Objektrelationale Erweiterungen in DB2

- Neue Basisdatentypen
- Benutzerdefinierte Typen
  - Distinct-Typen
  - Strukturierte Typen (mit Typvererbung)
- Referenztyp
- Typisierte Tabellen (mit Subtabellenbildung)
- Typisierte Sichten (mit Subsichtenbildung)

Neue Basisdatentypen

- Large Objects
  - BLOB (Binary Large Object)
  - CLOB (Character Large Object)
  - Operationen
    - SUBSTR, POSSTR, || (CONCAT)
    - IS [NOT] NULL, LIKE, LENGTH
  - Nicht erlaubt
    - BLOB/CLOB-Attribute als Teil eines Schlüssels oder in booleschen bzw. arithmetischen Ausdrücken
    - GROUP BY, ORDER BY, EQUALS, GREATER/LESS THAN
- BOOLEAN Datentyp nicht unterstützt

Verwendung neuer Basisdatentypen

Definition einer Tabelle mit LOB-Attributen:

```
CREATE TABLE MitarbeiterTupelTabelle (
        Name       VARCHAR(30),
        Bild       BLOB(1M),
        Bewerbung  CLOB(50k)
);
```

Erzeugen von LOB-Attributen:

```
INSERT INTO MitarbeiterTupelTabelle
VALUES  ('Billy',
         BLOB('Bilddaten innerhalb einer Anwendung zuweisen!'),
         CLOB('Bewerbung innerhalb einer Anwendung zuweisen!')
);
```


Distinct-Typen

- Distinct-Typ ist eine Kopie eines Basisdatentyps
- Syntax:

```
CREATE DISTINCT TYPE Distinct-Typname AS Basisdatentyp
WITH COMPARISONS
```

- Distinct-Typdefinition generiert automatisch
  - Cast-Funktionen zur Konversion zwischen Distinct-Typ und Ausgangstyp
  - Vergleichsoperatoren (=, <, >, <=, <=, <>) für Instanzen des Distinct-Typs
    - Gilt nicht für die Typen BLOB, CLOB, LONG VARCHAR, LONG VARGRAPHIC (hier darf WITH COMPARISONS nicht verwendet werden)

Beispieldefinition und Verwendung von Distinct-Typen

Definition von Distinct-Typen:

```
CREATE DISTINCT TYPE Franken AS DECIMAL(9,2) WITH COMPARISONS;
CREATE DISTINCT TYPE Euro AS DECIMAL(9,2) WITH COMPARISONS;
```

Verwendung von Distinct-Typen in Tabellendefinitionen:

```
CREATE TABLE CHBank(Nr INTEGER, Stand Franken);
CREATE TABLE EuroBank(Nr INTEGER, Stand Euro);
```

Anfrage basierend auf Distinct-Typen und automatisch generierten Cast-Funktionen:

```
SELECT c.Nr, c.Stand
FROM   CHBank c, EuroBank e
WHERE CAST(c.Stand AS DECIMAL) < CAST(e.Stand AS DECIMAL);
```


Strukturierte Typen

- Abstrakter Objekttyp mit Attributen und Methoden
  - Defaultwerte und Integritätsbedingungen werden nicht unterstützt
  - Keine kollektionswertigen Attribute
  - Attribut eines strukturierten Typs darf weder denselben Typ noch einen Subtyp dieses strukturierten Typs haben
  - Typvererbung: Ein Subtyp erbt die Attribute und Methoden des Supertyps
  - Substituierbarkeit: Subtypinstanz ist als Supertypinstanz verwendbar
- Typdefinition generiert automatisch
  - Methoden zum Zugriff auf und Ändern der Attributwerte
  - Konstruktoren zum Erzeugen von Instanzen
  - Operatoren zum Vergleichen von Instanzen

Strukturierte Typen - Definition von Wurzeltypen

- Syntax:

```
CREATE TYPE Typname AS
(Attributdefinitionsliste)
[[NOT] INSTANTIABLE]
[NOT FINAL]
MODE DB2SQL
[REF USING Typ]
[CAST (SOURCE AS REF) WITH Funktionsname]
[CAST (REF AS SOURCE) WITH Funktionsname]
[Methodendeklarationsliste]
```

  - Jede Instanz eines strukturierten Typs besitzt eine OID
    - Default-Typ des OID ist VARCHAR(16)
    - Typ des OID kann mit REF USING geändert werden

Strukturierte Typen - Beispiele

```
CREATE TYPE AdresseTyp AS (
  Strasse VARCHAR(30),
  Nr      DECIMAL(4),
  PLZ     DECIMAL(5),
  Ort     VARCHAR(40),
  Land    VARCHAR(25)
) MODE DB2SQL;
```

```
CREATE TYPE KundeTyp AS (
  KNr       INTEGER,
  Name      VARCHAR(30),
  Anschrift AdresseTyp
) REF USING INTEGER
MODE DB2SQL;
```

```
CREATE TYPE LieferantTyp AS (
  LNr       INTEGER,
  Name      VARCHAR(25),
  Anschrift AdresseTyp
) MODE DB2SQL;
```

```
CREATE TYPE AuftragTyp AS (
  ANr        INTEGER,
  Kunde      REF(KundeTyp)
  Eingang    DATE,
  Bearbeitet DATE,
  Lieferant  REF(LieferantTyp)
) MODE DB2SQL;
```

Referenztyp

- REF: Referenz auf ein Objekt eines strukturierten Typs
  - Attributwert ist eine OID (object identifier)
- Beispiel: Ein Teil referenziert ein anderes Teil

```
CREATE TYPE TeilTyp AS (
          Nr                DECIMAL(10),
          Bezeichnung       VARCHAR(25),
          Farbe             VARCHAR(15),
          IstTeilVon        REF(TeilTyp),
          Preis             Franken
) MODE DB2SQL
METHOD Gesamtpreis() RETURNS Franken LANGUAGE SQL;
```


Lesestoff

Methodenimplementierung

- Methodendeklaration und -implementierung erfolgt analog zu SQL-99
  - Deklaration erfolgt innerhalb der Definition/Änderung des Objekttyps

```
ALTER TYPE TeilTyp ADD METHOD AnzahlUnterteile()
                       RETURNS INTEGER LANGUAGE SQL;
```

  - Implementierung in einer separaten Anweisung (hier SQL-Variante)

```
CREATE METHOD AnzahlUnterteile()
          RETURNS INTEGER
          FOR TeilTyp
          RETURN (SELECT COUNT(*)
                     FROM Teil
                     WHERE IstTeilVon->Bezeichnung = SELF..Bezeichnung);
```

Subtypbildung - Aufbau von Typhierarchien

- Subtypdefinition mittels UNDER-Klausel
  - Subtyp erbt alle Attribute und Methoden des Supertyps
  - Supertyp muss selbst ein strukturierter Typ sein
  - Subtyp darf nur maximal einen direkten Supertyp haben
    - Keine (direkte) Mehrfachvererbung möglich
  - Geerbte Attribute und Methoden sind nicht überschreibbar
- Syntax:

```
CREATE TYPE Typname
UNDER Supertypname AS
(Attributdefinitionsliste)
[[NOT] INSTANTIABLE]
[NOT FINAL]
MODE DB2SQL
[Methodendeklarationsliste]
```

Beispiel:

```
CREATE TYPE BWKundeTyp
UNDER KundeTyp AS
( Interessen VARCHAR(50) )
MODE DB2SQL;
```


Typisierte Tabelle

- Basiert auf einem strukturierten Typen
- Speichert Instanz des strukturierten Typs als Zeile der Tabelle
- OID-Attribut ist sichtbar!
  - Muss für alle Wurzeltabellen explizit angegeben werden
  - Wert wird innerhalb von INSERT belegt und ist danach unveränderlich
  - Wert muss NOT NULL und eindeutig innerhalb der Tabellenhierarchie sein
- Kann Integritätsbedingungen enthalten
  - Primär-/Unique-/Fremdschlüssel, Not Null, Check-Klausel, Scope-Klausel

