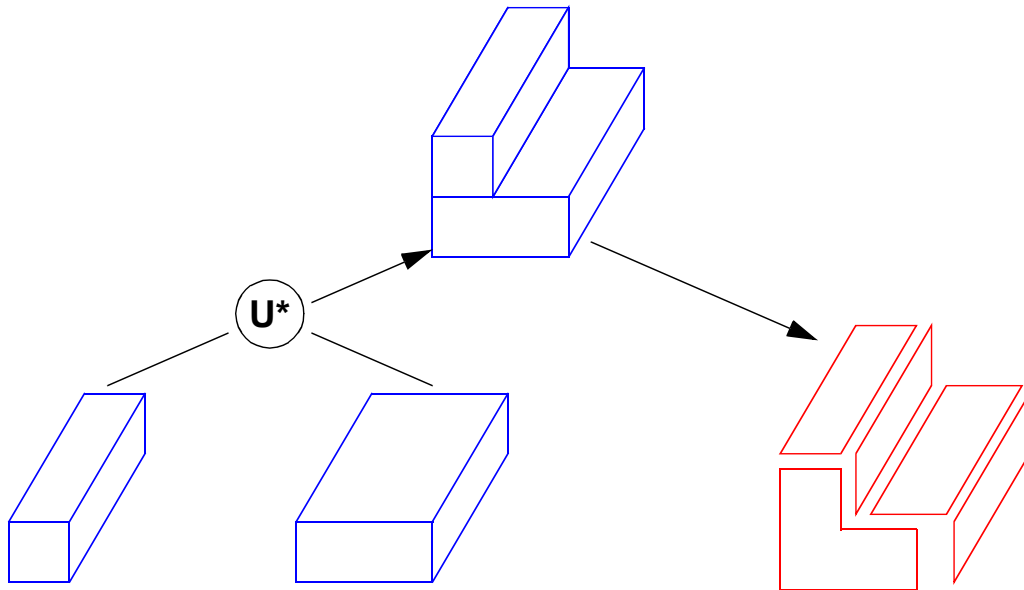
- Technische Objekte sind i. Allg. **dreidimensional**.
- Die räumlichen Daten bilden nur einen kleinen, aber wichtigen Teil der Daten des Systems.
- Die **interaktive Arbeitsweise erfordert einen schnellen Zugriff auf die räumlichen Daten** (Zeichnen am Bildschirm).
- Jedes technischen Objekt besitzt sein eigenes Bezugssystem.
- Die geometrische und topologische Information hängt vom gewählten Darstellungsschema ab.
- Die räumlichen Daten können im Raster-, Vektor- oder Hybridmodus dargestellt werden.

- **Geometrische Modellierung**

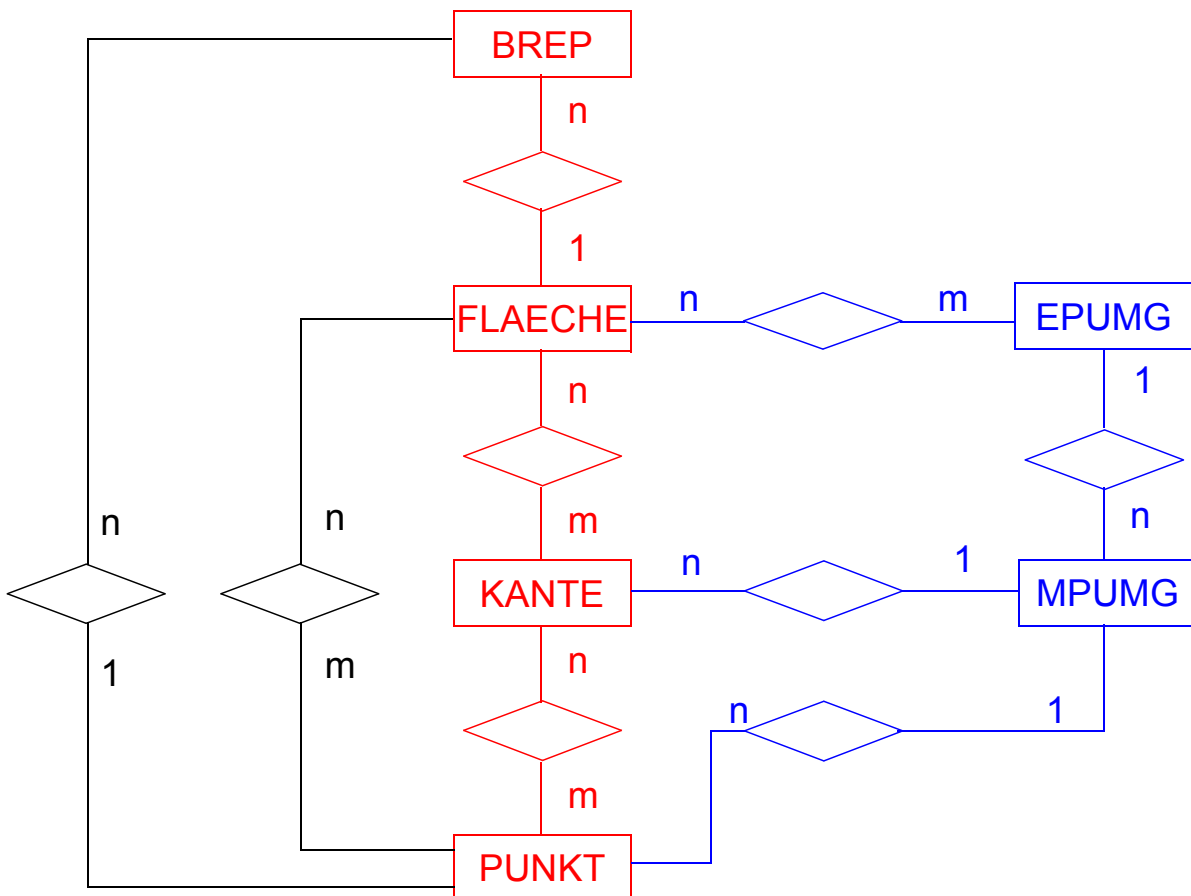
- **Konstruktionsprozess für dreidimensionale Objekte**  
(zusammengesetzte Körper)
- Volumenorientierter geometrischer Modellierer  
(CSG = Constructive Solid Geometry)
- Auswahl aus einer Menge von parametrisierten **primitiven Objekten**
- Anwendung **regulärer Operatoren** (Vereinigung, Differenz, Translation, Rotation) zur schrittweisen Konstruktion komplexerer Objekte
- System leitet automatisch **Begrenzungsflächendarstellung** ab  
(BREP = Boundary Representation)

## CAD-Systeme (2)

- CSG- und BREP-Modellierung



- DB-Repräsentation: Geometriemodell als Entity-Relationship-Diagramm:



# Darstellung im Relationenmodell

- **Abbildung**

- wegen der vielen (n:m)-Beziehungen mehr als 10 Relationen!
- Einzelteile des Körpers und ihre Beziehungen durch "unabhängige" Tupeln verschiedenen Typs dargestellt

**Schema:**

```
BREP (BID, ...)  
FLÄCHE (FID, ..., BID)  
FK-KA (FID, KID)  
KANTE (KID, K-LÄNGE, ...)  
KA-PKT (KID, PID, ...)  
PUNKT (PID, X, Y, Z, ...)  
...
```

- **Operationen**

**Anfrage:** alle Punkte, die zum Bauteil 7853 gehören und Kanten mit einer Länge von mehr als 10 Einheiten begrenzen

```
SELECT PID, X, Y, Z  
FROM PUNKT  
WHERE PID IS IN  
  
  (SELECT PID FROM KA-PKT  
   WHERE KID IS IN  
  
     (SELECT KID FROM KANTE  
      WHERE K-LÄNGE > 10  
      AND KID IS IN  
  
        (SELECT KID FROM FK-KA  
         WHERE FID IS IN  
  
           (SELECT FID  
            FROM FLÄCHE  
            WHERE BID = 7853)))));
```

## Darstellung im Relationenmodell (2)

- **Operationen (2)**

- Bohrung an einem Werkstück anbringen:  
„Subtraktion“ eines Zylinders vom bisher konstruierten Körper

- **Fazit**

- Komplexes Objekt durch heterogene Tupelmengemenge verkörpert.  
In vielfältiger Weise über Wertegleichheit von Attributen verknüpft

- **„Atomisierte“ Sicht** einzelner Tupeln

- Verlorengegangen:  
ganzheitliche Sicht des 3D-Körpers und  
die Möglichkeit seiner integrierten Behandlung

- **Objektzugriff** (Folge komplexer Verbundoperationen),
- Kontrolle von **Integritätszusicherungen**,
- **Objektmanipulationen** gemäß dem (semantisch weit höheren) Anwendungsmodell müssen mit den verfügbaren Operationen des Relationenmodells nachgebildet werden.

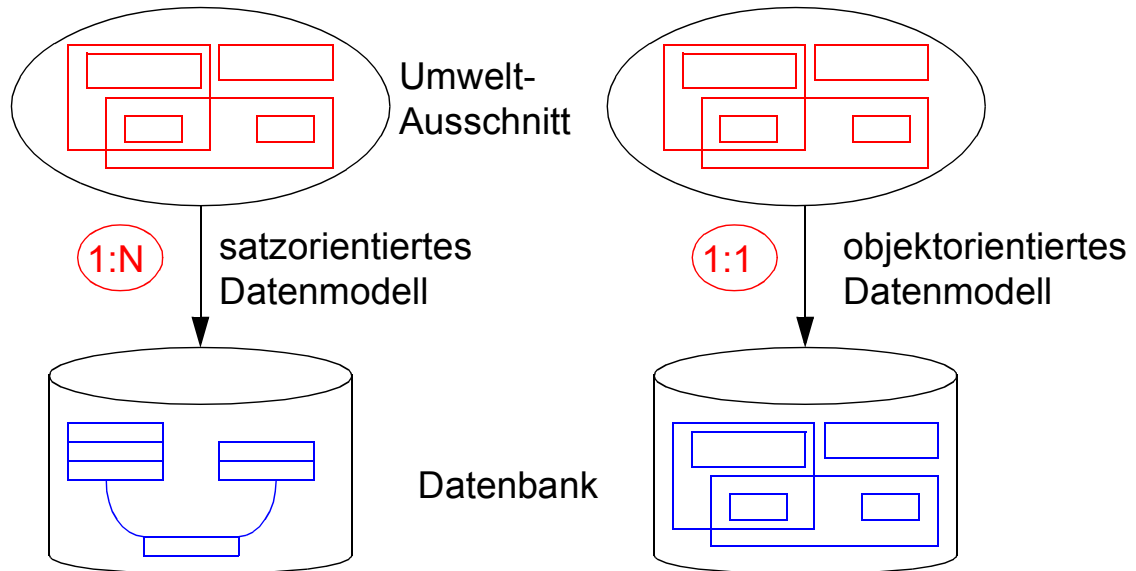
➔ **Oft Tausende von Operationen!**

- Unnatürliche Präsentation/Ausgabe komplexer Objekte
  - Ergebnis ist riesige Tabelle (oder mehrere) mit enormer Redundanz –  
wo eine Sammlung verschiedener Tupel benötigt wird
  - Anwendung muss Beziehungen der Objektstruktur auswerten

➔ **Strukturierte Ausgabe ist im RM nicht möglich!**

# Objektorientierung bei DBS

- Kernidee



- Speicherung und Suche von Objekten?
- Wirkung von Aktualisierungsoperationen?
  - Einfügen
  - Löschen
  - Kopieren, ...
- Erhaltung der Konsistenz?
- Leistungsaspekte?



