Edinburgh Napier UNIVERSITY

Advanced Database Systems SET09107

Object-Relational Databases

Inheritance References & Methods

Feedback

Edinburgh Napier UNIVERSITY

- Use editor Notepad ++
- Check spelling first if there is an error
- Use alias with “.” function.

Select p.name.surname

From People p;

The following won’t work!

Select name.surname

From People;

Select People.name.surname

From People;

Feedback – Cont'd

Edinburgh Napier UNIVERSITY

- Add "/" between queries, if you want to run more than one at a time.

```
create type peopleType as object
      ( pname Name,
      paddress  Address,
      dateOfBirth date)
      not final
/
create table peopleTable of peopleType;
```

Feedback – Cont'd

Edinburgh Napier UNIVERSITY

- Use of quotation marks in Insert, ‘ or '

Insert into *peopleTable*
values
(Name(‘John’, ‘Smith’),
Address(‘10 Merchiston’, ‘Edinburgh’, ‘EH10 5DT’),
’21-Feb-89’
);

Not Correct!

Insert into peopleTable
values
(Name('John', 'Smith'),
Address('10 Merchiston', 'Edinburgh', 'EH10 5DT'),
'21-Feb-89'
);

Edinburgh Napier UNIVERSITY

Contents

- Structured Types & Subtypes -- Review
- Inheritance
- References
- Methods
- Summary

Structured Types

Edinburgh Napier UNIVERSITY

- Structured types can be declared:

create type *Name* **as object**
(*firstname* **varchar2**(20),
surname **varchar2**(20))
final

create type *Address* **as object**
(*street* **varchar2**(20),
city **varchar2**(20),
postal_code **varchar2**(8))
not final

Edinburgh Napier UNIVERSITY

Types & Tables

- Tables can be defined as

```
drop type peopleType force;
-- if previously created

create type peopleType as object
      ( pname Name,
      paddress  Address,
      dateOfBirth date)
      not final
      /
create table peopleTable of peopleType;
```

Edinburgh Napier UNIVERSITY

Insert values

```
Insert into peopleTable
   values
   (Name('John', 'Smith'),
   Address('10 Merchiston', 'Edinburgh', 'EH10 5DT'),
   '21-Feb-89'
    );
```

Edinburgh Napier UNIVERSITY

Access Component Attributes

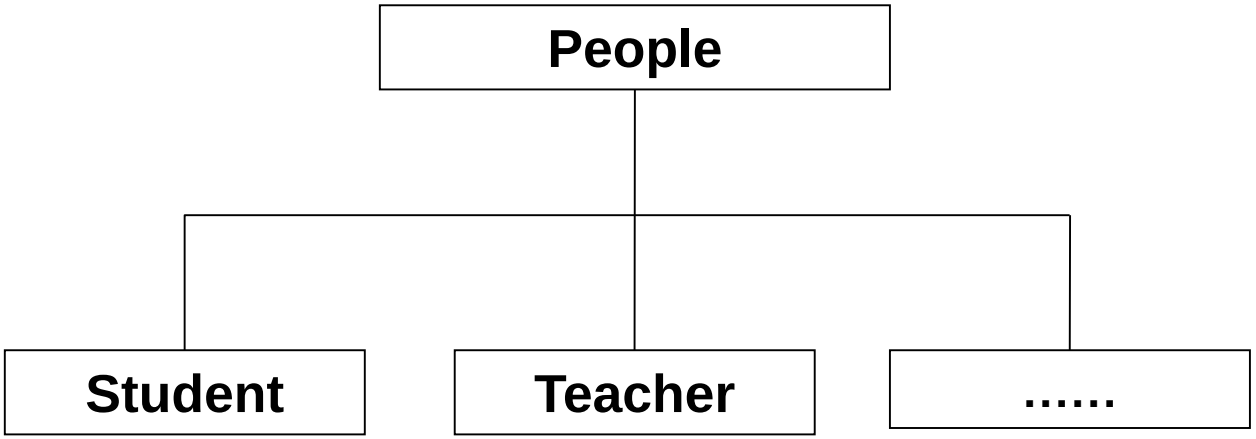
select *p.pname.surname, p.paddress.city*
from *peopleTable* p*;*

- components of a composite attribute can be accessed using a “dot” notation, such as pname.firstname