Typisierte Tabellen - Definition von Wurzeltabellen

- Syntax:

```
CREATE TABLE Tabellenname OF StrukturierterTyp (
        REF IS OID-Attributname USER GENERATED
        [Attributoptionsliste]
)
```

- Attributoption:

```
Attributname WITH OPTIONS Optionsliste
| CHECK-Integritätsbedingung
| UNIQUE-Integritätsbedingung
```

- Optionen:

```
SCOPE TypisierteTabelle
| DEFAULT Wert
| Integritätsbedingung
```


Typisierte Tabelle - Beispiele

```
CREATE TABLE Teil OF TeilTyp (
  REF IS oid USER GENERATED,
  Nr WITH OPTIONS NOT NULL PRIMARY KEY,
  IstTeilVon WITH OPTIONS SCOPE Teil
);
```

```
CREATE TABLE Kunde OF KundeTyp (
  REF IS oid USER GENERATED,
  KNr WITH OPTIONS NOT NULL PRIMARY KEY,
  Name WITH OPTIONS NOT NULL
);
```

Typisierte Tabelle - Weitere Beispiele

```
CREATE TABLE Auftrag OF AuftragTyp (
  REF IS oid USER GENERATED,
  ANr WITH OPTIONS NOT NULL PRIMARY KEY
);
```

```
CREATE TABLE Lieferant OF LieferantTyp (
  REF IS oid USER GENERATED,
  LNr WITH OPTIONS NOT NULL PRIMARY KEY,
  Name WITH OPTIONS NOT NULL
);
```


Operationen auf typisierten Tabellen

```
INSERT INTO Kunde                        -- OID (erstes Attribut) durch
VALUES (KundeTyp(17), 17, 'Billy',       -- Referenzkonstruktoraufruf erzeugt
        AdresseTyp()..Strasse('Seefeldstrasse')..Nr(31)..Plz(8008)
                   ..Ort('Zurich')..Land('Schweiz'));
```

```
UPDATE Kunde
SET Anschrift= AdresseTyp()..Strasse('Kreuzstrasse')..Nr(31)..Plz(8008)
                          ..Ort('Zurich')..Land('Schweiz')
WHERE Name = 'Billy';
```

```
DELETE FROM Kunde
WHERE Anschrift..Ort = 'Zurich';
```

Zugriff auf Attribute/Methoden einer Instanz eines strukturierten Typs erfolgt mittels dem Doppel-Dot-Operator ..

Attributbelegungen

- Ändern von strukturierten Attributen

```
UPDATE Kunde
SET Anschrift..Stadt = 'Zuerich'
WHERE Anschrift..Stadt = 'Zurich';
```

- Ändern von Referenzattributen

```
UPDATE Teil
SET IstTeilVon =(SELECT oid FROM Teil WHERE Bezeichnung = 'Fahrrad')
WHERE Bezeichnung = 'Rad';
```


Überladen des Konstruktors

Definition eines Konstruktors:

```
CREATE FUNCTION AdresseTyp (S   VARCHAR(30),
                            N   DECIMAL(4),
                            P   DECIMAL(5),
                            O   VARCHAR(25),
                            L   VARCHAR(20))
RETURNS AdresseTyp
LANGUAGE SQL
RETURN AdresseTyp()..Strasse(S)..Nr(N)..PLZ(P)..Ort(O)..Land(L);
```

Einfügen eines neuen Kunden mit Hilfe des neuen Konstruktors:

```
INSERT INTO Kunde
VALUES (KundeTyp(17), 17, 'Billy',
        AdresseTyp('Seefeldstrasse', 31, 8008, 'Zurich', 'Schweiz'));
```

OID-Erzeugung

- OIDs sind "systemgenerierbar", wenn der strukturierte Typ ohne REF USING definiert wurde

```
INSERT INTO Teil
VALUES (TeilTyp(GENERATE_UNIQUE()), 13, 'Fahrrad', 'Silber-Blau', NULL);
```

- Nutzer kann weiterhin OIDs vergeben

```
INSERT INTO Teil
VALUES (TeilTyp('RYWZBA12WS'), 18, 'Rahmen', 'Blau',
          (SELECT oid FROM Teil WHERE Nr =13));
```

  – Referenzattributbelegung mit einer Anfrage (oben) oder Direktangabe (unten)

```
INSERT INTO Teil
VALUES (TeilTyp(GENERATE_UNIQUE()), 56, 'Gabel', 'Blau',
         TeilTyp('RYWZBA12WS'));
```


Einfache Anfragen - Beispiele

Zugriff auf OID-Attribut:

```
SELECT oid FROM Kunde;
```

Zugriff auf eine Komponente eines objektwertigen Attributs:

```
SELECT Anschrift..Stadt FROM Kunde;
```

Zugriff auf ein Referenzattribut (OID des referenzierten Objekts):

```
SELECT IstTeilVon FROM Teil;
```

Zugriff auf ein Attribut eines referenzierten Objekts (Dereferenzierung mit anschliessendem Komponentenzugriff):

```
SELECT IstTeilVon->Bezeichnung FROM Teil;
SELECT DEREF(IstTeilVon)..Bezeichnung FROM Teil;
```

Subtabellenbildung - Aufbau von Tabellenhierarchien

- Syntax:

```
CREATE TABLE Tabellenname OF StrukturierterTyp
UNDER Supertabelle INHERIT SELECT PRIVILEGES
[( Attributoptionsliste )]
```

  - Typ der Subtabelle muss ein direkter Subtyp des Typs der Supertabelle sein
  - (Tiefe) Extension der Subtabelle muss immer eine Untermenge der (tiefen) Extension der Supertabelle sein
    - Instanzen der Subtabelle sind auch Mitglieder der zugehörigen Supertabellen
  - Subtabelle darf nur maximal eine direkte Supertabelle haben
    - Keine (direkte) Mehrfachspezialisierung möglich
  - Subtabelle kann neue Integritätsbedingungen hinzudefinieren
  - Subtabellen besitzen keine eigene Referenzklausel

Subtabelle - Beispiel

```
CREATE TABLE BWKunde OF BWKundeTyp UNDER Kunde
                                   INHERIT SELECT PRIVILEGES (
        Interessen WITH OPTIONS NOT NULL
);
```

- Achtung:
  - Schlüsselwort INHERIT SELECT PRIVILEGES ist nicht optional
  - Strukturierter Typ der Subtabelle mindestens ein "eigenes" Attribut definieren

Anfragen an Tabellenhierarchien

```
SELECT Name FROM Kunde;
```

liefert die Namen aller Kunden (inklusive der BWKunden)

Zugriff auf tiefe Extension einer Supertabelle

```
SELECT Name FROM BWKunde;
```

liefert die Namen aller BWKunden

Zugriff auf tiefe Extension einer Subtabelle

```
SELECT Name FROM ONLY(Kunde);
```

liefert nur die Namen der Kunden, die keine BWKunden sind

Zugriff auf flache Extension einer Supertabelle

```
SELECT Name, Interessen FROM OUTER(Kunde);
```

liefert Namen und Interessen aller Kunden

Zugriff auf Attribute der Subtabelle auf Supertabellenebene

Typisierte Sichten

- Syntax: Definition einer typisierten Sicht

```
CREATE VIEW Sichtenname OF StrukturierterTyp MODE DB2SQL (
          REF IS OID-Attributname USER GENERATED
          [Attributoptionsliste]
)
AS Anfrageausdruck
[WITH CHECK OPTION]
```

  – Referenzklausel ist zwingend notwendig für Wurzelsichten
  – Sichtanfrage über genau einer Tabelle bzw. Sicht
  – nur benutzerdefinierte OIDs sind erlaubt

