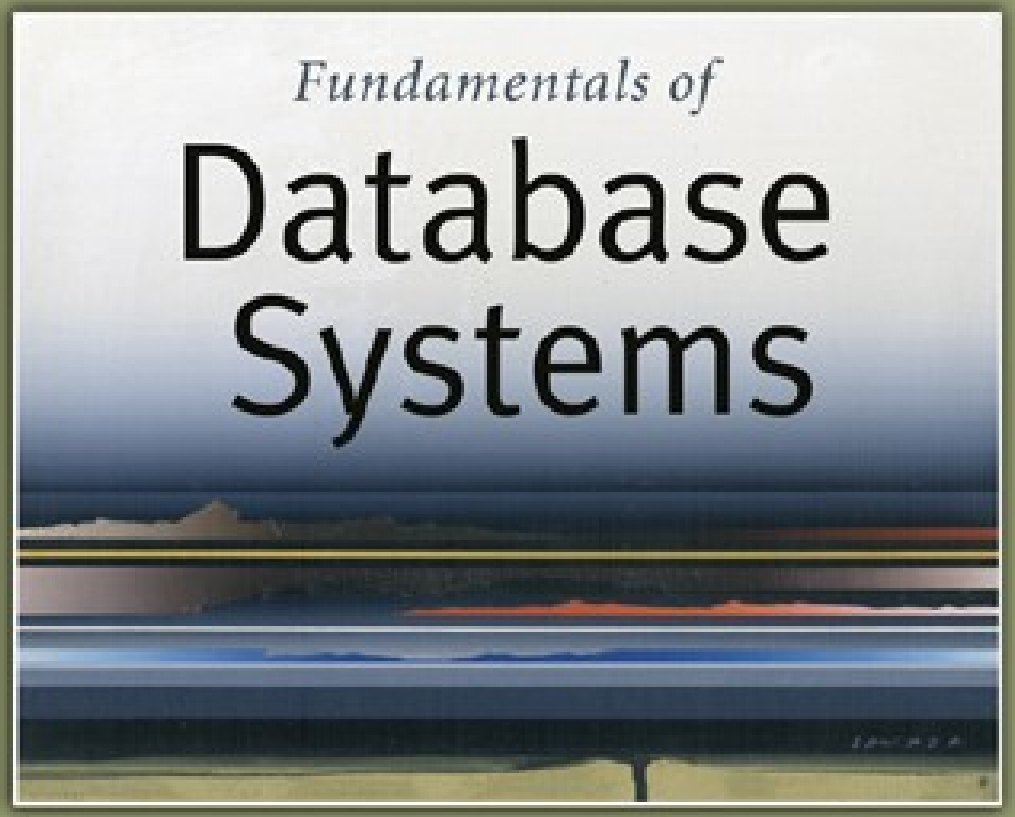
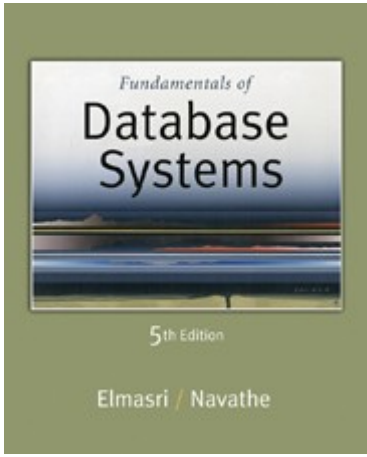


5th Edition

Elmasri / Navathe

Chapter 22

Object-Relational and Extended-Relational Systems



Chapter Outline

- 22.1 Overview of Object-Relational Features of SQL
- 22.2 Evolution and Current Trends
- 22.3 The Informix Server
- 22.4 Object-Relational Features of Oracle
- 22.5 Implementation and Related Issues for Extended Type Systems
- 22.6 The Nested Relational Model
- 22.7 Summary

Chapter Objectives

- To address the following questions:
  - What are the shortcoming of the current DBMSs?
  - What has led to these shortcomings?
- Identify new challenges
  - How Informix Universal Server and Oracle have addressed some of the challenges

