

Riga Technical University Computer Systems Department

Data Models in Database Systems

"***Design and implementation of an object-relational database for "The Logistics Center"***\*

work prepared by

**Shamil Mammadrzayev 211ADB022**

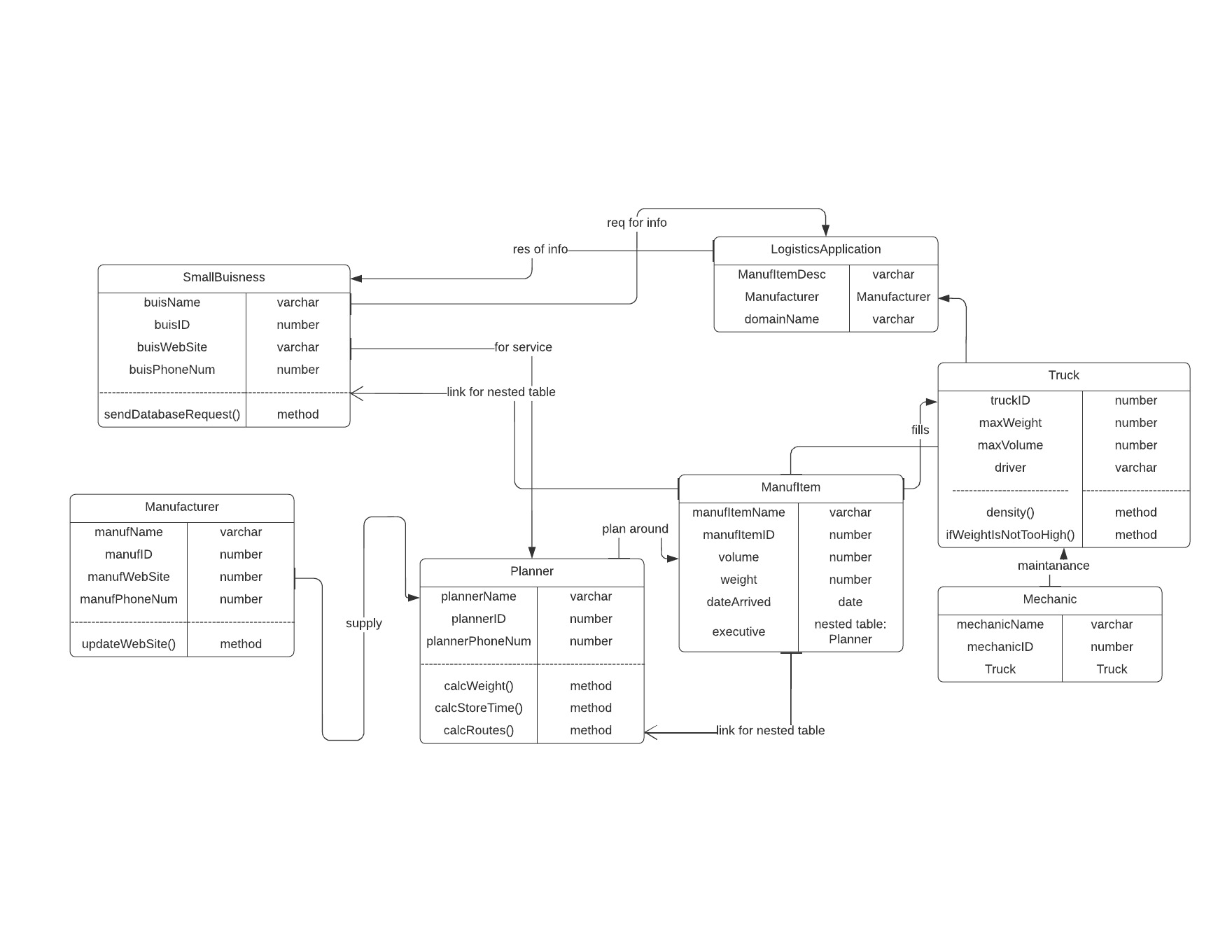
**INDEX**

**1. Develop ER diagram of structures (it is expected you have 3-5 tables and correspondingly: 3-5 object types).**

**2. All the queries, their semantic and technical explanations**

**3. Conclusions**

**CONTENT OF THE WORK:**

**1. Develop ER diagram of structures (it is expected you have 3-5 tables and correspondingly: 3-5 object types).** 

**2. The reference to student’s Oracle server profile**

**2.All the queries, their semantic and technical explanations**

***Query descriptions:***

***1) DEREF() queries***

**The first DEREF query uses the DEREF function to retrieve the website of the manufacturer for a given logistics application, based on the manufacturerr column. The WHERE clause filters the rows to return only the row for the specified manufacturer item.**

|  |
| --- |
| ***SELECT DEREF(l.manufacturerr).manufWebSite FROM LogisticsApplication l WHERE l.manufItemDesc = :manufItemDesc;*** |

**The second DEREF query returns the names of the mechanics in the Mechanic table and the names of the drivers for the trucks that are associated with each mechanic. It does this by using the DEREF function to dereference the REF value in the Truckk column and retrieve the truckID from the Trucks table. It then joins the Trucks table to the Mechanic table on the truckID column to retrieve the driver's name.**

|  |
| --- |
| ***SELECT m.mechanicName, t.driver FROM Mechanic m JOIN Trucks t ON t.truckID = DEREF(m.Truckk).truckID;*** |

**The third DEREF query retrieves the description and manufacturer name for all rows in the LogisticsApplication table. It uses the DEREF function to dereference the REF value in the manufacturerr column and retrieve the manufName attribute from the Manufacturers table.**

|  |
| --- |
| ***SELECT l.manufItemDesc, DEREF(l.manufacturerr).manufName AS manufName FROM LogisticsApplication l;*** |

