Chapter 4

Object-Relational SQL

(Oracle)

eid	ename	position	dependents	
			dep_name	dep_age
e001	Smith, John	Agent	Michael J.	9
			Susan R.	7
e002	Andrews, David	Superintendent	David M. Jr.	10
e003	Jones, Franklin	Agent	Andrew K.	11
			Mark W.	9
			Louisa M.	4

Figure 4.1 An employees Table with a Multi-Valued dependents Attributes

sid	name		class	telephone	enrollment	
	lname	fname			cno	major
1	Jones	Allan	2	555-1234	101	No
					108	Yes
2	Smith	John	3	555-4321	105	No
3	Brown	Harry	2	555-1122	101	Yes
					108	No
5	White	Edward	3	555-3344	102	No
					105	No

Figure 1.1b Object-Relational Student Enrollment Database

Content

1. Object Types

   The REF Object Reference

2. Collection Types

   nested tables and varrays

3. PL/SQL Procedures, UDFs, and Methods

4. Others

复习指导

- 对象类型（Object Type）
  - 类型创建命令
  - 构造函数
  - 对象及其成员属性的访问
  - REF类型

- 集合类型（Collection Type）
  - 区分Oracle数据库中两种不同的集合类型：Nested Table ＆ Array

1. Object Types

1.1 定义新的数据类型（对象类型）

1.2 对象类型的使用方法

1.3 对象值的创建、查询与更新

1.4 对象的引用类型

1. Object Types

1.1 定义新的数据类型（对象类型）

1.2 对象类型的使用方法

1.3 对象值的创建、查询与更新

1.4 对象的引用类型

1.1 定义新的数据类型

- 对象类型的创建与删除

```
CREATE TYPE typename AS OBJECT
  (attrname datatype, ……);
```

```
DROP TYPE typename;
```

- 这些数据类型一经定义，便以持久形式保存在数据库系统中，用户可以像使用系统内置的数据类型一样使用这些复杂的数据类型，以此来扩充系统的数据类型。

1.1 定义新的数据类型

Example 4.2.1：创建一个’姓名’(name_t)类型

```
CREATE TYPE name_t AS OBJECT (
    lname  varchar(30),
    fname  varchar(30),
    mi  char(1)
);
```

1.1 定义新的数据类型

- 例如：创建一个 ‘文档’ 类型

```
CREATE TYPE Document AS OBJECT (
    name  varchar(50),
    author   varchar(30),
    date Date
);
```

创建「表」的命令

```
CREATE TABLE  customers (
        cid  char(4) not null,
        cname  varchar(13),
        city  varchar(20),
        discnt  real,
        PRIMARY KEY(cid) );
```

1. Object Types

1.1 定义新的数据类型（对象类型）

1.2 对象类型的使用方法

1.3 对象值的创建、查询与更新

1.4 对象的引用类型

1.2 对象类型的使用

1) 使用所创建的对象类型来创建新类型
   - 用于定义新类型中的属性 (类型的嵌套)

2) 使用所创建的对象类型来创建新的表
   - 用于定义表中的属性

如同使用普通数据类型一样，可以用对象类型来定义表或对象类型中的属性

3) 使用对象数据类型来直接创建一张表
   - 新创建的‘表’的结构与对象类型的结构相同
   - 可以在创建的‘表’中增加完整性约束定义

1&2) 使用已有的对象类型来创建新的类型或表

➢**Example 4.2.3: 创建新的数据类型**

```
CREATE TYPE person_t AS OBJECT
(
    ssno        int,
    pname       name_t,
    age         int
);
```

1&2) 使用已有的对象类型来创建新的类型或表

Exp. 4.2.3: 用类型 name_t 创建新的类型 person_t

```
CREATE  TYPE  person_t  AS  OBJECT (
        ssno          int,
        pname         name_t,
        age           int  );
```

```
CREATE TYPE name_t AS OBJECT (
    lname  varchar(30),
    fname  varchar(30),
    mi  char(1)
);
```

类型的嵌套定义关系

1&2) 使用已有的对象类型来创建新的类型或表

- **Example 4.2.2:** 定义表中的属性

```
CREATE  TABLE  teachers (
        tid          int,
        tname    name_t,
        room       int    );
```

3) 使用对象数据类型来直接创建一张表

```
CREATE TABLE tablename OF typename
    { ( constraint-define ) } ;
```

