QUERYING

Multimedia Databases (Exercises)

Prof. Dr. Harald Kosch/ Prof. Dr. Mario Döller
Tutor: Kanishka Ghosh Dastidar, Dr. Alaa Alhamzeh

Task 1: Relational and object-relational databases

Relational databases (RD)

- Only Predefined data types can be used.
- Many aspects of objects can not be expressed directly in relational databases (e.g. inheritance, pointer, polymorphism, collections).
- RDB are good for managing large amounts of data
- Relational queries are easier to optimize automatically than OR-queries

Object-Relational databases (ORD)

- New data types and functions can be implemented.
- Several OO structural features (e.g. , inheritance, pointer, polymorphism, collections) are part of the OR data model.
- ORDB are good at expressing complex relationships among objects.
- Scalable and extensible
- More complex Queries

Task 2: Definitions

- User-defined data types/Object types
  - Attributes: hold the data about the object.
    - Primitive data type: integer, varchar2(), char, etc..
    - Other object type
  - Relationships: specifies the connection between different objects.
    - one to one, one to many, many to one, many to many
  - Methods: to manipulate the object content
    - Procedure
    - Function

Task 2: Definitions

- User-defined data types/Object types

Object Type person_type	
Attributes	**Methods**
first_name last_name street city	get_firstname

```
CREATE OR REPLACE TYPE person_type AS OBJECT (
first_name    VARCHAR2(150),
last_name     VARCHAR2(150),
street        VARCHAR2(250),
city          VARCHAR2(200),
MEMBER FUNCTION get_firstname
RETURN VARCHAR2(150)
);
/
```

→ This makes oracle process the statement.

Object	
first_name :	Vanessa
last_name :	El-Khoury
Street :	Innstrasse 31
city :	Passau

Object	
first_name :	David
last_name :	Coquil
Street :	Innstrasse 21
city :	Passau

Task 2: Definitions

- User-defined data types/Object types

Object Type address_type
Attributes
street city

Object Type person_type	
Attributes	**Methods**
first_name last_name address	get_firstname

```
CREATE OR REPLACE TYPE  address_type AS OBJECT (
street          VARCHAR2(250),
city            VARCHAR2(200)
);
/

CREATE OR REPLACE TYPE  person_type AS OBJECT (
first_name      VARCHAR2(150),
last_name       VARCHAR2(150),
address         address_type ,
MEMBER FUNCTION get_firstname
 RETURN VARCHAR2(150)
);
/
```

Object	
first_name :	Vanessa
last_name :	El-Khoury
Street :	Innstrasse 31
city :	Passau

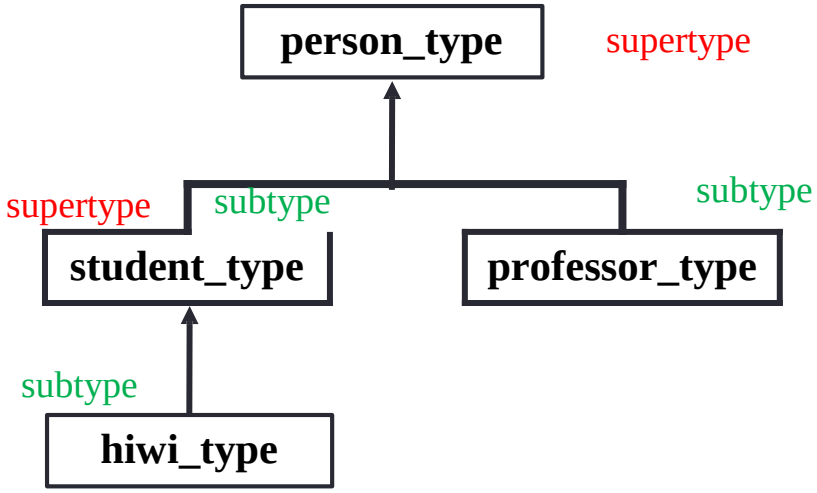
Object	
first_name :	David
last_name :	Coquil
Street :	Innstrasse 21
city :	Passa

Task 2: Definitions

- Inheritance
  - **object inheritance** is based on a family tree of object types that forms a **type hierarchy**. The type hierarchy consists of a parent object type, called a supertype, and one or more levels of child object types, called subtypes, which are derived from the parent.

Task 2: Definitions

- Inheritance



- The subtypes automatically inherit the attributes and methods of their parent type (supertype)
- Update in the supertype yields an update in the subtypes.

