12. SQL:1999 - Neue Funktionalität

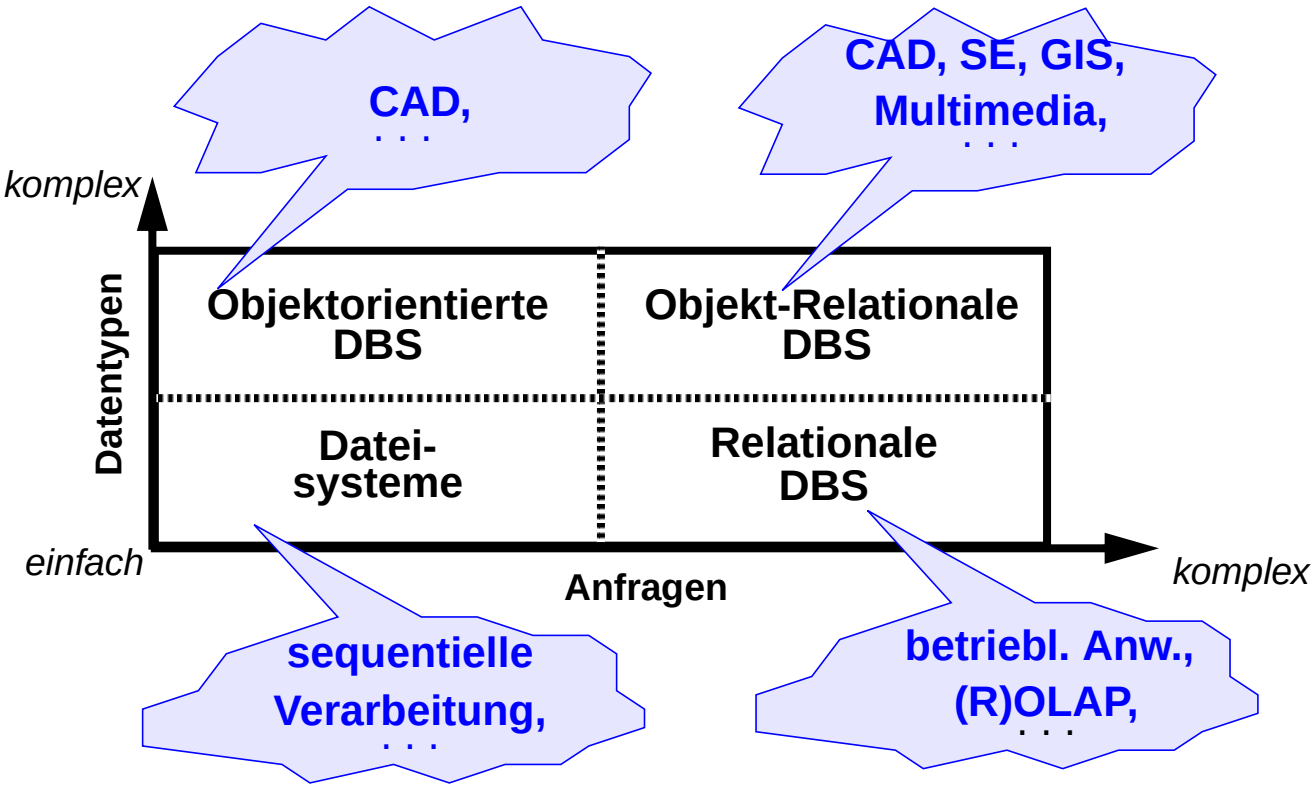
- **Objekt-relationale DBS - Vision**
  - verschiedene Systemarchitekturen
  - objekt-relationale DB-Technologie
  - Erweiterbarkeitsinfrastruktur

- **Standardisierung von SQL – Überblick**[1]

- **Erhöhung der Ausdrucksmächtigkeit**
  - Allgemeine Tabellenausdrücke
  - Rekursion
  - Rekursion mit Berechnungen