Section 22.1
SQL's Object-Relational Features

- SQL was specified in 1970s
- SQL was enhanced substantially in 1989 and 1992
- A new standard called SQL3 added object-oriented features
- A subset of SQL3 standard, now known as SQL-99 has been approved

Component of the SQL Standard

- SQL/Framework, SQL/Foundation, SQL/Bindings, SQL/Object
- New parts addressing temporal, transaction aspects of SQL
- SQL/CLI (Call Level Interface)
- SQL/PSM (Persistent Stored Modules)

SQL/Foundation

- New types
- New predicates
- Relational operators
- Rules and triggers
- User defined types
- Transaction capabilities
- Stored routines

SQL/CLI

- **SQL/CLI** stands for SQL Call Level Interface
- SQL/CLI provides rules that allow execution of application code without providing source code
  - Avoids the need for preprocessing
  - Contains about 50 routines for tasks such as connection to the SQL server

SQL/PSM

- **PSM** = Persistent Stored Modules
  - Specifies facilities for partitioning an application between a client and a server
  - Enhances performance by minimizing network traffic
  - SQL Bindings included Embedded SQL
  - SQL/Temporal deals with historical data

Object-Relational Support in SQL-99

- Type constructors to specify complex objects
- Mechanism to specify object-identity
- Mechanism for encapsulation of operations
- Mechanism to support inheritance
  - I.e., specify specialization and generalization

Type Constructors (1)

- Two types: **row** and **array**
- Known as **user-defined types (UDTs)**
- Syntax for a row type
  - **CREATE TYPE row_type_name AS [ROW] (<component decln>)**
- An example:

```
CREATE TYPE Addr_type AS (
        street VARCHAR (45),
        city VARCHAR (25),
        zip CHAR (5));
```

Type Constructors (2)

- An array type is specified for an attribute whose value will be a collection
- Example:

```
CREATE TYPE Comp_type AS (
        comp_name VARCHAR (2).
        location VARCHAR (20) ARRAY [10]
);
```

- Dot notation is used to refer to components
  - E.g., `comp1.comp_name` is the `comp_name` part of `comp1` (of type `Comp_type`)

Object-Identifiers Using References

- A user-defined type can also be used to specify the row types of a table:

```
CREATE TABLE Company OF Comp_type
(REF IS comp_id SYSTEM GENERATED,
PRIMARY KEY (comp_name));
```

- Syntax to specify object identifiers:

```
REF IS <oid_attribute>
<value_generation_method>
```

- Options:
  - SYSTEM GENERATED
  - or DERIVED

Attributes as References

- A component attribute of one tuple may be a reference:

```
CREATE TYPE Employment_type AS (
    employee REF (Emp_type) SCOPE (Employee),
    company REF (Comp_type) SCOPE (Company));
```

- Keyword **SCOPE** specifies the table whose tuples can be referenced by a reference attribute via the dereferencing notation `->`
  - E.g., `e.company->comp_name`

Encapsulation of Operations

- A construct similar to the class definition
- Users can create a named user-defined type with its own methods in addition to attributes:

```
CREATE TYPE <type-name> (
    list of attributes
    declaration of EQUAL and LESS THAN methods
    declaration of other methods
);
```

Method Syntax

- Syntax:

```
METHOD <name> (<arg-list>) RETURNS <type>;
```

- An example

```
CREATE TYPE Addr_type AS (
    street VARCHAR (45),
    city VARCHAR (25),
    zip CHAR (5)
)
METHOD apt_no ( ) RETURNS CHAR(8);
```

Inheritance in SQL

- **Inheritance** is specified via the **UNDER** keyword
- Example

```
CREATE TYPE Manager_type UNDER Emp_type
    AS (dept_managed CHAR (20));
```

- `Manager_type` **inherits** all features of `Emp_type`
  - and it has an additional attribute called `dept_managed`

Other Operations and New Features

- **WITH RECURSIVE** is used to specify recursive queries
- User accounts may have a **role** that specifies the level of authorization and privileges;
  - Roles can change
- **Trigger granularity** allows row-level and statement-level triggers
- SQL3 also supports programming languages facilities

