UNIVERSITÉ LUMIÈRE LYON 2

dkm² Erasmus Mundus master course in Data Mining and Knowledge Management a european master

Complex data warehouses

Academic year 2014-2015
Jérôme Darmont
http://eric.univ-lyon2.fr/~jdarmont/

Complex DW news

http://eric.univ-lyon2.fr/~jdarmont/?page_id=457

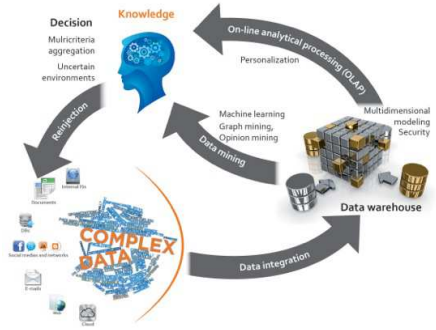
http://eric.univ-lyon2.fr/~jdarmont/?feed=rss2

https://twitter.com/darmont_lyon2 *hashtag* #cdw

Nature of data

- Well-mastered data are "simple".
  - Numerical
  - Symbolic
- Need to process and analyze more diverse and heterogeneous data
  - Multimedia
  - Originating from the Web
  - …

From complex data to decision @ERIC lab



(syntax-oriented)

One definition of complex data

Multiformat
and/or
Multistructure
and/or
Multisource
and/or
Multimodal
and/or
Multiversion

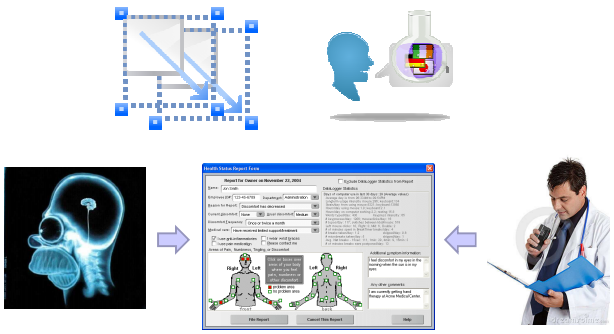
(At least 2 features)

Multiformat data

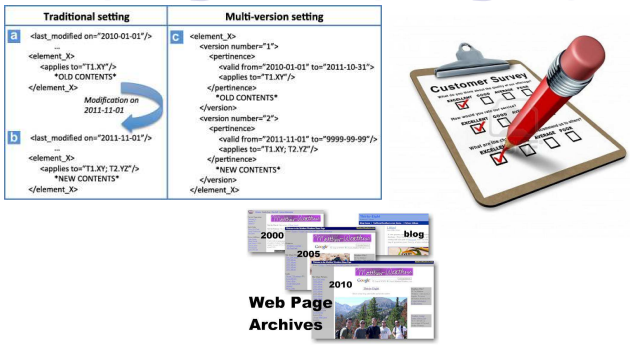
Multistructure data

Multisource data

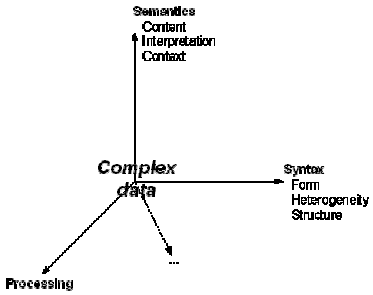
Multimodal data



Multiversion data



More axes of complexity



Issues

- Represent/model complex data
- Efficiently store complex data
- Analyze complex data

➲ New decision-support approaches are needed.

➲ What technology to support them?

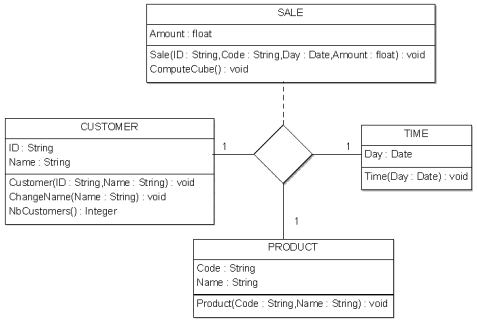
Go XML!