Subsichtenbildung - Aufbau von Sichtenhierarchien

- Syntax: Definition einer Subsicht

```
CREATE VIEW Sichtenname OF StrukturierterTyp MODE DB2SQL
UNDER Supersicht INHERIT SELECT PRIVILEGES
[( Attributoptionsliste )]
AS Anfrageausdruck
[WITH CHECK OPTION]
```

  - Typ der Subsicht muss ein direkter Subtyp des Typs der Supersicht sein
  - Supersicht muss auf eine flache Extension zugreifen (ONLY)
  - Supersicht darf keine andere Subsicht desselben Subtyps besitzen
  - Subsicht erweitert Supersicht: Alle Instanzen der Subsicht sind auch automatisch in der Supersicht sichtbar

Typisierte Sichten - Beispiele

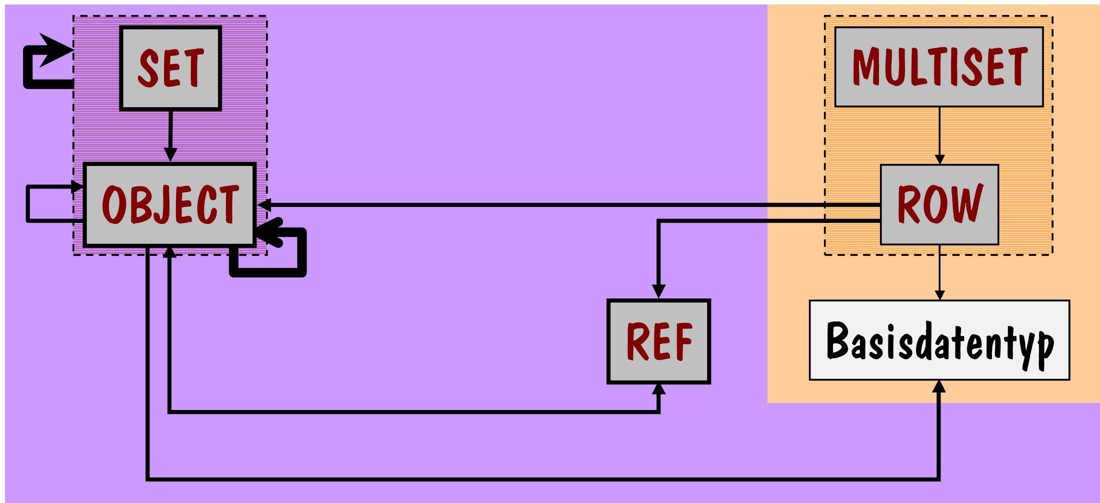
Definition einer Wurzelsicht:

```
CREATE VIEW ZuercherKunde OF KundeTyp MODE DB2SQL
(REF IS oid USER GENERATED)
AS (SELECT * FROM ONLY(Kunde) WHERE Anschrift..Ort = 'Zurich');
```

Definition einer Subsicht:

```
CREATE VIEW AKunde OF BWKundeTyp MODE DB2SQL
UNDER ZuercherKunde INHERIT SELECT PRIVILEGES
AS (SELECT * FROM BWKunde WHERE Name LIKE 'A%');
```

DB2 - Datenmodell

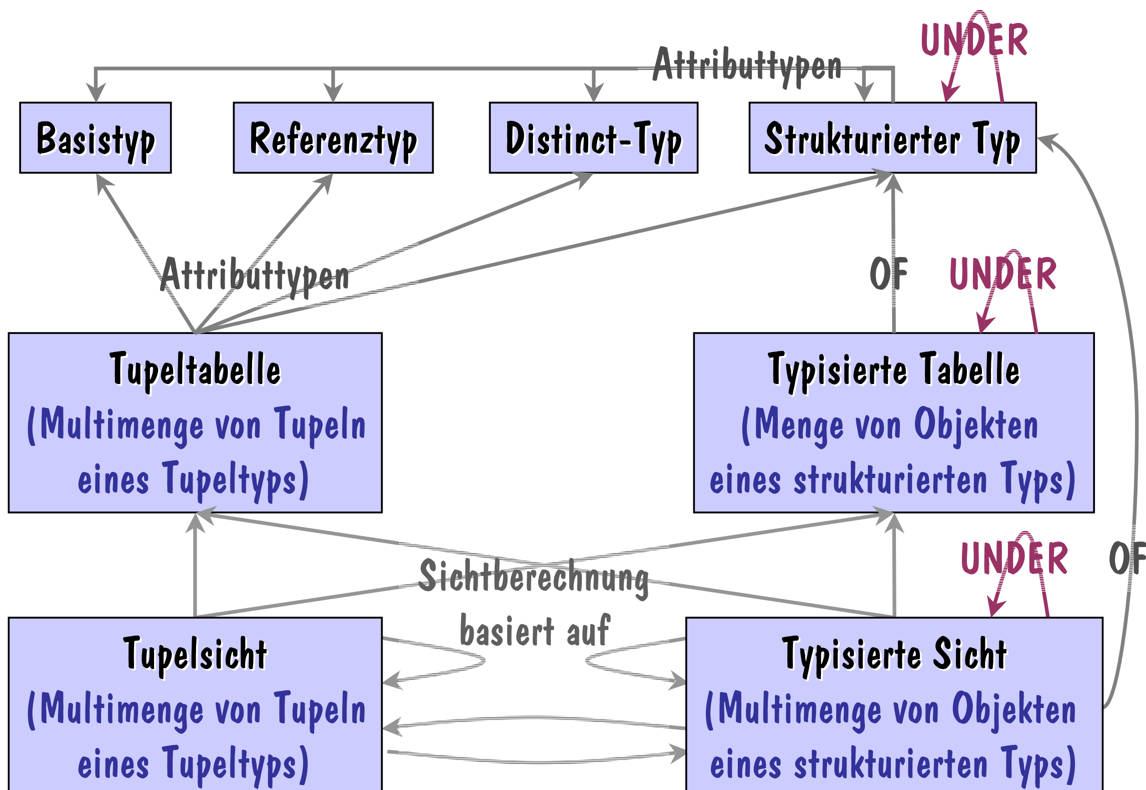


Einstiegspunkte

Typisierte Tabelle: SET(OBJECT(...))

Untypisierte Tabelle: MULTISET(ROW(...))

DB2 - Datenmodellüberblick



Definition einer Funktion - Beispiel

```
CREATE FUNCTION AnzahlAuftraege (k VARCHAR(30))
RETURNS INTEGER
LANGUAGE SQL
RETURN (SELECT COUNT(*)
          FROM Auftrag
          WHERE Kunde->Name = k);
```

Funktion, die einen Wert liefert!

```
CREATE FUNCTION AnzahlAuftraege ()
RETURNS TABLE(Name VARCHAR(30), Anzahl INTEGER)
LANGUAGE SQL
RETURN (SELECT Kunde->Name, COUNT(*)
          FROM Auftrag
          GROUP BY Kunde->Name);
```

Funktion, die eine Tabelle liefert!

SFW-Block (1)

- SELECT: Was darf in der Projektionsliste stehen?
  - Attribute (auch abgeleitete, berechnete) ✓
  - Methoden-/Funktionsaufrufe ✓
  - Unterabfragen (✓)

Nur skalare Unterabfragen, die genau einen Wert liefern!

- Beispiel:

```
SELECT Name, AnzahlAuftraege(Name),
       (SELECT COUNT(*)
        FROM Auftrag a
        WHERE a.Kunde->Name = k.Name)
FROM Kunde k;
```

SFW-Block (2)

- **FROM**: Welche Tabellenausdrücke sind erlaubt?
  - Tupeltabellen ✓
  - Objekttabellen (auch flache Extensionen) ✓
  - Kollektionsabgeleitete Tabellen —
  - Methoden-/Funktionsaufrufe (die Tabellen liefern) ✓
  - Unterabfragen (abgeleitete Tabellen) ✓ auch rekursive Tabellen!
- **Beispiel:** Funktionsabgeleitete bzw. mittels Unterabfragen abgeleitete Tabellen

```
SELECT *
FROM TABLE(AnzahlAuftraege()) a;
```

```
SELECT *
FROM (SELECT Kunde FROM Auftrag) k;
```

Korrelationsvariable muss in beiden Fällen angegeben werden!