- **Example 4.2.4**

```
CREATE TABLE people OF person_t
(
   PRIMARY KEY( ssno )
) ;
```

Attributes within table people

Attributes within pname

ssno	pname			age
	lname	fname	mi	
123550123	March	Jacqueline	E	23
245882134	Delaney	Patrick	X	59
023894455	Sanchez	Jose	F	30

Figure 4.2 ORACLE Object Table people of Example 4.2.4

1. Object Types

1.1 定义新的数据类型（对象类型）

1.2 对象类型的使用方法

1.3 对象值的创建、查询与更新

1.4 对象的引用类型

1.3 对象值的创建、查询与更新

1.3.1 对象值的创建

- 构造函数

1.3.2 对象值的查询

1.3.3 对象值的更新

1.3.1 对象值的创建

- 两个重要的函数

1. 对象构造函数

   typename (argument, ……)

   例： name_t (‘Einstein’, ‘Albert’, ‘E’)

2. 返回对象取值的函数

   value (…)

1.3.1 对象值的创建

- 使用对象类型name_t来定义teachers表中的属性tname
- 当我们使用插入操作在表teachers中插入新的元组时，会自动生成类型为name_t的对象值

```
Insert into teachers values
(1234, name_t('Einstein', 'Albert', 'E'), 120);
```

对象构造函数

1.3.2 对象值的查询

```
CREATE TYPE name_t
AS OBJECT (
    lname  varchar(30),
    fname  varchar(30),
       mi  char(1) );
```

```
CREATE  TABLE  teachers (
        tid          int,
        tname    name_t,
        room      int   );
```

例1: 查询位于123号房间的教师的编号(*普通查询*)

```
Select   t.tid
From     teachers  t
Where  t.room = 123
```

1.3.2 对象值的查询

```
CREATE TYPE name_t
AS OBJECT (
    lname  varchar(30),
    fname  varchar(30),
       mi  char(1) );
```

```
CREATE  TABLE  teachers (
        tid          int,
        tname    name_t,
        room      int   );
```

例2: 查询位于123号房间的教师的编号和姓名（first name & last name)

```
Select   t.tid, t.tname.fname, t.tname.lname
From     teachers  t
Where  t.room = 123
```

1.3.3 对象值的更新

1. 可以修改整个对象值

① 当表中的一个属性的值域是对象类型时，可以用‘对象’值直接对该属性进行赋值

② 如果一张‘表’是基于‘对象类型’创建的，那么可以用‘对象’值直接修改整个元组

2. 可以修改对象中成员属性的值

```
CREATE TYPE person_t
AS OBJECT (
        ssno        int,
        pname       name_t,
        age         int  );
```

```
CREATE TABLE people
OF person_t
(
    PRIMARY KEY( ssno )
) ;
```

例1: 修改元组中的对象属性值

```
update people p
set p.pname = name_t('Gould', 'Ben', null)
where ssn = 321341223;
```

pname是类型为name_t的属性

```
CREATE  TYPE  person_t
AS  OBJECT (
        ssno          int,
        pname       name_t,
        age           int  );
```

```
CREATE  TABLE  people
OF  person_t
(
    PRIMARY KEY( ssno )
) ;
```

例2: 修改某个对象属性值的成员属性

```
update  people  p
set  p.pname.mi = 'C'
where  ssno = 321341223;
```

```
CREATE TYPE person_t
AS OBJECT (
        ssno        int,
        pname       name_t,
        age         int  );
```

```
CREATE TABLE people
OF person_t
(
    PRIMARY KEY( ssno )
) ;
```

people是一个用对象类型person_t直接创建的表

例3: 修改整个元组

```
update people p
set p = person_t (
        332341223,
        name_t('Gould', 'Glen', 'A'),
        55 )
where ssno = 321341223;
```

1. Object Types

1.1 定义新的数据类型（对象类型）

1.2 对象类型的使用方法

1.3 对象值的创建、查询与更新

1.4 对象的引用类型

Definition of the REF Object Reference

- 对象的引用类型：**REF <object type>**

  ① 是指向某个元组对象的指针类型

  ② 可用于实现对象类型之间的嵌套引用

- 在使用含有**REF**类型的对象类型**(Object Type)**来创建关系表时，必须使用**Scope for**子句来限制**REF**属性的取值范围。

