STSCI 5060 Homework 4

Pete Rigas (pbr43 cornell.edu)

November 27, 2020

1 Problem: Determining Appropriate Oracle Data Types

To determine which types of Oracle provided data are the best for the given normalized relations of a furniture company, we observe the following. Because we know that character data types are processed more quickly if we are not performing any mathematical calculations with the data, it is best that we use character values, in the format of Varchar2 for some fixed length depending on the length of the vendor ID, address, and contact name, to store the values in the vendor normalized relation. Although it would be best to specify the number of characters that are included in the Varchar2 Oracle data type, the actual precision depends on how long the vendor ID is, in addition to the address of the vendor, and contact name. Although the address of a specific vendor could include the state and zip code of that particular state, we would need to have an idea of the length that the normalized vendor relation would have to store if we want to specify the precision of each character data type.

As for the item normalized relation, we similarly know that the item ID and description of the item, because they are not data types that will be mathematically manipulated, are best kept as characters because they will be processed more quickly. Finally, we know that the price quote normalized relation, unlike the vendor and item normalized relations, would require an integer data type from Oracle, specifically because the price of an item, which is listed as a component of the price quote normalized relation, would require the number Oracle data type. This covers all of the Oracle data types that we would introduce for each of the 3 normalized relations.

2 Problem: Creating Normalized Relations

To detail which opportunities may exist for denormalization when defining the physical records for such a databse, we recall that for 1 : 1 or many to many relationships, it is possible for us to make use of denormalization to 'collapse,' or perhaps combine, attributes of different entities so that the common attributes make it easier for the database to perform queries on information. More specifically, as mentioned on the prelim, denormalization strives to redefine relations for a database, in such a way so that the relations between different objects in the database have more commonm attributes than completely separating them in normalized relations. Thus, the circumstances from the 1 : 1 or many to many relations that would allow for a denormalized relation include examining the 1 : 1 relations, particularly in the case when there is an optional relationship, because we know by definition that the fields that do not have attributes will be set to null. Also, we know that many to many relationships similarly warrant for denormalization because from this many to many relation, we can combine the attributes from 2 relations. After combining attributes from the relevant entities, it is possible for us to introduce properly denormalized relations. By definition, the resulting relations will absolutely become denormalized because in combining the attributes from 2 entities together, we can prevent the relations from being completely normalized because there will be some information that overlaps between the 2 relations that we are analyzing.

From the denormalized relations, this illustrates the type of situation under which we would expect to introduce denormalized relations, as the denormalized relation would be useful when we are trying to maximize the database performance. Although this may not always be the best option, denormalization, in some cases, can serve the purpose of allowing the database to more easily query for given information. From the structure of a query, in addition to how the information is stored, we know that the denormalized relation can reduce the query time that a database must carry out to retrieve, and process, given information. Specifically, the denormalized records from the physical database would help with structuring the relations of the database differently, so that the query time of the database would be reduced. From such a more efficient implementation of the database, it would be more easy to reference useful data.

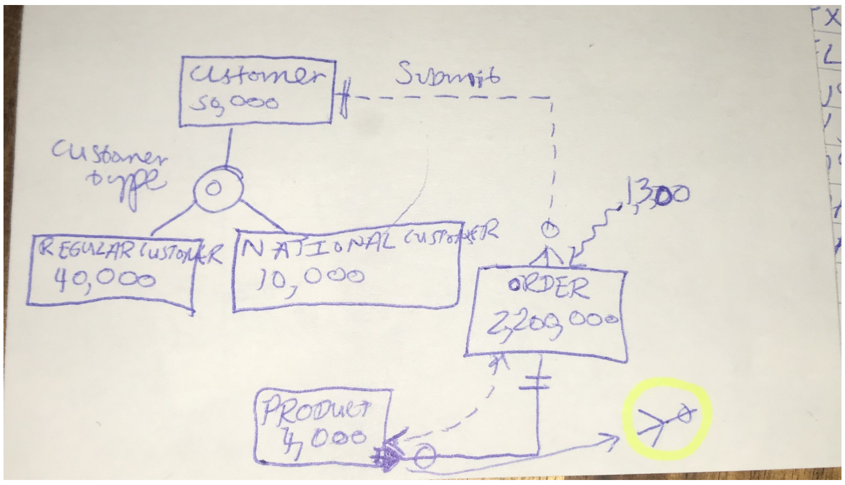
3 Problem: Drawing a Usage Map for the EER Diagram