SFW-Block (3)

- **WHERE**: Welche Prädikate sind erlaubt?
  - Prädikate über Attribute ✓
  - Prädikate mit Methoden-/Funktionsaufrufen ✓
  - Prädikate mit Unterabfragen ✓
- **Beispiel:**

```
SELECT *
FROM Kunde k
WHERE Name LIKE 'C%' AND
      AnzahlAuftraege(Name) > 5 AND
      EXISTS(SELECT *
             FROM Lieferant l
             WHERE k.Name = l.Name);
```

Objektrelationale Erweiterungen in Informix

- Neue Basisdatentypen
- Typkonstruktoren
- Benutzerdefinierte Typen
  - Distinct-Typen
  - Benannte Tupeltypen (mit Typvererbung)
- Typisierte Tabellen (mit Subtabellenbildung)
- Typisierte Sichten

Neue Basisdatentypen

- BOOLEAN
  - Werte: 'T', 't', 'F', 'f', NULL
- Large Objects
  - BLOB (Binary Large Object)
  - CLOB (Character Large Object)
  - Operationen
    - FILETOBLOB, FILETOCLOB, LOTOFILE, LOCOPY
    - = (bitweise Gleichheit), IS [NOT] NULL
  - Nicht erlaubt
    - BLOB/CLOB-Attribute als Teil eines Schlüssels oder in booleschen bzw. arithmetischen Ausdrücken
    - GROUP BY, ORDER BY

Verwendung neuer Basisdatentypen

Definition einer Tabelle mit BOOLEAN- und LOB-Attributen:

```
CREATE TABLE MitarbeiterTupelTabelle (
        Name        VARCHAR(30),
        Vollzeit    BOOLEAN,
        Bild        BLOB,
        Bewerbung   CLOB
);
```

Erzeugen von BOOLEAN- bzw. LOB-Attributen:

```
INSERT INTO MitarbeiterTupelTabelle
VALUES ('Billy', 'T',
        FILETOBLOB('D:\ImgSamp\billy.gif', 'client'),
        FILETOCLOB('D:\RsmSamp\billy.rsm', 'server'));
```


Verwendung neuer Basisdatentypen (Forts.)

Kopieren von LOB-Attributwerten:

```
UPDATE MitarbeiterTupelTabelle
SET Bewerbung = (SELECT LOCOPY(Lebenslauf)
                 FROM PersonenTupelTabelle
                 WHERE Name = 'Billy')
WHERE Name = 'Billy';
```

Verwendung von BOOLEAN- und LOB-Attributen:

```
SELECT LOTOFILE(Bewerbung, 'D:\RsmSamp\' || Name || '.rsm', 'server')
FROM MitarbeiterTupelTabelle
WHERE Vollzeit = 'T';
```

Typkonstruktoren

- Tupeltypkonstruktor: ROW
  - Tupelkomponentenselektion mittels Dot-Operator
- Kollektionstypen: SET, MULTISET, LIST
  - Enthaltenseinstest (IN)
  - Kardinalität (CARDINALITY)
- Einschränkungen
  - Kollektionselemente müssen NOT NULL sein
  - Für tupelwertige bzw. kollektionswertige Attribute sind abgesehen von NOT NULL keine Integritätsbedingungen erlaubt
- Typkonstruktoren sind orthogonal (beliebig kombinierbar)

Verwendung von Typkonstruktoren in Tabellendefinitionen

```
CREATE TABLE PersonenTupelTabelle (
        Name            VARCHAR(30),
        Anschrift       ROW(Strasse     VARCHAR(30),
                            Nr          DECIMAL(4),
                            PLZ         DECIMAL(5),
                            Ort         VARCHAR(40),
                            Land        VARCHAR(25)),
        Telefone        LIST(VARCHAR(20) NOT NULL),
        Hobbies         SET(ROW(Hobby   VARCHAR(15),
                                Aktiv   BOOLEAN) NOT NULL),
        Lebenslauf      CLOB
);
```

Verwendung von Typkonstruktoren in Einfügeoperationen

- Einfügen eines neuen Kunden

```
INSERT INTO PersonenTupelTabelle
VALUES ('Billy',
        ROW('Seefeldstrasse', 31, 8008, 'Zurich', 'CH'),
        "LIST{'0041-1-6327248', '0041-1-7337947'}",
        "SET{ROW('Reisen', 'T'), ROW('Sport', 'F')}" ,
        FILETOCLOB('D:\RsmSamp\billy.rsm', 'server'));
```

  – Ungewöhnliche Instanziierung von Kollektionstypen mit Anführungsstrichen

Selektion und Update von Tupelkomponenten

- Zugriff auf Tupelkomponenten mittels Dot-Notation möglich

```
SELECT Name, Anschrift.Ort
FROM KundeTupelTabelle
WHERE Anschrift.Land = 'CH';
```

- Tupelwertige Attribute sind nur als Ganzes änderbar
  – Ändern einzelner Tupelkomponenten nur innerhalb von SPL-Routinen möglich

```
UPDATE KundeTupelTabelle
SET Anschrift = ROW('Kreuzstrasse', 21, 8008, 'Zurich', 'Schweiz')
WHERE Name = 'Billy';
```

Behandlung einzelner Tupelkomponenten

- Nur innerhalb von SPL-Routinen unterstützt

```
CREATE PROCEDURE Prozedurname(Parameterliste)
        ...
END PROCEDURE;
```

- Deklaration einer Tupelvariablen

```
DEFINE a ROW(Strasse VARCHAR(30), Nr DECIMAL(4),
             PLZ DECIMAL(5), Ort VARCHAR(40), Land VARCHAR(25));
```

- Zuweisung eines Werts an eine Tupelkomponente

```
LET a.Ort = 'Wiesbaden';
```


Prozedur zum Ändern des Wohnortes einer Anschrift

```
CREATE PROCEDURE NeuerOrt(kname VARCHAR(30), kort VARCHAR(40))
    DEFINE a ROW(Strasse VARCHAR(30),
                 Nr DECIMAL(4),
                 PLZ DECIMAL(5),
                 Ort VARCHAR(40),
                 Land VARCHAR(25));
    SELECT Anschrift INTO a
    FROM KundeTupelTabelle
    WHERE Name = kname;
    LET a.Ort = kort;
    UPDATE KundeTupelTabelle
    SET Anschrift = a
    WHERE Name = kname;
END PROCEDURE;
```

Änderung durch Prozeduraufruf:

```
CALL PROCEDURE
NeuerOrt('kname', 'kort');
```

Wünschenwert wäre dennoch eine direkte Manipulutation der Form:

```
UPDATE KundeTupelTabelle
SET Anschrift.Ort = 'kort';
WHERE Name = 'kname';
```

Selektion und Update von Kollektionen

- Zugriff auf ganze Kollektion

```
SELECT Name, Telefone
FROM KundeTupelTabelle
WHERE Anschrift.Land = 'CH';
```

- Kollektionswertige Attribute sind nur als Ganzes änderbar
  - Einfügen, Ändern oder Löschen einzelner Kollektionselemente nur innerhalb von SPL-Routinen möglich

```
UPDATE KundeTupelTabelle
SET Telefone = "LIST{'0041-1-6327248', '0041-1-7337947',
                     '0049-6151-295479'}"
WHERE Name = 'Billy';
```


Operationen auf Kollektionen

- Enthaltenseinstest (Nicht-Enthaltensein NOT IN)

```
SELECT Name
FROM KundeTupelTabelle
WHERE '0041-1-9439470' IN Telefone;
```

liefert den Namen der Kunden mit der Telefonnummer '0041-1-9439470'

- Kardinalität

```
SELECT Name, CARDINALITY(Telefone)
FROM KundeTupelTabelle;
```

liefert für jeden Kunden die zugehörige Anzahl von Telefonnummern

Operationen auf Kollektionen (Forts.)

