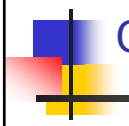


Overview

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

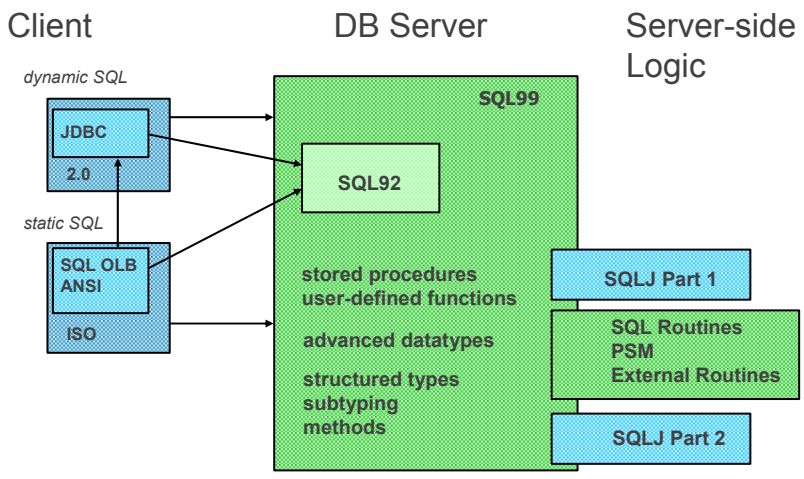
Major Extensions in SQL:1999 and 2003

- Mechanism for "users" to extend the database with application "objects" (specific types and their behavior - functions/methods)
  - User Defined Types (UDTs): Text,Image, CAD/CAM Drawing, Video ...
  - User Defined Functions (UDFs): Contains, Display, Rotate, Play, ...
- Support for storage/manipulation of large data types
  - Large Object Support (LOBs): Binary, Character
- Mechanism to improve the DB integrity and to allow checking of business rules inside the DBMS
  - Triggers: Auditing, Cross-Referencing, Alerts ...
- Means to express complex data relationships such as hierarchies, bills-of-material, travel planning ...
  - Recursion, Common Table Expressions, …
- Support for data analysis, online analytic processing
  - CUBE, ROLLUP, SQL Windows, …
- XML support
  - XML data type, publishing functions, mapping, …

Object-Relational Support

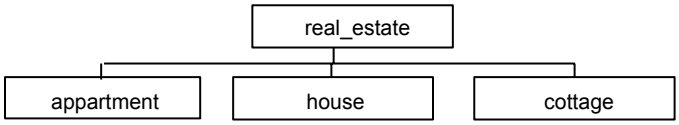
- Major goals
  - support management of complex business objects
  - provide extensibility for defining new, complex data types and behavior
- Key features
  - Large Objects (LOBs)
    - Binary, Character
  - User-Defined Data Types
    - Distinct types, Structured types
  - Type Constructors
    - Row types, Reference types
  - Collection Types
    - Arrays, Multisets
  - User-Defined Methods, Functions, and Procedures
  - Typed tables and views
    - Table hiearchies, View hierarchies (object views)

The "Big Picture"



Subtyping and Inheritance

- Structured types can be a subtype of another UDT
- UDTs inherit structure (attributes) and behavior (methods) from their supertypes
- Example
  - CREATE TYPE real_estate ... NOT FINAL
  - CREATE TYPE appartment UNDER real_estate ... NOT FINAL
  - CREATE TYPE house UNDER real_estate ... NOT FINAL



Structured Types – Nesting and Behavior

```
CREATE TYPE envelope (
    xmin      INTEGER,
    ymin      INTEGER,
    xmax      INTEGER,
    ymax      INTEGER);
```

```
CREATE TYPE point UNDER geometry;
CREATE TYPE line UNDER geometry;
CREATE TYPE polygon UNDER geometry;
```

```
CREATE TYPE geometry (
    gtype       INTEGER,
    refsystem   INTEGER,
    tolerance   FLOAT,
    area        FLOAT,
    length      FLOAT,
    mbr         envelope,
    numparts    INTEGER,
    numpoints   INTEGER,
    points      BLOB(1m),
    zvalue      BLOB(500k),
    measure     BLOB(500k));
```

```
CREATE FUNCTION distance
(s1 geometry, s2 geometry)
RETURNS BOOLEAN
EXTERNAL NAME
'/usr/lpp/db2se/gis!shapedist'
...
```

```
CREATE FUNCTION within
(s1 geometry, s2 geometry)
RETURNS BOOLEAN
EXTERNAL NAME
'/usr/lpp/db2se/gis!shapewithin'
...
```