1. http://www.jtc1sc32.org

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 DB-Konzepte: Motivation

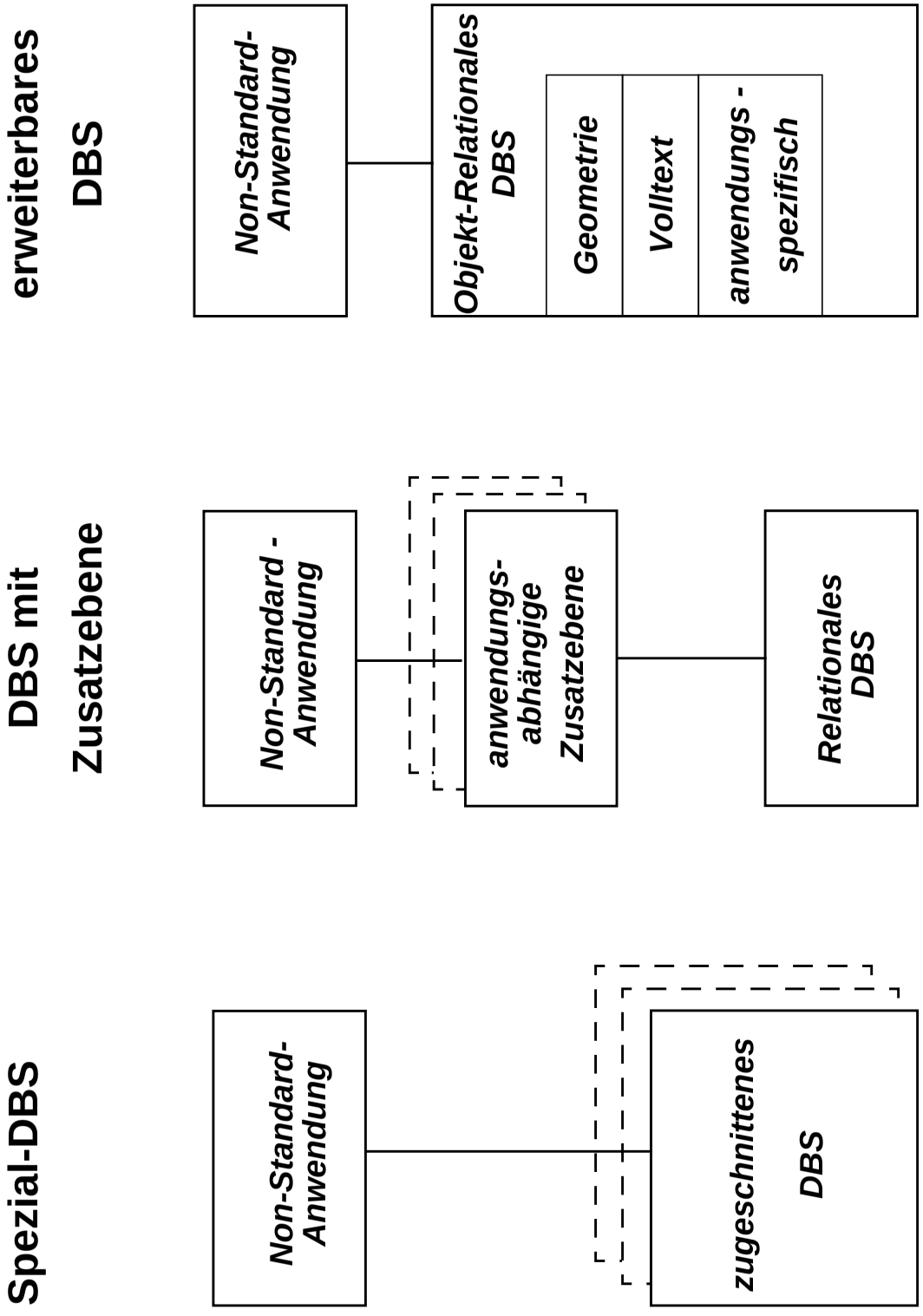
- **Relationale Datenbankverwaltungssysteme bieten**
  - eine Menge von Datentypen, um Anwendungsdaten darstellen zu können
  - eine Menge von Operationen, um diese Datentypen manipulieren zu können

• **Beispiele:**	**Datentypen**	**Funktionen**
	INTEGER	+, - , *, /, AVG, SUM, ...
	CHARACTER	Manipulation von Zeichenketten: suchen, anfügen, ...
	DATE	Tag, Monat, Jahr, +, -, ...