- XML dialects help represent easily any type of loosely structured data.
- XML is now a standard for representing business data for decision-support purposes.
- XML supports:
  - logical modeling (with DTDs or XML Schemas);
  - storage (in native or compatible XML DBMSs);
  - elaborate querying (with XQuery).
- Recurring issue: performance

Go objects!

- Objects help model more complex structures than flat tables.
- Object-oriented conceptual data warehouse models are easy to design.
- Objects are efficiently supported by modern DBMSs (object-relational), e.g., Oracle, DB2, PostgreSQL…

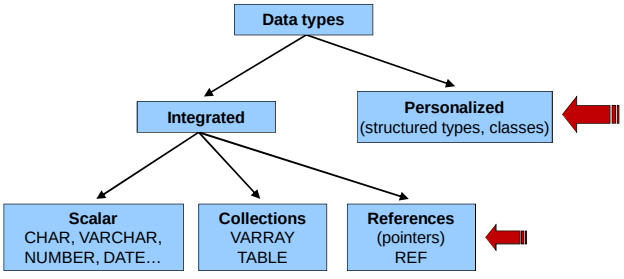
Sample datamart



Outline

- ✓ Introduction
- Classes and objects
- Persistence
- References
- Integrity constraints
- Methods
- Schema evolutions
- Object views
- Data dictionary
- Privileges

Oracle data types



Classes and attributes

- Class creation
  - CREATE TYPE T_CUSTOMER AS OBJECT (
    ID VARCHAR(25),
    Name VARCHAR(255))
    /
- Inheritance
  - CREATE TYPE T_PERSON AS OBJECT (…) NOT FINAL
    /
    CREATE TYPE T_CUSTOMER UNDER T_PERSON (…)
    /

Classes and attributes

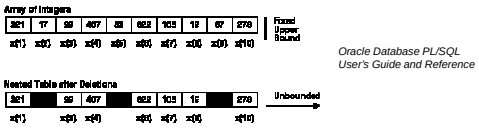
- Recursive class
  - CREATE OR REPLACE TYPE T_CUSTOMER AS OBJECT (
    ID VARCHAR(25),
    Name VARCHAR(255),
    RefBoss REF T_CUSTOMER)
    /
- Display class structure
  - DESC T_CUSTOMER
- Destroy class
  - DROP TYPE T_CUSTOMER;

Mutually dependent classes

- Declaration of incomplete classes
  - CREATE TYPE T_PROFESSOR
    / -- Incomplete type
    CREATE TYPE T_COURSE AS OBJECT (…,
    RefProf REF T_PROFESSOR)
    /
    CREATE TYPE T_PROFESSOR AS OBJECT(…,
    RefCourse REF T_COURSE)
    / -- Incomplete type completion
- Destruction of dependent classes
  - DROP TYPE T_PROFESSOR FORCE;

Collections

- Bounded size, dense collection
  - CREATE TYPE TAB_RATING1 AS
    VARRAY(10) OF INTEGER;
    /
- Extensible, sparse collection
  - CREATE TYPE TAB_RATING2 AS
    TABLE OF INTEGER;
    /



Oracle Database PL/SQL User's Guide and Reference

Collection management

- EXISTS(i) returns TRUE if the i^{th} element of the collection exists.
- COUNT returns the number of elements in the collection.
- LIMIT returns the maximum size of the collection (NULL for TABLE collections).
- EXTEND(n) increases the collection's size by n elements. EXTEND ⇔ EXTEND(1).

Collection management

- TRIM(n) trims n elements at the end of the collection (collection size decreases). TRIM ⇔ TRIM(1).
- DELETE(i) and DELETE delete the i^{th} element of and all elements in the collection, respectively (TABLE collections).
- FIRST and LAST return the indices of the first and last element in the collection, respectively.
  NB: FIRST=1 and LAST=COUNT in a VARRAY collection.
- PRIOR(n) and NEXT(n) return the indices of the previous and next elements of enement of index n, respectively.