Section 22.2
Evolution of Database Technology

- Several families of DBMS products
- Two important ones:
  - RDBMS
  - ODBMS
- Two major legacy DBMSs:
  - Network
  - Hierarchical
- Interoperability concerns:
  - While legacy systems are replaced by new offerings, we may encounter various issues

Current Trends

- Main force behind development of ORDBMSs: meet the challenges of new applications:
  - Text
  - Images
  - Audio
  - Streamed data
  - **BLOBs** (binary large objects)

Section 22.3
The Informix Universal Server

- Combines relational and object database technologies
- Consider two dimensions of DBMS applications:
  - Complexity of data (*x*)
  - Complexity of queries (*y*)
- Observe the possible quadrants

Four Quadrants of DBMS Applications

- Observe the possible quadrants
  - Quadrant 1 (*x=0, y=0*): simple data, simple query
  - Quadrant 2 (*x=0, y=1*): simple data, complex query
  - Quadrant 3 (*x=1, y=0*): complex data, simple query
  - Quadrant 4 (*x=1, y=1*): complex data, complex query
- Traditional RDBMSs belong to Quadrant 2
- Many object DBMSs belong to Quadrant 3
- Informix Universal belongs to Quadrant 4
  - It extends the basic relational model by incorporating a variety of features that make it **object-relational**

How Informix Universal Server Extends the Relational Data Model

- Support for extensible data types
- Support for user-defined routines
- Implicit notion of inheritance
- Support for indexing extensions
- Database Blade API

Informix Universal Server's Extensible Data Types

- DBMS is treated as razor into which **data blade modules** can be inserted
- A number of new data types are provided
  - Two-dimensional geometric objects
  - Images
  - Time series
  - Text
  - Web pages

Informix Universal Server's Constructs to Declare Additional Types

- **Opaque type**:
  - Encapsulates a type (hidden representation)
- **Distinct type**:
  - Extends an existing type thru inheritance
- **Row type**:
  - Represents a composite type (like C's `struct`)
- **Collection type**:
  - Lists, sets, multi-sets (bags)

Informix Universal Server's Support for User-Defined Routines

- Informix supports **user-defined functions** and routines to manipulate **user-defined types**
- Functions are implemented
  - Either in **Stored Procedure** (**SPL**)
  - Or in a high-level programming language (such as C or Java)
- Functions can define operations like
  - **plus, times, divide, sum, avg, negate**

Informix Universal Server's Support for Inheritance

- Informix supports inheritance at two levels:
  - Data
  - Operation
- Data inheritance is used to create sub-types (thru the **RETURN** keyword):

```
CREATE ROW TYPE employee_type (…);
CREATE ROW TYPE engineer_type ( …)
UNDER employee_type;
CREATE ROW TYPE engineer_mgr_type ( …)
UNDER engineer_type;
```

Informix Universal Server's Support for Indexing

- Informix supports indexing on user-defined routines in a single table or a table hierarchy:

```
CREATE INDEX empl_city
    ON employee (city (address));
```

- The above line creates an index on the table employee using the value of the city function

Informix Universal Server's Support for External Data Source

- Informix supports **external data sources**
  - E.g., data stored in a file system
- External data are mapped to a table in the database called virtual table interface
- The **interface** enables the user to defined operations that can be used as proxies

Informix Support for Data Blade Application Programming Interface

- Two dimensional (**spatial**) data types
  - E.g., a point, line, polygon, etc.
- **Image** data types:
  - tiff, gif, jpeg, FAX
- **Time series** data type
- **Text data** type:
  - a single data type called **doc** whose instances are large objects

Section 22.4
Object-Relational Features of Oracle

- **VARRAY** for representing multi-valued attributes

```
CREATE TYPE phone_type
    AS OBJECT (phone_number CHAR (10));
CREATE TYPE phone_list_type
    AS VARRAY (5) of phone_type;
CREATE TYPE customer_type AS
    OBJECT (customer_name(VARCHAR (20),
          phone_numbers phone_list_type);
CREATE TABLE customer of customer_type;
SELECT customer_name phone_numbers FROM customer;
```

