ORDB COMMANDS

Create type

(ROW OBJECTS)

```
CREATE TYPE Student_t AS OBJECT (
        Sno CHAR (5),
        Sname VARCHAR (30),
        Saddress VARCHAR (30),
        Gpa FLOAT
);
```

- Create type with REF

```
        CREATE TYPE Marks_t AS OBJECT (
                Subno CHAR (5),
                Subname VARCHAR (30),
                Saddress VARCHAR (30),
                SDetails REF Student_t
        );
```

Create table

```
CREATE TABLE Student_table OF Student_t;
```

- Create table with single constraints

```
        CREATE TABLE Student_table OF Student_t(
                Sno     primary key,
```

Sname Not null

);

- Create table with multiple constraints

CREATE TABLE Student_table OF Student_t(

Constraint pk_Student PRIMARY KEY(Sno, Sname),

Constraint fk_Student1 FOREIGN KEY(Saddress) REFERENCE Marks

);

Insert data

- Insert data as multi-column table

INSERT INTO Student_table VALUES ('S123','Saman','Malabe',2.35);

(Without calling TYPE CONSTRUCTOR we can store data in row objects)

- Insert data as single-column table

INSERT INTO Student_table VALUES (Student_t ('S123','Saman','Malabe',2.35));

Select data

- Retrieve as multi-column table

SELECT * FROM Student_table;

- Retrieve as Single-column table

SELECT VALUE (s) FROM Student_table s;

Varray

(ROW OBJECT)

- Create Varray

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

- Create table with Varray

```
CREATE TABLE pricelist (
    pno integer,
    prices price_arr  );
```

- Insert data into the table with Varray

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

- Select data from the table with Varray
  - SELECT * FROM pricelist;

    Output:

PNO	PRICES
1	PRICE_ARR(2.5, 3.75, 4.25)

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

    Output:

PNO	PRICE
1	2.5
1	3.75
1	4.25

Nested tables

(ROW OBJECT)

1 CREATE TYPE proj_t AS OBJECT (

```
    projno NUMBER,
```

```
        projname VARCHAR (15));
```

2 CREATE TYPE proj_list AS TABLE OF proj_t;

3 CREATE TYPE employee_t AS OBJECT (

```
        eno number,
        projects proj_list);
```

4 CREATE TABLE employees of employee_t (eno primary key)

NESTED TABLE projects STORE AS employees_proj_table;

- Insert data into nested table

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

- Select data only from the inner nested table

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

PROJNO PROJNAME

----

101 Avionics

102 Cruise control

- Select data from the inner + outer nested table

  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 collection

  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;

Member methods

```
1  CREATE TYPE MenuType AS OBJECT (
        bar REF BarType,
        beer REF BeerType,
        price FLOAT,
        MEMBER FUNCTION priceInYen(rate IN FLOAT)
        RETURN FLOAT
    );
2  CREATE TYPE BODY MenuType AS
        MEMBER FUNCTION
        priceInYen(rate FLOAT)
        RETURN FLOAT IS
            BEGIN
                RETURN rate * SELF.price;
            END;
        END;

3   CREATE TABLE Sells OF MenuType;
```

(SELF == this)

- Add new method to exsisting type

```
ALTER TYPE MenuType
ADD MEMBER FUNCTION priceInUSD(rate FLOAT)
RETURN FLOAT CASCADE;
```

```
CREATE OR REPLACE TYPE BODY MenuType AS
MEMBER FUNCTION
priceInYen(rate FLOAT)
RETURN FLOAT IS
        BEGIN
                RETURN rate * SELF.price;
        END priceInYen;
MEMBER FUNCTION
priceInUSD(rate FLOAT)
RETURN FLOAT IS
        BEGIN
                RETURN rate * SELF.price;
        END priceInUSD;
END;
```

- Object comparison using MAP

```
CREATE TYPE Rectangle_type AS OBJECT
( length NUMBER,
  width NUMBER,
 MAP MEMBER FUNCTION area RETURN NUMBER
);
CREATE TYPE BODY Rectangle_type AS MAP MEMBER FUNCTION area
RETURN NUMBER IS
        BEGIN
                RETURN length * width;
```

```
  END area;
END;
```

```
CREATE TABLE rectangles OF Rectangle_type;
INSERT INTO rectangles VALUES (1,2);
INSERT INTO rectangles VALUES (2,1);
INSERT INTO rectangles VALUES (2,2);

SELECT DISTINCT VALUE(r) FROM rectangles r;
```

Output:

VALUE(R)(LEN, WID)

RECTANGLE_TYP(1, 2)

RECTANGLE_TYP(2, 2)

- Object comparison using ORDER

```
1  CREATE TYPE Customer_typ AS OBJECT
   ( id NUMBER,
   name VARCHAR2(20),
   addr VARCHAR2(30),
   ORDER MEMBER FUNCTION match (c Customer_typ) RETURN INTEGER );
```

```
2 ORDER MEMBER FUNCTION match (c Customer_typ) RETURN INTEGER IS
 BEGIN
        IF id < c.id THEN RETURN -1; -- any num <0
        ELSIF id > c.id THEN RETURN 1; -- any num >0
        ELSE RETURN 0;
        END IF;
 END;
 END;
```

