COS221
L19 - Object-Relational Model
(Chapter 11 in Edition 6 and Chapter 12 in Edition 7)

Linda Marshall

5 April 2023

Object-Relational Model

- Chamberlin and Boyce (1974)
- Enhancements and Standardised (1989, 1992)
- SQL3 became SQL:99
- SQL/Object incorporated into SQL/Foundation in SQL:2008
- Relational Model + Object database enhancements *referred to* **Object-relational Model**

Object Database Features

- Type Constructors - Row type constructor corresponds to tuple (struct) constructor
- Array Type Constructors - Set, list, bag was added to SQL:2008
- Object Identity - Reference type
- Encapsulation of operations - Used Defined Type (UDTs) may include operations
- Encapsulation of operations - Used Defined Routines (UDRs) definition of general operations
- Inheritance

User-Defined Types (UDTs)

- Type/class declaration abstracted from table declaration
- **CREATE TYPE** TYPE_NAME **AS** (⟨component declarations⟩)
- UDT can either be a type or attribute
- UDT as attribute allows complex nesting
- No operations use **ROW TYPE**
- Use dot notation to refer to row elements in **ROW TYPE**

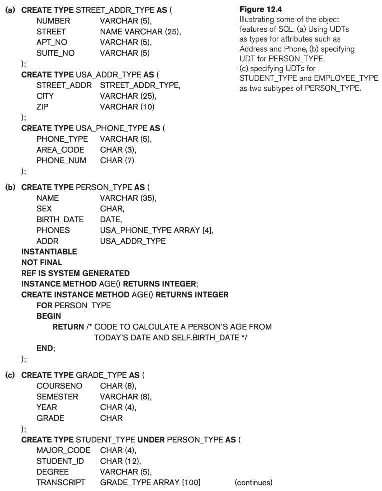
Array Types

- **ARRAY** keyword
- Initially only **ARRAY** type
- Other collections types added later
- Use [ ] common notation to reference element
- **CARDINALITY** - Returns number of elements

User-Defined Types - Row Type

```
CREATE TYPE USA_ADDR_TYPE AS (
    STREET_ADDR     ROW (  NUMBER          VARCHAR (5),
                           STREET_NAME     VARCHAR (25),
                           APT_NO          VARCHAR (5),
                           SUITE_NO        VARCHAR (5) ),
    CITY            VARCHAR (25),
    ZIP             VARCHAR (10)
    );
```

User-Defined Types - Complex



```
(a) CREATE TYPE STREET_ADDR_TYPE AS (
        NUMBER          VARCHAR (5),
        STREET          NAME VARCHAR (25),
        APT_NO          VARCHAR (5),
        SUITE_NO        VARCHAR (5)
    );
    CREATE TYPE USA_ADDR_TYPE AS (
        STREET_ADDR     STREET_ADDR_TYPE,
        CITY            VARCHAR (25),
        ZIP             VARCHAR (10)
    );
    CREATE TYPE USA_PHONE_TYPE AS (
        PHONE_TYPE      VARCHAR (5),
        AREA_CODE       CHAR (3),
        PHONE_NUM       CHAR (7)
    );

(b) CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR (35),
        SEX             CHAR,
        BIRTH_DATE      DATE,
        PHONES          USA_PHONE_TYPE ARRAY [4],
        ADDR            USA_ADDR_TYPE
    INSTANTIABLE
    NOT FINAL
    REF IS SYSTEM GENERATED
    INSTANCE METHOD AGE() RETURNS INTEGER;
    CREATE INSTANCE METHOD AGE() RETURNS INTEGER
        FOR PERSON_TYPE
        BEGIN
            RETURN /* CODE TO CALCULATE A PERSON'S AGE FROM
                   TODAY'S DATE AND SELF.BIRTH_DATE */
        END;
    );

(c) CREATE TYPE GRADE_TYPE AS (
        COURSENO        CHAR (8),
        SEMESTER        VARCHAR (8),
        YEAR            CHAR (4),
        GRADE           CHAR
    );
    CREATE TYPE STUDENT_TYPE UNDER PERSON_TYPE AS (
        MAJOR_CODE      CHAR (4),
        STUDENT_ID      CHAR (12),
        DEGREE          VARCHAR (5),
        TRANSCRIPT      GRADE_TYPE ARRAY [100]
```

(continues)

Figure 12.4
Illustrating some of the object features of SQL. (a) Using UDTs as types for attributes such as Address and Phone, (b) specifying UDT for PERSON_TYPE, (c) specifying UDTs for STUDENT_TYPE and EMPLOYEE_TYPE as two subtypes of PERSON_TYPE.

User-Defined Types - Complex (Cont.)

