Nest & Unnest

Slides by:

- Osmar Zaiane (sfu.ca)
- oracletutorial (https://www.oracletutorial.com/plsql-tutorial/plsql-nested-tables/)
- Orestes Appel (MRU)

Relation-Valued Attributes

- Extended SQL allows an expression evaluating to a relation to appear anywhere that a relation name may appear.
  - The ability to use subexpressions freely makes it possible to take advantage of the structure of nested relations.
- Schema for a relation *pdoc (right)*

```
create table pdoc
        (name MyString,
         author-list setof(ref(people)),
         date MyDate,
         keyword-list setof(MyString))
```

Relation-Valued Attributes

Find ALL documents having keyword = database

```
select name

from pdoc

where ``database'' in keyword-list
```

Find pairs of form doc-name, author-name for each document and author of the documents

```
select B.name, Y.name

from pdoc as B, B.author-list as Y
```

Aggregate functions can be applied to any relation-value expression

```
select name, count(author-list)
from pdoc
```

Nesting & Unnesting

1. The transformation of a nested relation into 1NF is called ***unnesting***.
2. To complete unnest the doc relation, we have:

```
select name, A as author, date.day, date.month, date.year,
K as keyword

from doc as B, B.author-list as A, B.keyword-list
as K
```

1. The reverse operation of transformation of a 1NF relation into a nested relation is called ***nesting***.
2. Example. To nest the relation flat-doc on the attribute keyword, we have

```
select title, author, (day, month, year) as date,
set(keyword) as keyword-list

from flat-doc

groupby title, author, date
```

Nesting & Unnesting

The **select** in the previous slide will generate the following table:

title	author_list	date	keyword_list
		day month year	
sales plan	Smith	1 April 89	{profit, strategy}
sales plan	Jones	1 April 89	{profit, strategy}
status report	Jones	17 July 94	{profit, personnel}
status report	Frick	17 July 94	{profit, personnel}

To convert *flat-doc* back to the nested table *doc*, we have:

```
        select title, set(author) as author-list,
(day, month, year) as date,
set(keyword) as keyword-list


        from flat-doc


        groupby title, date
```

PL/SQL Nested Tables (Oracle)

Nested tables are single-dimensional, unbounded collections of homogeneous elements.

- First, a nested table is single-dimensional, meaning that each row has a single column of data like a one-dimension array.
- Second, a nested table is unbounded. It means that the number of elements of a nested table is predetermined.
- Third, homogeneous elements mean that all elements of a nested table have the same data type.

Declaring a nested table variable

Declaring a nested table is a two-step process.

First, declare the nested table type using this syntax:

```
TYPE nested_table_type
    IS TABLE OF element_datatype [NOT NULL];
```

Then, declare the nested table variable based on a nested table type:

```
nested_table_variable nested_table_type;
```

It is possible to create a nested table type located in the database:

```
CREATE [OR REPLACE] TYPE nested_table_type
    IS TABLE OF element_datatype [NOT NULL];
```

If you want to drop a type, use the following `DROP TYPE` statement:

```
DROP TYPE type_name [FORCE];
```

Initializing a nested table variable

When you declare a nested table variable, it is initialized to NULL.

To initialize a nested table, you can use a constructor function. The constructor function has the same name as the type:

```
nested_table_variable := nested_table_type();
```

You can also declare a nested table and initialize it in one step using the following syntax:

```
nested_table_variable nested_table_type := nested_table_type();
```

Add elements to a nested table

To add an element to a nested table, you first use the `EXTEND` method:

```
nested_table_variable.EXTEND;
```

Then, use the assignment operator (:=) to add an element to the nested table:

```
nested_table_variable := element;
```

If you want to add multiple elements, you use the `EXTEND(n)` method, where `n` is the number of elements that you want to add:

```
nested_table_variable.EXTEND(n);

nested_table_variable := element_1;
nested_table_variable := element_2;
..
nested_table_variable := element_n;
```

Accessing elements & Iteration

To access an element at a specified index, you use the following syntax:

```
nested_table_variable(index);
```

Nested tables have the `FIRST` and `LAST` methods that return the first and last indexes of elements respectively.

Therefore, you can use these methods to iterate over the elements of a nested table using a `FOR` loop:

```
FOR l_index IN nested_table_variable.FIRST..nested_table_variable.LAST
LOOP
    -- access element

END LOOP;
```