# Definition eines objektorientierten DBS<sup>2</sup>

- **OODBS muss zwei Kriterien erfüllen**

- Es muss ein DBS sein.
- Es muss ein objektorientiertes System sein.

- **DBS-Aspekte:**

- Persistenz (Dauerhaftigkeit von Daten über Programmausführung hinaus)
- Sehr große Datenmengen (d. h. Zwang zur Verwendung von Externspeichern)
- Synchronisation (d. h. Mehrbenutzerbetrieb)
- Logging und Recovery (Datensicherung und -wiederherstellung)
- Deskriptive Anfragesprache (Ad-hoc-Anfragen)

- **OOS-Aspekte:**

- Grundkonzepte der Objektorientierung**

- Objektidentität
- Direkte Darstellung Komplexer Objekte
- Datenkapselung
- Typen oder Klassen, Typ-/Klassenhierarchien
- Vererbung
- Erweiterbarkeit (neue Typen, nicht unterscheidbar von systemdefinierten Typen)
- Polymorphie: Überladen (overloading) und spätes Binden
- Volle Berechenbarkeit (Mächtigkeit einer Programmiersprache)

- **Wahlweise Aspekte:**

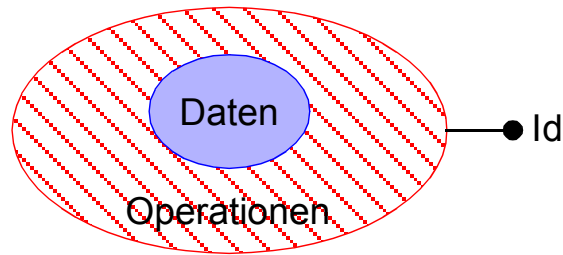
- Mehrfach-Vererbung, Typprüfung und -herleitung,
- Verteilung, Lange Transaktionen

---

2. M. P. Atkinson, et. al: „**The Object-Oriented Database System Manifesto**“, in: Won Kim, Jean-Marie Nicolas, and Shojiro Nishio (eds.), Proc. First Intl. Conf. on Deductive and Objekt-Oriented Databases, Elsevier Science Publishers, Amsterdam, 1989.

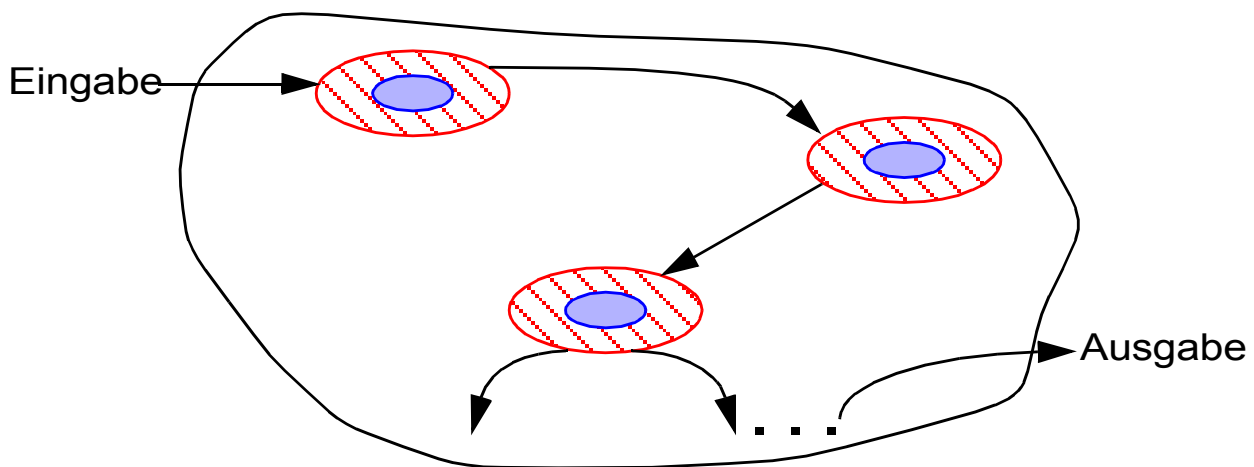
# Fundamentale Idee

- „Alles“ ist ein Objekt?



- **Objekteigenschaften**

- Objekte haben einen Identifikator
- Objekte haben einen internen Zustand, beschrieben durch Attribute (Variable, Slots, . . .)
- Objekte haben eine Schnittstelle zur externen Welt, definiert durch die Menge an Operationen
- Objekte kommunizieren über Nachrichten



- **Verarbeitungsaspekte**

- Suchen/Aktualisieren durch Methodenaufrufe
- Integritätskontrolle
- Autorisierung/Zugriffskontrolle

➔ **Ist diese Sichtweise bei DBS angemessen?**  
(Deskriptivität, Mengenorientierung, Wertbezug usw.)

# Objektidentität

- **Objektidentität**

- keine anwendungsspezifischen Werte (wie im RM)
- Identitätskonzept des Relationenmodells zu schwach (identity thru contents)
- sondern durch eindeutige Objekt-Identifikatoren (Surrogate)

- **Objekt-Identifikatoren (OIDs, Surrogate)**

- **tragen keine Semantik** ( $\leftrightarrow$  Primärschlüssel im RM)
- während der Objektlebensdauer konstant
- üblicherweise systemverwaltet

- **Eigenschaften/Konsequenzen**

- Existenz des Objektes ist **unabhängig von seinem Objektzustand**
  - Änderungen beliebiger Art (auch des Primärschlüssels im RM) ergeben *dasselbe* Objekt
- Identität  $\neq$  Gleichheit (beides ist ausdrückbar)
  - Objekte können **identisch** (dasselbe Objekt) oder **gleich** (derselbe Wert) sein
- OID zur Darstellung von **Referenzen/Beziehungen**
  - Realisierung gemeinsamer Teilobjekte ohne Redundanz möglich (referential sharing)
  - keine irreführenden Referenzen auf Objekte

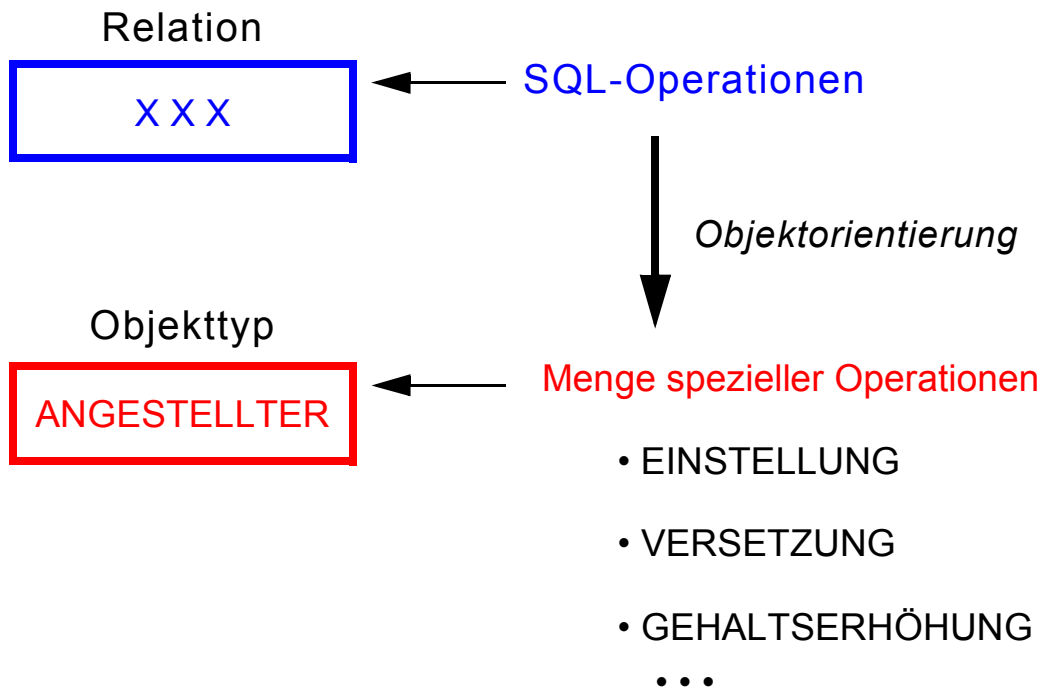
# Komplexe (strukturierte) Objekte

- **Anwendung von Typkonstruktoren –**  
Wünschenswerte Konstruktoren:
  - ARRAY-Konstruktor (VECTOR)
  - RECORD / TUPLE
  - LIST / SEQUENCE
  - SET
  - MULTISSET / BAG
- **Eigenschaften:**
  - Orthogonalität der Konstruktoren
  - beliebige (rekursive) Kombination von Konstruktoren zum Aufbau komplex strukturierter Objekte
  - Operationen zur Verarbeitung der (beliebig) strukturierten Objekte
- **Ein OODBS sollte wenigstens unterstützen:**
  - die Typkonstruktoren TUPLE, LIST und SET und
  - ihre beliebige Kombination