Outline

- ✓ Introduction
- ✓ Classes and objects
- Persistence
- References
- Integrity constraints
- Methods
- Schema evolutions
- Object views
- Data dictionary
- Privileges

Non-persistent objects

- In PL/SQL blocks

```
DECLARE
  me T_CUSTOMER;

BEGIN
  me := NEW T_CUSTOMER ('CUST001', 'Darmon');
  me.Name := 'Darmont';
  DBMS_OUTPUT.PUT_LINE(me.ID || ':' || me.Name);
END;
/
```


Persistent objects

- Object table
  - CREATE TABLE CUSTOMER OF T_CUSTOMER (
    CONSTRAINT CUSTOMER_PK PRIMARY KEY (ID));
- SQL instantiation
  - INSERT INTO CUSTOMER VALUES (
    T_CUSTOMER('CUST001', 'Darmont'));
- Attribute-based SQL query/update
  - SELECT Name FROM CUSTOMER
    WHERE ID = 'Cust001';
  - DELETE FROM CUSTOMER WHERE ID = 'Cust001';

Loading persistent objects

- In a SQL query
  - SELECT VALUE(c) FROM CUSTOMER c;
- In a PL/SQL block

```
DECLARE
  me T_CUSTOMER;
BEGIN
   SELECT VALUE(c) INTO me
        FROM CUSTOMER c WHERE ID = 'Cust001';
END;
/
```


Nested tables

- Class definition
  - CREATE TYPE T_ORDER AS OBJECT (OID VARCHAR(25))
    /
  - CREATE TYPE TAB_ORDER AS TABLE OF T_ORDER
    /
  - CREATE OR REPLACE TYPE T_CUSTOMER AS OBJECT (…,
    OrderList TAB_ORDER)
    /
- Object table definition
  - CREATE TABLE CUSTOMER OF T_CUSTOMER
    (CONSTRAINT CUST_PK PRIMARY KEY(ID))
    NESTED TABLE OrderList STORE AS ORDER;

Multi-level collections

- Sample classes
  - CREATE TYPE TAB_MARKS AS
    TABLE OF NUMBER(4,1)
  - CREATE TYPE T_EXAM AS OBJECT(
    RefCourse REF T_COURSE,
    Marks TAB_MARKS)
  - CREATE TYPE TAB_EXAMS AS TABLE OF T_EXAM
  - CREATE TYPE T_STUDENT AS OBJECT(
    StudentID NUMBER(5), …,
    Results TAB_EXAMS)
- Object table
  - CREATE TABLE STUDENT OF T_STUDENT(
    CONSTRAINT student_pk PRIMARY KEY(StudentID))
    NESTED TABLE Results STORE AS ET1_RES
    (NESTED TABLE Marks STORE AS ET2_MARKS);

Nested table update

- Inserting a parent element
  - INSERT INTO CUSTOMER VALUES (
    T_CUSTOMER('CUST001', 'Darmont',
    TAB_ORDER(T_ORDER('O1'), T_ORDER('O2')));
- Inserting in the collection
  - INSERT INTO TABLE (SELECT OrderList FROM CUSTOMER
    WHERE ID = 'CUST001')
    VALUES (T_ORDER('O9'), T_ORDER('O3'));

Nested table update

- Collection update
  - UPDATE TABLE (SELECT OrderList FROM CUSTOMER WHERE ID = 'CUST001') o SET o.OID = 'o1' WHERE o.OID = 'O1';
  - UPDATE TABLE (SELECT OrderList FROM CUSTOMER WHERE ID = 'CUST001') o SET VALUE(o) = T_ORDER('O0') WHERE o.OID = 'o1';
- Collection deletion
  - DELETE TABLE (SELECT OrderList FROM CUSTOMER WHERE ID = 'CUST001') o WHERE o.OID = 'O0';

Nested table querying

- Unnesting queries
  - SELECT c.ID, o.OID FROM CUSTOMER c, TABLE(c.OrderList) o;
- Collection extraction
  - SELECT o.OID FROM TABLE (SELECT OrderList FROM CUSTOMER WHERE ID = 'CUST001') o ;
- Embedded cursor
  - SELECT c.ID, CURSOR(SELECT o.OID FROM TABLE (OrderList) o) FROM CUSTOMER c;