1.4 对象的引用类型

1.4.1 定义类型之间的引用关系

1.4.2 创建含有引用类型的关系表

1.4.3 引用关系查询

1.4.4 函数与谓词

1.4.5 类型的循环嵌套定义

1.4.6 其它约束

1.4.1 定义类型之间的引用关系

- 定义方法
  - 先定义一个对象类型**X**
  - 然后再定义一个对象类型**Y**，在类型**Y**中含有一个对类型**X**的引用属性(**REF**属性)，从而构成了类型**X**与类型**Y**之间的引用关系。
- 其中：
  - **X** 被称为‘基本对象类型’(被引用类型)
  - **Y** 被称为‘引用类型’

1.4.1 定义类型之间的引用关系

- 定义基本的对象类型：

customer_t, agents_t, products_t

```
create type customer_t as object (
    cid  char(4),
    cname  varchar(13),
    city  varchar(20),
    discnt  real
);
```

1.4.1 定义类型之间的引用关系

```
create type agent_t as object (
    aid  char(3),
    aname  varchar(13),
    city  varchar(20),
    percent  smallint
);
```

1.4.1 定义类型之间的引用关系

```
create type product_t as object (
    pid  char(3),
    pname  varchar(13),
    city  varchar(20),
    quantity  integer,
    price  double precision
);
```

1.4.1 定义类型之间的引用关系

- 定义类型之间的引用关系

```
create type order_t as object  (
    ordno       int,
    month       char(3),
    cid         char(4),
    aid         char(3),
    pid         char(3),
    qty         int,
    dollars     double precision,
    ordcust     ref  customer_t,
    ordagent    ref  agent_t,
    ordprod     ref  product_t
);
```

新增三个REF属性，分别引用（指向）三个不同的元组对象（Row Object）

1.4.2 创建含有引用类型的关系表

- 基本方法如下：

① 先使用基本对象类型创建相应的基本关系表

② 再使用含有REF属性的引用类型创建对应的关系表

1.4.2 创建含有引用类型的关系表

- 使用前面定义的对象类型创建基本关系表

```
create table customers of customer_t
    (primary key (cid));

create table products of product_t
    (primary key (pid));

create table agents of agent_t
    (primary key (aid));
```

1.4.2 创建含有引用类型的关系表

- 创建含有引用类型的关系表

```
create table orders of order_t  (
      primary key (ordno),
      scope for (ordcust)  is  customers,
      scope for (ordagent)  is  agents,
      scope for (ordprod)  is  products
  );
```

- **Scope for 子句用于限制REF属性的取值范围**
- **为什么要有 Scope for 子句？**

```
create type customer_t
as object (……);
```

```
create type agent_t
as object (……);
```

```
create type product_t
as object (……);
```

```
create type order_t
as object (
  ……
  ordcust ref customer_t,
  ordagent ref agent_t,
  ordprod ref product_t
);
```

```
create table customers of customer_t (……);
create table products of product_t (……);
create table agents of agent_t (……);
```

```
create table orders of order_t ( ……,
    scope for (ordcust) is customers,
    scope for (ordagent) is agents,
    scope for (ordprod) is products   );
```

1.4.3 引用关系查询

- 根据元组之间的REF引用关系进行查询

```
select   o.ordno,  o.ordcust.cname
from     orders  o
where  o.dollars > 200.00
```

```
select distinct o.ordcust.cname,
                o.ordagent.aname
from  orders  o
```

1.4.3 引用关系查询

- **Example 4.2.11:** find pid values of products that have been ordered by at least two customers.

```
select  distinct  x1.pid
from    orders x1, orders x2
where  x1.pid = x2.pid and x1.ordcust < x2.ordcust
```

ordcust 承担起了 cid 所担负的责任!

```
select  distinct  x1.pid
from    orders x1, orders x2
where  x1.pid = x2.pid and x1.cid < x2.cid
```

1.4.4 函数与谓词

- 两个函数
  - 获取对象(元组)的引用指针: **REF(……)**
  - 返回引用指针所指向对象的值: **DEREF(……)**

- 两个谓词
  - **IS DANGLING**
  - **IS NULL**

1.4.4 函数与谓词

- 函数 REF (…)：返回对象的引用指针

Example 4.2.12: retrieve all customer names where the customer does not place an order through agent a05.

```
select  c.cname
from   customers  c
where  not exists (
    select  *  from orders x
    where  x.ordcust = ref(c) and x.aid = 'a05' );
```