## ADTs / Kapselung – Beispiele

- **Objektebene**

- Unterschied zum RM



- **Attributebene: Erzeugung problembezogener Datentypen**

- mit zugeschnittenen Operatoren und Funktionen

**Beispiel:** ADT 'DATE', Operator '-'

	Normalfall	Finanzwelt
15. April – 15. März	31	30
15. März – 15. Febr.	28	30

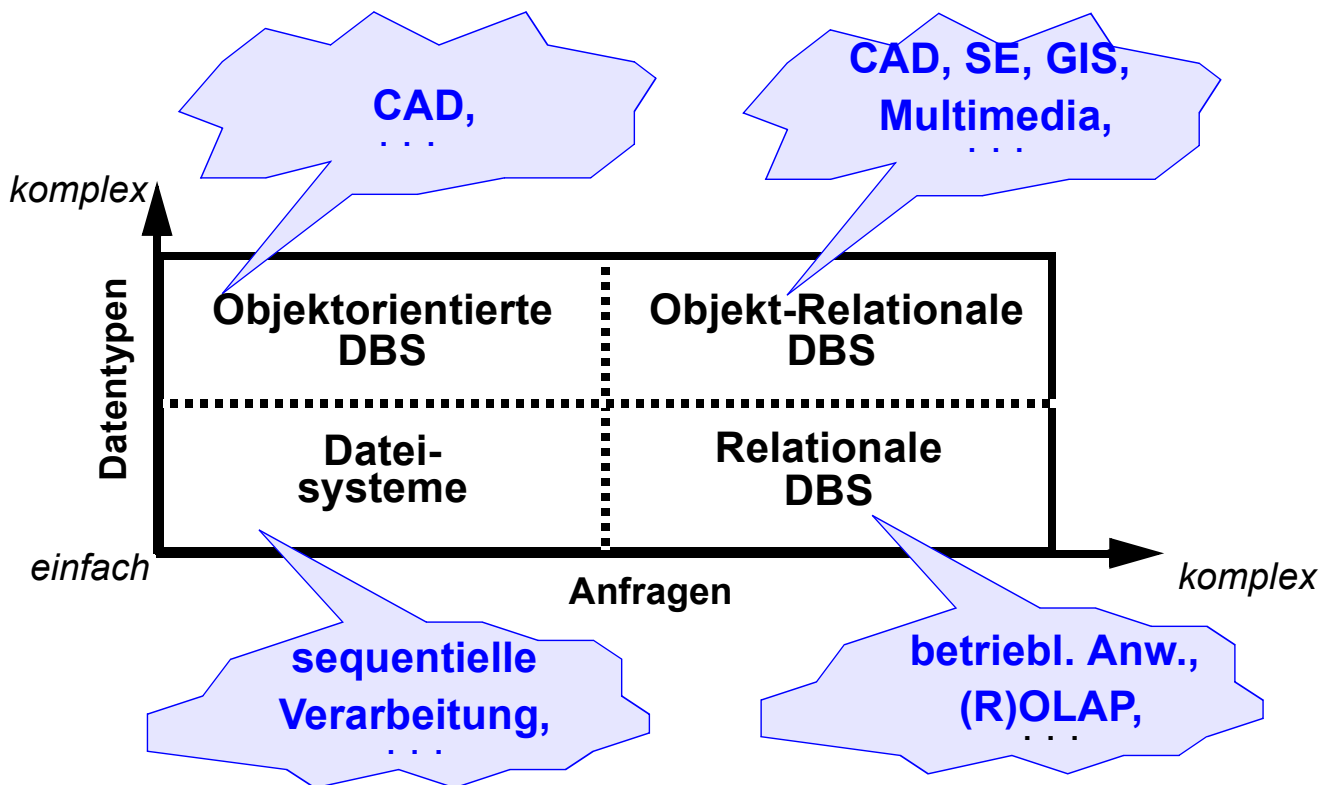
SELECT 'Beschäftigungsdauer:', HEUTE () – P.EDATUM  
FROM PERS P

- **erhöhte Datenunabhängigkeit**

- **Verwaltung der Funktionen im DBS (stored procedures)**

- ➔ **reduzierter Kommunikationsaufwand mit DBS**

# Objekt-Relationale DBS – Vision



## • Erwünschte Eigenschaften von *Objekt-Relationalen DBS (ORDBS)*

### - Eigenschaften von RDBS

- + ADTs/Kapselung
- + Klassen, Vererbung
- + mengenwertige Attribute, OIDs/Referenzen
- + benutzerdefinierte Funktionen
- + navigierende, prozedurale Verarbeitung
- + Multimedia-Integration
- + Erweiterbares Typsystem und Erweiterungsinfrastruktur
- + Client/Server-Verarbeitung
- + Offenheit
- + ... ?

### - Integration

(Leistungsverhalten, Skalierbarkeit, Bereitstellung auf Client)?

## Objekt-Relationale DBS – Entwicklungstrend

Daten- strukturen	komplex	OODBS	Universal Server
	einfach	Dateisysteme Video-Server	
		einfach	komplex
Anfragen			

- **DBS**, die VITA-Daten (*Video, Image, Text, Audio*) handhaben können, werden auch **Universal Server** genannt; viele Erweiterungen (*spatial types, time series, ...*) werden laufend entwickelt
- **Erweiterbare DBS erfordern erweiterbare Konzepte**
  - Integration von AW-Funktionen (in 3GL) in den DB-Server (Weiterentwicklung des Konzeptes der Stored Procedures)
  - Benutzung einer CALL-Schnittstelle oder von eingebettetem SQL
  - 4G-Sprachen (z. B. NewEra) lassen sich erweitern mit C++ und OLE
  - Plattformunabhängigkeit lässt sich durch Web-Applikationen erzielen
  - Java-Client-Applikationen können für sich Anwendungscode in Form von Java-Applets aus dem Web laden
    - Java-Applets werden in Intranets oder im Internet gespeichert
    - Applets laufen als Client-Applikationen ab
- **Wettbewerber**
  - Oracle 11i (10g Express Edition für Studenten), Informix Dynamic Server, IBM DB2 Univ. Database V8.2 (5 Produktlinien u. a. Enterprise Server Edition)
  - Sybase Adaptive Server, CA Associates (OpenIngres ++?)
  - SAG (Adabas D++), Microsoft SQLServer, . . .

# Objekt-Relationale DB-Konzepte: Motivation

- **Relationale Datenbankverwaltungssysteme bieten**

- eine Menge von Datentypen, um Anwendungsdaten darstellen zu können
- eine Menge von Operationen, um diese Datentypen manipulieren zu können

- **Beispiele:**

**Datentypen**

**Funktionen**

INTEGER

+, -, \*, /, AVG, SUM, ...

CHARACTER

Manipulation von Zeichenketten:  
suchen, anfügen, ...

DATE

Tag, Monat, Jahr, +, -, ...

- **Neue Anwendungen erfordern neue Datentypen und Funktionen!**

- **Beispiele:**

**Datentypen**

**Funktionen**

TEXT

Volltextsuche,  
Rechtschreibkorrektur, ...

POLYGON

Durchmesser, Schnitt  
von Polygonen, Fläche, ...

RASTER

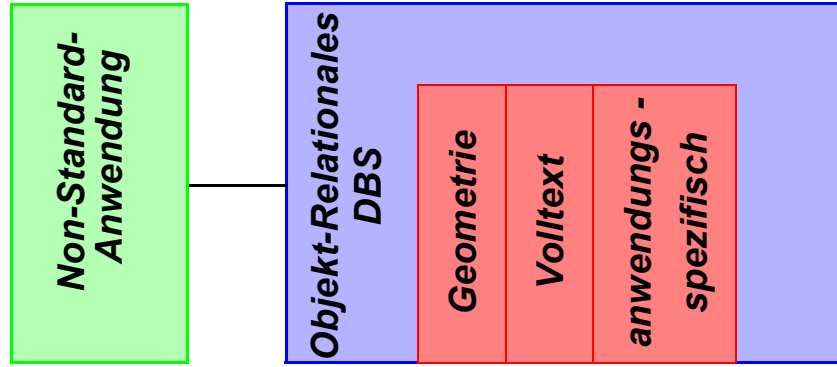
Konversion zwischen  
Formaten, Farbanalyse