Edinburgh Napier UNIVERSITY

Subtypes

- Generally related to the **supertype**



Edinburgh Napier UNIVERSITY

Subtypes-- Cont'd

- Subtypes can be created with some extra attributes:

create type *Student* **under** *peopleType*
(*programme* **varchar2**(20),
school **varchar2**(20))
final
create type *Teacher* **under** *peopleType*
(*salary* **number**,
school **varchar2**(20))
final

Edinburgh Napier UNIVERSITY

Subtypes-- Examples

- Insert values to sub-tables:

```
Insert into studentTable
    values
      (Name('John', 'Smith'),
       Address('10 Merchiston', 'Edinburgh', 'EH10 5DT'),
      '21-Feb-89',
      'BEng Computing',
      'Computing'
      );
```

Edinburgh Napier UNIVERSITY

Subtypes-- Cont'd

- A supertype can be changed even after some subtypes have been created

  alter type *peopleType*
  add attribute (*gender* **varchar2(8)) cascade**;

- The **cascade** option propagates a type change to dependent types and tables

Edinburgh Napier UNIVERSITY

Subtypes-- Cont'd

- The supertype must be **not final**
- If it is **final**, it must be changed to **not final**

alter type *peopleType* **not final cascade**;

Edinburgh Napier UNIVERSITY

Inheritance

- Subtypes inherit attributes from their supertypes
- Type Student should have *programme, school* in addition of *pname, paddress* and *dateOfBirth*
- Subtypes can redefine methods by using overriding method in place of method in the method declaration
- If the definition of peopleType changes, so do the definitions of any subtypes.
- More examples after Methods

Multiple Inheritance

Edinburgh Napier UNIVERSITY

- If your type system supports multiple inheritance, you can define a type for teaching assistant as follows:

  create type TeachingAssistant

  under Student, Teacher

- To avoid a conflict between the two occurrences of school we can rename them

  create type TeachingAssistant

  under Student with (school as student_sch),

  Teacher with (school as teacher_sch)

Note: SQLPlus doesn't support Multiple Inheritance

Edinburgh Napier UNIVERSITY

References

- Object-oriented languages provide the ability to create and refer to objects.
- In SQL:
  - References are to tuples, and
  - References can only be scoped when defining a table,
    - I.e., can only point to tuples in one specified table
- References as foreign keys in 1-n relationships
- Using references in a query can replace joins.

Edinburgh Napier UNIVERSITY

System-generated object ID

- A reference is generated by the system automatically for each row in a table, which is a unique object identifier

- The hidden column sys_nc_oid$ stores the object ids in an object table, which is a 32-character string

```
select sys_nc_oid$
from job_table;
```

Reference Declaration – Cont'd

- Define a type employment with a field employee and a field position which are references to types employee and job respectively

```
create type employment as object(
    employee_r ref employee,
    position ref job)
```

- A ref is a logical pointer to an instance object (a tuple (row)) in the ref type.
- It makes references behave like foreign keys
- The reference points to object types employee and job respectively, not the relevant tables

Edinburgh Napier UNIVERSITY

Reference Declaration – Cont'd

- "**scope is**" can be used to restrict the references to actual object tables

```
create table employment_table(
        employee ref employee scope is employee_table,
        position ref job scope is job_table);
```

- This can only be defined when creating a table, not when creating a type
- The referenced table must have an object identifier for each tuple, which is automatically generated.

References -- Functions

Edinburgh Napier UNIVERSITY

Three functions supporting queries involving objects:

- ref() – takes as its argument a table alias associated with a row of an object table and returns the ref to that object
- value() -- takes as its argument a table alias associated with a row of an object table and returns object instances stored in the object table.
- deref() – take the ref to an object as its argument and returns the instance of the object type

Reference Functions – Cont'd

- Use ref() to insert data

  e.g., find the reference for the manager in job_table:

```
insert into employment_table
    select ref(e), ref(j)
    from job_table j, employee_table e
    where e.emp_ID = 2
    and j.job_ID = 1;
```

- The function **ref**() provides the pointers to the objects in the two corresponding tables, which are then inserted into employment_table

Edinburgh Napier UNIVERSITY

Reference Functions – Cont'd

- Use ref() to find references

e.g., find the reference for the manager in job_table:

```
select ref(j)
from job_table j
where jobtitle = 'manager';
```

```
REF(J)
--------------------------------------------------------------------------------
000028020929E6A948197C813DE040007F0100132429E6A9481978813
   DE040007F0100132401C0095D0003
```