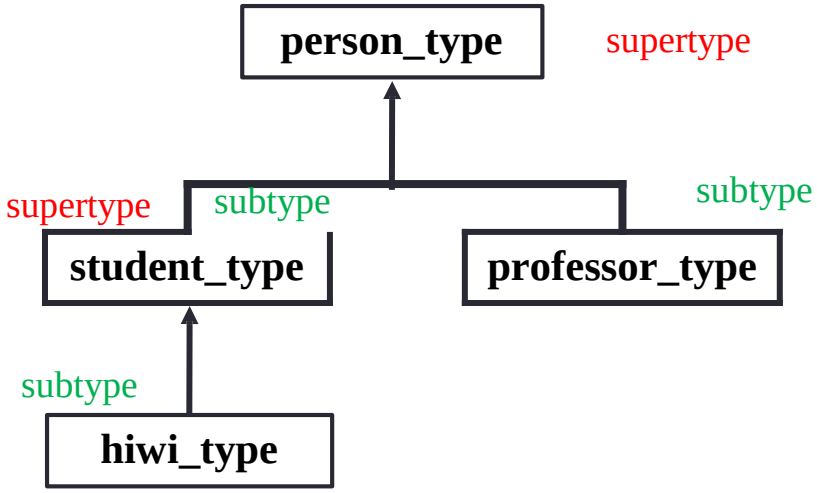
```
CREATE OR REPLACE TYPE  address_type AS OBJECT (
street          VARCHAR2(250),
city            VARCHAR2(200)
);
/

CREATE OR REPLACE TYPE  person_type AS OBJECT (
first_name      VARCHAR2(150),
last_name       VARCHAR2(150),
address         address_type ,
MEMBER FUNCTION get_firstname
RETURN VARCHAR2(150)
) NOT FINAL;  → This allows the inherintance.
/

CREATE OR REPLACE TYPE  student_type  UNDER
person-type (
student_id    INTEGER     → additional attributes.
);
/
```

Task 2: Definitions

- Inheritance



- The subtypes automatically inherit the attributes and methods of their parent type (supertype)
- Update in the supertype yields an update in the subtypes.

```
CREATE OR REPLACE TYPE  address_type AS OBJECT (
street          VARCHAR2(250),
city            VARCHAR2(200)
);
/

CREATE OR REPLACE TYPE  person_type AS OBJECT (
first_name    VARCHAR2(150),
last_name     VARCHAR2(150),
address       address_type ,
MEMBER FUNCTION get_firstname
RETURN VARCHAR2(150),
NOT FINAL MEMBER FUNCTION calculate_salary
RETURN NUMBER
) NOT FINAL;
/

CREATE OR REPLACE TYPE  student_type UNDER
person-type (
student_id    INTEGER
OVERRIDING  MEMBER FUNCTION calculate_salary
RETURN NUMBER
);
/
```

Task 2: Definitions

- Object Tables
  - **Object tables:** store only objects such that each row represents an object, which is referred to as a **row object.**
  - **Relational tables:** store objects with other table data objects that are stored as columns of a relational table, or are attributes of other objects, are called **column objects** (also known as stored inline object or an embedded object)**.**

Task 2: Definitions

- Object Tables

Object Type address_type
Attributes
street city

Object Type person_type	
Attributes	**Methods**
first_name last_name address	get_firstname

Restrictions can only be defined at the level of tables and not in the specification of types

```
CREATE OR REPLACE TYPE
address_type AS OBJECT (
street          VARCHAR2(250),
city            VARCHAR2(200)
);
/
```

```
CREATE TABLE Address _table OF
address_type (
street          NOT NULL,
city            NOT NULL
);
/
```

```
INSERT INTO Address _table VALUES (
(address_type ( 'Innstrasse 31', 'Passau')),
(address_type ( 'Innstrasse 21', 'Passau')),
);
/
```

Object	
first_name :	Vanessa
last_name :	El-Khoury
Street :	Innstrasse 31
city :	Passau