# Drei verschiedene DBS-Architekturen

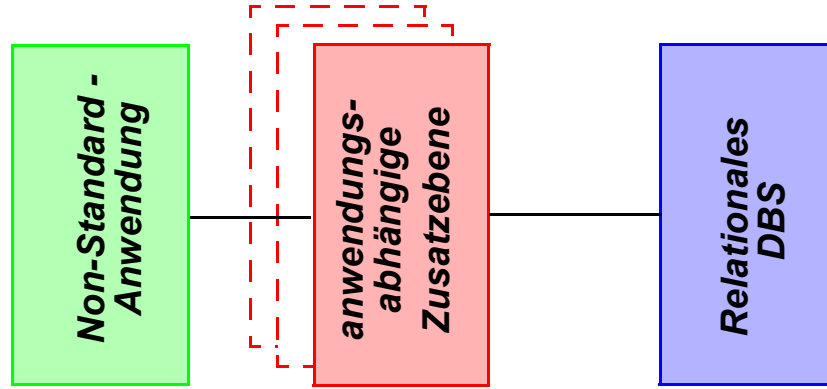
## Erweiterbares DBS

### DBS

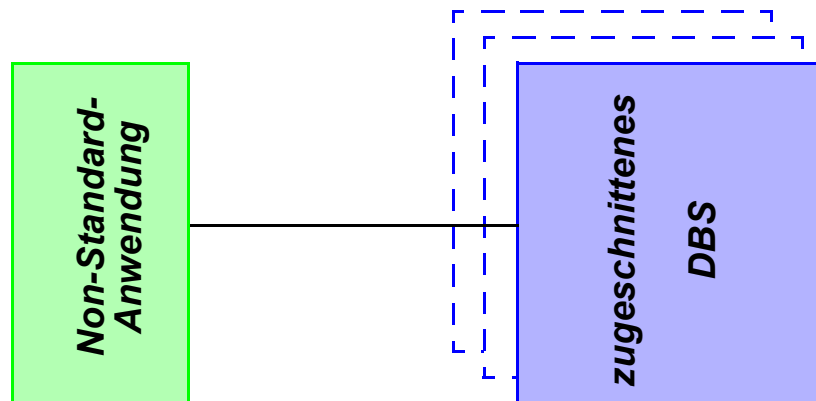


## DBS mit Zusatzebene

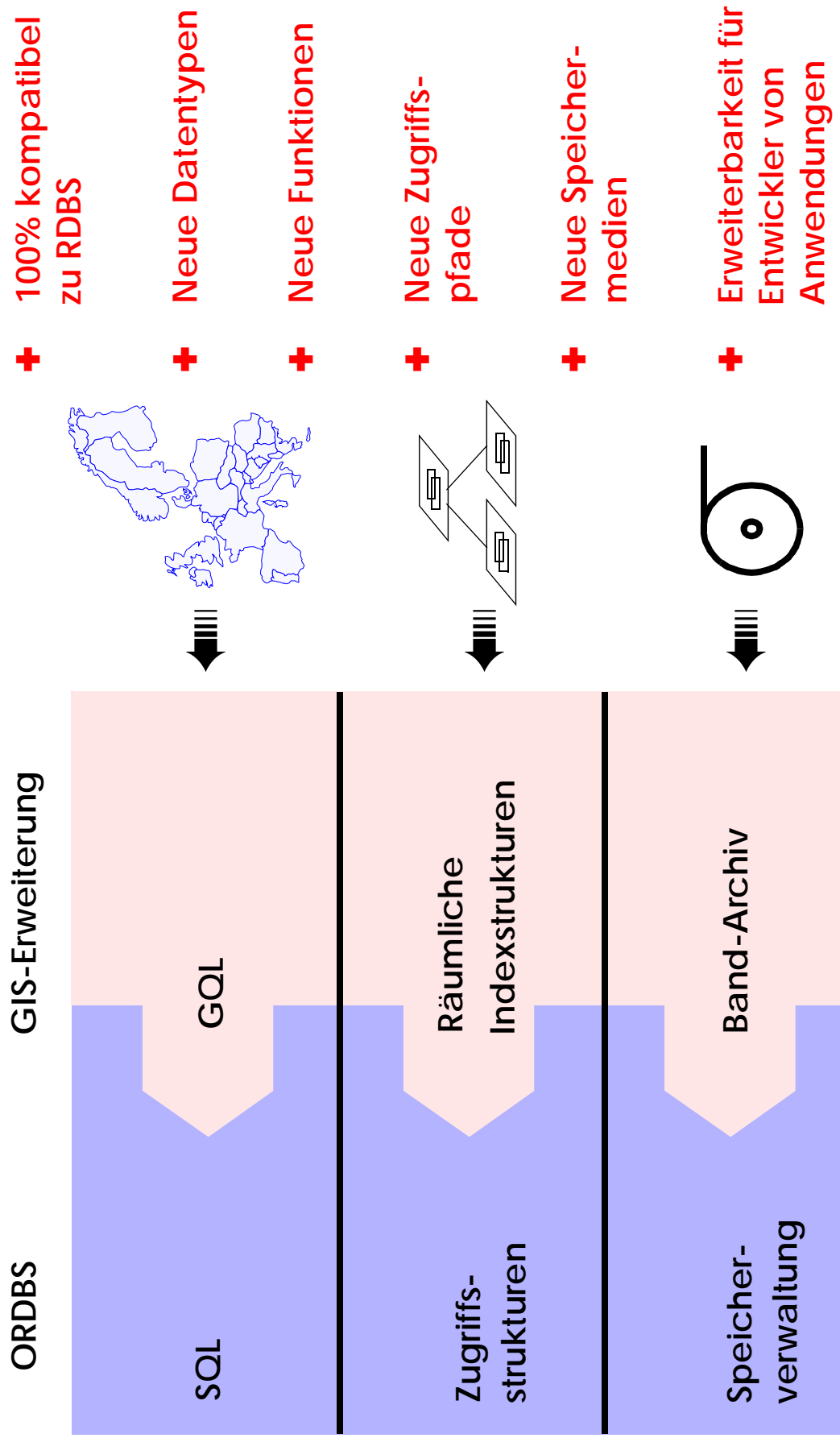
### Zusatzebene



## Spezial-DBS



# Objekt-Relationale DB-Technologie am Beispiel Geographische Informationssysteme (GIS)



# Objekt-Relationale DB-Technologie

- **Funktionalität wird derzeit im wesentlichen durch den Standard SQL:1999 beschrieben**
- **Erhöhung der Anfragemächtigkeit**
  - Allgemeine Tabellenausdrücke
  - Rekursion
  - Große Objekte
- **Unterstützung von benutzerdefinierten Typen (UDT) bzw. Objektorientierung**
  - komplexe Datenstrukturen mit
  - komplexer Funktionalität definierbar
  - Vererbungshierarchie
  - . . .

➔ **Repräsentation von Anwendungswissen im DB-System (Klassen-Bibliotheken)**
- **Erweiterung von herkömmlichen Tabellen**
  - komplexe Spalten (Attribute, Wertebereiche)
  - Schachtelung
  - Referenzierung/Dereferenzierung
  - Tabellen mit Typbindung (typed tables) und Tabellenhierarchien
  - . . .
- **Erweiterungsinfrastruktur**
  - benutzerdefinierte Datentypen und Funktionen lassen sich in das ORDBS integrieren und sind in SQL nutzbar
  - Unterstützung durch spezielle Zugriffspfade und Speicherungsstrukturen
  - Integration mit DBS-Komponenten wie Anfrageoptimierer, Synchronisation, Logging und Recovery

# Standardisierung von SQL

- **Standardisierung durch ISO JTC 1/SC 32/WG 3 DBL**

SC 32: Data Management and Interchange

WG 3: Database – Rapporteur Groups

DBL: Database Languages

- **Geschichte der SQL-Normung:**

SQL-86	ISO 9075	1987
SQL-89	ISO/IEC 9075	1989
SQL-92	(SQL2)	ISO/IEC 9075 1992
SQL:1999	(SQL3)	ISO/IEC 9075-1/2 1999
SQL:2003 <sup>3</sup>	(SQL4)	ISO/IEC 9075-1/2 2003

(IEC= Intl. Electrotechnical Commission)

- **Arbeit seit 1990 an SQL:1999**

- weitreichende Erweiterung von SQL-92

- **Parallel dazu: vorbereitende Arbeiten an SQL:2003 seit 1996 und jetzt an SQL:2007**

---

3. Information Technology – Database Language SQL - Part 1 and Part 2: Framework (for SQL:200n) and Foundation (SQL:200n), International Standard, Dezember 2003 ([www.jtc1sc32.org](http://www.jtc1sc32.org))

# SQL:1999 als richtungsweisender DB-Standard

- **Standardisierungsprozess**

- Teilnehmer: DB-Hersteller und Anwender, mehr als 20 Länder, ANSI
- Konsens zwischen Teilnehmern wird angestrebt

- **SQL:1999 hat mehrere Teile**

- SQL/Foundation (Part 2), SQL/CLI (Part 3), SQL/PSM (Part 4)
- SQL/Language Bindings (Part 5), **SQL/MED** (Mgmt. of External Data) (Part 9)
- SQL Object Language Bindings (Part 10)
- SQL/JRT (Part 13), . . .
- für SQL:200n zusätzlich noch: SQL/Schemata (Part11), SQL/XML (Part 14)

- **Weiterer auf SQL:1999 aufbauender Standard:**

- SQL Multimedia and Application Packages (SQL/MM)**

- Framework, Full-Text
- Spatial, Still Image
- Data Mining

## SQL als Datenbanksprache: DDL, DML, DCL

- **DDL: Definition von Daten**

Wie sehen die Daten der Anwendung aus?

- **DML: Manipulation von Daten**

Wie können die Daten abgefragt und manipuliert werden?

- **DCL: Kontrolle des Datenbankzugriffs**

Wer hat Zugriff auf welche Daten?

- **Administration von Datenbanken**

Leistung des Systems, ...

# Objekt-Relationale Anfragemöglichkeiten – Beispiel

- **Integrierte Suche über Inhalt**

- SQL ermöglicht den einheitlichen Zugriff auf herkömmliche und neue Datentypen
- Eine Anfrage kann sich auf ALLE Datentypen zugleich erstrecken
- Es können dabei benutzerdefinierte Datentypen und Funktionen ausgenutzt werden

- **Intuitives Anfragebeispiel**

„Finde die Kunden und ihre Versicherungsnummern, die Unfälle hatten, wobei Motorhauben von roten Autos schwer beschädigt wurden und die sich innerhalb von 5 km von Ausfahrten der Autobahn 61 ereigneten“

**SELECT** Kundenname, Versicherungsnummer

**FROM** Unfälle U, Autobahnausfahrten A

**WHERE** **CONTAINS**(U.Bericht, “Schaden”

**Textdaten**



**IN SAME SENTENCE AS**

“schwer” **AND** (“Motorhaube” **OR** “Blech”))

**AND** A.Nummer = 61

**herkömmliche Attribute**



**AND** **SCORE**(U.Bild, “rot”) > 0.6

**Bilddaten**



**AND** **DISTANCE**(A.Ausfahrt, U.Ort) < km (5);

**räumliche Daten**



# Große Objekte

- **Anforderungen**

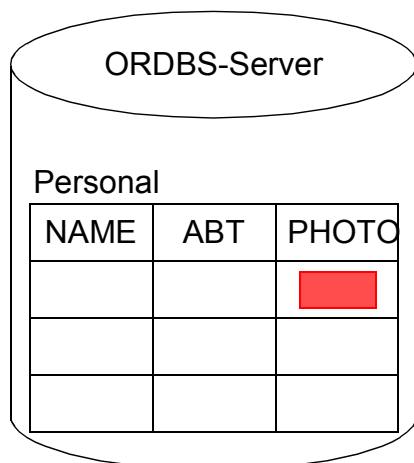
- idealerweise keine Größenbeschränkung
- allgemeine Verwaltungsfunktionen
- zugeschnittene Verarbeitungsfunktionen, . . .

- **Beispiele für große Objekte (heute bis n (=2) GByte)**

- Texte, CAD-Daten
- Bilddaten, Tonfolgen
- Videosequenzen, . . .