Structured Types as Column Types

```
CREATE TABLE customers (
    cid       INTEGER,
    name      VARCHAR(20),
    income    INTEGER,
    addr      CHAR(20)
    loc       point);
```

```
CREATE TABLE stores (
    sid       INTEGER,
    name      VARCHAR(20),
    addr      CHAR(20),
    loc       point,
    zone      polygon);
```

```
CREATE TABLE  sales (
    sid       INTEGER,
    cid       INTEGER,
    amount    INTEGER);
```


Structured Types as Column Types

```
SELECT * FROM  stores s, customers c
WHERE  within(c.loc, s.zone)=1
       or distance(c.loc, s.loc)<100
ORDER BY  s.name, c.name;
```

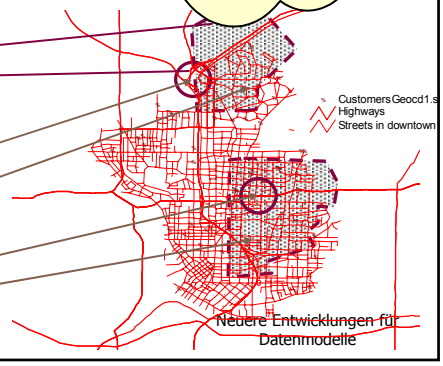
"Tell me all the information I have about each customer who either lives within a stores' zone or within 100 miles of the store."

CUSTOMERS

CID	NAME	INCOME	ADDR	LOC

STORES

SID	NAME	ADDR	LOC	ZONE



Structured Types as Row Types: Typed Tables

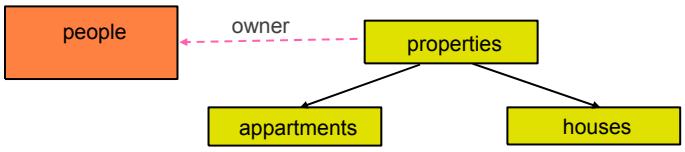
- Structured types can be used to define typed tables
  - Attributes of type become columns of table
  - Plus one column to define REF value for the row (object id)

```
CREATE TYPE real_estate AS
(owner              REF (person),
price               money,
rooms               INTEGER,
size                DECIMAL(8,2),
location            address,
text_description    text,
front_view_image    bitmap,
document            doc) NOT FINAL

CREATE TABLE properties OF real_estate
(REF IS oid USER GENERATED)
```

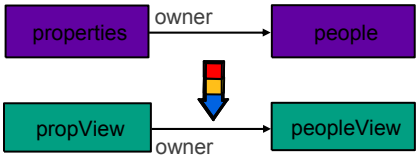

Subtables: Table Hierarchies

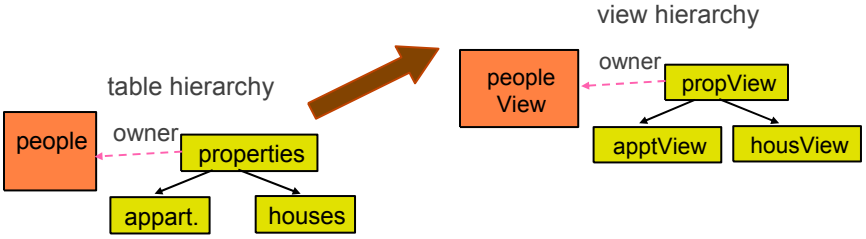
- Typed tables can have subtables
  - Inherit columns, contraints, triggers, ... from the supertable



Object Views

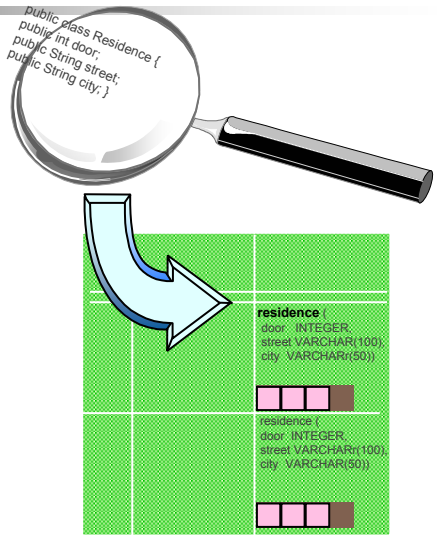
- Views have been extended to support
  - Typed views
  - View hierarchies
  - References on base tables can be mapped to references on views





