Database Systems

ORDB: Collections

Last Lecture

- Object types
  - Declaring
  - Row objects/ column objects
  - References
    - Dereferencing (implicit join)
- Constraints on object tables

- Any questions?

NAME	ADDRESS	INVESTMENTS			
		COMPANY	PURCHASE PRICE	DATE	QTY
John Smith	3 East Av Bentley WA 6102	BHP	12.00	02/10/01	1000
		BHP	10.50	08/06/02	2000
		IBM	58.00	12/02/00	500
		IBM	65.00	10/04/01	1200
		INFOSYS	64.00	11/08/01	1000
Jill Brody	42 Bent St Perth WA 6001	INTEL	35.00	30/01/00	300
		INTEL	54.00	30/01/01	400
		INTEL	60.00	02/10/01	200
		FORD	40.00	05/10/99	300
		GM	55.50	12/12/00	500

COMPANY	CURRENT PRICE	EXCHANGES TRADED	LAST DIVIDEND	EARNING PER SHARE
BHP	10.50	Sydney New York	1.50	3.20
IBM	70.00	New York London Tokyo	4.25	10.00
INTEL	76.50	New York London	5.00	12.40
FORD	40.00	New York	2.00	8.50
GM	60.00	New York	2.50	9.20
INFOSYS	45.00	New York	3.00	7.80

Collection Types

- Useful for modelling one-to-many relationships.
  - Example: An investor makes many share purchases.
- Collection datatypes in Oracle:
  - varrays
  - nested tables.

VARRAYs

- Arrays of variable size.
  - Specify a maximum size when you declare the array type.
  - Creating an array type does not allocate space.
    - Since it is only a type definition.
- Examples:

```
CREATE TYPE price_arr AS VARRAY(10) OF
    NUMBER(12,2);
```

  - The VARRAYs of type PRICES have no more than ten elements, each of data type NUMBER(12,2).

VARRAYs

- A varray type can be used as:
  - the data-type of a column of a relational table;
  - an attribute data-type in an object type definition;
- Example:
  - Create type excharray as varray(5) of varchar(12)
    /
    Create type share_t as object(
    cname varchar(12),
    cprice number(6,2),
    exchanges excharray,
    dividend number(4,2),
    earnings number(6,2))
    /

Creating a VARRAY

- To insert a collection use its type constructor method.
  - The type constructor method has same name as the type.
  - Its argument is a comma-separated list of collection elements.
- Example:
  - create table shares of share_t(
    cname primary key);
  - insert into shares values(‘BHP’, 10.50,
    excharray(‘Sydney’ ,’New York’), 1.50, 3.20);

VARRAY Example

```
CREATE TYPE price_arr AS
   VARRAY(10) OF NUMBER(12,2)
/
CREATE TABLE pricelist (
   pno integer,
   prices price_arr);

INSERT INTO pricelist
   VALUES(1, price_arr(2.50,3.75,4.25));
```

Retrieving from a VARRAY

- SELECT * FROM pricelist;

```
PNO    PRICES
  ----------------------------------
  1      PRICE_ARR(2.5, 3.75, 4.25)
```

- SELECT pno, s.COLUMN_VALUE price FROM pricelist p, TABLE(p.prices) s;

```
PNO     PRICE
---------------
 1        2.5
 1        3.75
 1        4.25
```

Nested Tables

- Allow values of tuple components to be whole relations.
- If *T* is a UDT, we can create a table type *S:*

  CREATE TYPE *S* AS TABLE OF *T* ;

  - Values of type S are relations with rowtype *T*.
  - S can be the type of an attribute in another UDT or in a relation.
  - See the example on next slide.

Example: Nested Table Type

```
CREATE TYPE BeerType AS OBJECT (
   name CHAR(20),
   kind   CHAR(10),
   colour CHAR(10))
/
CREATE TYPE BeerTableType AS
   TABLE OF BeerType
/
```

Example - Continued

- CREATE TABLE Manfs (
  name CHAR(30),
  addr CHAR(50),
  beers beerTableType)
  NESTED TABLE beers STORE AS beer_table;
- BeerTableType used in Manfs relation to store the set of beers by each manufacturer in one tuple.

Storing Nested Tables

- Oracle doesn't really store each nested table as a separate relation
  - it just makes it look that way.
  - tuples of all the nested tables for one attribute *A* are stored in one relation *R*.
- Declare a storage of nested tuples in CREATE TABLE by:

  NESTED TABLE *A* STORE AS *R*
- In previous example,

  NESTED TABLE beers STORE AS beer_table;

Querying a Nested Table

- We can retrieve the value of a nested table like any other value.
- But these values have two type constructors:
  - For the table.
  - For the type of tuples in the table.
- Find the beers by Anheuser-Busch:

```
SELECT beers FROM Manfs
WHERE name = 'Anheuser-Busch';
```

- Produces one value like:

```
BeerTableType(
   BeerType('Bud', 'lager', 'yellow'),
   Beertype('Lite', 'malt', 'pale'),
…)
```

Querying Within a Nested Table

- A nested table can be converted to an ordinary relation by applying TABLE(…).
- This relation can be used in FROM clauses like any other relation.
- Find the ales made by Anheuser-Busch:

```
SELECT b.name
FROM TABLE( SELECT beers
            FROM Manfs
            WHERE name = 'Anheuser-Busch') b
WHERE b.kind = 'ale';
```