**The fourth DEREF query finds all trucks that can carry a specific manufacturer's item, sorted by maximum weight in descending order. It does this by using the DEREF function to dereference the REF value in the Truckk column and retrieve the truckID from the Trucks table. It then filters the Trucks table based on the maximum weight and volume of the truck, and the weight and volume of the manufacturer's item.**

|  |
| --- |
| **SELECT m.mechanicName FROM Mechanic m WHERE DEREF(m.Truckk).truckID IN (SELECT t.truckID  FROM Trucks t  WHERE t.maxWeight >= (SELECT mi.weight FROM ManufItem mi WHERE mi.manufIttemID = :itemID)  AND t.maxVolume >= (SELECT mi.volumee FROM ManufItem mi WHERE mi.manufIttemID = :itemID));** |

***2) Value() queries***

**The first VALUE query uses the VALUE function to return the names of the planners in the Planners table and their phone numbers.**

|  |
| --- |
| ***SELECT VALUE(p).plannerName AS plannerName, VALUE(p).plannerPhoneNum AS plannerPhoneNum FROM Planners p;*** |

**The second VALUE query finds all trucks that can carry a specific manufacturer's item, sorted by maximum weight in descending order. It filters the Trucks table based on the maximum weight and volume of the truck, and the weight and volume of the manufacturer's item. It then uses the VALUE function to retrieve the entire Truck object and display the truckID, maxWeight attributes.**

|  |
| --- |
| ***SELECT value(t),t.truckID, t.maxWeight FROM Trucks t WHERE t.maxWeight >= (SELECT mi.weight FROM ManufItem mi WHERE mi.manufIttemID = :itemID) AND t.maxVolume >= (SELECT mi.volumee FROM ManufItem mi WHERE mi.manufIttemID= :itemID) ORDER BY t.maxWeight DESC;*** |

**The third VALUE query uses the VALUE function to return the names of the drivers in the Trucks table and the maximum weight and volume their trucks can handle, filtered by a specific maximum weight range. It filters the Trucks table based on the maxWeight attribute and uses the VALUE function to retrieve the entire Truck object and display the driver and maxWeight and maxVolume attributes.**

|  |
| --- |
| ***SELECT VALUE(t).driver AS driver, t.maxWeight AS maxWeight, t.maxVolume AS maxVolume FROM Trucks t WHERE t.maxWeight BETWEEN 5000 AND 10000;*** |