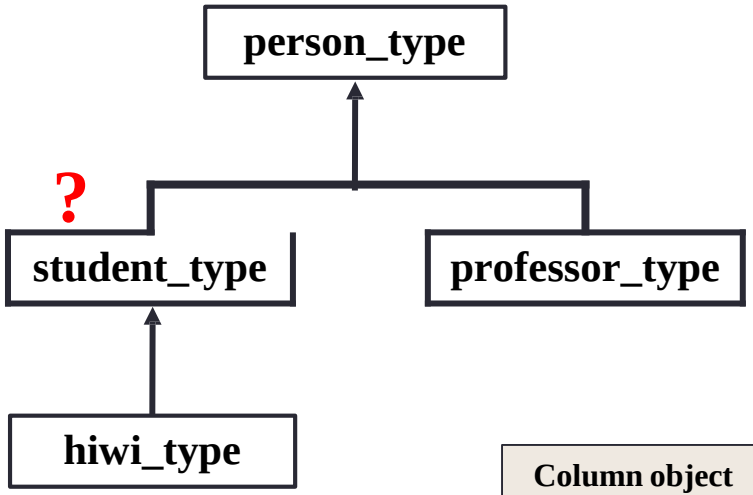
Object	
first_name :	David
last_name :	Coquil
Street :	Innstrasse 21
city :	Passau

Address_table
(Innstrasse 31, Passau)
(Innstrasse 21, Passau)
…

Row object

Task 2: Definitio

```
CREATE OR REPLACE TYPE  person_type AS OBJECT (
first_name    VARCHAR2(150),
last_name     VARCHAR2(150),
address       address_type ,
MEMBER FUNCTION get_get_firstname
 RETURN VARCHAR2(150)
) NOT FINAL;
/
CREATE OR REPLACE TYPE  student_type UNDER
person-type (
student_id    INTEGER     → additional attributes.
);
/
```




```
CREATE TABLE  students OF student_type (
last_name     NOT NULL,
student_id    CONSTRAINT pk_students PRIMARY KEY
);
/
```

Column object

students			
first_name	last_name	address	student_id
Vanessa	EL-Khoury	(Innstrasse 31, Passau)	35418

```
INSERT INTO students VALUES (
'Vanessa ',
'El-Khoury',
address_type ( 'Innstrasse 31', 'Passau'),
35418
);
/
```

Task 2: Definitions

- Polymorphism
  - It means that rows of an object table of type A can contain instances of this type or instances of any subtype of A.
  - **In polymorphic overriding**, subtypes redefine a method they have inherited.
  - **In polymorphic overloading**, there may be several versions of the same method, with different parameters or parameters having different types and order.

Task 2: Definitions

- **Object Identifier OID**
- It uniquely identify row objects in object tables.
- There are two types of **object identifiers:**

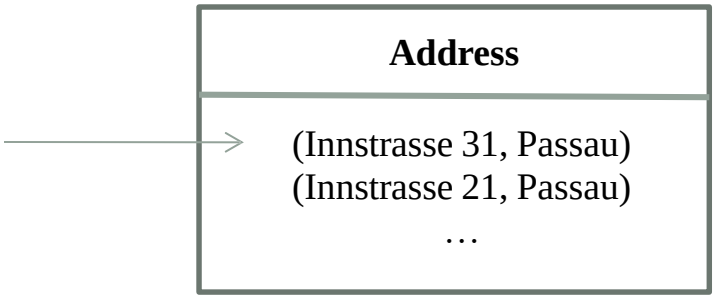
  - System-Generated Object Identifiers (default)
    - Oracle automatically creates system-generated object identifiers for row objects in object tables **unless you choose the primary-key based option.**

  - Primary-Key Based Object Identifiers
    - You have the option to create primary-key based OIDs when you create the table using the CREATE TABLE statement.

Task 2: Definitions

- **Object Identifier OID**
- It uniquely identify row objects in object tables.

0000220208424E801067C2EABBE040578CE70A0707424E8010
67C1EABBE040578CE70A0707



students			
first_name	last_name	address	student_id
Vanessa	EL-Khoury	(Innstrasse 31, Passau)	35418