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- References
- Integrity constraints
- Methods
- Schema evolutions
- Object views
- Data dictionary
- Privileges

Reference definition

```
CREATE TYPE T_SALE AS OBJECT (
        RefCustomer REF T_CUSTOMER,
        RefProduct REF T_PRODUCT,
        RefTime REF T_TIME,
        Amount FLOAT )
/
```


Reference instantiation

```
INSERT INTO SALE VALUES (
    T_SALE(
        (SELECT REF(c) FROM CUSTOMER c
                WHERE c.ID = 'Cust001'),
        (SELECT REF(p) FROM PRODUCT p
                WHERE p.Code = Pr32blue'),
        (SELECT REF(t) FROM TIME t
                WHERE t.Day = '15-09-2010'),
        500.75)
);
```


Reference update

```
UPDATE SALE s
    SET s.RefProduct =
        (SELECT REF(p) FROM PRODUCT p
                WHERE p.Code = 'LuxuryGood')
    WHERE s.Amount > 5000;
```

Implicit joins

- In a WHERE clause
  - SELECT s.Amount FROM SALE s
    WHERE s.RefCustomer.Name = 'Darmont';
- In a SELECT clause
  - SELECT DISTINCT s.RefCustomer.Name FROM SALE s
    WHERE s.Amount > 1000;
- Dereferencing
  - SELECT DEREF(RefCustomer) FROM SALE s
    WHERE ROWID = 'AAANB6 AAF AADmRx AAC';

Type substitution

- Objects: An object may be an instance of any subtype (class) of its own type.
- Object tables: An object table may store any instance of a subtype of the type it is built upon.
- References: A reference targeting a type may point to an instance of any of its subtypes.

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- Integrity constraints
- Methods
- Schema evolutions
- Object views
- Data dictionary
- Privileges

Fact unique identifier

- In Oracle, a primary key *cannot* be defined on references.
- Solution 1: Add a separate identifier attribute in class T_SALE that can be used as primary key in table SALE.
- Solution 2: Enforce integrity with a trigger.

Trigger fact unicity enforcement

```
CREATE TRIGGER TRIG_SALE_PK
BEFORE INSERT OR UPDATE ON SALE
FOR EACH ROW

DECLARE
        n INTEGER;
        null_problem EXCEPTION;
        double_problem EXCEPTION;

BEGIN
        IF :NEW.RefCustomer IS NULL OR :NEW.RefProduct IS NULL
        OR :NEW.RefTime IS NULL THEN
                RAISE null_problem;
        END IF;
```


Trigger fact unicity enforcement

```
        SELECT COUNT(*) INTO n FROM SALE s
                WHERE s.RefCustomer = :NEW.RefCustomer
                AND s.RefProduct = :NEW.RefProduct
                AND s.RefTime = :NEW.RefTime;

        IF n > 0 THEN
                RAISE double_problem;
        END IF;

EXCEPTION
        WHEN null_problem THEN
           RAISE_APPLICATION_ERROR(-20501, 'Primary key with NULL
                                                        value(s).');
        WHEN double_problem THEN
           RAISE_APPLICATION_ERROR(-20502, 'Primary key already exists.');

END;
```

Referential integrity

- Parent-child consistency
  - Deleting a dimension instance that is referenced in a fact must *not* be allowed.
  - CREATE TABLE SALE OF T_SALE (
    CONSTRAINT SALE_CUSTREF RefCustomer
    REFERENCES CUSTOMER,
    CONSTRAINT SALE_PRODREF RefProduct
    REFERENCES PRODUCT,
    CONSTRAINT SALE_TIMEREF RefTime
    REFERENCES TIME);

Referential integrity

- Child-parent consistency
  - Inserting a reference to a non-existent dimension instance must *not* be allowed.
  - Achieved by construction (SELECT REF()…)
  - Prevent NULL values with domain constraints, e.g.,
    CONSTRAINT SALE_CUSTREF_NOTNULL
    CHECK (RefCustomer IS NOT NULL)
    (redundant if trigger is used for enforcing primary key)