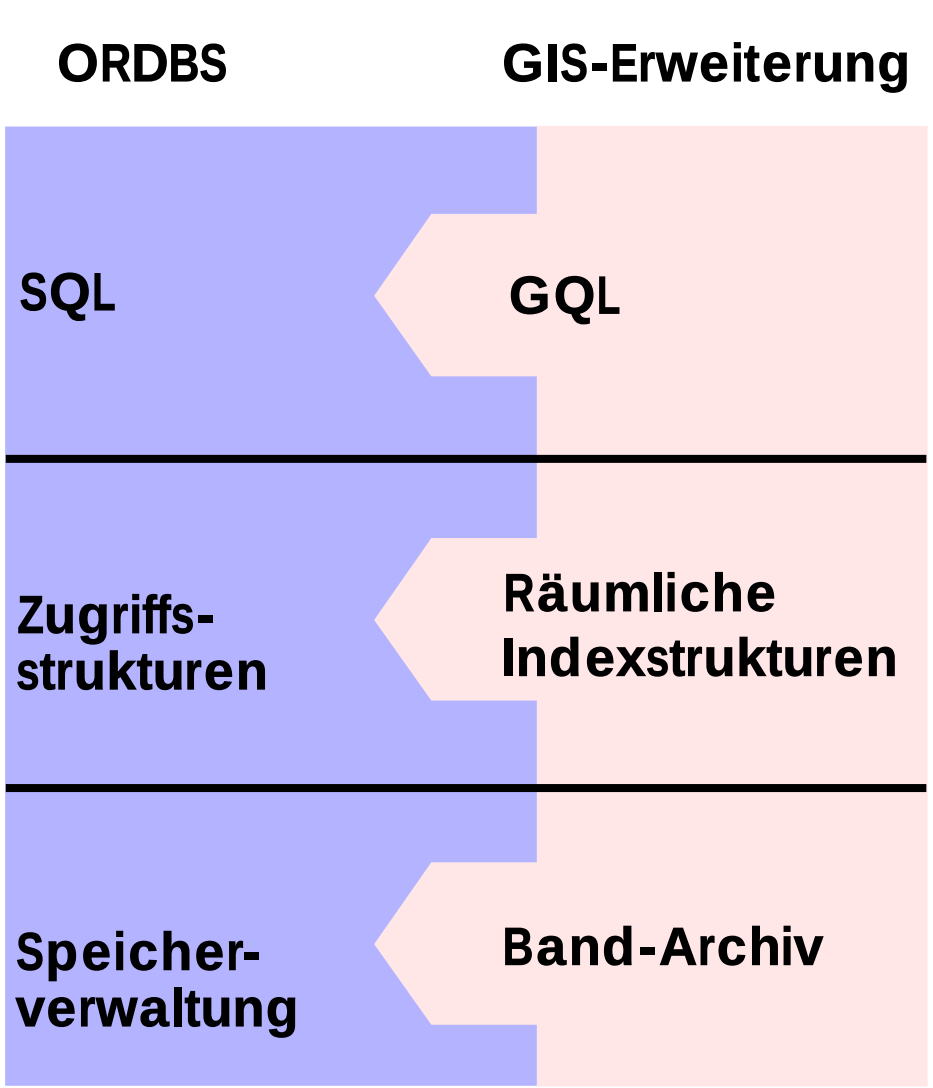
- **Neue Anwendungen erfordern neue Datentypen und Funktionen!**