- **Prinzipielle Möglichkeiten der DB-Integration**

**Speicherung als LOB in der DB** (meist indirekte Speicherung)

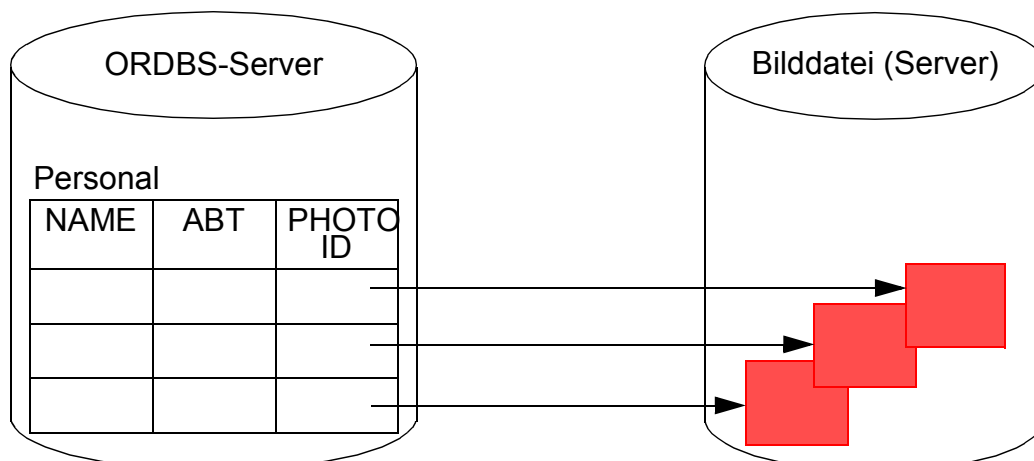


**BLOB** - Binary Large Object  
für Tonfolgen, Bilddaten usw.

**CLOB** - Character Large Object  
für Textdaten

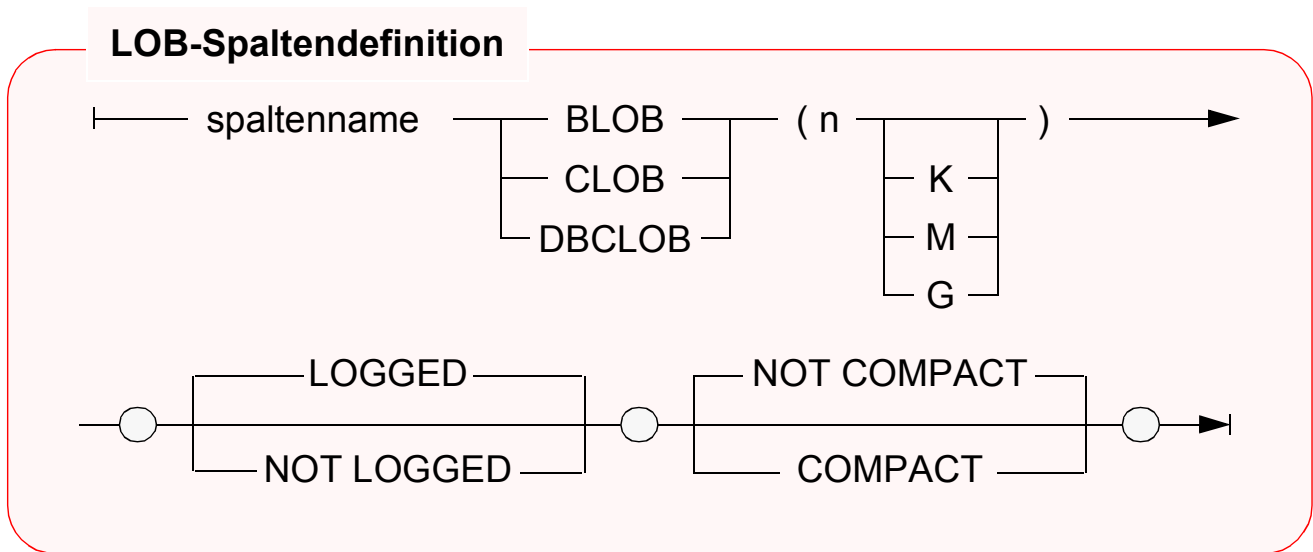
**DBCLOB** - Double Byte Character  
Large Object (DB2)  
für spezielle Graphikdaten usw.

**Speicherung mit DataLinks-Konzept in externen Datei-Servern**



## Große Objekte (2)

- Erzeugung von LOB-Spalten<sup>4</sup>



- Beispiele

CREATE TABLE	Absolvent	
(Lfdnr	Integer,	
Name	Varchar (50),	
...		
<b>Photo</b>	BLOB (5 M) NOT LOGGED COMPACT,	-- <b>Bild</b>
<b>Lebenslauf</b>	CLOB (16 K) LOGGED NOT COMPACT);	-- <b>Text</b>
CREATE TABLE	Entwurf	
(Teilnr	Char (18),	
Änderungsstand	Timestamp,	
Geändert_von	Varchar (50)	
<b>Zeichnung</b>	BLOB (2 M) LOGGED NOT COMPACT);	-- <b>Graphik</b>
ALTER TABLE	Absolvent	
ADD COLUMN	<b>Diplomarbeit</b> CLOB (500 K)	
	LOGGED NOT COMPACT;	

4. Die Realisierungsbeispiele beziehen sich auf DB2 – Universal Database



## Große Objekte (3)

- Spezifikation von LOBs erfordert Sorgfalt

- maximale Länge

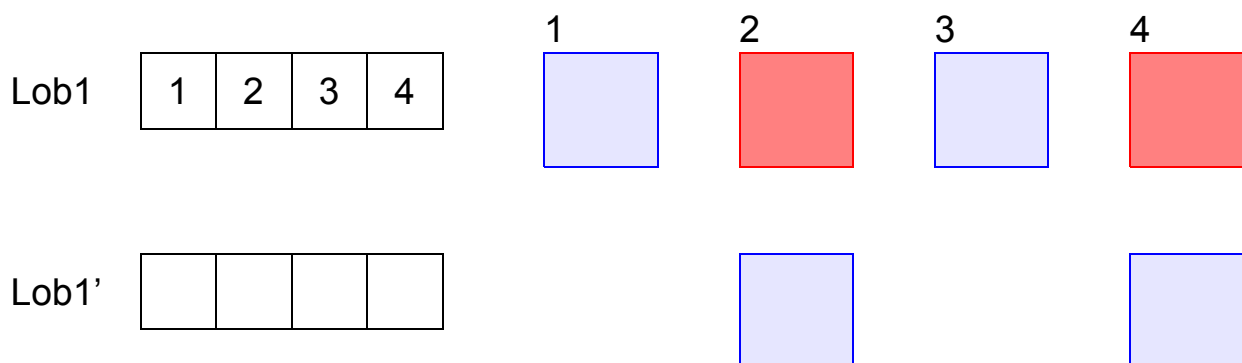
- Reservierung eines Anwendungspuffers
- Clusterbildung und Optimierung durch indirekte Speicherung; Deskriptor im Tupel ist abhängig von der LOB-Länge (72 Bytes bei <1K - 316 Bytes bei 2G)
- bei kleinen LOBs (< Seitengröße) direkte Speicherung möglich

- kompakte Speicherung

- **COMPACT** reserviert keinen Speicherplatz für späteres Wachstum
  - ➔ Was passiert bei einer LOB-Aktualisierung?
- **NOT COMPACT** ist Default

- Logging

- **LOGGED**: LOB-Spalte wird bei Änderungen wie alle anderen Spalten behandelt (ACID!)
  - ➔ Was bedeutet das für die Log-Datei?
- **NOT LOGGED**: Änderungen werden nicht in der Log-Datei protokolliert. Sog. Schattenseiten (shadowing) gewährleisten Atomarität bis zum Commit



➔ Was passiert bei Gerätefehler?

## Große Objekte (4)

- **Wie werden große Objekte verarbeitet?**

- BLOB und CLOB sind keine Typen der Wirtssprache

- ➔ **Spezielle Deklaration von BLOB, CLOB, ... durch SQL TYPE ist erforderlich,** da sie die gleichen Wirtssprachentypen benutzen. Außerdem wird sichergestellt, dass die vom DBS erwartete Länge genau eingehalten wird.

- **Vorbereitungen im AWP erforderlich**

- SQL TYPE IS CLOB (2 K) c1 (oder BLOB (2 K))  
wird durch C-Precompiler übersetzt in

```
static struct c1_t
{
    unsigned long length;
    char data [2048];
} c1;
```

- Erzeugen eines CLOB

```
c1.data = 'Hello';
c1.length = sizeof ('Hello')-1;
```

kann durch Einsatz von Makros (z. B. `c1 = SQL_CLOB_INIT('Hello');`)  
verborgen werden

- **Einfügen, Löschen und Ändern**

kann wie bei anderen Typen erfolgen, wenn genügend große AW-Puffer  
vorhanden sind

- **Hole die Daten des Absolventen mit Lfdnr. 17 ins AWP**

```
...
SELECT Name, Photo, Lebenslauf
INTO :x, :y :yindik, :z :zindik
FROM Absolvent
WHERE Lfdnr = 17;
```

## Große Objekte (5)

```
void main ( ) /* Beispielprogramm */
{ /* Verarbeitung von Filmkritiken auf */
    /* Tabelle Filme (Titel, Besetzung, Kritik) */

    EXEC SQL INCLUDE SQLCA;
    EXEC SQL BEGIN DECLARE SECTION;
        char dbname[9] = "Filmdb"; /* Name der Datenbank*/
        char msgbuffer[500]; /* Puffer für DB2-Fehlermeldungen*/
        char titel[100]; /* für Varchar-Daten*/
        SQL TYPE is CLOB (50 K) kritik; /* Ausgabe-Clob-Struktur*/
        SQL TYPE is CLOB (50 K) neuekritik; /* Eingabe-Clob-Struktur*/
        short indikator1, indikator2; /* Indikator-Variable */
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR GO TO schlechtenachrichten;
    EXEC SQL CONNECT TO :dbname;

    strcpy (neuekritik.data, "Bullet ist ein ziemlich guter Film.");
    neuekritik.length = strlen (neuekritik.data);
    indikator1 = 0;
    EXEC SQL
        UPDATE Filme
        SET Kritik = :neuekritik :indikator1
        WHERE Titel = 'Bullet';
    EXEC SQL COMMIT;
    EXEC SQL DECLARE f1 CURSOR FOR
        SELECT Titel, Kritik
        FROM Filme
        WHERE Besetzung LIKE '%Steve McQueen%';
    EXEC SQL WHENEVER NOT FOUND GO TO close_f1;
    EXEC SQL OPEN f1;
    WHILE (1)
    {
        EXEC SQL FETCH f1 INTO :titel, :kritik :indikator2;
        /* Angabe eines eigenen Nullterminierers */
        kritik.data[kritik.length] = '\0';
        printf(„\nTitel: %s\n“, titel);
        if (indikator2 < 0)
            printf ("Keine Kritik vorhanden\n");
        else
            printf(“%s\n“, kritik.data);
    }
    close_f1:
        EXEC SQL CLOSE f1;
        return;
    schlechtenachrichten:
        printf ("Unerwarteter DB2-Return-Code.\n");
        sqlaintp (msgbuffer, 500, 70, &sqlca);
        printf ("Message: &s\n“, msgbuffer);
    } /* End of main */
```

## Große Objekte (6)

- Welche Operationen können auf LOBs angewendet werden?