Nested Table Example 2

- -- ‘/’ after each type definition omitted to save space

```
CREATE TYPE proj_t AS OBJECT (
    projno NUMBER,
    projname VARCHAR (15));
CREATE TYPE proj_list AS TABLE OF proj_t;
CREATE TYPE employee_t AS OBJECT (
    eno number,
    projects proj_list);
CREATE TABLE employees of employee_t (eno primary key)
NESTED TABLE projects STORE AS employees_proj_table;
```

Inserting and Retrieving

- Insert a row into employees table:

```
INSERT INTO employees VALUES(1000, proj_list(
    proj_t(101, 'Avionics'),
    proj_t(102, 'Cruise control')
));
```

- To retrieve the projects of eno 1000:

```
SELECT *
FROM TABLE(SELECT t.projects FROM employees t
WHERE t.eno = 1000);
```

```
PROJNO  PROJNAME
--------------------
 101    Avionics
 102    Cruise control
```

Collection Unnesting

- Unnest or flatten the collection attribute of a row
  - by joining each row of the nested table with the row that contains the nested table.
  - Example:

```
SELECT e.eno, p.*
FROM employees e, TABLE (e.projects) p;
 ENO       PROJNO   PROJNAME
---- ----------- ----------------
1000       101     Avionics
1000       102     Cruise control
2000       100     Autopilot
```

DML on Collections:

- Use a TABLE expression to identify the nested table values.

```
INSERT INTO TABLE(SELECT e.projects
                  FROM employees e
                  WHERE e.eno = 1000)
  VALUES (103, 'Project Neptune');
UPDATE TABLE(SELECT e.projects
             FROM …) p
  SET p.projname = 'Project Pluto'
  WHERE p.projno = 103;
DELETE TABLE(SELECT e.projects
             FROM …) p
  WHERE p.projno = 103;
```

DML on Nested Tuples

- To drop a particular nested table, set the nested table column in the parent row to NULL.

```
UPDATE employees e
    SET e.projects = NULL
    WHERE e.eno = 1000;
```

- To add back a nested table row:

```
UPDATE employees e
    SET e.projects = proj_list(proj_t(103, 'Project Pluto'))
    WHERE e.eno=1000;
```

DML on Nested Tuples

- There is a difference between a NULL value and an empty constructor. To add back a nested table row, we could have done it in two steps as follows:

```
UPDATE employees e
   SET e.projects = proj_list() // Creates a nested table w/o any rows
   WHERE e.eno=1000;

INSERT INTO TABLE
   (SELECT e.projects FROM employees e WHERE e,eno = 1000)
   VALUES (proj_t(102, 'Project Pluto'));
```

Multilevel Collection Types

- Collection types whose elements are themselves another collection type.

- Possible multilevel collection types are:
  - Nested table of nested table type
  - Nested table of varray type
  - Varray of nested table type
  - Varray of varray type
  - Nested table or varray of a user-defined type that has an attribute that is a nested table or varray type

Multilevel Collection Types Example

- Models a system of stars in which each star has a collection of the planets revolving around it, and each planet has a collection of its satellites.
  - CREATE TYPE sat_t AS OBJECT (name VARCHAR2(20), orbit NUMBER);

    /
  - CREATE TYPE sat_ntt AS TABLE OF sat_t

    /
  - CREATE TYPE planet_t AS OBJECT (name VARCHAR2(20), mass NUMBER, satellites sat_ntt)

    /

Multilevel Collection Types Example… (contd.)

- CREATE TYPE planet_ntt AS TABLE OF planet_t;
  /
- CREATE TYPE star_t AS OBJECT (name VARCHAR2(20), age NUMBER, planets planet_ntt)
  /
- CREATE TABLE stars_tab of start_t (name PRIMARY KEY)
  NESTED TABLE planets STORE AS planets_nttab
  (NESTED TABLE satellites STORE AS satellites_nttab);
- Separate nested table clauses are provided for the outer planets nested table and for the inner satellites one.

Multilevel Collection Types Example… (contd.)

- Inserting a new star called ‘Sun’…

- INSERT INTO stars VALUES('Sun',23,

```
  nt_pl_t( planet_t( 'Neptune',10,
                  nt_sat_t(satellite_t('Proteus',67),
                           satellite_t('Triton',82))),
               planet_t('Jupiter',189,
                  nt_sat_t(satellite_t('Callisto',97),
                           satellite_t('Ganymede', 22)) ) ));
```

Multilevel Collection Types Example… (contd.)

- Inserting a planet called ‘Saturn’ to the star ‘Sun’…

- INSERT INTO TABLE(SELECT planets FROM stars WHERE name = 'Sun')

```
  VALUES ('Saturn', 56,
    nt_sat_t(
      satellite_t('Rhea', 83)
    )
  );
```

Multilevel Collection Types Example… (contd.)

- Inserting a satellite called ‘Miranda’ to planet ‘Uranus’ of the star ‘Sun’…

- INSERT INTO TABLE(
  SELECT p.satellites
  FROM TABLE(SELECT s.planets
  FROM stars s
  WHERE s.name = 'Sun') p
  WHERE p.name = 'Uranus')
  VALUES ('Miranda', 31);

Summary

- Collection Types
  - VARRAYs
  - Nested Tables
- DDL, DML and SELECTs on Collection Types
- Multilevel collection types