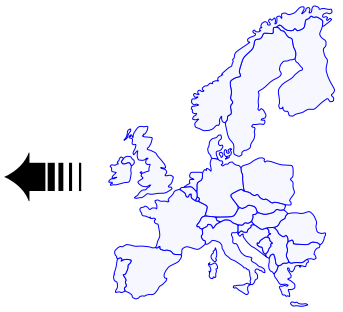
• **Beispiele:**	**Datentypen**	**Funktionen**
	TEXT	Volltextsuche, Rechtschreibkorrektur, ...
	POLYGON	Durchmesser, Schnitt von Polygonen, Fläche, ...
	RASTER	Konvertierung zwischen Formaten, Farbanalyse

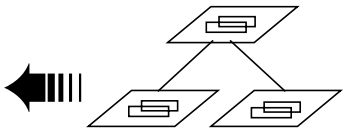
Drei verschiedene System-Architekturen



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







- ✚ 100% kompatibel zu RDBS
- ✚ Neue Datentypen
- ✚ Neue Funktionen
- ✚ Neue Zugriffs-pfade
- ✚ Neue Speicher-medien
- ➠ Erweiterbarkeit für Entwickler von Anwendungen

Objekt-Relationale Datenbank-Technologie

- **Funktionalität wird derzeit im wesentlichen durch den Standard SQL99 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 Optimizer, Synchronisation, Logging und Recovery

Standardisierung von SQL

- **Standardisierung durch ISO JTC1/SC21/WG3 DBL**

  SC21: Information Retrieval, Transfer and Management

  WG3: 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	1999

  (IEC= Intl. Electrotechnical Commission)

- **Arbeit seit 1990 an SQL:1999**
  - weitreichende Erweiterung von SQL-92

- **Parallel dazu: vorbereitende Arbeiten an SQL4 seit 1996**

SQL:1999 als richtungsweisender DB-Standard

- **Teilnehmer am Standardisierungsprozess:**
  DB-Hersteller und Anwender, 11 Länder, ANSI

- **Konsens zwischen Teilnehmern wird angestrebt**

- **SQL:1999 hat mehrere Teile**
  - SQL/Foundation, SQL/Object
  - SQL/PSM, SQL/Binding, SQL/CLI
  - **SQL/MED** (Management of External Data)
  - **SQL Object Language Bindings** (SQLJ) . . .

- **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”
                    IN SAME SENTENCE AS
                    “schwer” AND (“Motorhaube” OR “Blech”))
  AND A.Nummer = 61
  AND SCORE (U.Bild, “rot”) > 0.6
  AND DISTANCE(A.Ausfahrt, U.Ort) < km (5);
```

Textdaten

herkömmliche Attribute

Bilddaten

räumliche Daten

Allgemeine Tabellenausdrücke

- **Gegeben: Pers (Pnr, Anr, Mnr, Gehalt, Bonus)**

- **Gesucht: Abteilung (Anr) mit höchster Gehaltssumme**

- **Versuch**

```
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
	K 03	389 K
	K 51	794 K
	K 55	1012 K

  - Anfrage Q1:

```
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?**

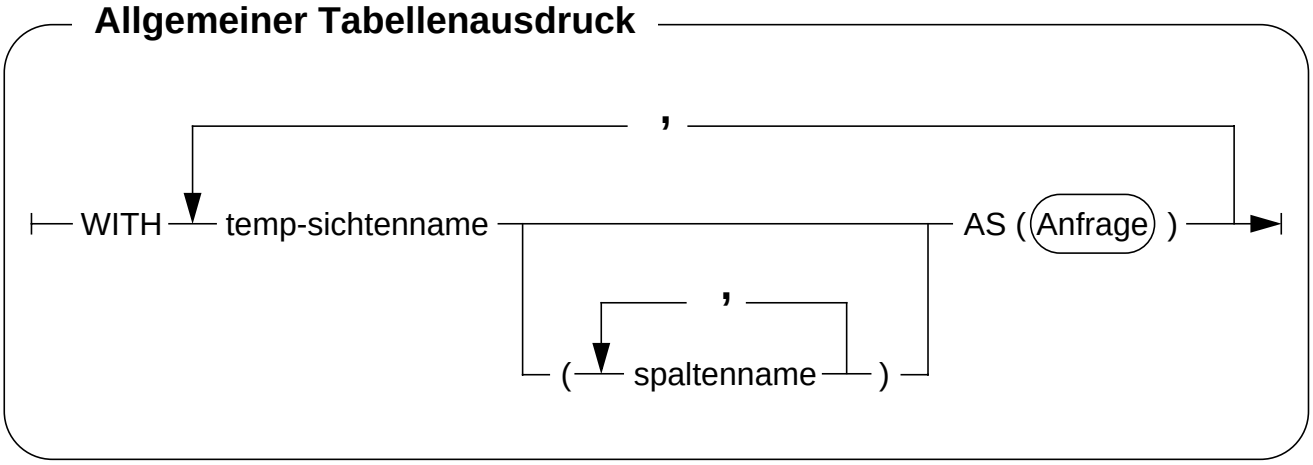
Allgemeine Tabellenausdrücke (2)

- **Anfrage Q2**