- Vergleichsprädikate: =, <>, <, <=, >, >=, IN, BETWEEN

- LIKE-Prädikat

- Eindeutigkeit oder Reihenfolge bei LOB-Werten

- PRIMARY KEY, UNIQUE, FOREIGN KEY
- SELECT DISTINCT, . . ., COUNT (DISTINCT)
- GROUP BY, ORDER BY

- Einsatz von Aggregatfunktionen wie MIN, MAX

- Operationen

- UNION, INTERSECT, EXCEPT
- Joins von LOB-Attributen

- Indexstrukturen über LOB-Spalten

- Wie indexiert man LOBs?

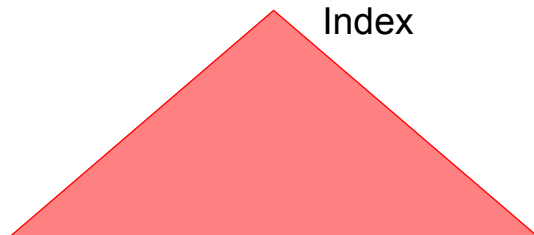
- Benutzerdefinierte Funktion ordnet LOBs Werte zu

- Funktionswert-Indexierung

$f(\text{blob1}) = x$

Blob-  
Index

blob1



## Große Objekte (7)

- **Verarbeitungsanforderungen bei LOBs**

- Verkürzen, Verlängern und Kopieren
- Suche nach vorgegebenem Muster, Längenbestimmung
- Stückweise Handhabung (Lesen und Schreiben), . . .
  - ➔ Einsatz von Funktionen bietet manchmal Ersatzlösungen

- **Funktionen für CLOBs und BLOBs**

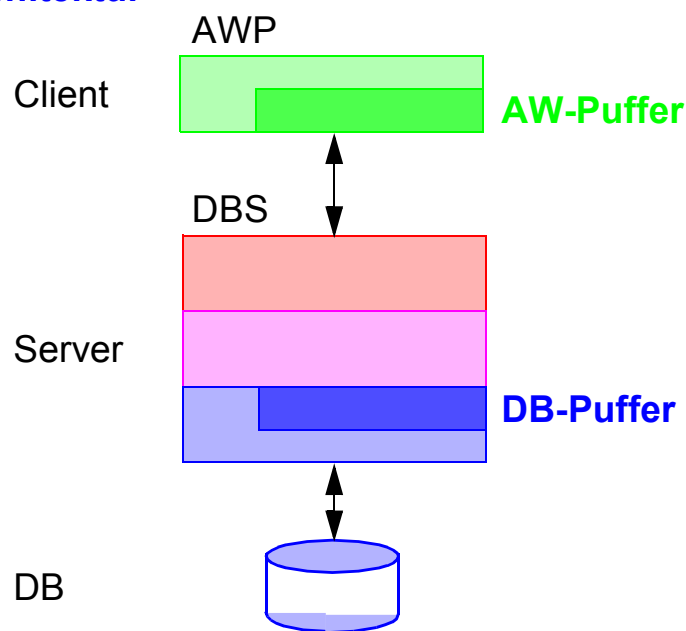
- string1 || string2 oder **CONCAT** (string1, string2)
  
- **SUBSTRING** (string **FROM** start [ **FOR** length ])
  
- **LENGTH** (expression)
  
- **POSITION** (search-string **IN** source-string)
  
- **OVERLAY** (string1 **PLACING** string2 **FROM** start [ **FOR** length ])
  
- **TRIM** ([ [ {**LEADING** | **TRAILING** | **BOTH**} ] [ string1 ] **FROM** ] string2)
  
- . . .

## Große Objekte (8)

- Ist die direkte Verarbeitung von LOBs im AWP realistisch?

Bücher		EXEC SQL
(Titel	Varchar (200),	SELECT Kurzfassung, Buchtext, Video
BNR	ISBN,	INTO :kilopuffer, :megapuffer, :gigapuffer
Kurzfassung	CLOB (32 K),	
Buchtext	CLOB (20 M),	FROM Bücher
Video	BLOB (2 G))	WHERE Titel = 'American Beauty'

- Client/Server-Architektur



- Allokation von Puffern?
- Transfer eines ganzen LOB ins AWP?
- Soll Transfer über DBS-Puffer erfolgen?
- „Stückweise“ Verarbeitung von LOBs durch das AWP erforderlich!
  - ➔ Lokator-Konzept für den Zugriff auf LOBs

# Lokator-Konzept

## • Ziel

- Minimierung des Datenverkehrs zwischen Client und Server:  
Es sollen „stückweise“ **so wenig** LOB-Daten **so spät wie möglich** ins AWP übertragen werden
- **noch besser:** Bereitstellung von Server-Funktionen  
Durchführung von Operationen auf LOBs durch das DBMS

## • Lokator-Datentyp

- Wirtsvariable, mit der ein LOB-Wert referenziert werden kann
  - In C wird long als Datentyp benutzt (4-Byte-Integer)
  - Jedoch Typisierung erforderlich
    - SQL TYPE IS BLOB\_LOCATOR
    - SQL TYPE IS CLOB\_LOCATOR
- Identifikator für persistente und flüchtige DB-Daten

## • Anwendung

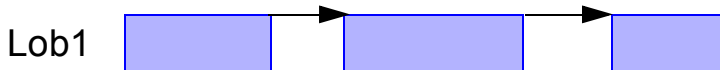
```
EXEC SQL BEGIN DECLARE SECTION;  
  SQL TYPE IS BLOB_LOCATOR Video_Loc;  
EXEC SQL END DECLARE SECTION;  
EXEC SQL  
  SELECT  Video  
  INTO    :Video_Loc  
  FROM    Bücher  
  WHERE   Titel = 'American Beauty'
```

- Ein Lokator kann überall dort eingesetzt werden, wo ein LOB-Wert verwendet werden kann
  - Wirtsvariable (z. B. in UPDATE-Anweisung)
  - Parameter von Routinen
  - Rückgabewerte von Funktionen
- **Wie lange ist eine Lokator-Referenz gültig?**

## Lokator-Konzept (2)

- Lokatoren können LOB-Ausdrücke repräsentieren

- Innerhalb des DB-Servers entspricht jeder Lokator einer Art „**Rezept**“ zum Zusammenbau eines LOB-Wertes aus an unterschiedlichen Stellen gespeicherten Fragmenten



- Ein LOB-Ausdruck ist ein Ausdruck, der auf eine LOB-Spalte verweist oder einen LOB-Datentyp als Ergebnis hat. Er kann LOB-Funktionen beinhalten.
- LOB-Ausdrücke können andere Lokatoren referenzieren.

- **Beispiel**

```
SELECT
  SUBSTRING (Buchtext FROM
    POSITION ('Kapitel 1' IN Buchtext) FOR (
    POSITION ('Kapitel 2' IN Buchtext) -
    POSITION ('Kapitel 1' IN Buchtext)))
  INTO      : Kap1Loc
FROM      Bücher
WHERE     Titel = 'American Beauty'
```



## Lokator-Konzept (3)

- **Mächtigkeit des Lokator-Konzeptes**

- Ein Lokator repräsentiert immer einen konstanten Wert
- Operationen auf LOBs werden nach Möglichkeit indirekt mit Hilfe ihrer Verweise („Rezepte“) vorgenommen

**CONCAT** (:loc1, :loc2) erzeugt einen neuen Verweis, **ohne die physische Konkatenation der LOBs vorzunehmen**

- Ein Anlegen oder Kopieren von LOBs erfolgt nur
  - beim Aktualisieren einer LOB-Spalte
  - bei der Zuweisung eines LOB-Wertes zu einer Wirtsvariablen

- **Einsatz von Lokator-Variablen**

- LENGTH ( :loc1 )
- POSITION ( 'Schulabschluss' IN :loc2 )
- SUBSTRING ( :loc3 FROM 1200 FOR 200 )
- EXEC SQL VALUES  
SUBSTRING ( :loc1 FROM POSITION ( 'Schulabschluss' IN :loc1 )  
FOR 100 ) INTO :loc2

- **Lebensdauer von Lokatoren**

- **Explizite Freigabe**

```
EXEC SQL  
FREE LOCATOR :loc1, :loc2;
```

- **Transaktionsende** (non-holdable locators)
- **Sitzungsende**

```
EXEC SQL  
HOLD LOCATOR :loc1;
```

## Lokator-Konzept (4)

- **Beispielprogramm Theaterstück:**

Korrektur eines Textes in Tabelle Theaterstücke (Titel, Text, ...)

```
void main ( )
{
    EXEC SQL INCLUDE SQLCA;

    EXEC SQL BEGIN DECLARE SECTION;
        char dbname[9] = "Stüickedb";           /* Name der Datenbank */
        char msgbuffer[500];                   /* Puffer für DB2-Fehlermeldungen */
        SQL TYPE IS CLOB_LOCATOR loc1, loc2;
        long n;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR GO TO schlechtenachrichten;
    EXEC SQL CONNECT TO :dbname;
    EXEC SQL      SELECT  Text INTO :loc1
                  FROM    Theaterstücke
                  WHERE   Titel = 'As You Like It';
    EXEC SQL VALUES POSITION ( 'colour' IN :loc1 ) INTO :n;

    while (n > 0)
    {
        EXEC SQL VALUES SUBSTRING ( :loc1 FROM 1 FOR :n-1 ) || 'color'
                  || SUBSTRING (:loc1 FROM :n+6) INTO :loc2;

        /*
        ** Gib alten Lokator frei und behalte den neuen.
        */
        EXEC SQL FREE LOCATOR :loc1;
        loc1 = loc2;
        EXEC SQL VALUES POSITION ( 'colour' IN :loc1 ) INTO :n;
    }

    /*
    ** Es wurden noch keine Daten bewegt; es wurden lediglich neue Lokatoren erzeugt.
    */
    EXEC SQL UPDATE Theaterstücke SET Text = :loc1
              WHERE Titel = 'As You Like It';

    /*
    ** Jetzt wird der neue Text zusammengesetzt
    ** und der DB-Tabelle Theaterstücke zugewiesen.
    */
    EXEC SQL COMMIT;
    return;
    ...
}
```

# Allgemeine Tabellenausdrücke

- Gegeben: Pers (Pnr, Anr, Mnr, Gehalt, Bonus)
- Q1: Finde Abteilung (Anr) mit höchster Gehaltssumme
- Versuch einer Lösung für Q1

```
CREATE VIEW Gehaltsliste (Anr, Gesamt) AS
  SELECT Anr, SUM (Gehalt) + SUM (Bonus)
  FROM Pers
  GROUP BY Anr;
```

- Viele DBS erlauben auch komplexe Anfragen auf Sichten (ggf. über eine Sichtenmaterialisierung)
- Beispiel:

Gehaltsliste	Anr	Gesamt
	K03	389 K
	K51	794 K
	K55	1012 K

- **Referenz auf Sicht**

```
SELECT Anr, Gesamt
FROM Gehaltsliste
WHERE Gesamt = (SELECT MAX(Gesamt) FROM Gehaltsliste);
```

- Sicht muss nur für die Anfrage im Systemkatalog angelegt und wieder gelöscht werden

➔ Umständliche Vorgehensweise

- Gibt es, auch für die mehrfache Verwendung von Sichten, bessere Lösungen?

