DB2 and IMS Technical Conference
5 - 9 March 2001, Hamburg, Germany

IBM

OO Feature Overview

Serge Rielau, IBM Toronto Lab (srielau@ca.ibm.com)

Content

- Motivation
- OR extensions up to V7.2
- How it all works together
- Developers@Work
- Summary

Motivation

- Relational Model has many goodies:
  - Efficient query optimization and execution
  - Well-defined transaction semantics and support
  - Excellent multi-user performance and robustness
  - Views for data independence, authorization
  - Constraints, triggers, and stored procedures for (shared) business-rule capture/enforcement
  - "The" success story for parallel computing

But...

- World is becoming more complex
  - New data types are appearing
  - Real-world data doesn't fit neatly into tables
    - Entities and relationships (vs. tables)
    - Variance among entities (vs. homogeneity)
    - Set-valued attributes (vs. normalization)
  - Advanced applications bring complex data
    - E.g., CAD/CAM, web data, GIS, ...

The new employee

- Beyond name, rank, and serial number
  - New attribute types (Location ((2-D point))
  - with their methods (distance(*point*, *point*))
  - Employees with flavors
    - Emp, RSM, Programmer, Manager, Temp, ...
  - and relationships
    - Manager, department, projects, ...
  - Employees have behavior
    - age(Emp), qualified(Emp, Job), hire(Emp), ...
- The employee as business object

Where to go....

- So maybe objects are the answer...?
  - Yes,
  - ...., if we can keep all the relational "goodies"

Need some add ons

- Abstract data type
  - e.g. video, text, shape
  - Describes complex facts
- Row type
  - Template to create tables
  - Inheritance
  - Methods on rows
  - Navigation between tables
- SQL99: Merged into Structured Type
  - One type for columns and rows

OR extensions up to V7.2

- structured types
  - inheritance
  - methods
  - nesting
- typed tables
  - user defined object id
  - inheritance
  - methods
  - triggers, RI and check constraints
  - reference types

OR extension up to V7.2 cont'd

- typed views (a.k.a. object views)
  - inheritance
  - methods
  - reference types
  - on multiple legacy tables
  - on multiple typed table hierarchies

Let's get moving!

Address Datatype

- Address type with subclasses
  - CREATE TYPE address AS (number INT, street VARCHAR(20),city VARCHAR(20)) MODE DB2SQL
  - CREATE TYPE us_address UNDER address AS (postalcode INT, state CHAR(2)) MODE DB2SQL
  - CREATE TYPE ger_address UNDER address AS (plz INT) MODE DB2SQL

Constructor for Addresses

- CREATE FUNCTION address
  (num INT, street VARCHAR(20),
  city VARCHAR(20), state VARCHAR(2),
  postal INT, country VARCHAR(20))
  RETURNS address RETURN
  CASE country WHEN 'USA' THEN
  us_address()..state(state)
  ..postalcode(postal)
  WHEN 'GERMANY' THEN
  ger_address()..plz(postal) END
  ..number(num)..street(street)..city(city)

Method on Address

- ALTER TYPE address ADD METHOD print() RETURNS VARCHAR(80) LANGUAGE SQL
  - Method prototype separate from implementation
  - Can also be written in external language
    - Needs to use "transform function" to map.
  - Implementation uses SELF for its own instance
  - IS OF DYNAMIC TYPE for type info on SELF
  - TREAT AS to access dynamic type specific attributes