***3) Table() queries***

**The first TABLE query finds the names and phone numbers of all planners for a specific manufacturer's item. It does this by using the TABLE function to extract the executive nested table from the ManufItem table and join it to the Planners table. The WHERE clause filters the rows to return only the rows for the specified manufacturer's item.**

|  |
| --- |
| **SELECT e.plannerName FROM ManufItem mi, TABLE(mi.executive) e WHERE mi.manufIttemID = :itemID;** |

**The second TABLE query finds the names and phone numbers of all planners who worked with items with volume greater than a specific value. It does this by using the TABLE function to extract the executive nested table from the ManufItem table and join it to the Planners table. The WHERE clause filters the rows based on the volume of the manufacturer's item.**

|  |
| --- |
| **SELECT p.plannerName, p.plannerPhoneNum FROM ManufItem mi, TABLE(mi.executive) p WHERE mi.volumee > :volumee ORDER BY p.plannerName ASC;** |

**The third TABLE query finds the number of planners who have worked on each manufacturer's item, sorted by the number of planners in descending order. It does this by using the TABLE function to extract the executive nested table from the ManufItem table and joining it to the Planners table. It then groups the rows by the manufIttemID attribute and uses the COUNT function to count the number of planners for each manufacturer's item. It sorts the results by the num\_planners column in descending order.**

|  |
| --- |
| ***SELECT mi.manufIttemID, COUNT(p.plannerID) AS num\_planners FROM ManufItem mi, TABLE(mi.executive) p GROUP BY mi.manufIttemID ORDER BY num\_planners DESC;*** |

**The fourth query retrieves the average weight of all manufacturer's items in the ManufItemtable. It does this by using the TABLE function to extract the executive nested table from the ManufItem table and joining it to the ManufItem table. It then calculates the average weight of the manufacturer's items by using the AVG function on the weight attribute. The result is returned as the avg\_weight column**

|  |
| --- |
| ***SELECT AVG(mi.weight) AS avg\_weight FROM ManufItem mi, TABLE(mi.executive) p;*** |

***Methods description :***

**The first procedure, "density", is a map member function of the Truck object type. It calculates and returns the density of the truck by dividing the maxWeight attribute by the maxVolume attribute.**

**The second procedure, "ifWeightIsNotTooHigh", is a member procedure of the Truck object type. It checks if the weight of the truck is too high (greater than 20000), and if it is, it raises an application error. If the weight is not too high, it prints a message to the console saying"Weight is okay".**

**Implementation :**

|  |
| --- |
| ***CREATE OR REPLACE TYPE BODY Truck AS   MAP MEMBER FUNCTION density RETURN NUMBER IS  BEGIN  RETURN maxWeight / maxVolume;  END density;  MEMBER PROCEDURE ifWeightIsNotTooHigh  IS  BEGIN  -- Check if the weight of the truck is not too high  IF (self.maxWeight > 20000) THEN  -- Raise an exception if the weight is too high  RAISE\_APPLICATION\_ERROR(-20000, 'Weight of truck is too high');  else DBMS\_OUTPUT.PUT\_LINE('Weight is okay');  end if;  END;  END;*** |

**Realization:**

|  |
| --- |
| ***DECLARE  po Truck;   BEGIN  po :=NEW Truck(1,10,5,'ben');   DBMS\_OUTPUT.PUT\_LINE('AREA:' || po.density()); -- prints density:2  END;   DECLARE  t Truck; BEGIN  t := Truck(123, 10000, 100, 'John Smith');  t.ifWeightIsNotTooHigh; END;*** |

**The "update\_website" procedure is a member procedure of the Manufacturer object type. It updates the website for a manufacturer in the Manufacturers table based on the manufIDattribute. First, it selects the Manufacturer object based on the manufID and stores it in the "manuf" variable. Then it calls the "update\_website" procedure on the "manuf" variable, passing in the new website as an argument. Finally, it updates the manufWebSite column in the Manufacturers table with the new website value.**