## Allgemeine Tabellenausdrücke (2)

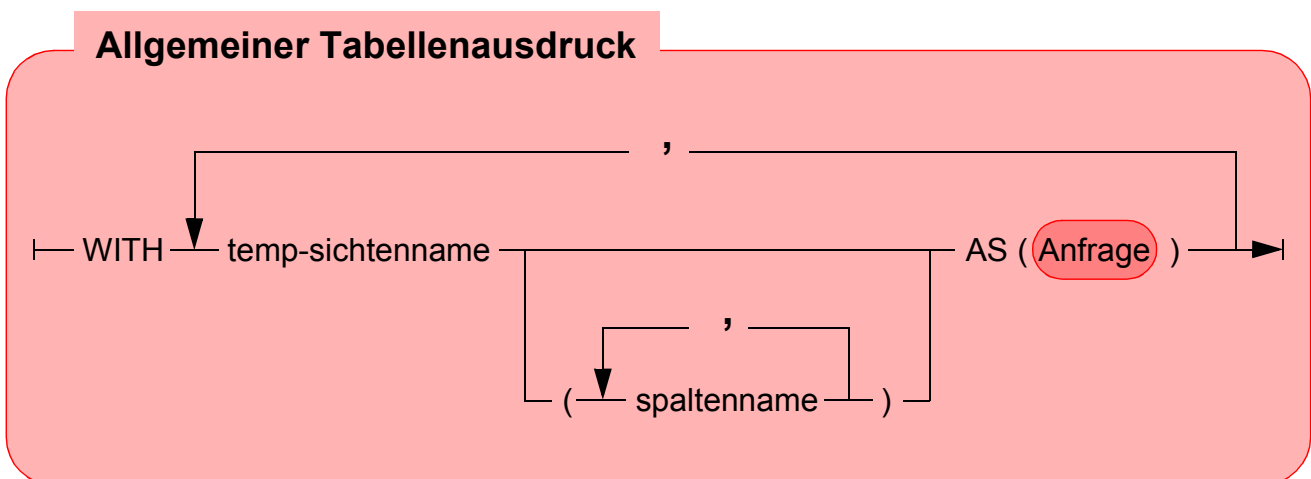
- **Geschlossene Lösung für Q1**

```
SELECT Anr, Gesamt
FROM ( SELECT Anr, SUM (Gehalt) + SUM (Bonus) AS Gesamt
      FROM Pers
      GROUP BY Anr) AS Gehaltsliste1

WHERE Gesamt =
      ( SELECT MAX (Gesamt)
        FROM ( SELECT Anr, SUM (Gehalt) + SUM (Bonus) AS Gesamt
              FROM Pers
              GROUP BY Anr) AS Gehaltsliste2);
```

- Derselbe Tabellenausdruck wird in einer Anfrage mehrfach ausgewertet
- Auswertung erfolgt unabhängig, was zu Inkonsistenzen führen kann (bei einer Konsistenzstufe schwächer als „Repeatable Read“)

- **Neues Konzept**



- erlaubt mehrfache Referenz, ohne eine Sicht materialisieren zu müssen
  - ➔ **Allgemeiner Tabellenausdruck definiert eine oder mehrere Sichten für die Verarbeitung der SQL-Anweisung**

## Allgemeine Tabellenausdrücke (3)

- **Neuformulierung von Q1**

```
WITH Gehaltsliste (Anr, Gesamt) AS  
  ( SELECT Anr, SUM (Gehalt) + SUM (Bonus)  
    FROM Pers  
    GROUP BY Anr)
```

```
SELECT Anr, Gesamt  
FROM Gehaltsliste  
WHERE Gesamt =  
  ( SELECT MAX (Gesamt)  
    FROM Gehaltsliste);
```

- einmalige Auswertung der Sicht, Optimierung durch das DBS

- **Größere Flexibilität**

- Explizite Sichten sind im Systemkatalog „kontextlos“ definiert und erlauben keine Parametrisierung
- WITH-Sichten sind im Kontext einer SQL-Anweisung definiert
  - Parametrisierung möglich, z. B. alle Abteilungen kleiner x
- Wann werden die Wirtsvariablen gebunden?
- Verbunde und Selbstverbunde sind möglich  
(Abteilungen mit mehr als der doppelten Gehaltssumme als andere)

# Rekursion

- **Was ist rekursives SQL?**

- Ein allgemeiner Tabellenausdruck ist rekursiv, falls er in seiner Definition (WITH-Klausel) auf sich selbst Bezug nimmt
- Einsatz von selbstreferenzierenden Tabellenausdrücken
  - bei temporären und permanenten Sichten
  - bei INSERT-Anweisungen

- **Warum nutzt man Rekursion in SQL?**

- **deskriptive und mengenorientierte Formulierung**
  - Gewinn an Ausdrucksmächtigkeit
  - verbessertes Leistungsverhalten
- **Traversierung von Baum- und Netzwerkstrukturen**
  - Stücklistenauflösung
  - Wegesuche in Graphen

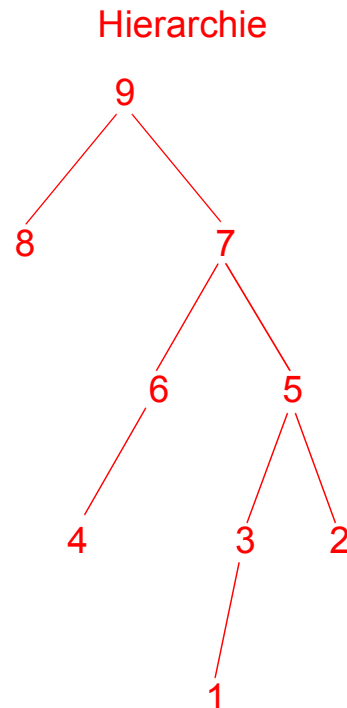
- **Integration in SQL**

- Syntax analog zu DataLog
- lineare Rekursion, verschränkte Rekursion
- Graphtraversierung mit „depth first“ oder „breadth first“ möglich
- **Herausforderungen**
  - Integration mit verschiedenen Verbundoperationen
  - Zulassung von Duplikaten
  - Zykluskontrolle

## Rekursion (2)

- **Beispiel**

Pers	Pnr	Gehalt	Mnr
	9	180 K	–
	8	110 K	9
	7	70 K	9
	6	120 K	7
	5	50 K	7
	4	150 K	6
	3	90 K	5
	2	50 K	5
	1	110 K	3



- **Q2: Finde alle Angestellten, deren direkter Manager MNR = 7 hat und die mehr als 100 K verdienen**

```
SELECT Pnr, Gehalt
FROM Pers
WHERE Mnr = 7 AND Gehalt > 100 K;
```

- **Q3 mit Erweiterung: Manager mit MNR = 7 kann höherer Manager sein**

- **Lösungsstrategie**

- Bilde anfängliche Sicht mit direkten Untergebenen (initial subquery)
- Erweitere diese Sicht rekursiv um die Untergebenen der Untergebenen solange, bis keine Untergebenen mehr hinzukommen (rekursive subquery)
- **UNION ALL** erlaubt die rekursive Ausführung

## Rekursion (3)

- Lösung für Q3**

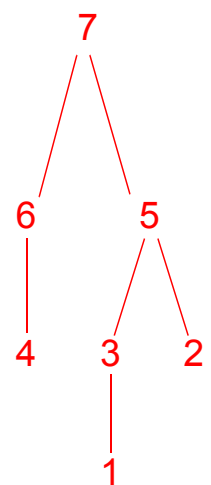
```

WITH RECURSIVE Untergebene (Pnr, Gehalt) AS
  ( ( SELECT Pnr, Gehalt
    FROM Pers
    WHERE Mnr = 7)
  UNION ALL
  ( SELECT P.Pnr, P.Gehalt
    FROM Untergebene AS U, Pers AS P
    WHERE P.Mnr = U.Pnr) )
SELECT Pnr
FROM Untergebene
WHERE Gehalt > 100 K;
  
```

- Auswertung**

Pers	Pnr	Gehalt	Mnr
	9	180 K	–
	8	110 K	9
	7	70 K	9
	6	120 K	7
	5	50 K	7
	4	150 K	6
	3	90 K	5
	2	50 K	5
	1	110 K	3

Unter- gebene	Pnr	Gehalt
	6	120 K
	5	50 K
	4	150 K
	3	90 K
	2	50 K
	1	110 K

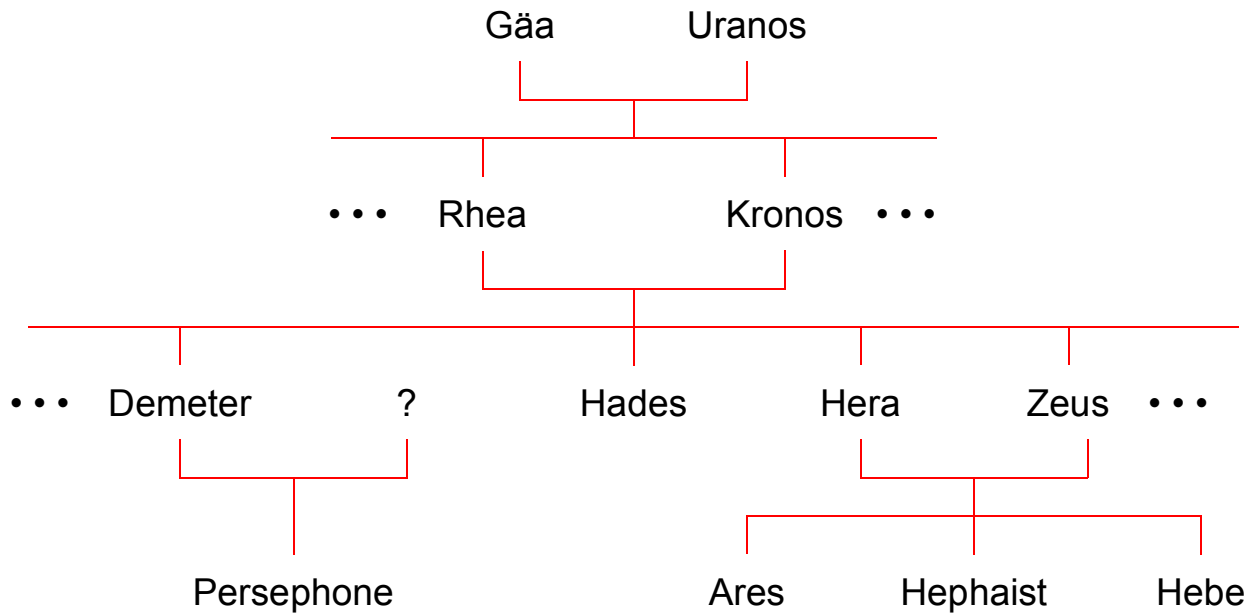


Ergebnis	Pnr



## Rekursion (4)

- Weltausschnitt



- Q4: Finde alle Vorfahren

Gegeben: Eltern (Kind, Elternteil)