3.1 Explanation

To complete a usage analysis from the given EER, we observe the following. Namely, from the portion of the business that is represented, in which a customer has relationships with the order, and that the order has a relationship with the order line, which in turn has a relationship with products that compose a customer's order, we can represent the usage analysis by taking into account the cardinality of the given relationships. From the illustrated cardinality between the customer and the order that he or she places , we know that this is a one to many relationship, because one customer can place more than one order that needs to be fulfilled by the order line that puts together the products that a customer orders. Also, we know that the assumptions below, as the business rules, reflect the fact that from the total of 50, 000 customers, the regular and national customer base are clearly different, which is reflected in the overlap condition from the customer supertype, in which the subtypes national customer and regular customer share attributes with the customer supertype, as the regular and national customers both have name and identification that are helpful for the database to process the orders that are placed. Next, it is important to recognize that from the average number of products per order, the given EER reflects the volume of daily order by adding in dashed arrows to not only represent **how** many orders are placed, but also **which** customers place a specific order, which can be useful for querying the database. Therefore, with the frequency of orders that are placed, we are able to compute a usage analysis of the given EER because from the given entities and attributes, the relationships, in addition to the cardinality of such relationships, determines, according to the business rules, whether a customer can place more than one order. Because the business rules of most organizations are alright with having one single customer place more than one order, it is clear that the regular customer, or national customer, subtype instances of the customer supertype share attributes of the customer supertype; the customer supertype instance has the ability to place orders, and from the order line that has a relationship with the products that it assembles for different orders that customers place, can store different orders that are placed.

3.2 Drawing a Usage Map

The Usage Map below demonstrates the observations mentioned above.



4 Problem: Creating a join index

4.1 Explanation

To appropriately draw a join index, we want to keep in mind the quantities that we would like to combine from the given data. That is, from the given SQL tables Customer$_T$ and Order$_T$, we know that the relevant columns that we would want to combine to form the desired join index, from SQL, which can be achieved with code of the form

```
    SELECT * FROM Customer_T

FULL OUTER JOIN Order_T

ON Customer_T.CustomerID = Order_T.CustomerID
```

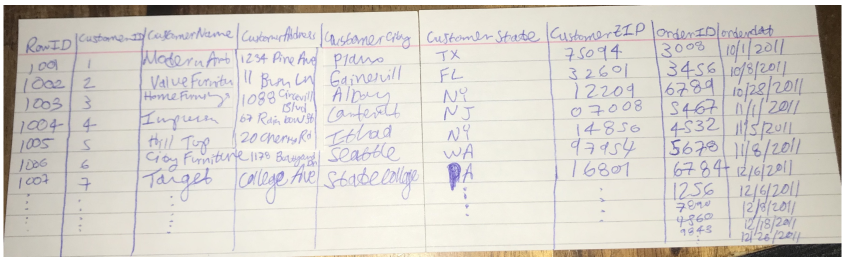
Alternatively, because I remember that the above code reproduced the customerId at the beginning and end of the table created, a similar SQL command, namely

```
    SELECT * FROM Customer_T

FULL INNER JOIN Order_T

ON Customer_T.CustomerID = Order_T.CustomerID
```

allows us to creat a table of the form below. Again, my apologies but I ran into issues with my laptop that I emailed Yang about, but the join index from the given data **must** include 11 columns. I was not completely sure if the number of elements should exactly match up, as the number of elements from my join index below is not equal to the number of elements in the other row, which are organized in this way after matching up the corresponding rows with the same CustomerID and same CustomerName have been joined from the Customer$_T$ and Order$_T$ tables.

4.2 join index illustration

In particular, from the description of the join index and the SQL code that we have offered above, we observe that the join index that we have created is specifically what we want, in the fact that as performing the join index operation, it is clear that the table that we have obtained has the ID of each customer, in addition to the row number detailing the order of each customer, as well as the customer number that the customer has given after placing the order. I was not sure whether Dr. Yang wanted us to mimic the output from SQL or to just include it. Either way, I have included the code that could be used to create such a table in SQL, from which we would obtain an appropriate join indexing of the given data, because, from either the customer number of the ID that a customer provides, we are able to categorically organize, and join, the relevant information together.



RowID	CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerZIP	OrderID	orderdat
1001	1	Modern Art	1234 Pine Ave	Plano	TX	75094	3008	10/1/2011
1002	2	Value Furnitu	11 Bush Ln	Gainesvill	FL	32601	3456	10/8/2011
1003	3	Home Furnishings	1088 Cirevill Blvd	Albany	NY	12209	6789	10/28/2011
1004	4	Impressi	67 Rainbow St	Canterelt	NJ	07008	5467	11/1/2011
1005	5	Hill Top	20 Cherry Rd	Ithaca	NY	14850	4532	11/5/2011
1006	6	City Furniture	1178 Burtyard Dr	Seattle	WA	97954	5678	11/8/2011
1007	7	Target	College Ave	State College	PA	16801	6784	12/6/2011
⋮	⋮	⋮			⋮	⋮	1256	12/6/2011
							7890	12/8/2011
							4860	12/18/2011
							9843	12/26/2011
							⋮	