1.4.4 函数与谓词

- **Example 4.2.13，取得通过所有New York的代理商发订单的顾客的cid值。**

```
select c.cid  from customers c
where not exists (
    select * from agents a
    where a.city = 'New York' and not exists (
        select * from orders x
        where x.ordcust = ref(c) and
                x.ordagent = ref(a)
    )
);
```

1.4.4 函数与谓词

- **DEREF 函数**
  - 检索整个被引用对象，而不是仅仅获得该对象的引用指针

```
select  value ( p ),  deref ( p.partner )
from    police_officers   p
```

```
create  type  police_officer_t  as  object (
    pol_person   person_t,
    badge_number   integer,
    partner  ref  police_officer_t    );
```

1.4.4 函数与谓词

- 有关REF属性的判定谓词：**IS DANGLING**
  - 用于判断所引用的元组对象是否存在
  - 如果所引用的元组对象不存在，那么该谓词返回逻辑真(TRUE)，否则返回逻辑假(FALSE)。
  - 该谓词主要用于检查那些错误的对象引用指针。

1.4.4 函数与谓词

- example 4.2.14

```
select  o.cid    from  orders o
where  o.ordcust  IS DANGLING;
```

- 等价于：

```
select o.cid   from orders o
where o.ordcust <>
    (select ref(c) from customers c
     where c.cid = o.cid)
```

1.4.4 函数与谓词

- 也可以使用 **IS NULL** 谓词来查找取值为空 **(NULL)** 的 **REF** 属性
  - 但是，‘is dangling’不等于‘is null’

- 对象引用指针的使用规则

1. A dangling REF is non-null but useless.
2. If o.ordcust is null or dangling, then o.ordcust.cname is null.

1.4.5 类型的循环嵌套定义

- 对象类型(object type)不能嵌套定义，但 REF 关系可以实现嵌套引用。

```
create type police_officer_t as object
(
    pol_person   person_t,
    badge_number   integer,
    partner ref police_officer_t
);
```

```
create type police_officer_t as object (
    pol_person   person_t,
    badge_number   integer,
    partner  ref  police_officer_t   );
```

- 可以使用对象类型 police_officer_t 来创建表

```
create table police_officers of police_officer_t
(
    primary key (badge_number),
    scope for (partner) is police_officers
);
```

```
create type police_officer_t as object (
    pol_person   person_t,
    badge_number   integer,
    partner ref police_officer_t   );
create table police_officers of police_officer_t (……);
```

- 在表 police_officers 上的对象查询的例子
  - Retrieve the last names of all police officers who have partners over sixty years of age.

```
select p.pol_person.pname.lname
from   police_officers p
where  p.partner.pol_person.age > 60;
```

1.4.6 其它约束

- 有关REF定义的其它约束 (REF Dependencies)
  1) 两张表之间的相互REF关系的定义
  2) 两个具有相互REF关系的表/类型的删除
  3) REF属性数据的加载

1.4.6 其它约束

1) 两张表之间的相互REF关系的定义

- 首先，定义两个具有相互REF关系的对象类型 (create type ……)
  - ① 部分创建(partially create)第一个对象类型(只给出类型名，没有类型的详细定义)
  - ② 详细定义第二个对象类型(包含对第一个类型的引用属性)
  - ③ 再详细定义第一个对象类型
- 再用创建好的对象类型创建关系表

1.4.6 其它约束

2) 两个具有相互REF关系的表/类型的删除

- 在删除类型(drop type)之前需要先删除表(drop table)
- 在删除类型(drop type)时需要采用强制删除的方式

```
DROP TYPE typename FORCE;
```

1.4.6 其它约束

3) REF属性数据的加载

- 方法一：先不管REF属性的赋值(先置为NULL)，然后再使用UPDATE操作修改REF属性上的取值

```
update  orders  o
set   ordcust = (select ref(c) from customers c
                        where c.cid = o.cid),
      ordagent = (select ref(a) from agents a
                        where a.aid = o.aid),
      ordprod = (select ref(p) from products p
                        where p.pid = o.pid);
```

1.4.6 其它约束

3) REF属性数据的加载

- 方法二：使用带有子查询的插入操作

```
insert into police_officers
    select value(p), 1000, ref(p0)
    from   people p, police_officers p0
    where  p.ssno = 033224445 and
           p0.badge_number = 990;
```

2. Collection Types

- Collection types allow us to put multiple values (collections of values) in a column of an individual row.