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- ✓ Integrity constraints
- Methods
- Schema evolutions
- Object views
- Data dictionary
- Privileges

Methods in Oracle

- Encapsulation is *not* automatic.
- Overloading is possible.
- Three types of methods:
  - MEMBER methods apply onto objects.
  - STATIC methods apply onto types.
  - CONSTRUCTOR methods instantiate objects.
- Invoking methods: queries, other methods, PL/SQL, external program,

Method declaration

```
CREATE TYPE T_CUSTOMER AS OBJECT (

    ID VARCHAR(25),
    Name VARCHAR(255),

    CONSTRUCTOR FUNCTION T_Customer (
        ID VARCHAR,
        Name VARCHAR) RETURN SELF AS RESULT,
    MEMBER PROCEDURE ChangeName (Name VARCHAR),
    STATIC FUNCTION NbCustomers RETURN NUMBER )
/
```


Method implementation

```
CREATE TYPE BODY T_CUSTOMER AS

    CONSTRUCTOR FUNCTION T_Customer (
        ID VARCHAR,
        Name VARCHAR) RETURN SELF AS RESULT IS
BEGIN
        SELF.ID := ID;
        SELF.Name := Name;
        INSERT INTO CUSTOMER VALUES (SELF);
        RETURN;
END;
```

Method implementation

```
MEMBER PROCEDURE
                ChangeName (Name VARCHAR) IS
BEGIN
        SELF.Name := Name;
        UPDATE CUSTOMER
                SET OBJECT_VALUE = SELF
                WHERE ID = SELF.ID;
END;
```


Method implementation

```
    STATIC FUNTION NbCustomers RETURN NUMBER IS
            n INTEGER;
    BEGIN
            SELECT COUNT(*) INTO n FROM CUSTOMER;
            RETURN n;
    END;

END; -- End body
/
```


Method invocation

```
DECLARE
    me T_CUSTOMER;
    nbcust INTEGER;

BEGIN
    me := NEW T_CUSTOMER('Cust001', 'Darmon');
    me.ChangeName('Darmont');
    nbcust := T_CUSTOMER.NbCustomers;
END;
/
```


Overloading within a class

- Interface
  - CREATE TYPE T_CUSTOMER AS OBJECT (
    …
    MEMBER PROCEDURE ChangeName (Name VARCHAR),
    MEMBER PROCEDURE ChangeName,
    …)
- Body
  - MEMBER PROCEDURE ChangeName IS
    BEGIN
      SELF.Name = UPPER(SELF.Name);
      UPDATE CUSTOMER SET OBJECT_VALUE = SELF
            WHERE ID = SELF.ID;
    END;

Overloading in a subclass

- Superclass
  - CREATE TYPE T_PERSON AS OBJECT (
    …
    NOT FINAL MEMBER PROCEDURE Display,
    …
) NOT FINAL
- Subclass
  - CREATE TYPE T_CUSTOMER UNDER T_PERSON (
    …
    OVERRIDING MEMBER PROCEDURE Display,
    …
)

MAP object comparison

- Definition
  - CREATE TYPE T_FRACTION AS OBJECT (
    Numerator INTEGER,
    Denominator INTEGER,
    MAP MEMBER FUNCTION Compute RETURN REAL)
    /
  - CREATE TYPE BODY T_FRACTION AS
    MAP MEMBER FUNCTION Compute RETURN REAL IS
    BEGIN
            RETURN Numerator / Denominator;
    END;
    END;
    /

MAP object comparison

- Usage
  - DECLARE
    f1 T_FRACTION;
    f2 T_FRACTION;

    BEGIN
    f1 := NEW T_FRACTION (10, 4);
    f2 := NEW T_FRACTION (249, 100);
    IF f1 > f2 THEN
    -- (…)
    END IF;
    END;
    /

ORDER object comparison

- Used for comparing objects that are too complex for a MAP method
- Only one MAP or ORDER method per class
- Interface
  - CREATE TYPE T_FRACTION AS OBJECT (
    Num INTEGER,
    Den INTEGER,
    ORDER MEMBER FUNCTION Compare (f T_FRACTION)
    RETURN INTEGER)
    /