- Methods on nested tables

```
1  CREATE TYPE proj_t AS OBJECT (projno number,
        Projname varchar(15));
  CREATE TYPE proj_list AS TABLE OF proj_t;
  CREATE TYPE emp_t AS OBJECT
   ( eno number,
        projects proj_list,
    MEMBER FUNCTION projcnt RETURN INTEGER
   );
2  CREATE OR REPLACE TYPE BODY emp_t AS MEMBER FUNCTION projcnt
RETURN INTEGER IS
        pcount INTEGER;
        BEGIN
            SELECT count(p.projno) INTO pcount
            FROM TABLE(self.projects) p;
            RETURN pcount;
```

```
    END;
END;
```

```
3  CREATE TABLE emptab OF emp_t
   (Eno PRIMARY KEY)
      NESTED TABLE projects STORE AS emp_proj_tab;
```

```
SELECT e.eno, e.projcnt() projcount
FROM emptab e;
```

Inheritance

- Create super type

```
CREATE TYPE Person_type AS OBJECT
( pid NUMBER,
name VARCHAR2(30),
address VARCHAR2(100) )   NOT FINAL;
```

- Create sub type

```
CREATE TYPE Student_type UNDER Person_type
      ( deptid NUMBER,
  major VARCHAR2(30)) NOT FINAL;
```

```
CREATE TYPE Employee_type UNDER Person_type
      ( empid NUMBER,
        mgr VARCHAR2(30)
```

);

- Change final type → not final / not final → final

  ALTER TYPE Person_type FINAL;

- Create super type table

  Create table person_tab of person_type

  (pid primary key);

- Inserting a subtype object/row

  Insert into person_tab values

  (student_type(4, 'Edward Learn',

  '65 Marina Blvd, Ocean Surf, WA, 6725',

  40, 'CS'));

- Select values from inheritance

  SELECT VALUE(p) FROM person_tab p;

  Output:

  VALUE(P)(PID, NAME, ADDRESS)

  Person_type(21937, 'Fred', '4 Ambrose Street')

  Student_type(27362, `Peter', … , 21, 'Oragami')

  PartTimeStudent_type(2134, 'Jack',…, 13, 'Physics', 5)

  Person_type(21362, 'Mary', …)

  Student_type(18437, `Susan', … , 13, 'Maths')

  PartTimeStudent_type(4318, 'Jill',…, 21, 'Pottery', 2)

Person_type(39374, 'George', …)

- Select values using IS OF

```
SELECT VALUE(s)
  FROM person_tab s
  WHERE VALUE(s) IS OF (Student_type);
```

Output:

VALUE(P)(PID, NAME, ADDRESS)

Student_typ(27362, `Peter', … , 21, 'Oragami')
PartTimeStudent_type(2134,'Jack',…,13,'Physics', 5)
Student_typ(18437, `Susan', … , 13, 'Maths')
PartTimeStudent_type(4318,'Jill',…,21,'Pottery', 2)

- Select values using IS OF ONLY

```
SELECT VALUE(s)
FROM person_tab s
WHERE VALUE(s) IS OF (ONLY student_type);
```

Output:

VALUE(P)(PID, NAME, ADDRESS)

Student_typ(27362, `Peter', … , 21, 'Oragami')
Student_typ(18437, `Susan', … , 13, 'Maths')

- Select values using TREAT

```
SELECT Name, TREAT(VALUE(p) AS PartTimeStudent_type).numhours hours
    FROM person_tab p
    WHERE VALUE(p) IS OF (ONLY PartTimeStudent_type);
```

Output:

NAME hours

Jack 5

Jill **2**

Not instantiable types

- CREATE TYPE Address_typ AS OBJECT(...) NOT INSTANTIABLE NOT FINAL;
- CREATE TYPE AusAddress_typ UNDER Address_typ(...);
- CREATE TYPE IntlAddress_typ UNDER Address_typ(...);

```
CREATE TYPE T AS OBJECT (
        x NUMBER,
        NOT INSTANTIABLE MEMBER FUNCTION func1()* RETURN NUMBER )
    NOT INSTANTIABLE NOT FINAL;
```

Not instantiable types with overriding

```
CREATE TYPE MyType AS OBJECT
  (  ...,
    MEMBER PROCEDURE Print,
    FINAL MEMBER FUNCTION foo(x NUMBER) …, ...
  )   NOT FINAL;

CREATE TYPE MySubType UNDER MyType
  ( ...,
   OVERRIDING MEMBER PROCEDURE Print,
```

```
            ...);
```

Not instantiable types with overloading

```
CREATE TYPE MyType AS OBJECT
          ( ...,
            MEMBER FUNCTION fun(x NUMBER)…,
            ...) NOT FINAL;
/
CREATE TYPE MySubType UNDER MyType
          ( ...,
     MEMBER FUNCTION fun(x DATE) …,
            ...);
/
```