Primary Key

Task 2: Definitions

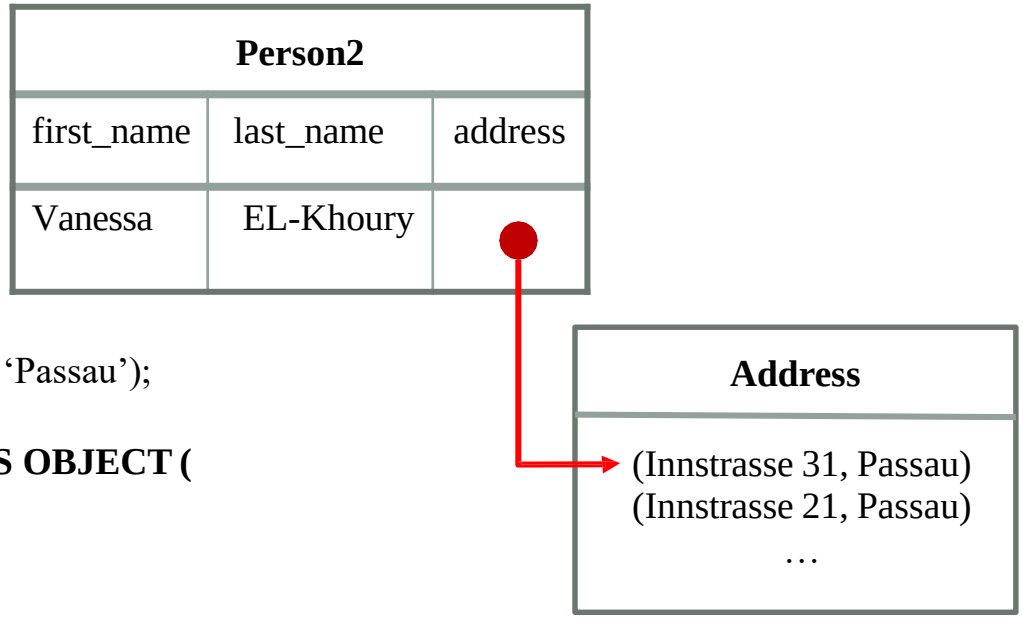
- **Relationships between types**
  - Similar to relational DBMS, the semantic connections among objects are modeled in object relational systems with the help of relations and their cardinality. Three classes of relationships exist:
  - 1:1 relationship connects exactly 2 objects (one of type A and one of type B).
  - 1:n (n:1) relationship specifies that exactly one object of type A is connected to n objects of type B.
  - n:m relationship offers the possibility to connect n objects of type A with m objects of type B.

Task 2: Definitions

- **REFERENCE REF**
- You cannot directly access object identifiers (OID)s, but you can make references (REFs) to the object identifiers and directly access the REFs.
- So what is **REF**?
  - **REF** is a logical pointer or a reference to a single row object of an object table.
  - It uses the OID to point to an object.
  - **REF** is used to obtain, examine, or update the object.

STEPS TO FOLLOW:

```
1)  CREATE OR REPLACE TYPE  address_type AS
street          VARCHAR2(250),
city            VARCHAR2(200)
);
/
2)  CREATE TABLE address OF address_type;
/
3)  INSERT INTO address VALUES ( 'Innstrasse 31', 'Passau');
/
4)  CREATE OR REPLACE TYPE  person_type 2 AS OBJECT (
first_name    VARCHAR2(150),
last_name     VARCHAR2(150),
address       REF address_type ,
) NOT FINAL;
/
5)  CREATE TABLE person2 OF person_type 2 ;
/
6)  INSERT INTO person2 VALUES (
'Vanessa ',
'El-Khoury',
(SELECT REF(a)
FROM address a
WHERE a.street LIKE 'Innstrasse 31'AND a.city LIKE 'Passau')
);
/
```



•The **REF** statement in the select clause allow to retrieve the OID of the wanted row in the table.

•In order to access the object that REF is referring to, we use **DREF**. This is called dereferencing of the **REF**.

•Select **DREF(**p.address**) From** Person2 p;

Task 2: Definitions

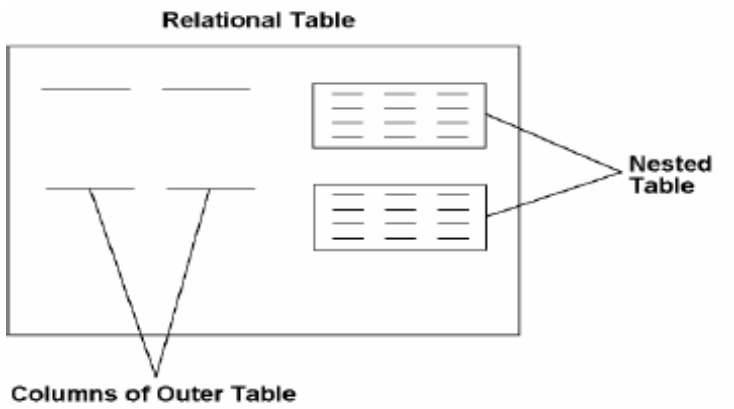
- **Collections**
- **What is a collection?**
  - A collection is a group of values where all values are of the **same data type**.
- **Why do we need collections?**
  - To model the 1:n relationships.