- Kollektion in Tabelle umwandeln (Doppelklammerung bei SFW-Ausdrücken nötig)

```
SELECT *
FROM TABLE((SELECT Auftraege
            FROM KundeTupelTabelle
            WHERE Name='Billy')) a
WHERE a.Anzahl > 10;
```

liefert alle Auftraege vom Kunden 'Billy', die eine bestimmte Anzahl überschreiten

```
SELECT Name, (SELECT a.Anzahl
              FROM TABLE(KundeTupelTabelle.Auftraege) a
              WHERE a.ProduktNr = '435-663-AB')
FROM KundeTupelTabelle;
```

liefert die Kunden und die Anzahl ihrer bestellten Produkte mit der Nummer 17

Behandlung einzelner Kollektionselemente (1)

- Nur innerhalb von SPL-Routinen unterstützt

```
CREATE PROCEDURE Prozedurname(Parameterliste)
        ...
END PROCEDURE;
```

- Deklaration einer Kollektionsvariablen sowie einer Elementvariablen

```
DEFINE telefonnummern LIST(VARCHAR(20) NOT NULL);
DEFINE telefonnummer VARCHAR(20);
```

- Belegen einer Kollektionsvariablen

```
LET telefonnummern = "LIST{'0041-1-9876543'}";
```

Behandlung einzelner Kollektionselemente (2)

- Belegen einer Kollektionsvariable mittels SFW-Klausel

```
SELECT Telefone INTO telefonnummern
FROM KundeTupelTabelle
WHERE Name = 'Billy';
```

- Einfügen eines Elements in eine Liste auf der Position 1

```
INSERT AT 1 INTO TABLE(telefonnummern) VALUES('0041-1-1872321');
```

- Änderungen von Kollektionsvariablen haben keine Wirkung auf der DB
  - Änderungen müssen explizit mittels UPDATE in die DB eingebracht werden

```
UPDATE KundeTupelTabelle
SET Telefone = telefonnummern
WHERE Name = 'Billy';
```


Behandlung einzelner Kollektionselemente (3)

- Cursor für Kollektionsvariablen

```
FOREACH c FOR
   SELECT Telefone INTO telefonnummern FROM KundeTupelTabelle
   FOREACH d FOR
      SELECT * INTO telefonnummer FROM TABLE(telefonnummern)
      IF telefonnummer[1,4] = '0049' THEN
        DELETE FROM TABLE(telefonnummern) WHERE CURRENT OF d;
      END IF;
   END FOREACH;
   UPDATE KundeTupelTabelle
   SET Telefone = telefonnummern
   WHERE CURRENT OF c;
END FOREACH;
```

Prozedur zum Einfügen einer neuen Telefonnummer

```
CREATE PROCEDURE NeueNummer(kname VARCHAR(30), tnr VARCHAR(20))
        DEFINE telefonnummern LIST(VARCHAR(20) NOT NULL);
        DEFINE telefonnummer VARCHAR(20);
        SELECT Telefone INTO telefonnummern
        FROM KundeTupelTabelle
        WHERE Name = kname;
        INSERT AT 1 INTO TABLE(telefonnummern)
        VALUES(tnr);
        UPDATE KundeTupelTabelle
        SET Telefone = telefonnummern
        WHERE Name = kname;
END PROCEDURE;
```

Einfügen durch Prozeduraufruf:

```
CALL PROCEDURE
NeueNummer('kname', 'tnr');
```

Wünschenwert wäre dennoch eine direkte Manipulutation der Form:

```
INSERT AT 1 INTO
TABLE(SELECT Telefone
      FROM KundeTupelTabelle
      WHERE Name = 'kname')
VALUES('tnr');
```


Prozedur zum Entfernen einer bestimmten Telefonnummer

```
CREATE PROCEDURE EntferneTelefonnummer(tnr VARCHAR(20))
  DEFINE nummern LIST(VARCHAR(20) NOT NULL);
  DEFINE nummer VARCHAR(20);
  FOREACH c FOR
    SELECT Telefone INTO nummern FROM KundeTupelTabelle WHERE tnr IN Telefone
    FOREACH d FOR
      SELECT * INTO nummer FROM TABLE(nummern)
      IF nummer = tnr THEN
        DELETE FROM TABLE(nummern) WHERE CURRENT OF d;
      END IF;
    END FOREACH;
    UPDATE KundeTupelTabelle SET Telefone = nummern WHERE CURRENT OF c;
  END FOREACH;
END PROCEDURE;
```

Distinct-Typen

- Kopie eines bereits existierendes Typs
  - Distinct-Typen dürfen auch auf benutzerdefinierten Typen basieren
  - Explizite Cast-Funktionen zur Konversion zwischen Distinct-Typ und Quelltyp und umgekehrt automatisch bereitgestellt
  - Definition impliziter Cast-Funktionen erfordert vorheriges Löschen obiger Funktionen
- Syntax:

```
CREATE DISTINCT TYPE Distinct-Typname AS Quelltyp
```


Beispieldefinition und Verwendung von Distinct-Typen

Definition von Distinct-Typen:

```
CREATE DISTINCT TYPE Franken AS DECIMAL(12,2);
CREATE DISTINCT TYPE Euro AS DECIMAL(12,2);
```

Verwendung von Distinct-Typen in Tabellendefinitionen:

```
CREATE TABLE CHBank(Nr INTEGER, Stand Franken);
CREATE TABLE EuroBank(Nr INTEGER, Stand Euro);
```

Anfrage basierend auf Distinct-Typen:

```
SELECT c.Nr, c.Stand
FROM   CHBank c, EuroBank e
WHERE c.Stand > e.Stand;
```

Ungültige Anfrage: Schweizer Franken und Euro nicht vergleichbar
→ Konvertierung ist notwendig!

Verwendung von Cast-Funktionen

- Alternative, äquivalente Formulierungen

```
SELECT c.Nr, c.Stand
FROM   CHBank c, EuroBank e
WHERE c.Stand::DECIMAL > e.Stand::DECIMAL;

... WHERE CAST(c.Stand AS DECIMAL) < CAST(e.Stand AS DECIMAL);

... WHERE c.Stand::DECIMAL < CAST(e.Stand AS DECIMAL);

... WHERE CAST(e.Stand AS DECIMAL) < e.Stand::DECIMAL;
```


Benutzerdefinierte Cast-Funktionen

- Konversion zwischen unterschiedlichen Datentypen
- Syntax:

```
CREATE [EXPLICIT|IMPLICIT] CAST (Quelltyp AS Zieltyp
                                 WITH Konvertierungsfunktion)
```

  - Konvertierungsfunktion hat genau einen Parameter vom Typ Quelltyp sowie Zieltyp als Rückgabetyp
  - Quelltyp und Zieltyp dürfen nicht vom Typ LIST, SET, MULTISET, ROW, BLOB oder CLOB sein
  - IMPLICIT: Implizite Cast-Funktion (wird automatisch aufgerufen)
  - Defaulteinstellung: EXPLICIT

Explizite benutzerdefinierte Cast-Funktionen - Beispiel

Definition einer Konvertierungsfunktion:

```
CREATE FUNCTION EuroToFranken(e Euro)
RETURNS Franken
RETURN CAST((e::DECIMAL * 1.5) AS Franken);
END FUNCTION;
```

Definition einer expliziten Cast-Funktion:

```
CREATE CAST (Euro AS Franken WITH EuroToFranken);
```

Vorherige Anfrage nun mit direkter Konvertierung von Euro nach Franken:

```
SELECT c.Nr, c.Stand
FROM   CHBank c, EuroBank e
WHERE  c.Stand > CAST(e.Stand AS Franken);
```


Implizite benutzerdefinierte Cast-Funktionen - Beispiel

Definition einer Konvertierungsfunktion:

```
CREATE FUNCTION FrankenToEuro(f Franken)
RETURNS Euro
RETURN CAST((CAST(f AS DECIMAL) * 0.65) AS Euro);
END FUNCTION;
```

Definition einer impliziten Cast-Funktion:

```
CREATE IMPLICIT CAST (Franken AS Euro WITH FrankenToEuro);
```

Vorherige Anfrage nun mit impliziter Konvertierung von Franken nach Euro:

```
SELECT c.Nr, c.Stand
FROM   CHBank c, EuroBank e
WHERE  c.Stand > e.Stand;
```

Achtung: Falls implizite Cast-Funktionen in beiden Richtungen vorhanden sind, wird automatisch nur der rechte Ausdruck des Vergleichsprädikats konvertiert!