```
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 und Q2**

```
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

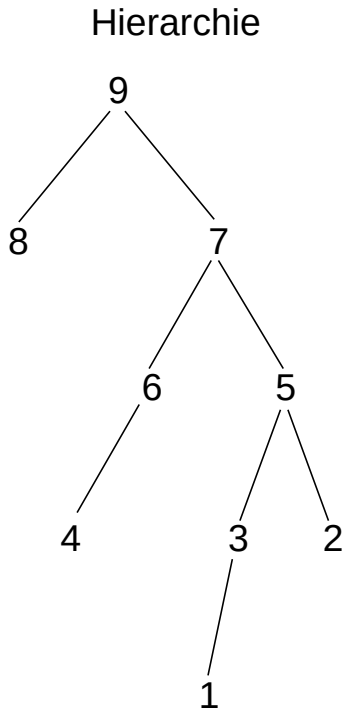
- **Warum nutzt man Rekursion in SQL?**
  - deskriptive und mengenorientierte Formulierung
    - Gewinn an Ausdrucksmächtigkeit
    - verbessertes Leistungsverhalten
  - Traversierung von Baum- und Netzwerkstrukturen
    - Stücklistenauflösung
    - Wegesuche in Graphen

- **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



- **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;
```

- **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)

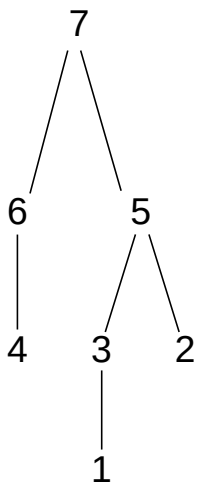
- **Lösung**

```
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

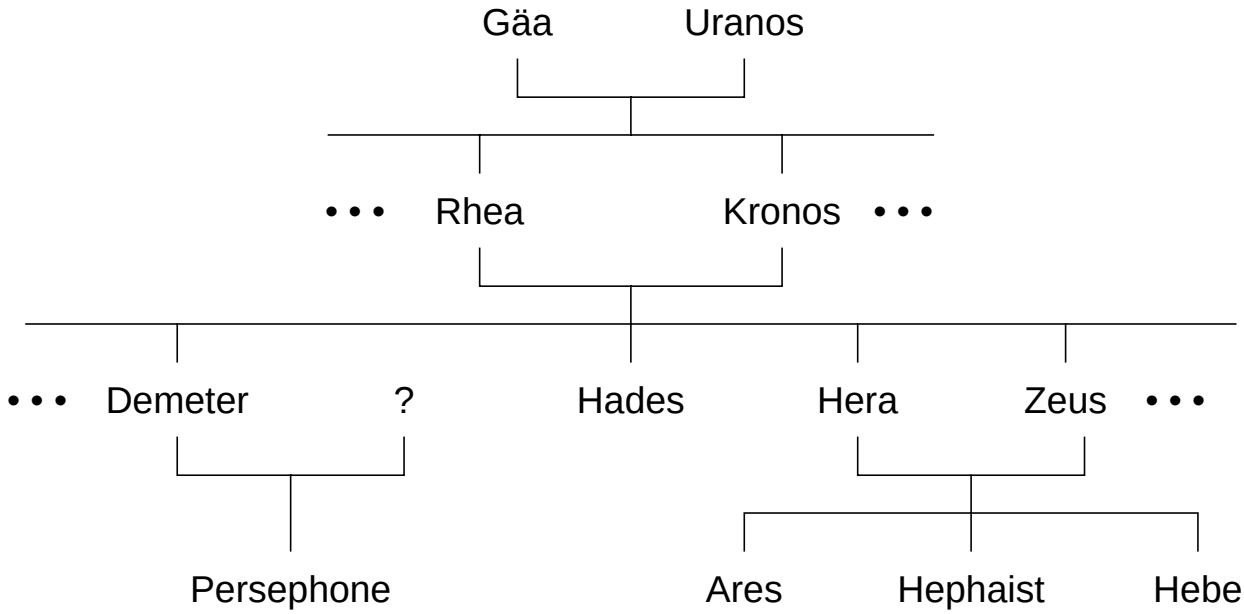
Unter-gebene	Pnr	Gehalt
	6	120 K
	5	50 K
	4	150 K
	3	90 K
	2	50 K
	1	110 K



Ergebnis	Pnr

Rekursion (4)

- **Weltausschnitt**



- **Bestimmung aller 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
  - Bestimmung aller Vorfahren von Ares