- **What can we declare as a collection?**
  - A column of a table can be declared as a collection type.
  - An attribute of an object can be of a collection type.
  - A collection can contain a collection of object types.

- **How to do it? two collection data types exist:**
  - VARRAYS
  - Nested Tables

Task 2: Definitions

Nested Tables

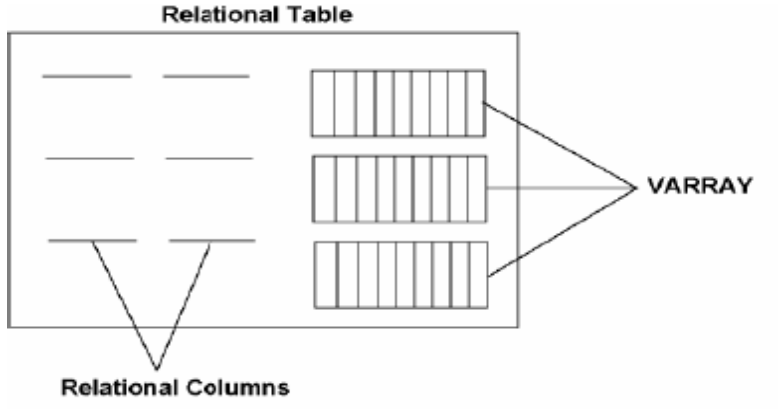


- **Syntax**

CREATE TYPE nested_table_type **AS TABLE OF OF** table_type

nested_table_type : is the name of the Nested table
table_type : is the type of the table

VARRAYS



- **Syntax**

CREATE TYPE array_name **AS VARRAY (**limit**)**
OF data_type

Array_name: is the name of the VARRAY data type.
Limit: is the maximum number of elements that the array can have.
Data_type: is the type of each element of the array. It can be any standard type or object type.

Task 2: Definitions

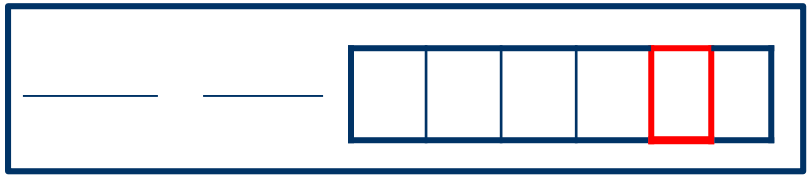
When to use What?

Nested Tables

- Handle arbitrary number of elements.
- Running efficient query on a collection.
- Perform a lot of update and delete operations.

VARRAYS

- Store a fixed number of elements.
- Loop through the elements in order.
- Retrieve and manipulate the entire collection as a single value. **It is not possible to access details of a single elements.**