Gesucht: Vorfahren (Kind, Vorfahr)

WITH RECURSIVE Vorfahren (Kind, Vorfahr) AS

( ( SELECT Kind, Elternteil FROM Eltern)

UNION ALL

( SELECT V.Kind, E.Elternteil

FROM Vorfahren AS V, Eltern AS E

WHERE V.Vorfahr = E.Kind) )

SELECT \*

FROM Vorfahren;

## Rekursion (5)

- **Rekursive Sicht**

Verwendung einer rekursiven Anfrage innerhalb von CREATE VIEW

- **Q5: Finde alle Vorfahren von Ares (als rekursive Sicht Ahnen)**

```
CREATE VIEW Ahnen (Kind, Vorfahr) AS
  WITH RECURSIVE Vorfahren (Kind, Vorfahr) AS
  ( ( SELECT Kind, Elternteil FROM Eltern)
  UNION ALL
  ( SELECT V.Kind, E.Elternteil
    FROM Vorfahren AS V, Eltern AS E
    WHERE V.Vorfahr = E.Kind) )
```

```
SELECT *
FROM Vorfahren
WHERE Kind = 'Ares';
```

- Optimierung und Ergebnis

Eltern	Kind	E-teil
	A	H
	A	Z
	H	R
	H	K
	Z	R
	Z	K
	R	G
	R	U
	K	G
	K	U
	...	...

## Rekursion (6)

- **Rekursives Einfügen**

- Ergebnis einer rekursiven Anfrage kann mit INSERT in eine Tabelle eingefügt werden
- Technik zur Erzeugung synthetischer Tabellen

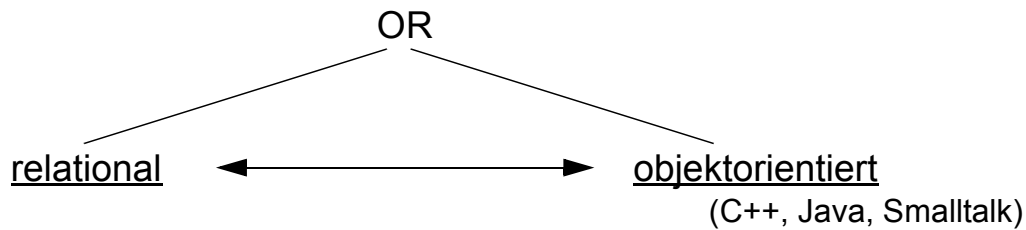
- **Beispiel**

```
CREATE TABLE Zahlen (Zähler Integer, Zufall Integer);
```

```
INSERT INTO Zahlen (Zähler, Zufall)
  WITH RECURSIVE Temp(n) AS
    ( (VALUES (1))
      UNION ALL
      ( SELECT n+1 FROM Temp
        WHERE n < 1000) )
  SELECT n, integer (rand ( ) * 1000)
  FROM Temp;
```

- Ergebnis

# Vergleich von Begriffen/Konzepten



## Intensional:

- Relationenschema
- Objekttyp

## Extensional:

- Relation
- Klasse/Kollektion

## Struktur:

- sichtbar im Schema
- unsichtbar:  
gekapselt, Signatur

## Sprache:

- generische Operationen  
(„Insert into Pers“)
- typspezifische Operationen  
(„Einstelle Angestellter“)

## Identität:

- wertbasiert (Primärschlüssel)
- objektbasiert (OID)

## Zugriff:

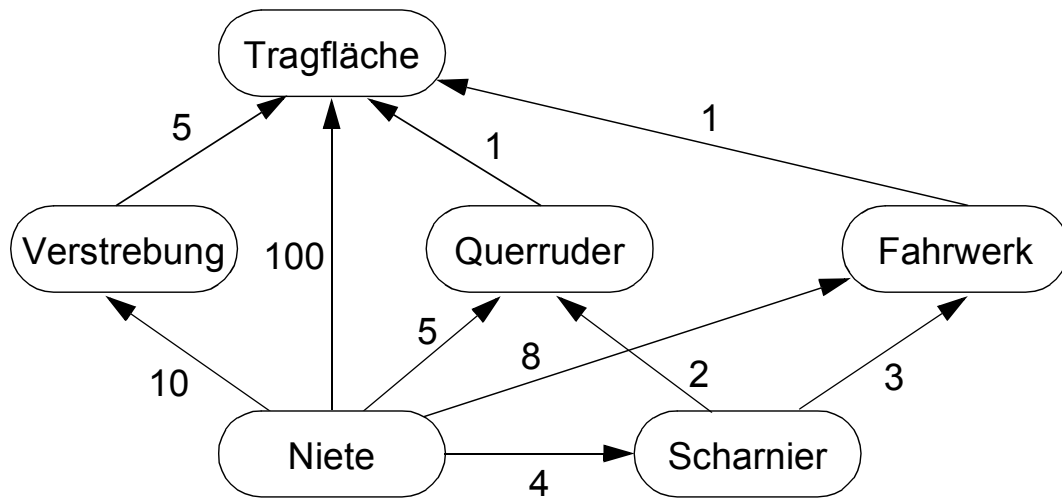
- mengenorientiert  
(n Tupel)
- deskriptiv  
(Anfragen über n Relationen)
- n-mengenorientiert
- satzorientiert  
(1 Objekt)
- navigierend  
(Iterator mit Suchargument)
- 1-mengenorientiert

# Zusammenfassung

- **OODM liefern leistungsfähige Konzepte für den Umgang mit komplexen Objekten und mächtigen Operationen**
  - Sie eignen sich für Non-Standard-Anwendungen
  - Es gibt bereits leistungsfähige Implementierungen von OODBS
- **OO-Manifesto ist nicht allgemein anerkannt**
  - Wieviele Eigenschaften sind essentiell?
  - Welche Eigenschaften sind eher ergänzend?
    - ➔ **Es werden noch viele weitere Forderungen gestellt!**
- **Es gibt ein durch SQL:1999 standardisiertes ORDM**
  - Es wurden die wesentlichen OODM-Konzepte übernommen
  - Typkonstruktoren, benutzerdefinierte Typen und Funktionen
  - Typ- und Tabellenhierarchien sowie Referenzen
  - Regelsystem (Triggerkonzept), Erweiterungsinfrastruktur, . . .
- **Spezifikation großer Objekte hat großen Einfluss auf die DB-Verarbeitung**
  - Speicherungsoptionen, Logging
  - Einsatz benutzerdefinierter und systemspezifischer Funktionen
  - Deklarative SQL-Operationen, aber Cursor-basierte und „navigierende“ Verarbeitung von LOB-Werten
- **Lokator-Konzept**
  - Identifikation von LOBs oder Positionen in LOBs
  - Minimierung des Datenverkehrs zwischen Client und Server
  - Bereitstellung von Server-Funktionen bei der LOB-Verarbeitung
- **Deskriptive Anfragesprache von SQL:1999 ist sehr mächtig**
  - Nutzung von allgemeinen Tabellenausdrücken
  - Einsatz von Rekursion
  - Rekursion mit Berechnungen

# Rekursion mit Berechnungen

- Gozinto-Graph



- Q6: Wie viele Nieten werden insgesamt für eine Tragfläche benötigt?

- Abbildung des Gozinto-Graph

Teil (Tnr, Bezeichnung, ...)

Struktur (Otnr,	Utnr,	Anzahl)
T	V	5
T	Q	1
T	F	1
T	N	100
V	N	10
Q	N	5
Q	S	2
F	N	8
F	S	3
S	N	4

## Rekursion mit Berechnungen (2)

- **Temporäre rekursive Sicht Tragflächenteile (TFT)**

WITH RECURSIVE **Tragflächenteile** (Utnr, Anzahl) AS

( ( SELECT Utnr, Anzahl

FROM **Struktur**

WHERE Otnr = 'T')

UNION ALL

( SELECT S.Utnr, T.Anzahl \* S.Anzahl

FROM **Tragflächenteile** T, **Struktur** S

WHERE S.Otnr = T.Utnr) );

- **Ableitung von TFT**

Struktur (Otnr, Utnr, Anzahl)

TFT (Utnr, Anzahl)

T	V	5
T	Q	1
T	F	1
T	N	100
V	N	10
Q	N	5
Q	S	2
F	N	8
F	S	3
S	N	4

## Rekursion mit Berechnungen (3)

- **Q7: Bestimme die Gesamtzahl der Nieten in einer Tragfläche**

```
WITH RECURSIVE Tragflächenteile (Utnr, Anzahl) AS
  ( ( SELECT Utnr, Anzahl
    FROM Struktur
    WHERE Otnr = 'T')
  UNION ALL
  ( SELECT S.Utnr, T.Anzahl * S.Anzahl
    FROM Tragflächenteile T, Struktur S
    WHERE S.Otnr = T.Utnr )
```

```
SELECT SUM (Anzahl) AS NAnzahl
FROM Tragflächenteile
WHERE Utnr = 'N';
```

- Ergebnis: NAnzahl

- **Q8: Bestimme alle für eine Tragfläche benötigten Teile, zusammen mit der jeweiligen Anzahl**

```
WITH RECURSIVE Tragflächenteile (Utnr, Anzahl) AS
  ( ( SELECT Utnr, Anzahl
    FROM Struktur
    WHERE Otnr = 'T')
  UNION ALL
  ( SELECT S.Utnr, T.Anzahl * S.Anzahl
    FROM Tragflächenteile T, Struktur S
    WHERE S.Otnr = T.Utnr )
```

```
SELECT Utnr, SUM (Anzahl) AS TAnzahl
FROM Tragflächenteile
GROUP BY Utnr;
```

- Ergebnis: Utnr, TAnzahl