Figure 12.4 (continued)
Illustrating some of the object features of SQL. (c) (continued) Specifying UDTs for STUDENT_TYPE and EMPLOYEE_TYPE as two subtypes of PERSON_TYPE, (d) Creating tables based on some of the UDTs, and illustrating table inheritance, (e) Specifying relationships using REF and SCOPE.

```
    INSTANTIABLE
    NOT FINAL
    INSTANCE METHOD GPA( ) RETURNS FLOAT;
    CREATE INSTANCE METHOD GPA( ) RETURNS FLOAT
        FOR STUDENT_TYPE
        BEGIN
            RETURN /* CODE TO CALCULATE A STUDENT'S GPA FROM
                       SELF.TRANSCRIPT */
        END;
    );
    CREATE TYPE EMPLOYEE_TYPE UNDER PERSON_TYPE AS (
        JOB_CODE        CHAR (4),
        SALARY          FLOAT,
        SSN             CHAR (11)
    INSTANTIABLE
    NOT FINAL
    );
    CREATE TYPE MANAGER_TYPE UNDER EMPLOYEE_TYPE AS (
         DEPT_MANAGED CHAR (20)
    INSTANTIABLE
    );

(d) CREATE TABLE PERSON OF PERSON_TYPE
        REF IS PERSON_ID SYSTEM GENERATED;
    CREATE TABLE EMPLOYEE OF EMPLOYEE_TYPE
        UNDER PERSON;
    CREATE TABLE MANAGER OF MANAGER_TYPE
        UNDER EMPLOYEE;
    CREATE TABLE STUDENT OF STUDENT_TYPE
        UNDER PERSON;

(e) CREATE TYPE COMPANY_TYPE AS (
        COMP_NAME       VARCHAR (20),
        LOCATION        VARCHAR (20));
    CREATE TYPE EMPLOYMENT_TYPE AS (
        Employee REF (EMPLOYEE_TYPE) SCOPE (EMPLOYEE),
        Company REF (COMPANY_TYPE) SCOPE (COMPANY) );
    CREATE TABLE COMPANY OF COMPANY_TYPE (
        REF IS COMP_ID SYSTEM GENERATED,
        PRIMARY KEY (COMP_NAME) );
    CREATE TABLE EMPLOYMENT OF EMPLOYMENT_TYPE;
```

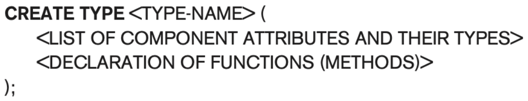
Object Identifiers

- **REF** keyword
- Object identifiers created via reference type
- **REF IS** ⟨ OID_ATTRIBUTE ⟩ ⟨VALUE_GENERATION_METHOD⟩
- **REF IS SYSTEM GENERATED** or **REF IS DERIVED** For SYSTEM GENERATED, a unique identifier will generated by the system to identify the object. DERIVED will use the given primary key as the unique identifier for the object.

Create Tables

- Requires the `INSTANTIABLE` keyword
- Allows table to be created using UDT
- In example create PERSON table using PERSON_TYPE UDT
- UDTs `STREET_ADDR_TYPE`, `USA_ADDR_TYPE` and `USA_PHONE_TYPE` are *noninstantiable*

Encapsulation of Operations



```
CREATE TYPE <TYPE-NAME> (
    <LIST OF COMPONENT ATTRIBUTES AND THEIR TYPES>
    <DECLARATION OF FUNCTIONS (METHODS)>
);
```

- Own behavioural specification
- Can specify code in external file
- Can specify code in type declaration

Encapsulation of Operations - Built-In

For a UDT called TYPE_T:

- **Constructor Function** `TYPE_T()` — Returns new object of that type
- **Observer Function** `A(X)` or `X.A` — Returns the value of attribute `A` of `TYPE_T` for variable `X`
- **Mutator Function** — Update an attribute
- `EXECUTE` privilege

Encapsulation of Operations - User-Defined Functions

Two types of functions can be defined:

- Internal Function — Written in extended PSM language of SQL
- External Function — Written in a host language and only signature appears in UDT
- Three categories
  - PUBLIC — Visible at the UDT interface
  - PRIVATE — Not visible at the UDT interface
  - PROTECTED — Visible only to subtypes
- Virtual Attributes - Computed using functions

Encapsulation of Operations - Internal Functions

INSTANCE METHOD <NAME> (<ARGUMENT_LIST>) **RETURNS** <RETURN_TYPE>;

Encapsulation of Operations - External Functions

```
DECLARE EXTERNAL <FUNCTION_NAME> <SIGNATURE>
LANGUAGE <LANGUAGE_NAME>;
```

Inheritance and Overloading Functions - Type Inheritance

- Inheritance can be applied to tables or attributes
- **UNDER** keyword
- Attributes and instance methods are inherited (Default)
- **NOT FINAL** — UDTs are closed/final by default, must therefore be specified in order to create a subtype.

Inheritance and Overloading Functions - Type Inheritance Rules

- All attributes are inherited.
- The order of supertypes in the UNDER clause determines the inheritance hierarchy.
- An instance of a subtype can be used in every context in which a supertype instance is used.
- A subtype can redefine any function that is defined in its supertype, with the restriction that the signature be the same.
- When a function is called, the best match is selected based on the types of all arguments.
- For dynamic linking, the types of the parameters are considered at runtime.

Inheritance and Overloading Functions - Table Inheritance

- Executed using the supertable/subtable facility
- **UNDER** keyword
- A new record inserted/updated/deleted in subtable is inserted/updated/deleted into supertable
- Corresponds to extent inheritance

Specify Relationship via Reference

- Component attribute of one tuple may reference a tuple of another table
- **SCOPE** keyword
- Specifies the name of the table whose tuples can be referenced by the reference attribute
- System generated value used instead of foreign key
- Use dereference symbol -⟩ for attribute type **REF**
- Use dot notation to build path expressions