Managing Large Objects

- Oracle can store extremely large objects:
  - **RBLOB** (binary large object)
  - **CLOB** (character large object)
  - **BFILE** (binary file stored outside the database)
  - **NCLOB** (fixed-width multibyte CLOB)

Section 22.5: Implementation and Related Issues

- The **ORDBMS** must dynamically link a user-defined function in its address space
- **Client-server issues**:
  - if a server needs to perform a function, it is best to do so in the DBMS (server) address space
- Queries should be possible to run inside functions
- Efficient storage and access of data
  - Especially given new types, is very important

Other Issues

- **Object-relational database design**
  - Object-relational design is more complicated
- Query processing and optimization
- Interaction of rules with transactions

Section 22.6
Nested Relational Model

- **Nested relational mode**:
  - Removes the restriction of the first normal form (1NF)
- No commercial database supports a nested relational model
- Visual representation:

Figure 22.1
Illustrating a nested relation. (a) DEPT schema. (b) Example of a Non-1NF tuple of DEPT. (c) Tree representation of DEPT schema.

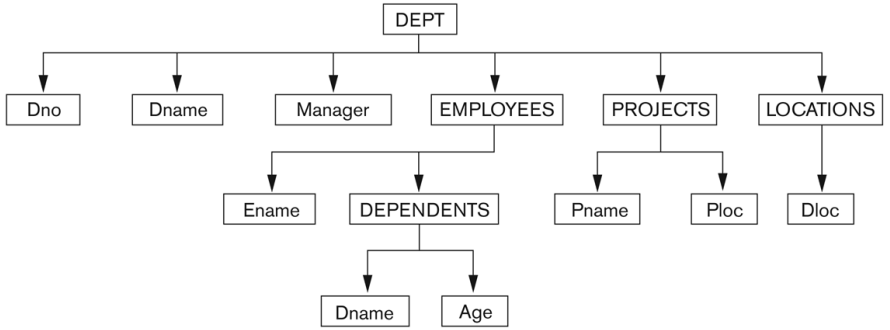
(a)

Dno	Dname	Manager	EMPLOYEES		PROJECTS		LOCATIONS
			Ename	DEPENDENTS	Pname	Ploc	Dloc
				Dname Age			

(b)

4	Administration	Wallace	Zelaya	Thomas	8	New benefits	Stafford	Stafford
				Jennifer	6	Computerization	Stafford	Greenway
			Wallace	Jack	18	Phone system	Greenway	
				Robert	15			
				Mary	10			
			Jabbar					

(c)



Attributes of Nested Relations

- **Simple value** attributes
- **Multi-valued simple** attributes
- **Multi-valued composite** attributes
- **Single-valued composite** attributes

Manipulating Nested Relations

- Extension made to
  - **Relational algebra**
  - **Relational calculus**
  - **SQL**
- Two operations for converting between nested and flat relations:
  - **NEST**
  - **UNNEST**

Example of NEST

- To nest un-nested attributes:

```
EMP_PROJ_FLAT ←
  Π SSN, ENAME, PNUMBER, HOURS (EMP_PRO)
EMP_PROJ_NESTED ←
  NEST PROJ = (PNUMBER,HOURS) (EMP_PROJ_FLAT)
```

- Nested relation **PROJS** within **EMP_PROJ_NESTED** groups together the tuples with the same value for the attributes that are not specified in the **NEST** operation

Example of UNNEST

- UNNEST operation is the inverse of NEST; thus we can recover **EMP_PROJ_FLAT**:

```
EMP_PROJ_FLAT ← UNNEST PROJS =
(PNUMBER,HOURS)  (EMP_PROJ_NESTED)
```

Summary

- An overview of the object-oriented features in SQL-99
- Current trends in DBMS that led to the development of object-relational models
- Features of Informix Universal Server and Oracle
- Nested relational models