Mapping Java Objects to Structured Types

- Support built into the DBMS
- Very flexible
  - DB understands internal structure of type
  - Based on SQL type system
  - Client applications written in other programming languages are supported
  - Can be used to define row types/typed tables
  - DB functions/methods can be implemented in other programming language
- Potential for better performance
- Requires conversion (Java <-> SQL)



SQLJ/JRT

- SQL Types using the Java™ Programming Language
- Use of Java classes to define SQL types
  - Can be mapped to structured types or "native" Java types (blobs)
  - Can be used to define columns in tables
  - Can be used to define SQL99 tables (structured types)
- Mapping of object state and behavior
  - Java methods become SQL99 methods on SQL type
  - Java methods can be invoked in SQL statements
- Automatic mapping to Java object on fetch and method invocation
  - Java Serialization
  - JDBC 2.0 SQLData interface

Mapping Java Classes to SQL

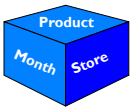
- Described using extended CREATE TYPE syntax
  - DDL statement, or
  - Mapping description in the deployment descriptor
- Supported Mapping

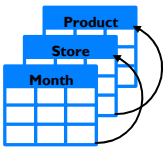
Java	SQL
class	user-defined (structured) type
member variable	attribute
method	method
constructor	constructor method
static method	static method
static variable	static observer method

  - SQL constructor methods
    - Have the same name as the type for which they are defined
    - Are invoked using the NEW operator (just like in Java)
  - SQL does not know static member variables
    - Mapped to a static SQL method that returns the value of the static variable
    - No support for modifying the static variable

SQL99 OLAP SQL Extensions

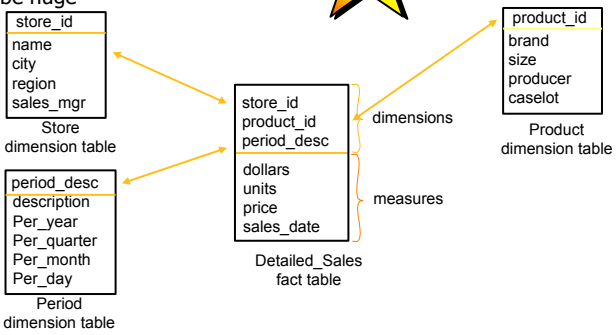
- Extension to GROUP BY clause
- Produces "super aggregate" rows
- ROLLUP equivalent to "control breaks"
- CUBE equivalent to "cross tabulation"
- GROUPING SETS equivalent to multiple GROUP BYs
- Provides "data cube" collection capability
  - Often used with data visualization tool





OLAP Schema

- Typically uses a "STAR" structure
  - Dimension tables tend to be small
  - Fact table tends to be huge

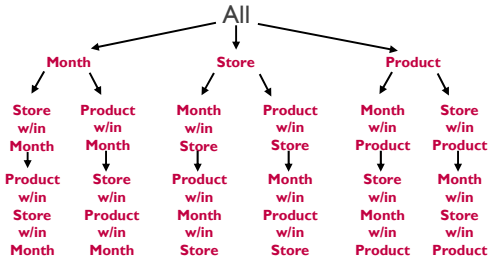



```
CREATE VIEW Sales AS
(SELECT ds.*, YEAR (sales_date) AS year, MONTH (sales_date) AS month, DAY
(sales_date) AS day
FROM (Detailed_Sales NATURAL JOIN Store NATURAL JOIN Product NATURAL JOIN
Period) ds
```

CUBE

- Extends grouping semantics to produce multidimensional grouping and "subtotal" rows
  - Produces "regular" grouped rows
  - Produces same groupings reapplied down to grand total
  - Produces additional groupings on all variants of the CUBE clause

```
SELECT month, city, product_id, SUM(units)
FROM Sales
WHERE year = 1998
GROUP BY CUBE (month, city, product.id)
```