- Oracle
  - 2.1 Table Types (Nested Tables)
  - 2.2 Array Types

    Containing items all of the same type – ***the element type***

2.1 Table Types and Nested Tables

- 使用table type来定义表中的属性可以实现多值属性的功能

- 创建一个新的表类型 (table type)

```
CREATE TYPE dependents_t
    AS TABLE OF person_t;
```

2.1 Table Types and Nested Tables

- 使用类型dependents_t来定义表employees中的属性并形成一个嵌套表(nested table)定义。

```
create table employees (
    eid       int,
    eperson   person_t,
    dependents  dependents_t,
    primary key (eid)
)
nested table dependents store as dependents_t;
```

```
create table employees (
    eid       int,
    eperson   person_t,
    dependents  dependents_t,
    primary key (eid)
) nested table dependents store as dependents_tab;
```

- 执行上述的建表命令将在数据库中创建两个关系表：
  - **employee**：存放职工记录
  - **dependents_t**：存放所有职工的家属信息（记录），被称为‘嵌套表’

- 在一条建表命令中可以定义多个**nested table**
  - 每一个**table-type**属性，都需要有一个对应的**nested table**

2.1 Table Types and Nested Tables

- **Nested table的访问**
  - **Example: Retrieve the nested table of all dependents of employee 101.**

```
select  dependents
from    employees
where   eid = 101;
```

2.1 Table Types and Nested Tables

- **Nested table的访问**
  - **Example: Retrieve the eids of employees with more than six dependents.**

```
select  eid
from  employees e
where  6 < ( select count(*)
                  from table(e.dependents));
```

转换函数: table(…)

❑Nested table的访问: table (…) 的使用

错误的使用方法：

```
Select  count(*)
From   (select e.dependents
          from employees e
          where e.eid = 101);
```

正确的使用方法：

```
Select  count(*)
From   table (select e.dependents
                 from employees e
                 where e.eid = 101);
```

2.1 Table Types and Nested Tables

- Oracle 数据库没有提供 nested table 的相等比较运算。
  - 可以使用 IN 操作符来实现某些需要通过 nested table 进行的查询功能

2.1 Table Types and Nested Tables

- **Example:** List eids of employees (there should be only one) with a dependent having Social Security Number 3451112222

```
select  eid
from    employees e
where  3451112222  in
          ( select  d.ssno
            from  table(e.dependents)  d);
```

通过子查询得到职工e的所有家属的SSN编号的集合

2.1 Table Types and Nested Tables

- Oracle提供了单个对象的相等比较功能
  - Example: retrieve eids with dependents that have name given by name_t('Lukas', 'David', 'E')

```
select eid from employees e
where name_t('Lukas', 'David', 'E') in
      ( select d.pname
        from table(e.dependents) d);
```

2.1 Table Types and Nested Tables

- Oracle 不支持直接对嵌套表属性的统计查询功能，即下述的统计查询操作是错误的：

```
select  count(e.dependents)
from    employees  e
where  e.eid = 101;
```

2.1 Table Types and Nested Tables

- 表与其自身的嵌套表的联接查询
- **Example:**
  - **display all employee identifiers and their dependents' ssno values**

```
select  e.eid, d.ssno
from  employees e,  table(e.dependents) d
```

2.1 Table Types and Nested Tables

- 如果希望同时列出那些没有 dependents 职工的信息，那么可以使用 outer join 来实现这样的查询。

```
select  e.eid, d.ssno
from   employees e,  table(e.dependents) (+) d
```

2.1 Table Types and Nested Tables

- **Nested Cursors （嵌套游标）**
  - 可以使查询结果的显示更清楚明了

- 例如：普通的查询操作

```
select e.eid, d.ssno as dep_sso
from employees e, table(e.dependents) d
where d.age < 16;
```

❑普通的查询操作

```
select e.eid, d.ssno as dep_sso
from employees e, table(e.dependents) d
where d.age < 16;
```

❑使用 nested cursor 的查询操作

```
select e.eid,
       cursor ( select d.ssno as dep_ssno
                from table(e.dependents) d
                where d.age < 16)  dep_tab
from employees e;
```