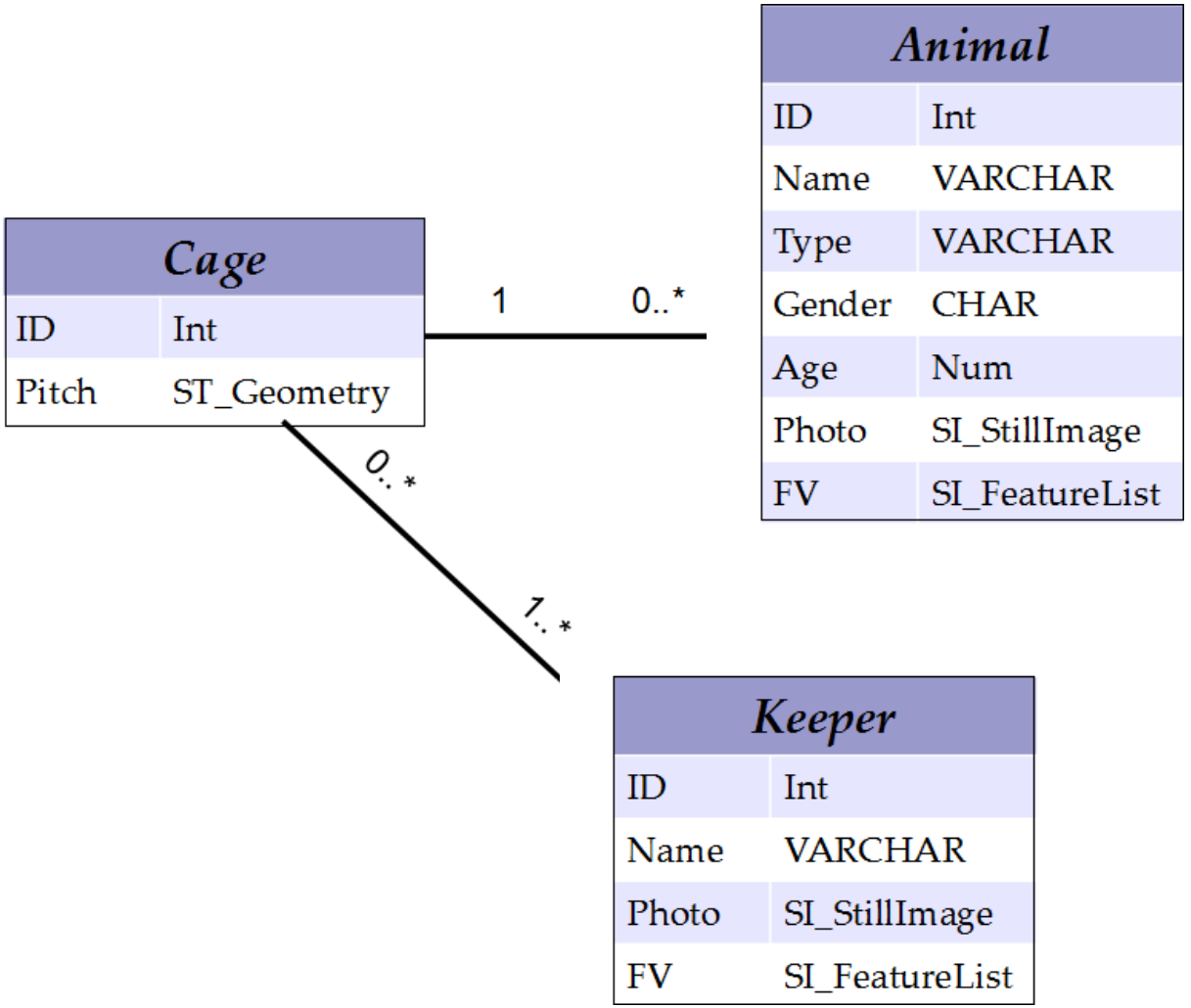
Task 2: Definitions

- Dot operator
- The advantage of references and nested tables is their easy and efficient query behavior. With the help of the dot operator one can easily navigate within the schema without using foreign keys.
  - E.g, to list the address of our students then the following select statement is sufficient:
    - SELECT s.address.street, s.address.city FROM students s;
- Advantages:
  - hides the real complexity of the schema.
  - We do not need any foreign keys and no JOIN between tables.

Task 3: DB-Application

- DB-application to manage a zoo. The DB must include information about:
  - Animals: Name, Gender, Type, Age and an image
  - Keepers: Name and a Pass-photo
  - Cages: Pitch (location), and at least 1 Keeper
  - The application offers the possibility to search for animals and keepers, based on SQL/MM features.
- Sketch the entity–relationship model (ER model) for the given DB.

ER Model


Animal
ID Int
Name VARCHAR
Type VARCHAR
Gender CHAR
Age Num
Photo SI_StillImage
FV SI_FeatureList
Cage
ID Int
Pitch ST_Geometry
1
0..*
0..*
1..*
Keeper
ID Int
Name VARCHAR
Photo SI_StillImage
FV SI_FeatureList

DB Creation: Animal

- Animal:

```
-- Object type
CREATE OR REPLACE TYPE AnimType AS OBJECT (
   ID INT,
   Name VARCHAR(50),
   Type VARCHAR(50),
   Gender CHAR,
   Age NUMBER,
   Photo SI_StillImage,
   FV SI_FeatureList
);

-- Object table: instance of the O-type
CREATE OR REPLACE TABLE Animal OF AnimType (
   ID PRIMARY KEY
);
```

- *SI_StillImage and SI_FeatureList are defined data-types in SQL/MM*

DB Creation: Keeper

```
CREATE TYPE KeeperType AS OBJECT (
   ID INT,
   Name VARCHAR(50),
   Photo SI_StillImage,
   FV SI_FeatureList
);

CREATE OR REPLACE TABLE Keeper OF KeeperType (
   ID PRIMARY KEY
);
```