```
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

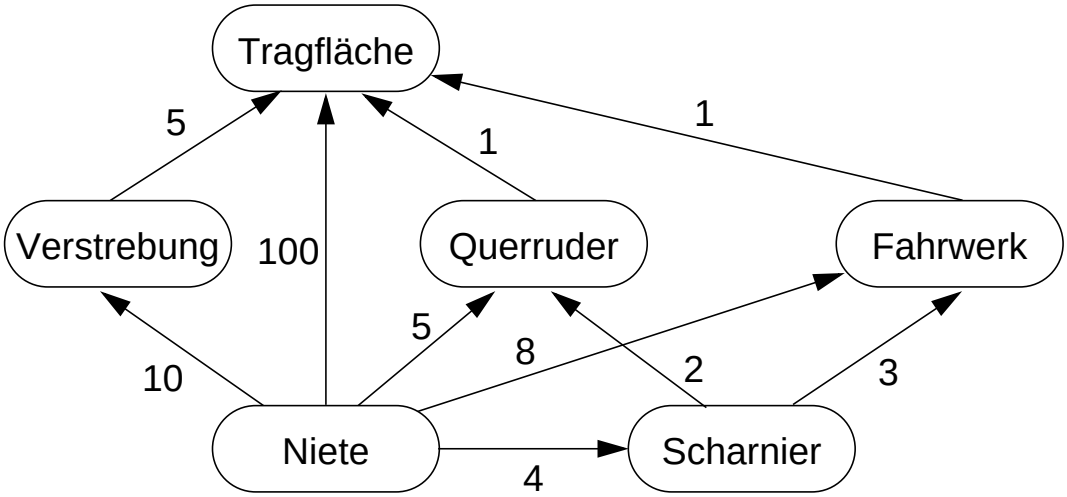
```
CREATE TABLE Zahlen (Zähler Integer, Zufall Integer);

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

Rekursion mit Berechnungen

- **Gozinto-Graph**



- **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)

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

TFT (Utnr, Anzahl)

Rekursion mit Berechnungen (3)

- **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

Rekursion mit Berechnungen (4)

- **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

Zusammenfassung

- **Es gibt ein durch SQL99 standardisiertes ORDM**
  - Es wurden die wesentlichen OODM-Konzepte übernommen
  - Typkonstruktoren, benutzerdefinierte Typen und Funktionen
  - Typ- und Tabellenhierarchien sowie Referenzen
  - Regelsystem (Triggerkonzept)
  - Erweiterungsinfrastruktur
  - . . .
- **Deskriptive Anfragesprache von SQL99 ist sehr mächtig**
  - Nutzung von allgemeinen Tabellen ausdrücken
  - Einsatz von Rekursion
  - Rekursion mit Berechnungen