Method implementation

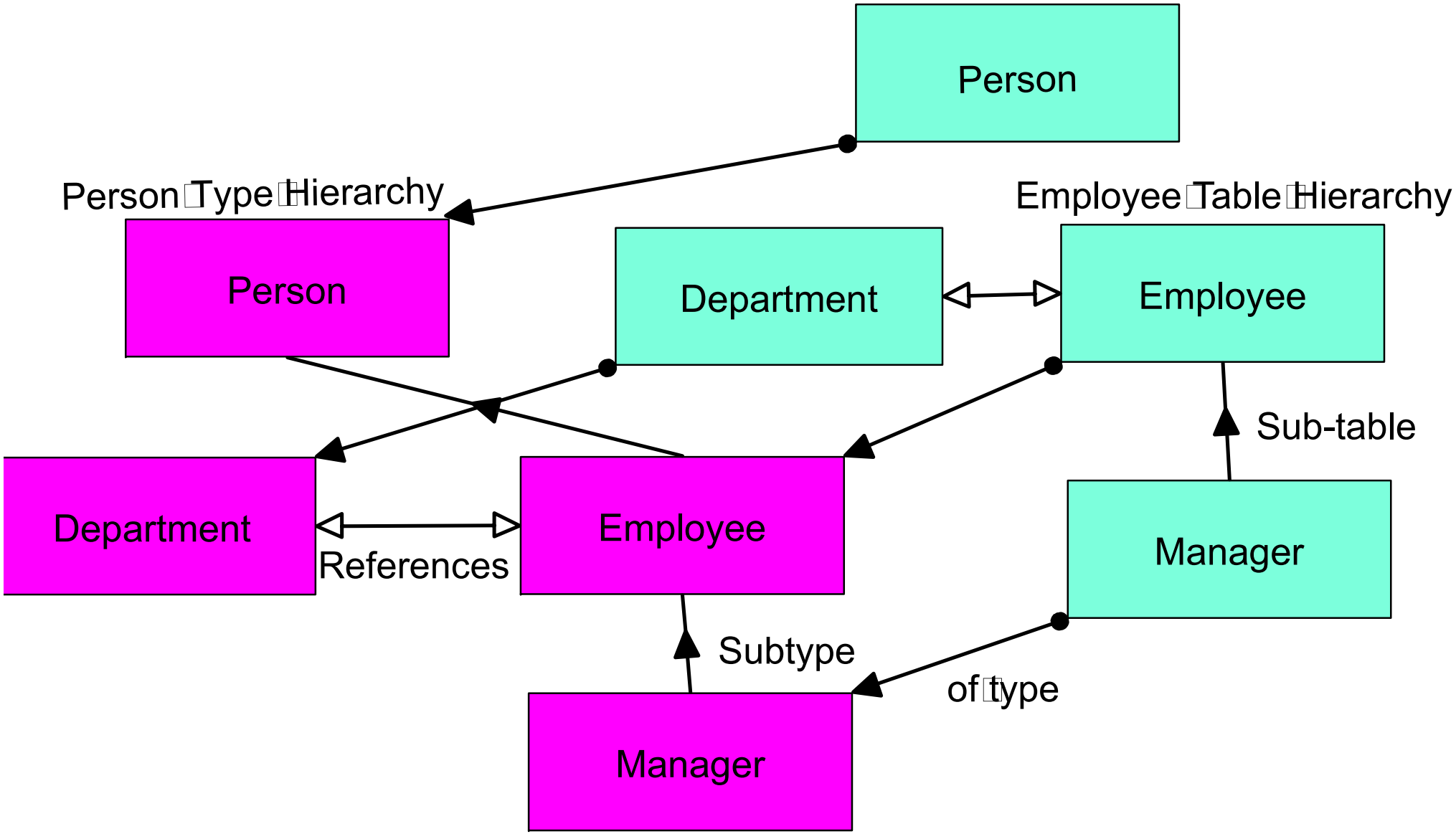
- CREATE METHOD print FOR address
  RETURN CASE
  WHEN SELF IS OF DYNAMIC TYPE (us_address) THEN
  CHAR(SELF..number) || ' ' || SELF..street || ', ' ||
  SELF..city || ', ' || TREAT(SELF AS us_address)..state ||
  ' ' || CHAR(TREAT(SELF AS us_address)..postalcode)
  || ', ' || 'USA'
  ELSE
  SELF..street || ' ' || CHAR(SELF..number) || ', ' ||
  CHAR(TREAT(SELF AS ger_address)..plz) || ' ' ||
  SELF..city || ', ' || 'GERMANY'
  END