ORDER object comparison

- Body
  - CREATE TYPE BODY T_FRACTION AS
    ORDER MEMBER FUNCTION Compare (f T_FRACTION)
    RETURN INTEGER IS
    BEGIN
    IF (f.Num/f.Den) < (SELF.Num/SELF.Den) THEN
    RETURN -1;
    ELSIF (f.Num/f.Den) > (SELF.Num/SELF.Den) THEN
    RETURN 1;
    ELSE
    RETURN 0;
    END IF;
    END;
    END;

ORDER object comparison

- Usage
  - DECLARE
    f1 T_FRACTION;
    f2 T_FRACTION;
    BEGIN
    f1 := NEW T_FRACTION (10, 4);
    f2 := NEW T_FRACTION (249, 100);
    IF f1.Compare(f2) > 0 THEN
    -- (…)
    END IF;
    END;
    /

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- ✓ Integrity constraints
- ✓ Methods
- Schema evolutions
- Object views
- Data dictionary
- Privileges

Possible operations

- Adding/suppressing attributes
- Adding/suppressing methods
- Augmenting (size, precision) an attribute
- Modifying inheritance characteristics (FINAL) et persistence (INSTANTIABLE)

Dependent elements

- Object tables based on the class
- Class referencing a table / Subclass
- PL/SQL programs using the class
- Index referencing a modified attribute of the class
- Views created from the class or a table depending on the class

Howto

- Modify class structure
  ALTER TYPE…
- Recompile body type
  CREATE OR REPLACE TYPE BODY…
- Make object table conform
  ALTER TABLE…
- Recompile PL/SQL programs

Examples

- Attributes
  - ALTER TYPE T_SALE DROP ATTRIBUTE RefCustomer CASCADE INCLUDING TABLE DATA;
  - ALTER TYPE T_SALE ADD ATTRIBUTE (Qty NUMBER(5)) CASCADE INCLUDING TABLE DATA;
  - ALTER TYPE T_SALE MODIFY ATTRIBUTE (Qty NUMBER(10)) CASCADE INCLUDING TABLE DATA;
- If INCLUDING TABLE DATA option not used
  - ALTER TABLE SALE DROP UNUSED COLUMNS;
  - ALTER TABLE SALE UPGRADE INCLUDING DATA;

Examples

- Methods
  - ALTER TYPE T_CUSTOMER DROP MEMBER PROCEDURE ChangeName (Name VARCHAR);
  - ALTER TYPE T_CUSTOMER ADD STATIC PROCEDURE Delete(ID NUMBER);
- Body recompilation
  - CREATE OR REPLACE TYPE BODY T_CUSTOMER AS…
  - Suppress procedure ChangeName's code
  - Add procedure Delete's code
  - Other procedures/functions remain the same

Examples

- Inheritance hierarchy
  - ALTER TYPE T_CUSTOMER NOT INSTANTIABLE;
    Possible only if a table does not depend on the class
  - ALTER TYPE T_CUSTOMER INSTANTIABLE CASCADE INCLUDING TABLE DATA;
    Always possible, does not affect tables
  - ALTER TYPE T_CUSTOMER FINAL CASCADE INCLUDING TABLE DATA;
    Possible if the class does not bear subclasses
  - ALTER TYPE T_CUSTOMER NOT FINAL CASCADE INCLUDING TABLE DATA; -- Substitution forbidden or CASCADE CONVERT TO SUBSTITUTABLE;

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- ✓ Integrity constraints
- ✓ Methods
- ✓ Schema evolutions
- Object views
- Data dictionary
- Privileges

Object views

- Same interest than that of relational views
  - Simplify data access
  - Indirectly save complex queries
  - Present data differently w.r.t. user role (security)
  - Support logical independence
- Easy migration to object technology
  - Use methods on table data
  - Easy interface with object-oriented languages

Object views

- Structure class
  - CREATE TYPE T_COMPLETE_SALE AS OBJECT (
    ID VARCHAR(25),
    Code VARCHAR(25),
    Day Date,
    Amount NUMBER(10,2))
    /
- View creation
  - CREATE VIEW COMPLETE_SALE OF T_COMPLETE_SALE
    WITH OBJECT IDENTIFIER (ID, Code, Day) AS
    SELECT s.RefCustomer.ID, s.RefProduct.Code,
    s.RefTime.Day, Amount
    FROM SALE s;