Benannte Tupeltypen (Named Row Types)

- Wiederverwendbare Tupeltypen
  - Typ eines Attributes kann auch ein Tupeltyp (ROW; benannt oder unbenannt) oder Kollektionstyp (SET, MULTISET, LIST) sein
  - Typvererbung (Subtyp erbt alle Attribute des Supertyp) und Substituierbarkeit
  - Nicht unterstützt werden
    - Referenztypen und Objektidentifikatoren
    - Kapselung und Methoden
    - Defaultwerte und Integritätsbedingungen (abgesehen von NOT NULL)
- Syntax für Wurzeltypdefinition:

```
CREATE ROW TYPE Typname
(Attributdefinitionsliste)
```

- Attributdefinition:

```
Attributname Typ [NOT NULL]
```


Benannte Tupeltypen - Beispiele

```
CREATE ROW TYPE AdresseTyp (
        Strasse    VARCHAR(30),
        Nr         DECIMAL(4),
        PLZ        DECIMAL(5),
        Ort        VARCHAR(40),
        Land       VARCHAR(25)
);
```

```
CREATE ROW TYPE AuftragTyp (
   ANr        INTEGER,
   Lieferant  VARCHAR(20),
   Positionen LIST(ROW(ArtikelNr INTEGER,
                       Anzahl INTEGER)
                  NOT NULL)
);
```

```
CREATE ROW TYPE KundeTyp (
        KNr          INTEGER NOT NULL,
        Name         ROW(Vorname VARCHAR(25),
                         Nachname VARCHAR(35)) NOT NULL,
        Anschrift    AdresseTyp,
        Telefone     LIST(VARCHAR(20) NOT NULL),
        Auftraege    SET(AuftragTyp NOT NULL)
);
```

Attributtypen für Tupeltabellen

- Mögliche Attributtypen:
  - Basisdatentypen
  - Mittels der Typkonstruktoren konstruierte (unbenannte) Typen
  - Benannte Tupeltypen

```
CREATE TABLE KundeTupelTabelle (
        KNr             INTEGER NOT NULL,
        Name            ROW(Vorname VARCHAR(25),
                              Nachname VARCHAR(35)) NOT NULL,
        Anschrift       AdresseTyp,
        Telefone        LIST(VARCHAR(20) NOT NULL),
        Auftraege       SET(AuftragTyp NOT NULL)
);
```


Typ-Instanziierung in Einfügeoperationen

- Einfügen eines neuen Kunden

```
INSERT INTO KundeTupelTabelle
VALUES (17,
        ROW('Billy', 'Schwarz'),
        ROW('Seefeldstrasse', 31, 8008, 'Zurich', 'Schweiz')::AdresseTyp,
        "LIST{'0041-1-6327248', '0041-1-7337947'}",
        "SET{ROW(13, 'ETHWorld', LIST{ROW(453, 10)}),
             ROW(70, 'XYZ', LIST{ROW(959, 50), ROW(911, 100)})}");
```

  - Tupeltypkonstruktoraufruf mit explizitem Casting (Doppel-Doppelpunkt) notwendig, um eine Instanz eines benannten Tupeltyps zu erzeugen
  - Achtung: Innerhalb von Schachtelungen fallen sowohl das Casting bei benannten Tupeltypen als auch die doppelten Anführungsstriche bei Kollektionstypen weg

Subtypbildung - Aufbau von Typhierarchien

- Syntax:

```
CREATE ROW TYPE Typname
(Attributdefinitionsliste)
UNDER Supertypname
```

  - Subtyp erbt alle Attribute des Supertyps
  - Supertyp muss ein selbst ein benannter Tupeltyp sein
  - Subtyp darf nur maximal einen direkten Supertyp haben
    - Keine (direkte) Mehrfachvererbung möglich

- Beispiel:

```
CREATE ROW TYPE BWKundeTyp (
        Interessen  LIST(VARCHAR(20) NOT NULL)
) UNDER KundeTyp;
```


Lesestoff

Überladen von Routinen und Substituierbarkeit

- Routinen (Prozeduren und Funktionen) sind überladbar
  - Routinen mit demselben Namen, aber unterschiedlicher Signatur sind erlaubt
- Instanz eines Subtyps kann in jedem Kontext benutzt werden, wo eine Instanz eines Supertyps nutzbar ist
  - Eingabeargumente für Routinen, deren formale Parameter auf dem Supertyp definiert sind
  - Rückgabewert einer Routine, für die der Supertyp als Typ definiert wurde
- Informix spricht hier von "Vererbung von Routinen" und sagt, dass "geerbte" Routinen überladen, aber nicht überschrieben werden dürfen

Definition von typisierten Tabellen

- Typisierte Tabelle basiert auf einem benannten Tupeltyp
  - Speichert Instanz dieses Typs als Zeile der Tabelle
  - Besitzt kein OID-Attribut
    - Konzept der Objektidentifikatoren und Referenzen wird nicht unterstützt
  - Kann Integritätsbedingungen enthalten
    - Primär-/Unique-/Fremdschlüssel, Check-Klausel
- Syntax:

```
CREATE TABLE Tabellenname OF TYPE BenannterTupeltyp
[(Integritätsbedingungsliste)]
```

- Beispiel:

```
CREATE TABLE Kunde OF TYPE KundeTyp (
        PRIMARY KEY(KNr)
);
```


Operationen auf typisierten Tabellen

```
INSERT INTO Kunde
VALUES (17,
        ROW('Billy', 'Schwarz'),
        ROW('Seefeldstrasse', 31, 8008, 'Zurich', 'CH')::AdresseTyp,
        "LIST{'0041-1-6327248', '0041-1-7337947'}",
        "SET{ROW(13, 'ETHWorld', LIST{ROW(453, 10)}),
             ROW(70, 'XYZ', LIST{ROW(959, 50), ROW(911, 100)})}");
```

```
UPDATE Kunde
SET Anschrift= ROW('Kreuzstrasse', 21, 8008, 'Zurich', 'CH')::AdresseTyp
WHERE Name.Vorname = 'Billy';
```

```
DELETE FROM Kunde
WHERE Anschrift.Ort = 'Zurich';
```

Subtabellenbildung - Aufbau von Tabellenhierarchien

- Syntax:

```
CREATE TABLE Tabellenname OF TYPE BenannterTupeltyp
[(Integritätsbedingungsliste)]
UNDER Supertabelle
```

  - Typ der Subtabelle muss ein direkter Subtyp des Typs der Supertabelle sein
  - (Tiefe) Extension der Subtabelle muss immer eine Untermenge der (tiefen) Extension der Supertabelle sein
    - Instanzen der Subtabelle sind auch Mitglieder der zugehörigen Supertabellen
  - Subtabelle darf nur maximal eine direkte Supertabelle haben
    - Keine (direkte) Mehrfachspezialisierung möglich
  - Subtabelle kann neue Integritätsbedingungen hinzudefinieren
- Beispiel:

```
CREATE TABLE BWKunde OF TYPE BWKundeTyp
(CHECK(Telefone IS NOT NULL))
UNDER Kunde;
```