Usage of print Method

- VALUES
  address(18, 'Neustr.', 'Emden', '',
  12345, 'GERMANY')..print,
  address(58, 'Bernal Rd.', 'San Jose', 'CA',
  55535, 'USA')..print
- 1

Neustr. 18; 12345 Emden; GERMANY
58 Bernal Rd.; San Jose, CA, 55535; USA

Type Hierarchy for Tables



Type Hierarchy Definition

- CREATE TYPE person AS
  (sin DEC(9,0), name VARCHAR(20),
  firstname VARCHAR(20), address ADDRESS)
  REF USING INT MODE DB2SQL
- CREATE TYPE emp UNDER person AS
  (salary INT, office INT) MODE DB2SQL
- CREATE TYPE mgr UNDER emp AS
  (bonus INT) MODE DB2SQL
- CREATE TYPE dept AS
  (deptname VARCHAR(20), budget INT,
  mgr REF(MGR)) REF USING INT MODE DB2SQL
- ALTER TYPE emp ADD ATTRIBUTE dept REF(DEPT)
  - Cyclic Reference!

Table Definitions

- CREATE TABLE emp OF emp
  (REF IS oid USER GENERATED)
- CREATE TABLE mgr OF mgr UNDER emp
  INHERIT SELECT PRIVILEGES
- CREATE TABLE dept OF dept
  (REF IS oid USER GENERATED,
  oid WITH OPTIONS PRIMARY KEY,
  mgr WITH OPTIONS SCOPE mgr)
- ALTER TABLE emp ALTER COLUMN dept
  ADD SCOPE dept

Constraints

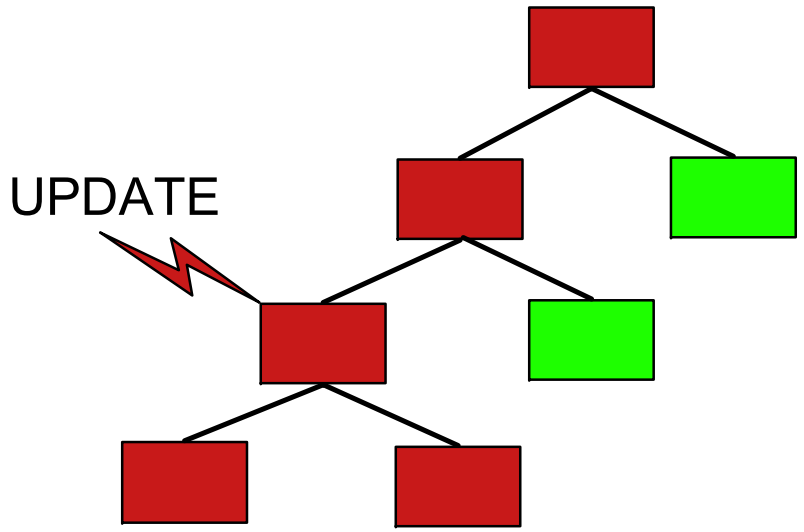
- ALTER TABLE emp ADD FOREIGN KEY (dept) REFERENCES dept(oid) ON DELETE RESTRICT ON UPDATE RESTRICT
  - RI inherits to sub-table mgr
  - Usually matches the scoped references
- ALTER TABLE mgr ADD CONSTRAINT ck1 CHECK (bonus < salary)
  - Check constraint inherited to sub-tables
  - Can include native and non-native attributes

