

9 Object-Relational Databases and SQL:1999

9.1 Shortcomings of the Conventional Database Technology

- The conventional database models were mainly developed to store alpha-numeric data that can be well organized as tables.
- In recent years, databases are needed to store data that do not fit tables, for example multimedia data.
- When dealing with such data, the conventional database models have limitations:
 - A conventional data model, especially the relational model, is too simple for modeling complex nested entities.
 - Conventional database systems support only a limited set of atomic data types.
 - The conventional database systems do not support operations needed to manipulation of such data, for example, playing music, display images.
 - The performance of conventional database systems, is unacceptable for various types of compute-intensive applications.
- Approaches to overcome:
 - The **extended relational approach** starts with the relational model and a relational query language, and extends them in various ways to allow the modeling and manipulation of additional semantic relationships and database facilities
 - The **object-oriented approach** starts with an object-oriented data model and a database language that captures it, and extends them in various ways to allow additional capabilities.

9.2 SQL:1999 and the Object-Relational Model

- SQL:1999 is an extension of SQL-86 and SQL-92. Applications in SQL-92 are compatible with SQL:1999.
- SQL:1999 supports the **Object-Relational Model**.

- An object-relational database consists of a set of tables (relations), which is the logical view of the database.
- In an object-relational database, the type of a table column (attribute) does not have to be a primitive (atomic) type (integer, floating, or character string).
- Major features of SQL:1999 include **Large Objects (LOBs)**, **User Defined Types (UDTs)**, **User Defined Functions (UDFs)**, and user defined **Triggers**.

9.3 Collection Types

- SQL:1999 allows a tuple to have a collection of values in a column.
- Such a column must be of a **collection type**.
- Collection types in SQL:1999 include **set** and **array**.
- An Example,

```
CREATE TABLE books
(
    ...
    keyword-set SETOF(VARCHAR(20)),
    ...
    author-array VARCHAR(20) ARRAY(10),
    ...
)
```

9.4 Large Object Data Types

- Two new data types that support very large objects:
 - BLOB (Binary Large Object)
 - CLOB (Character Large Object)

- An Example,

```
CREATE TABLE employee
( id          INTEGER,
  name        VARCHAR(30),
  salary      US_DOLLAR,
  ...
  resume      CLOB(75K),
  signature   BLOB(1M),
  picture     BLOB(12M))
```

9.5 User Defined Data Types

- SQL:1999 supports the Object-Relational Model via **User Defined Data Type (UDT)**.
- A UDT is a named, user-defined data type with behavior and an encapsulated internal structure.
 - UDTs support complex structures.
 - A UDT has a set of functions.
 - UDTs are completely encapsulated.
 - Types of UDT attributes may be primary data types or other UDTs.
- A UDT is quite similar to an object class.
- An Example

```
CREATE TYPE Address
( street     CHAR(30),
  city       CHAR(20),
  province   CHAR(2),
  postcode   CHAR(6));
```

```
CREATE DISTINCT TYPE bitmap AS BLOB;
```

```
CREATE TYPE Real-estate
(
  rooms      INTEGER,
  size       DECIMAL(8.2),
  location   Address,
  text_description
              VARCHAR(1024),
  front_view_image
              bitmap,
  ...
  FUNCTION   display_front_view()
              <code to display an image>
);
```

- A UDT can be used as
 - a type of attributes of other UDTs,
 - a type of parameters of functions and procedures, and
 - a type of columns in tables.
 - a type of tables.

* An example

```
CREATE TABLE real-estate of Real-estate
```

9.6 Unnamed Row Types

- In a UDT, an attribute can have a UDT as its type. The component UDT can be created using “CREATE TYPE” and is given a name, like the “Address” UDT in the above example.
- We can also use an unnamed **Row Type** to define a UDT.
- An Example

```

CREATE TYPE Real-estate
(
  rooms      INTEGER,
  size       DECIMAL(8.2),
  location   ROW (
    street   CHAR(30), city CHAR(20),
    province CHAR(2), zip  CHAR(6)),
  text_description VARCHAR(1024),
  ...
  FUNCTION   display_image(real_estate)
    <code to display an image>
);

```

9.7 Type Inheritance

- In SQL:1999, we can create subtypes that inherit attributes and functions of their super types. Multiple inheritance is allowed.
- An Example:

```

CREATE TYPE Person
(
  name VARCHAR(20), address Address,
  dept VARCHAR(20));

CREATE TYPE Student UNDER Person
(
  degree VARCHAR(20), semester VARCHAR(2));

CREATE TYPE Teacher UNDER Person
(
  salary INTEGER, office VARCHAR(10));

CREATE TYPE TeachingAssistant
  UNDER Student, Teacher;

```

9.8 Reference Types

- In SQL:1999, an attribute of a type can be a reference to an object of that type.
- An Example:

```
CREATE TYPE Department
( name VARCHAR(20),
  head REF(Person) SCOPE people
)
```

- In the above example, “people” is a table. The restriction of the SCOPE of a reference to tuples of a table is mandatory in SQL:1999.

9.9 User Defined Functions

- A **function** can be defined as

```
CREATE FUNCTION Function-name
(parameter1 ptype1, ... parametern ptypen)
RETURN return-type
BEGIN
  DECLARE
    variable1 vtype1,
    ...
    variablem vtypem,
    ...
END
```

- A function defined within a UDT is a **method**.
- A function can be defined independent of any UDTs.

```

CREATE TYPE Real_estate
(
  rooms      INTEGER,
  size       DECIMAL(8.2),
  location   address,
  text_description
             VARCHAR(1024),
  front_view_image
             bitmap);
CREATE FUNCTION display_image(r-e Real_estate)
  <code to display an image>

```

- A method can be invoked as

```

BEGIN
  DECLARE r Real_estate;
  ...
  r.display_image;
  ...
  display_image(r);
  ...
END

```

9.10 User Defined Procedures

- SQL:1990 also supports **procedures**.
- A procedure can be defined as

```

CREATE PROCEDURE Procedure-name
  (IN in-parameter1 in-ptype1, ...
  OUT out-parameter1 out-ptype1 ...)
BEGIN

```

```

DECLARE
    variable1 vtype1,
        ...
    variablem vtypem,
        ...
END

```

9.11 Constructs in a Function/Procedure Body

- An SQL query of SFW can be used in a function or procedure body:

```

CREATE PROCEDURE author-count
    (IN title VARCHAR(20),
    OUT a-count INTEGER)
BEGIN
    SELECT COUNT(author) into a-count
    FROM authors
    WHERE authors.title=title
END

```

- In addition, SQL:1999 supports a variety of procedural constructs, which gives it almost all the power of a general purpose programming language.
 - WHILE *condition* DO ... END WHILE
 - REPEAT ... UNTIL *condition* END REPEAT
 - FOR *tuples in a table* DO ... END FOR
 - IF *condition* ... ELSE ... END IF

9.12 Triggers

- A **Trigger** is a named database object that gets implicitly activated whenever a “triggering event” occurs.

- An example

```
CREATE TRIGGER take_action
  AFTER UPDATE OF balance ON accounts
  REFERENCING NEW AS new_value
  FOR EACH ROW
  WHEN (new_value.balance < 0)
  IF account_type = 'VIP' THEN
    INSERT INTO send_letters ...
  ELSE
    INSERT INTO blocked_accounts ...;
```

- Triggers can be used to
 - improve the database integrity, and
 - allow checking of business rules in databases.