Anfragen an Tabellenhierarchien

```
SELECT * FROM Kunde;
```

liefert alle Attributwerte der Kunden

Zugriff auf tiefe Extension einer Supertabelle

```
SELECT * FROM BWKunde;
```

liefert alle Attributwerte der "Besonders Wichtigen" Kunden

Zugriff auf tiefe Extension einer Subtabelle

```
SELECT * FROM ONLY(Kunde);
```

liefert alle Attributwerte der "Nicht Besonders Wichtigen" Kunden

Zugriff auf flache Extension einer Supertabelle

Typisierte Sichten

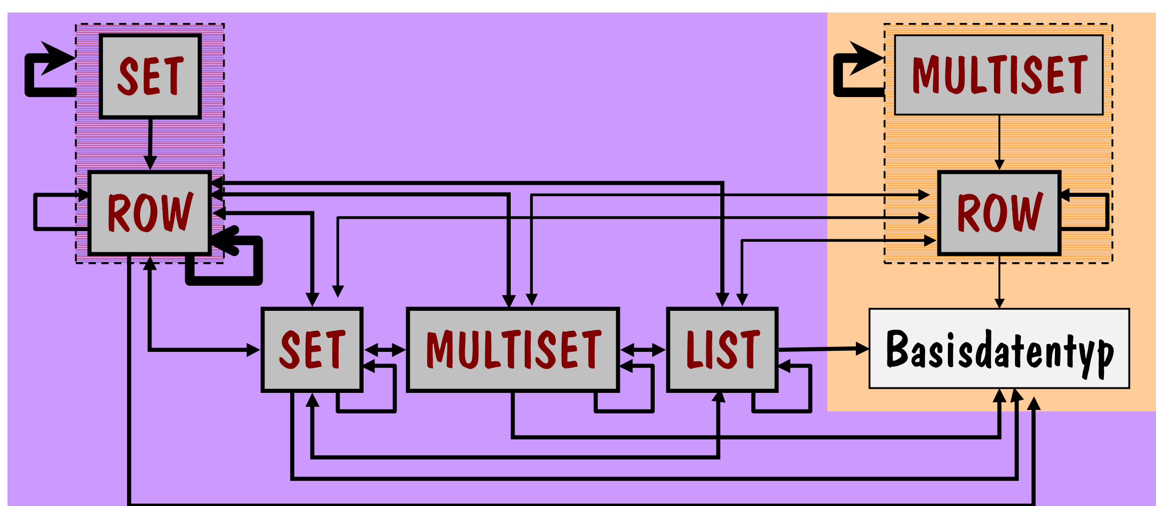
- Analog zu typsierten Tabellen sind typisierte Sichten definierbar
  - Subsichten nicht unterstützt
- Syntax:

```
CREATE VIEW Sichtenname OF TYPE BenannterTupeltyp
AS Anfrageausdruck
[WITH CHECK OPTION]
```

- Beispiel:

```
CREATE VIEW GuterKunde OF TYPE KundeTyp
AS (SELECT *
    FROM Kunde
    WHERE CARDINALITY(Auftraege) > 10);
```

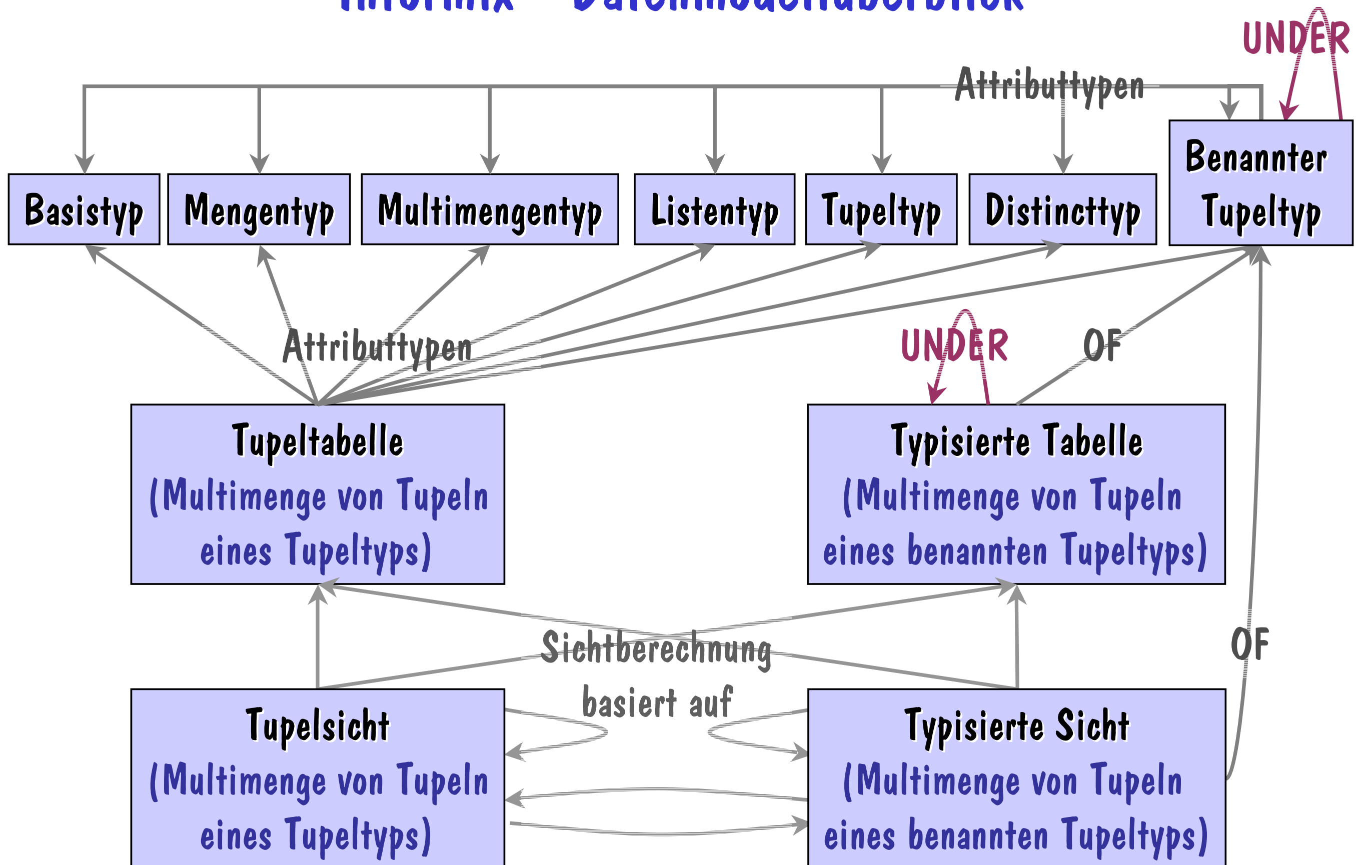

Informix - Datenmodell



Einstiegspunkte

Typisierte Tabelle:	MULTISET(ROW(...))
Untypisierte Tabelle:	MULTISET(ROW(...))