Modifiable object views

- Example: View V is based on object tables T1 and T2.
  - CREATE TRIGGER TRIG_CASCADE_INSERT
    INSTEAD OF INSERT ON V
    FOR EACH ROW

    BEGIN
    INSERT INTO T1 VALUES(…);
    INSERT INTO T2 VALUES(…);
    END;
    /

Collection-based object views

```
CREATE TYPE T_DEGREE AS OBJECT(Label VARCHAR(20))
/
CREATE TYPE TAB_DEGREE AS TABLE OF T_DEGREE
/
CREATE TYPE T_STUDENT AS OBJECT
        (StudentID NUMBER(10), Degrees TAB_DEGREE)
/

CREATE VIEW VIEW_STUDENT  OF STUDENT
   WITH OBJECT IDENTIFIER(StudentID) AS
      SELECT s.StudentID, CAST( MULTISET(
         SELECT Label FROM DEGREE d
         WHERE d.StudentID = s.StudentID) AS TAB_DEGREE
         ) AS Degrees
      FROM STUDENT s;
```


Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- ✓ Integrity constraints
- ✓ Methods
- ✓ Schema evolutions
- ✓ Object views
- Data dictionary
- Privileges

Classes, object tables, collections

- USER_TYPES (TYPE_NAME, SUPERTYPE_NAME, FINAL, INSTANTIABLE…)
- USER_TYPE_VERSIONS (TYPE_NAME, VERSION#…)
- USER_OBJECT_TABLES (TABLE_NAME, OBJECT_ID_TYPE, TABLE_TYPE, NESTED…)
- USER_COLL_TYPES (TYPE_NAME, COLL_TYPE, UPPER_BOUND, ELEM_TYPE_NAME…)

Attributes, methods

- USER_TYPE_ATTRS (TYPE_NAME, ATTR_NAME, ATTR_TYPE_NAME, LENGTH, PRECISION, INHERITED…)
- USER_TYPE_METHODS (TYPE_NAME, METHOD_NAME, METHOD_TYPE, FINAL, INSTANTIABLE, OVERRIDING, INHERITED…)
- USER_METHOD_PARAMS (TYPE_NAME, METHOD_NAME, PARAM_NAME, PARAM_MODE, PARAM_TYPE_NAME…)
- USER_METHOD_RESULTS (TYPE_NAME, METHOD_NAME, RESULT_TYPE_NAME…)

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- ✓ Integrity constraints
- ✓ Methods
- ✓ Schema evolutions
- ✓ Object views
- ✓ Data dictionary
- Privileges

Invocation privileges

- Executing member methods
  - With the privileges of the class creator
    CREATE TYPE class_name AUTHID DEFINER
    AS OBJECT(…)
    /
  - With the privileges of the current user
    CREATE TYPE class_name AUTHID CURRENT_USER
    AS OBJECT(…)
    /

Object privileges

- EXECUTE
  - Definition of an object table of the class
  - Declaration of a variable or parameter of the class
  - Invation of the class' methods
- UNDER
  - Creation of subclasses
- Privilege transmission
  - Statement GRANT privilege ON object TO user;

Ex.
```
GRANT EXECUTE, UNDER
ON COURSE TO PUBLIC
WITH GRANT OPTION;
```


Class synonyms

- Create
  - CREATE SYNONYM TypeCourse FOR darmont.T_COURSE;
- Modify
  - CREATE SYNONYM TypeCourse FOR darmont.T_COURSE2;
- Rename
  - RENAME TypeCourse TO CourseType;
- Delete
  - DROP SYNONYM CourseType;

Outline

- ✓ Introduction
- ✓ Classes and objects
- ✓ Persistence
- ✓ References
- ✓ Integrity constraints
- ✓ Methods
- ✓ Schema evolutions
- ✓ Object views
- ✓ Data dictionary
- ✓ Privileges