Reference Functions – Cont'd

- Use **value**() to find object instances in a table

  e.g., find object instances for managers in job_table:

  select value(j)
  from job_table j
  where jobtitle = 'manager';

  ```
  VALUE(J)(JOBTITLE, JOB_ID, SALARY_AMOUNT,
      YEARS_OF_EXPERIENCE)
  ---------------------------------------------------------------------
  JOB('manager', 3, 52000, 10)
  ```

- Try **select** * **from** job_table **where** jobtitle = 'manager'

Reference Functions – Cont'd

Edinburgh Napier UNIVERSITY

- Use deref() to return the tuple pointed to by a reference

  e.g., find the employee in the employee table

  select DEREF(p.employee)

  from employment p

```
DEREF(P.EMPLOYEE)(PNAME(FIRST, MIDDLE, LAST),
    PPHONE(HOMEPH, BUSINESSPH, MOBILE
--------------------------------------------------------------------------------
EMPLOYEE(NAME('Paul', 'R', 'Miller'), PHONE('123-4587', '838-4536',
    '73556-12')
, ADDRESS('High St', 'Edinburgh', 'EH3 4RF'), 0)

EMPLOYEE(NAME('Sally', NULL, 'Jones'), PHONE('322-5643', NULL,
    '744-6-56'), ADD
RESS('Princess St.', 'Edinburgh', 'EH1 3AB'), 1)
```

Edinburgh Napier UNIVERSITY

Reference Functions – Cont'd

Assuming that the employment table is generated with **refs**. What will it show if the following statement is used?

```
select *
from employment;
```

Initialising Reference Typed Values

To create a tuple with a reference value, do

- first create the tuple with a null reference
- then set the reference separately by using the function ref(p) applied to a tuple variable
  - E.g. to create a department with name CS and head being the person named John, we use

```
insert into departments
        values (' CS ', null)
update departments
        set head = (select ref(p)
                from people as p
                where name= ' John ')
        where name = ' CS '
```

Reference - Example

Edinburgh Napier UNIVERSITY

```
create type employee_ref under people () not final;
create table employee_ref_table of employee_ref;
INSERT INTO employee_ref_table VALUES (
   name('John', 'R', 'Smith'),
   phone('123-4567', NULL,'73746-56'),
   address('Merchiston','Edinburgh', 'EH10 5DT'));

INSERT INTO employee_ref_table VALUES (
   name('Mary', NULL, 'Miller'),
   phone('354-5643', '453-5746','73346-56'),
   address('Princess St.','Edinburgh', 'EH1 3AB'));

INSERT INTO employee_ref_table  VALUES (
   name('Mary', 'S', 'Miller'),
   phone('322-8484', NULL,'645-2929'),
   address('Merchiston Ave', 'Edinburgh','EH10 4AW'));
```

Reference – Example – cont'd

Edinburgh Napier UNIVERSITY

```
create type job_ref as object (
    jobtitle varchar(20),
    salary_amount int,
    years_of_experience int );

create table job_ref_table of  job_ref;

insert into job_ref_table values ('engineer', 30000,4);
insert into job_ref_table values ('programmer', 35000,3);
insert into job_ref_table values ('data analyst', 20000,15);
insert into job_ref_table values ('designer', 25000,2);
insert into job_ref_table values ('engineer', 33000,5);
```

Reference – Example – cont'd

Edinburgh Napier UNIVERSITY

```
create type employment_ref as object (
    employee ref employee_ref,
    position ref job_ref
    );
create table employment_ref_table of employment_ref;

INSERT INTO employment_ref_table
    SELECT REF(e), REF(j)
    FROM job_ref_table j, employee_ref_table e
    WHERE e.pname.first = 'John'
        AND j.jobtitle = 'engineer';

INSERT INTO employment_ref_table
    SELECT REF(e), REF(j)
    FROM job_ref_table j, employee_ref_table e
    WHERE e.pname.first = 'Mary'
        AND j.jobtitle = 'designer';
```

Edinburgh Napier UNIVERSITY

Reference – Example – cont'd

SQL> **SELECT** e.employee.pname **FROM** employment_ref_table e;