DB Creation: Cage

- Cage:
  - Animal: 1 to many
  - Keeper : many to many

```
-- Nested types
CREATE OR REPLACE TYPE AnimNTType AS TABLE OF REF AnimType;

CREATE OR REPLACE TYPE KeeperNTType AS TABLE OF REF KeeperType;
```

```
CREATE TABLE Cage (
   ID INT PRIMARY KEY,
   Pitch ST_Geometry,
   AnimalsIn AnimNTType, -- nested table
   Responsible KeeperNTType -- nested table
)
NESTED TABLE AnimalsIn STORE AS AnimalsIn_tbl,
NESTED TABLE Responsible STORE AS Responsible_tbl
;
```

Query

Write the SQL/MM queries to content-based search for animals (Query by Example)

All features have a method SI_Score, which
– Computes the distance between an image and a feature value and
– Returns a real value between 0 and 1.

```
SELECT p1, p2
FROM Picture1 p1, Picture2 p2
WHERE
  p1.photo1_color.SI_Score(p2. photo2) > 0.5 AND
  p1.photo1_texture.SI_Score(p2.photo2)  > 0.4
```

```
SELECT Photo INTO myAnim FROM Animal WHERE ID='90';

SELECT t1.ID, t1.Name FROM Animal t1, myAnim t2
       WHERE t1.FV.SI_Score(t2.Photo) > 0.5;
```

Task 4: MPEG Query Format (MPQF)

- XML-based
- MPQF is composed of three main categories:
  - Management
  - Input Query Format
  - Output Query Format
- The Input Query Format consists of three (optional) components:
  - QFDeclaration
  - OutputDescription
  - QueryCondition

MPQF: Input Structure

- XML Based

```
<?xml version="1.0" encoding="UTF-8"?>
<mpqf:MpegQuery mpqfID="" xmlns:mpqf="urn:mpeg:mpqf:schema:2008">
    <mpqf:Query>
        <mpqf:Input>

            <mpqf:QFDeclaration> ←
                Resources for query processing
            </mpqf:QFDeclaration>

            <mpqf:OutputDescription> ←
                Desired results' content and structure
            </mpqf:OutputDescription>

            <mpqf:Condition> ←
                Conditions
            </mpqf:Condition>

        </mpqf:Input>
    </mpqf:Query>
</mpqf>
```

Mario Doeller et al. Standardized Multimedia Retrieval based on Web Service technologies and the MPEG Query Format. January 2008
Journal of Digital Information Management

Query 1

- Search for images and their titles, which are similar to a presented example image, taken in Berlin, and not bigger than 2048K. Take the similarity criterion as the most important.

```
<OutputDescription>

        <ReqField typeName="MediaLocatorType">/MediaUri
                </ReqField>

        <ReqField typeName="CreationInformationType">/Creation/Title
                </ReqField>

</OutputDescription>
```

Query 1

- Search for images and their titles, which are similar to a presented example image, taken in Berlin, and not bigger than 2048K. Take the similarity criterion as the most important.

```
<QueryCondition>
..
<Condition xsi:type="QueryByMedia" preferenceValue="0.8" matchType="similar">
        <MediaResource>
                        <MediaUri>URI_to_Example_Image</MediaUri>
        </MediaResource>
</Condition>
..
</QueryCondition>
```

MPQF Query-1

- Search for images and their titles, which are similar to a presented example image, taken in Berlin, and not bigger than 2048K. Take the similarity criterion as the most important.

```
<QueryCondition>
..
<Condition xsi:type="QueryByFreeText" preferenceValue="0.2">
        <FreeText>City:Berlin</FreeText>
</Condition>
..
</QueryCondition>
```

Query 1

- Search for images and their titles, which are similar to a presented example image, taken in Berlin, and not bigger than 2048K. Take the similarity criterion as the most important.

```
<QueryCondition>
..
<Condition xsi:type="LessThan">
        <ArithmeticField typeName="MediaFormatType">
                /FileSize </ArithmeticField>
        <LongValue>2048</LongValue>
</Condition>
..
</QueryCondition>
```