- 可以使用 **nested cursor** 来实现对嵌套表属性的统计功能

```
select  eid,
        cursor ( select count(*)
                 from table(e.dependents) )
from employees e;
```

- 也可以使用下面的两种方式实现上述的统计查询功能

```
select  eid,  ( select count(*)
                from table(e.dependents) )
from employees e;
```

```
select  eid,  count(*)
from  employees e,  table(e.dependents)
group by  eid;
```

Q_1:

```
select  eid, cursor ( select count(*)
                      from table(e.dependents) )
from employees e;
```

Q_2:

```
select  eid,  ( select count(*)
                from table(e.dependents) )
from employees e;
```

Q_3:

```
select  eid,  count(*)
from  employees e,  table(e.dependents)
group by  eid;
```

2.2 Array Types

❑ Array Types for VARRAYs

```
create type extensions_t as varray(4) of int;
```

- **Type name:** extensions_t
- **Element type:** int
- **Maximum number:** 4

2.2 Array Types

- 使用 Array Types 定义表中的属性

```
create table phonebook (
    phperson   person_t,
    extensions extensions_t
);
```

- 可以使用 table (…) 将一个VARRAY属性转换成一张嵌套表。

2.2 Array Types

- Nested table 与 VARRAY 的比较

	Nested table	VARRAY
成员的排列次序	无 序	有 序
成员的最大数目	没有限制	确定的值
成员的存储组织	单独的存储表	直接存储在表中

2.2 Array Types

- **Nested table 与 VARRAY 的访问模式比较**
  - 可以对嵌套表属性执行**insert**操作，或通过**update**操作修改其成员的取值
  - 但对于 **VARRAY** 属性则不能执行上述的插入或修改操作，只能通过**update**语句修改整个 **VARRAY** 属性的取值

3. PL/SQL Procedures, UDFs, and Methods

- **Program block in PL/SQL**

```
declare                    -- local variable
    i  integer;
    total integer:= 0;
begin
    for i in 1..100 loop
        total := total + i;
    end loop;
    insert into result (rvalue) values (total);
end;
```

3. PL/SQL Procedures, UDFs, and Methods

- **Create function in PL/SQL**

```
create function sum_n(n integer) return integer is
    i integer;
    total integer:= 0;
begin
    for i in 1..n loop
        total := total + i;
    end loop;
    return total;            -- return result to SQL or
                             --  PL/SQL caller
end;
```

3. PL/SQL Procedures, UDFs, and Methods

- 创建对象类型 point_t

```
create type point_t as object (
    x int,       -- horizontal coordinate of the point
    y int        -- vertical coordinate of the point
);
```

❑创建带有成员函数的对象类型 rectangle_t

```
create type rectangle_t as object (
   pt1 point_t,       -- lower left-hand corner of rectangle
   pt2 point_t,       -- upper right-hand corner of rectangle
   member function inside(p point_t)
      return int,          -- return 1 if point inside, else 0
   member function area     -- method for area of rectangle
      return int           -- area value will always be an integer
);
```

❑成员函数的实现 (area)

```
create type body rectangle_t as
    member function area return int is
    begin
        return (self.pt2.x-self.pt1.x) *
                    (self.pt2.y-self.pt1.y)    ;
    end;
    ......
end;
```

❑成员函数的实现 (inside)

```
create type body rectangle_t as
    ......
    member function inside(p in point_t)
    return int is
    begin
       if (p.x>=self.pt1.x)and(p.x<=self.pt2.x) and
           (p.y>=self.pt1.y)and(p.y<=self.pt2.y)
       then
              return 1;
       else
              return 0;
       end if;
    end;
end;
```

Others

- External Functions and Packaged User-Defined Types (UDTs)
  - Binary Data and BLOBs
  - External Functions
  - Distinct Types
  - Packaged UDTs and Other Encapsulated UDTs

Others

- **External Functions**
  - 用其它的程序设计语言实现的可执行代码段，通过 **Create Function** 语句将其注册到数据库服务器中后，就可以如同调用使用 **PL/SQL**定义的成员函数一样调用这些 **external function** 的功能
  - 数据库管理系统无法直接处理**BLOBs**类型的值，但可以通过 **external function** 来存取、处理 **BLOBs** 字段上的值。

Others

- **Distinct Types**
  - **If type U is a distinct type based on type T, you cannot compare objects of these two types without a cast.**
  - **The idea of distinct type**
    - **Provide the advantages of type checking**