EMPLOYEE.PNAME(FIRST, MIDDLE, LAST)

NAME('John', 'R', 'Smith')

NAME('John', 'R', 'Smith')

NAME('Mary', NULL, 'Miller')

NAME('Mary', 'S', 'Miller')

Note: You should be able to access any tuple in both emplyee_re and job_ref

SELECT e.position.jobtitle FROM employment_ref_table e;

SELECT e.position.salary_amount FROM employment_ref_table e;

Reference – Example – cont'd

To retrieve individual values:

```
SELECT e.employee.pname.surname, e.position,salary_amount,
    e.position.jobtitle
 FROM employment_ref_table e;
```

```
LAST NAME       MONTHLY SALARY JOB TITLE
--------------- -------------- ---------
Smith                   30000 engineer
Smith                   33000 engineer
Miller                  25000 designer
Miller                  25000 designer
```

Edinburgh Napier UNIVERSITY

Methods

- Methods can be viewed as functions associated with structured types
  - They have an implicit first parameter called self which is set to the structured-type value on which the method is invoked
  - The method code can refer to attributes of the structured-type value using the self variable

    e.g. self.a

Methods SQL

- Member methods -- instance methods
- Static methods – class methods
- Methods can be part of the type definition of a structured type
- A method body needs to be created separately

Methods – SQL

Edinburgh Napier UNIVERSITY

- Methods can be part of the type definition of a structured type:

```
create type Employee as object(
    name varchar(20),
    salary integer)
    not final
method giveraise ( percent integer)
```

- the method body is created separately

```
create method giveraise ( percent integer) for Employee
begin
    set self. salary = self.salary + (self. salary * percent) / 100;
end
```

- The variable **self** refers to the structured type instance on which the method is invoked.

Methods SQL – Cont'd

- Values of structured types are created using constructor functions
  - E.g. Publisher('McGraw-Hill', 'New York')
  - Note: a value is not an object
- SQL:1999 constructor functions

```
create function Publisher ( n varchar(20), b varchar(20))
returns Publisher
begin
    set name = n;
    set branch = b;
end
```

  - Every structured type has a default constructor with no arguments, others can be defined as required

Edinburgh Napier UNIVERSITY

Methods Oracle

- Member methods -- instance methods
- Static methods – class methods
- An object type with methods must have a separate type body
- **create type** statement specifies
  - The name of the object type
  - Its attributes
  - Methods
  - Other properties
- **create type body** statement contains the code for the methods that implement the type

Edinburgh Napier UNIVERSITY

Methods Oracle – Cont'd

- **create type** statement

```
create type emp as object(
    name varchar2(20),
    salary number,
    member function giveraise ( percent number) return
    number);
```

Methods Oracle – Cont'd

Edinburgh Napier UNIVERSITY

- **create type body** statement

```
create or replace type body emp as
    Member function giveraise (percent number) return
      number is
    sal number;
    begin
      sal :=(self.salary+(self.salary*percent)/100);
      return sal;
      End giveraise;
    End;
```

- Note: use := in assignments

Methods Oracle – Cont'd

Edinburgh Napier UNIVERSITY

- **Access methods**

```
Select * from emp_table e
where e.giveraise(20)>60000;
```

- It needs to mention which table this method belongs to, e.g. e.giveraise

Edinburgh Napier UNIVERSITY

Adding methods

- Methods can be added to existing types

```
alter type  emp
add member function evaluate_qualif return string
cascade;
```

- **cascade** makes the alternations apply to existing dependent object tables

Edinburgh Napier UNIVERSITY

Debugging

- If a type creation is unsuccessful, Oracle usually only provides a warning "created with compilation errors".
- To obtain more information about the reasons for the compilation errors, type

  ***show error*;**

- The error message contains information about the line number and column in which the error occurred and about the type of error

Compilation errors

- To recreate a type after fixing the compilation errors, use

  ***create or replace*;**

  instead of ***drop type*** ... And ***create***
- But this will only work if the type does not yet have dependent tables.
- A type with dependent tables can only be ALTERed, not REPLACEd.

Edinburgh Napier UNIVERSITY

Summary

- Structured Types & Subtypes
  - Review
- Inheritance
  - Student and Teacher inherit attributes from People
- References
  - Declaration
  - Functions: **ref**(), **value**() and **deref**()
  - An example
- Methods