**Implementation:**

|  |
| --- |
| ***CREATE OR REPLACE TYPE BODY Manufacturer AS  MEMBER PROCEDURE update\_website(new\_website varchar2)  AS  BEGIN  UPDATE Manufacturers  SET manufWebSite = new\_website  WHERE manufID = self.manufID;  END; END;*** |

**Realization:**

|  |
| --- |
| ***-- First, let's select the current website for a manufacturer SELECT m.manufWebSite FROM Manufacturers m WHERE m.manufID = 1;   -- Now let's use the update\_website procedure to update the website DECLARE  manuf Manufacturer; BEGIN  SELECT VALUE(m) INTO manuf  FROM Manufacturers m  WHERE m.manufID = 1;  manuf.update\_website('www.accc.com'); END;   -- Now let's select the website again to verify that it was updated SELECT m.manufWebSite FROM Manufacturers m WHERE m.manufID = 1;*** |

**3.Conclusions**

*It was an exceptionally interesting and huge task. It was challenging and we learnt a lot by fixing our theoretical knowledge with real ORACLE PL/SQL programming.*

*At the designing the object-relational database stage we had a pretty clear general understanding based on our past experience (also including Practical #1 of this course). But when it came to differentiating row, column-types alongside object collection, we were a bit confused and now we thank you for great assistance on consultations. Afterwards, everything was clear and we started designing our database by carefully reading the task and dividing the text into valuable functional aspects and working processes, for representing them in a form of object-relational database.*

*So, we put our Logistics Center task into a design ER-diagram look and then started to implement our design in code.*

*At the beginning the type and table creation were pretty easy, because there is nothing extraordinary there. Except the nested table insertion and REF column-type insertion. They are presented below:*

|  |
| --- |
| *INSERT INTO ManufItem (manufIttemName, manufIttemID, volumee, weight, dataArrived, executive) VALUES ('Item 1', 1, 10, 5, TO\_DATE('01-01-2022','DD-MM-YYYY'),  planner\_table(Planner('Nina Garcia', 1, 1234567890)));* |

*Here, we can see that executive attribute is actually a nested table - planner\_table in which Planner with data is passed*

|  |
| --- |
| *INSERT INTO LogisticsApplication (manufItemDesc, manufacturerr, domainName) VALUES ('Widget A', (SELECT REF(m) FROM Manufacturers m WHERE m.manufID = 1), 'Widget');* |

*Here, we see a select for a REF Column-type attribute which retrieves the data from a table by a needed id.*

|  |
| --- |
| *INSERT INTO Mechanic (mechanicName, mechanicID, Truckk) VALUES  ('John', 1, (SELECT REF(T) FROM Trucks T WHERE T.truckID = 1));* |

*Here is the same as previously.*

*Next goes the Query part. The easiest part was the queries containing DEREF() function,I had pretty no problems while creating and compiling them. Talking about VALUE()-containing queries, I was trying to implement more complex ones, but as most of them should have a table function inside of it, I couldn’t achieve the exact goal. The TABLE() functions were the most hard for dealing with, as were continuously getting compilation errors when I was using our nested table. Actually it was one of the most critical obstacles along my way. I originally was going to implement much more complex queries with more semantic meaning, but because of the challenges with nested table usage, I downgraded to a more basic type, which is a pity.*

*The methods creation and implementation was the last part of my work.*

*Originally I has many more methods in our ER-diagram, but in fact they were pretty abstract and hard to implement. But I don’t consider it as my mistake, at that point I was directed on maximum task analysis and to dividing the processes into methods with later consideration of their realization-ability. I had a lot of difficulties with the syntax of the method implementation and realization. In the end I decided to construct some middle-level complexity methods.*

*All in all, it can be called an exceptional academic experience. I have gained a lot of practical knowledge as well as a lot of other things and skills.*