Windows in SQL

- Logical partition of a query
  - user-specified selection of rows
- Determines set of rows used to perform certain calculations for current row
- Example: Moving Average
  - Return, for each store and for each month in the last third of 2001, the total DVD sales during the month at that store, plus the running average of sales over that month and (at most) 2 preceding months.
  - SELECT h.store_id, h.month, h.total_dvd_sales,
    AVG (h.total_dvd_sales) OVER w AS moving_avg
    FROM sales_history AS h
    WHERE h.month BETWEEN 200109 AND 200112
    WINDOW w AS (PARTITION BY h.store_id
    ORDER BY h.month
    ROWS 2 PRECEDING)

XML Origin and Usages

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language, not a database language
  - Documents have tags giving extra information about sections of the document
  - For example:
    - <title> XML </title>
    - <slide> XML Origin and Usages </slide>
- Derived from SGML (Standard Generalized Markup Language)
  - standard for document description
    - enables document interchange in publishing, office, engineering, …
  - main idea: separate form from structure
- XML is simpler to use than SGML
  - roughly 20% complexity achieves 80% functionality

XML Origin and Usages (cont.)

- XML is heavily used for data exchange
- Data interchange is critical in today's networked world
  - Examples:
    - Banking: funds transfer, Order processing
    - Scientific data: Chemistry (ChemML), Genetics (BSML, Bio-Sequence markup Language)
- XML has become the basis for new application-specific data interchange formats
- XML is a key technology for interoperation
- Each XML based standard defines what are valid elements, using
  - XML type specification languages to specify the syntax
    - DTD (Document Type Descriptors)
    - XML Schema
  - Plus textual descriptions of the semantics
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Describing XML Data: XML Schema

- XML Schema is close to the general understanding of a (database) schema
- XML Schema supports
  - Typing of values
    - E.g. integer, string, etc
  - Constraints on min/max values
  - Typed references (for ID and IDREFS)
  - User defined types
  - Specified in XML syntax (unlike DTDs)
  - Integrated with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance ..

XQuery

- XQuery is a general purpose query language for XML data
- Currently being standardized by the World Wide Web Consortium (W3C)
- Alpha version of XQuery engine available free from Microsoft
- XQuery is derived from
  - the **Quilt** ("Quilt" refers both to the origin of the language and to its use in "knitting " together heterogeneous data sources) query language, which itself borrows from
  - **XPath**: a concise language for navigating in trees
  - **XML-QL**: a powerful language for generating new structures
  - **SQL**: a database language based on a series of keyword-clauses: SELECT - FROM – WHERE
  - **OQL**: a functional language in which many kinds of expressions can be nested with full generality

XQuery: FLWR Syntax

- XQuery is a functional language
  - Every query is an expression
  - Expressions can be nested with full generality:
    - XPath expressions
    - Element constructors
    - FLWR expressions
- Simple FLWR expression in XQuery
  - Find all accounts with balance > 400, with each result enclosed in an <account-number> .. </account-number> tag

```
for     $x in /bank-2/account
let     $acctno := $x/@account-number
where $x/balance > 400
return <account-number> {$acctno} </account-number>
```

- Let and Where clause not really needed in this query, and selection can be done in XPath.
  - Query can be written as:

```
for         $x in /bank-2/account[balance>400]
return <account-number> {$x/@account-number}
                                    </account-number>
```


SQL/XML Features

- SQL/XML includes the following:
  - XML data type
    - Enables storage and retrieval of XML documents as typed values
  - Host language bindings for values of XML type
  - XML "publishing functions"
  - Mapping SQL Tables to XML Documents
  - Mapping SQL identifiers to XML Names and vice versa
  - Mapping SQL data types to XML Schema data types
  - Mapping SQL data values to XML

XML Data Type

```
CREATE TABLE employee
   ( id CHAR(6),
     lastname VARCHAR (30),
     ...,
     resume XML
   )
```

ID	LASTNAME	...	RESUME
940401	Long	...	<?xml version="1.0"?> <resume xmlns="http://www.res.com/resume"> <name> … </name> <address> … </address> ... </resume>
862233	Nicks	...	null
766500	Banner	...	<resume ref="http://www.banner.com/resume.html"/>

XML Publishing Functions - *Example*

```
SELECT    e.id,
          XMLELEMENT (NAME "Emp",
                        e.fname || ' ' || e.lname) AS
"result"
FROM      employees e
WHERE     ... ;

==>
```

ID	result
1001	<Emp>John Smith</Emp>
1006	<Emp>Mary Martin</Emp>