Informix - Datenmodellüberblick



Definition einer Funktion - Beispiel

```
CREATE FUNCTION AnzahlBestellteArtikel(k VARCHAR(20))
   RETURNS INTEGER;
   DEFINE auftragsmenge SET(AuftragTyp NOT NULL);
   DEFINE auftrag AuftragTyp;
   DEFINE gesamt INTEGER;
   DEFINE i INTEGER;
   LET gesamt = 0;
   SELECT (CASE WHEN Auftraege IS NULL THEN 1 ELSE 0 END) INTO i
   FROM Kunde
   WHERE Name.Vorname = k;
   IF i <> 1 THEN
      SELECT Auftraege INTO auftragsmenge FROM Kunde WHERE Name.Vorname = k;
      FOREACH c FOR
         SELECT (SELECT SUM(Anzahl) FROM TABLE(a.Positionen)) INTO i
         FROM TABLE(auftragsmenge) a
         WHERE a.Positionen IS NOT NULL
         LET gesamt = gesamt + i;
      END FOREACH;
   END IF;
   RETURN gesamt;
END FUNCTION;
```

SFW-Block (1)

- **SELECT**: Was darf in der Projektionsliste stehen?
  - Attribute (auch abgeleitete, berechnete) ✓
  - Funktionsaufrufe ✓
  - Unterabfragen ✓
- **Beispiel:**

```
SELECT Name.Vorname, AnzahlBestellteArtikel(k),
       (SELECT COUNT(*)
        FROM TABLE(Telefone) (nummer)
        WHERE nummer LIKE '0041%'),
       (MULTISET(SELECT Anr
                 FROM TABLE(Auftraege))
FROM Kunde k;
```

Skalare Unterabfrage, die genau einen Wert liefert

Unterabfragen, die mehrere Werte liefern, müssen mit MULTISET umhüllt werden

SFW-Block (2)

- **FROM**: Welche Tabellenausdrücke sind erlaubt?
  - (Untypisierte) Tabellen ✓
  - Typisierte Tabellen (auch flache Extensionen) ✓
  - Kollektionsabgeleitete Tabellen ✓
  - Funktionsaufrufe (die Tabellen liefern) —
  - Unterabfragen (abgeleitete Tabellen) ✓
- **Beispiel:** Kollektionsabgeleitete bzw. mittels Unterabfragen abgeleitete Tabellen

```
SELECT *
FROM TABLE((SELECT Auftraege FROM Kunde WHERE Name.Vorname = 'Billy'));
```

```
SELECT *
FROM TABLE(MULTISET(SELECT Auftraege FROM Kunde));
```

SFW-Block (3)

- WHERE: Welche Prädikate sind erlaubt?
  - Prädikate über Attribute ✓
  - Prädikate mit Funktionsaufrufen ✓
  - Prädikate mit Unterabfragen ✓
- Beispiel:

```
SELECT *
FROM Kunde
WHERE Name.Vorname LIKE 'C%' AND
      AnzahlBestellteArtikel(Name.Vorname) > 5 AND
      Telefone IS NOT NULL AND
      EXISTS(SELECT *
             FROM TABLE(Telefone) (nummer)
             WHERE nummer LIKE '0041%');
```


Lesestoff

Vergleich der objektrelationalen Modelle und Resümee

- Neue Basistypen
- Typkonstruktoren
- Benutzerdefinierte Typen
- Objekttabellen
- Objektsichten
- SQL-Spracherweiterungen

Vergleich - Neue Basisdatentypen

Typ	SQL-99	Oracle	DB2	Informix
BOOLEAN	✓	—	—	✓
CLOB	✓	✓	✓	✓
BLOB	✓	✓	✓	✓

Vergleich - Typkonstruktoren

Typen		SQL-99	Oracle	DB2	Informix
Benannte	DISTINCT	✓	—	✓	✓
	OBJECT (strukturiert)	✓	✓	✓	—
	ROW	—	—	—	✓
	VARRAY	—	✓	—	—
	TABLE	—	✓	—	—
Unbenannte	ROW	✓	—	—	✓
	SET	—	—	—	✓
	MULTISET	—	—	—	✓
	LIST	—	—	—	✓
	ARRAY	✓	—	—	—
	REF	✓	✓	✓	—

Vergleich - Typ- und Tabellenhierarchien

	SQL-99	Oracle	DB2	Informix
Benannte Tupeltypen	—	—	—	✓
Benannte Objekttypen	✓	✓	✓	—
Typhierarchien	✓	(✓9i)	✓	✓
Typisierte Tupeltabellen	—	—	—	✓
Typisierte Objekttabellen	✓	✓	✓	—
Tabellenhierarchien	✓	—	✓	✓
Typisierte Tupelsichten	—	—	—	✓
Typisierte Objektsichten	✓	✓	✓	—
Sichtenhierarchien	✓	(✓9i)	✓	—

Vergleich - Allgemeine und weitere Konzepte

	SQL-99	Oracle	DB2	Informix
Objektkonzept mit OIDs & Methoden	✓	✓	✓	—
Benutzerdefinierte Gleichheit von Objekten (Abbildungs-/Ordnungsfunktionen)	✓	✓	—	—
Benutzerdefinierte Cast-Funktionen (Explizit/Implizit)	✓	—	—	✓
Benutzerdefinierte Prozeduren & Funktionen	✓	✓	✓	✓
Rekursion	✓	(✓)	✓	—

Vergleich - Projektionsliste - SELECT

	SQL-99	Oracle	DB2	Informix
Attribute (Attributausdrücke)	✓	✓	✓	✓
Methoden-/Funktionsaufrufe	✓	✓	✓	✓
Einwertige Unterabfragen	✓	✓	✓	✓
Mehrwertige Unterabfragen	—	✓	—	✓

Oracle & Informix: Unteranfragen, die mehr als einen (Tupel-)Wert liefern, müssen explizit mit MULTISET umhüllt werden

Vergleich - Tabellenausdrücke - FROM

	SQL-99	Oracle	DB2	Informix
Tupeltabellen	✓	✓	✓	✓
Objekttabellen (tiefe Extensionen)	✓	✓	✓	✓
Objekttabellen (flache Extensionen)	✓	—	✓	✓
Kollektionsabgeleitete Tabellen	✓	✓	—	✓
Funktionsaufrufe (die Tabellen liefern)	—	—	✓	—
Unterabfragen (abgeleitete Tabellen)	✓	✓	✓	✓

Umwandlung einer Kollektion in eine Tabelle
SQL-99: UNNEST vs. ORACLE & Informix: TABLE

Rekursive Tabellenausdrücke in SQL-99 & DB2 unterstützt!

Vergleich - Prädikate - WHERE

	SQL-99	Oracle	DB2	Informix
Prädikate über Attribute	✓	✓	✓	✓
Prädikate mit Methoden-/Funktionsaufrufen	✓	✓	✓	✓
Prädikate mit Unterabfragen	✓	✓	✓	✓

Resümee

- Objektrelationale Modellerweiterungen können sowohl Schemadefinition als auch Anfrageformulierung erleichtern (aber auch erschweren)
  - Intuitivere Anfrageformulierung
  - Effizientere Anfrageauswertung (Pfadausdrücke vs. Joins)
- Syntax und Semantik der in kommerziellen DBMS implementierten objektrelationalen Konzepte variieren teilweise
  - Datenmodellheterogenität erschwert Verständlichkeit und Portabilität
- Unterschiede besonders sichtbar bei
  - Typkonstruktoren und benutzerdefinierten Datentypen
  - Objektidentifikatoren und Referenzen
  - Mengenwertigen Attributen und geschachtelten Tabellen
  - Typ- und Tabellenhierarchien

Resümee (Fortsetzung)

- Weder SQL-99 noch kommerzielle DBMS definieren ein "sauberes" objektrelationales Datenmodell bzw. eine "saubere" Datenbanksprache
  - Fehlende Orthogonalität der Typkonstruktoren
  - Implizite Typumwandlungen
  - Unterschiedliche Anwendung einiger Typkonstruktoren innerhalb von geschachtelten Anfragen (Informix)
  - Benutzergenerierte Referenzen - OIDs (DB2)
  - Objektrelationale Spracherweiterungen von Standard-SQL sowie kommerzieller SQL-Dialekte sind derzeit zum Teil wenig intuitiv, das u.a. durch die fehlende Orthogonalität der Sprachkonstrukte begründet wird
- Forderung nach Aufwärtskompatibilität steht einem durchgängigen, sauberen Sprachentwurf entgegen