Triggers

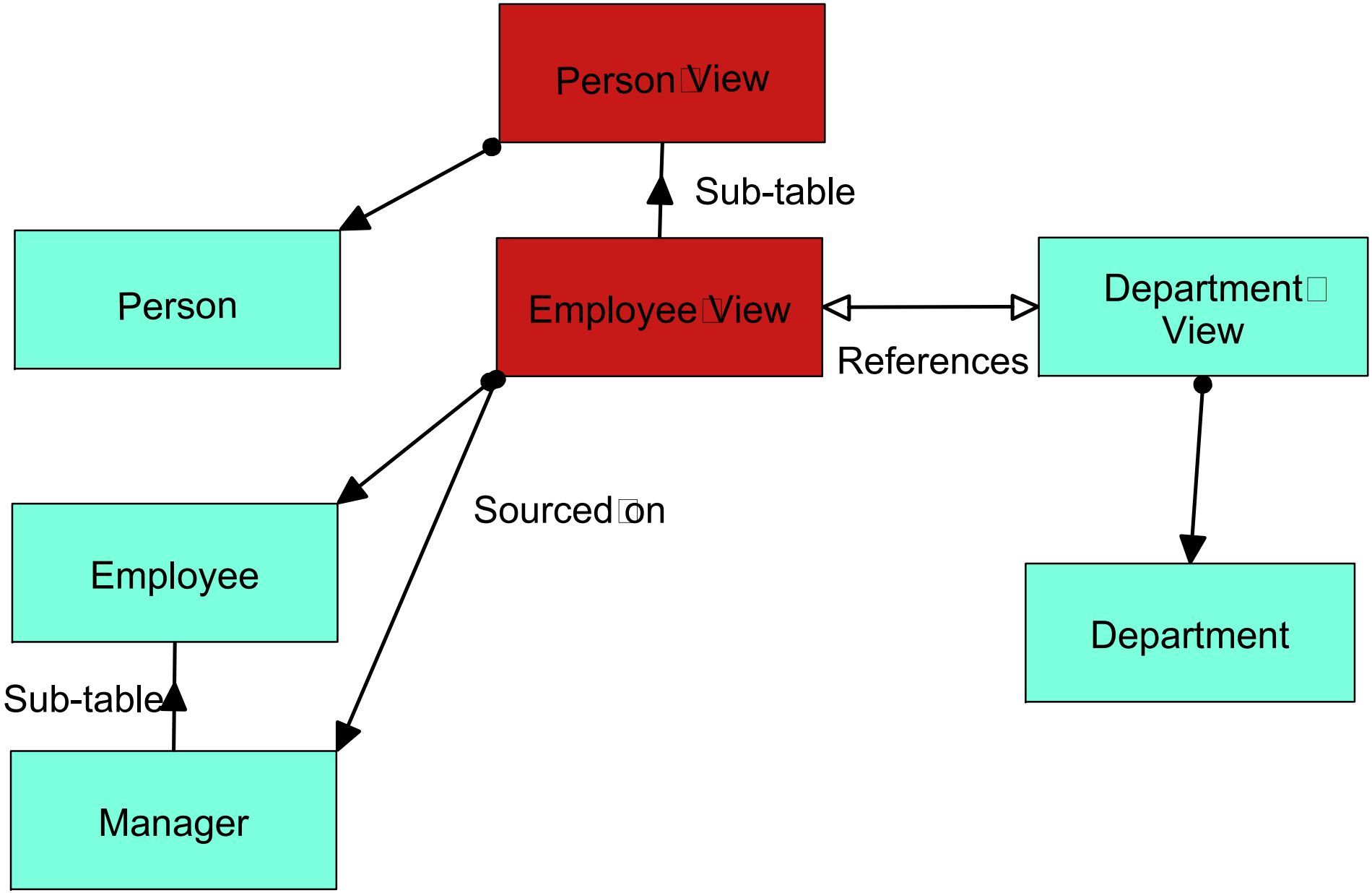
- CREATE TRIGGER emptrg1 NO CASCADE
  BEFORE INSERT ON emp
  REFERENCING NEW AS n
  FOR EACH ROW MODE DB2SQL
  WHEN ((SELECT COALESCE(SUM(salary), 0)
  + n.salary
  FROM emp WHERE dept = n.dept)
  > n.dept->budget)
  SIGNAL SQLSTATE '70001'
  SET MESSAGE_TEXT = 'Cannot afford'

Triggers *(rules)*

- INSERT fires triggers on target table and super-tables
- UPDATE/DELETE also fires statement triggers on sub-tables and row triggers if affected.