Query 2

- Search for images, which have in their northern part a house and their southern part a tree.
- Furthermore, the images must obey the following dominant color distribution (Red 30%, Green 50% and Blue 20%).The search must return only JEPG images, and 30 at the maximum. Results must sorted by the file size.
- Use resources ⇒ Manage resources
- Define all resources in the QFDeclaration section.
  - Example image: House upperArea.png
  - Example image: Tree lowerArea.png
  - MPEG-7 Dominant Color Descriptor

Query 2

```
<mpqf:QFDeclaration>
..
<mpqf:Resource xsi:type="mpqf:MediaResourceType" resourceID="House_1">
    <mpqf:MediaResource>
        <mpqf:MediaUri>http://upperArea.png</mpqf:MediaUri>
    </mpqf:MediaResource>
</mpqf:Resource>

<mpqf:Resource xsi:type="mpqf:MediaResourceType" resourceID="Tree_1">
    <mpqf:MediaResource>
        <mpqf:MediaUri>http://lowerArea.png</mpqf:MediaUri>
    </mpqf:MediaResource>
</mpqf:Resource>
```

Query 2

```
<mpqf:Resource xsi:type="mpqf:DescriptionResourceType" resourceID="DominantColor1">
    ..
    <mpeg7:DescriptionUnit xsi:type="mpeg7:DominantColorType">
        <mpeg7:ColorSpace>RGB</mpeg7:ColorSpace>

        <mpeg7:Value>
            <mpeg7:Percentage>30</mpeg7:Percentage>
            <mpeg7:Index>255 0 0 </mpeg7:Index>
            <mpeg7:ColorVariance>0 0 0 </mpeg7:ColorVariance>
        </mpeg7:Value>

        <mpeg7:Value>
            <mpeg7:Percentage>50</mpeg7:Percentage>
            <mpeg7:Index>0 255 0 </mpeg7:Index>
            <mpeg7:ColorVariance>0 0 0 </mpeg7:ColorVariance>
        </mpeg7:Value>

        <mpeg7:Value>
            <mpeg7:Percentage>20</mpeg7:Percentage>
            <mpeg7:Index>0 0 255 </mpeg7:Index>
            <mpeg7:ColorVariance>0 0 0 </mpeg7:ColorVariance>
        </mpeg7:Value>
    </mpeg7:DescriptionUnit>
    ..
</mpqf:Resource>
..
</mpqf:QFDeclaration>
```

Query 2

- Search for images, which have in their northern part a house and their southern part a tree. Furthermore, the images must obey the following dominant color distribution (Red 30%, Green 50% and Blue 20%). The search must return only JEPG images, and 30 at the maximum. Results must sorted by the file size.

```
<QueryCondition>
..
<mpqf:Condition xsi:type="mpqf:SpatialQuery">
    <mpqf:SpatialRelation relationType="urn:mpeg:mpqf:cs:SpatialRelationCS:2008:south"
            sourceResource="Tree_1" targetResource="House_1"/>
</mpqf:Condition>
..
</QueryCondition>
```

Query 2

- Search for images, which have in their northern part a house and their southern part a tree. Furthermore, the images must obey the following dominant color distribution (Red 30%, Green 50% and Blue 20%). The search must return only JEPG images, and 30 at the maximum. Results must sorted by the file size.

```
<QueryCondition>
..
<mpqf:Condition xsi:type="mpqf:QueryByDescription">
    <mpqf:DescriptionResourceREF>DominantColor1</mpqf:DescriptionResourceREF>
</mpqf:Condition>
..
</QueryCondition>
```

Query 2

- Search for images, which have in their northern part a house and their southern part a tree. Furthermore, the images must obey the following dominant color distribution (Red 30%, Green 50% and Blue 20%). The search must return only JPEG images, and 30 at the maximum. Results must sorted by the file size.

```
<QueryCondition>
..
<mpqf:TargetMediaType>image/jpeg</mpqf:TargetMediaType>
..
</QueryCondition>
```

Query 2

- Search for images, which have in their northern part a house and their southern part a tree. Furthermore, the images must obey the following dominant color distribution (Red 30%, Green 50% and Blue 20%). The search must return only JPEG images, and 30 at the maximum. Results must sorted by the file size.

```
<mpqf:OutputDescription maxItemCount="30">
..
   <mpqf:SortBy xsi:type="mpqf:SortByFieldType">
      <mpqf:Field typeName="MediaFormatType"> /FileSize</mpqf:Field>
   </mpqf:SortBy>

</mpqf:OutputDescription>
```

Thank you for your attention!

Next session is postponed to the week after

(see you on the 5th and 6th of July)