View Hierarchies



Type hierarchy for Views

- CREATE TYPE personv AS
  (name VARCHAR(20), firstname VARCHAR(20),
  address ADDRESS)
  REF USING INT MODE DB2SQL
- CREATE TYPE empv UNDER personv AS
  (office INT) MODE DB2SQL
- CREATE TYPE deptv AS
  (deptname VARCHAR(20), mgr REF(EMPV))
  REF USING INT MODE DB2SQL
- ALTER TYPE empv ADD ATTRIBUTE
  dept REF(DEPTV)

View Hierarchy Definition

- CREATE VIEW personv OF personv MODE DB2SQL (REF IS oid USER GENERATED UNCHECKED) AS SELECT personv(INTEGER(oid)), name, firstname, address FROM ONLY(person)
- CREATE VIEW empv OF empv MODE DB2SQL UNDER personv INHERIT SELECT PRIVILEGES AS SELECT empv(INTEGER(oid)), name, firstname, address, office, deptv(integer(dept)) FROM emp

View Hierarchy Def. (cont'd)

- CREATE VIEW deptv OF deptv MODE DB2SQL
  (REF IS oid USER GENERATED,
  mgr WITH OPTIONS SCOPE empv)
  AS SELECT
  deptv(INTEGER(oid)), deptname,
  empv(INTEGER(mgr))
  FROM dept

- ALTER VIEW empv ALTER COLUMN dept
  ADD SCOPE deptv

View Properties

- Can combine multiple hierarchies to make super-hierarchy
- Can include normal relational tables
- Can combine different sources within a sub-view using UNION ALL
- Supports scoped references
- Allows methods on views

Methods on Views

- ALTER TYPE empv ADD METHOD mgrname()
  RETURNS VARCHAR(20) LANGUAGE SQL

- CREATE METHOD mgrname FOR empv
  RETURN
  CAST(SELF..dept AS REF(DEPTV) scope deptv)
  ->mgr->firstname || ' ' ||
  CAST(SELF..dept AS REF(DEPTV) SCOPE deptv)
  ->mgr->name

Let's have some fun

- Create some OID generators
  - CREATE SEQUENCE personid AS INT
  - CREATE SEQUENCE deptid AS REF(DEPTV)
- Insert a department
  - INSERT INTO deptv(oid, deptname)
    VALUES(NEXTVAL FOR deptid, 'DB2'),
    (NEXTVAL FOR deptid, 'Compiler')
  - UPDATE dept SET budget = 1000000
  - OID DEPTNAME MGR
    1 DB2 -
    2 Compiler -

Some more fun

- Need a manager
  - INSERT INTO mgr(oid, name, firstname, salary, bonus, dept)
    VALUES(mgr(NEXTVAL FOR personid),
    'Manager', 'Manfred', 150000,
    50000, dept(1))
  - UPDATE deptv
    SET mgr = empv(PREVVAL FOR personid)
    WHERE oid = deptv(2)

Cont'd fun...

- and some employee to run the shop
  - INSERT INTO emp
    (oid, name, firstname, salary, dept)
    VALUES (emp(NEXTVAL FOR personid),
    'Employee', 'Elton', 50000,
    dept(2))
- Who is Elton's manager?
  - SELECT oid->mgrname AS MGR
    FROM empv WHERE firstname = 'Elton'
  - MGR
    Manfred Manager

Not covered in this talk

- Transform functions
  - bind in, bind out to application
  - map to and from external functions
- embedded SQL support

Developers@Work

- Dynamic dispatch for methods
- method constructors
- Tight integration with Java (SQLJ Part 2)
  - SQL types based on Java classes
  - No transform functions needed
- Further integration (LOAD, ...)
- Collections
- ... you tell us ...

Summary

- DB2 V7.2 supports
  - Structured column and row types
  - Inheritance, nesting and methods
  - advanced integration into RDBMS
    - views
    - triggers
    - RI/check constraints
- More to come