CS 550 Project Assignment

Part1: Database design

Consider the following "supply chain" information system description. This information system will support a collaborative supply chain in which items of two types (materials, products) move among business entities of various kinds (suppliers, manufacturers, shippers, end-customers):

- Manufacturers use items of materials to manufacture items of products for customers.
- Suppliers supply items of materials to manufacturers; they also supply items directly to customers.
- Shippers (e.g., UPS, Fedex etc) move items from one business entity (supplier, manufacturer, customer etc) to another.

All items have a *unique id*, and also have a *weight*.

Every business entity (suppliers, manufacturers, shippers, customers) is uniquely identified by its *id*, and also has an *address, phone, web location,* and *contact information*. A business entity also has a *shipping location* (used by Shippers for shipping orders).

Every product item (e.g., a table) has an associated list of required material items, along with the quantity of each, needed to manufacture 1 unit of that product item. For example, a table product item requires 1 table top material item, 4 leg material items and 8 screw material items.

Suppliers supply items at a *price per unit*. Different suppliers may supply the same item at a different price per unit. In addition, suppliers may also offer a *volume discount*, a discount applied to the dollar amount (computed based on price per unit) being ordered from that supplier. Volume discount is applied in a graduated manner, with the discount increasing for the parts of an order above a certain bound. Volume discount is described as a deduction applied to the amounts of an order between predetermined bounds. grey denotes that what cant be represented in the ER diagram

Manufacturers produce product items. Whenever they produce items, this has an associated *setUpCost*, plus a product *cost per unit*. When they sell

their items to customers, manufacturers may offer *volume discounts* to customers in the same way suppliers apply volume discounts.

Shippers move items (all types of items). Each shipper's shipping rates (*price per pound*) are determined separately for each (*source, destination*) pair, where sources and destinations are the shipping locations of business entities (suppliers, manufacturers, end-customers). Total pricing for a shipment is based on a) the total weight of shipment in pounds times *price per pound*, b) a volume discount applied on the total dollar amount of that shipment, c) a minimum price for any shipment (from a given souce to a given destination) called the *minPackagePrice*.

Customers have a demand for some *quantity* of various items.

What will the orders be represented as in the ER diagram??

Orders are recorded separately for shipping, manufacturing and supply.

- Shipping orders capture information about a shipper, sender, and receipient (who are all business entities) and the item being shipped. They also record the *quantity* of the item requested to be shipped in that order.
- Manufacturing orders capture information about a manufacturer, a manufactured item and the quantity requested to be manufactured in that order.
- Supply orders capture information about a supplier, item and the quantity requested to be supplied in that order.

Assignment:

1. Create an ER diagram for this information system. Specify all integrity constraints. If some information in the description cannot be modeled, please describe it. If some needed information is missing from this description, please suggest additional assumptions.
2. Compare the relational tables that would be generated from your ER design to the tables below (in Part 2). Please write out some of your thoughts discussing how these sets of tables differ from each other.
3. Submit your diagram and discussion as a PDF on blackboard.

Important note: In Part 1, DO NOT try to generate an ER diagram by reverse-engineering the tables in Part2! Develop your ER diagram from the textual description alone.

Part 2: Database implementation (NOTE: for your Project Part 2 (= HW3) you will only need to submit the script file ***"queries.sql"***; you do not need to submit a script creating the following tables, that is provided).

A. Consider the following tables (see file *create_empty_tables.sql* and *sample_db.sql).* For foreign key constraint definition, note that product items, and material items are both items (see table *items*). Note also that suppliers, manufacturers, shippers, customers, senders and recipients are business entities (see table *busEntities*).

1. ***items(item,unitWeight)***: item has *unitWeight*. Assume that all items, including supplied and manufactured items, must appear in this table.
2. ***busEntities(entity, shipLoc, address, phone, web, contact)***: every business entity has *shipLoc* (used by shipping companies), *address*, *phone*, *web* link, and *contact* info. Assume all business entities, including suppliers, manufacturers and customers, must appear in this table.
3. ***billOfMaterials(prodItem, matItem, QtyMatPerItem):*** to produce 1 unit of *prodItem*, manufacturers need *QtyPerItem* units of *matItems* (e.g., if a table (prodItem) needs 4 Legs (matItems), QtyPerItem = 4)
4. ***supplierDiscounts(supplier, amt1, disc1, amt2, disc2):*** supplier gives discount *disc1* for purchase $ amount between *amt1* and *amt2*, and *disc2* for purchase amount above *amt2*. Note that discounts will be expressed as fractions rather than percentages, e.g., 0.15 rather than 15 (%). use float value
5. ***supplyUnitPricing(supplier, item, ppu):*** *item* supplied by *supplier* has *ppu* (price per unit)
6. ***manufDiscounts(manuf, amt1, disc1):*** manufacturer *manuf* gives discount *disc1* for manufacturing cost in excess of *amt1* of the base cost, which is computed according to *manufUnitPricing* table - see below.
7. ***manufUnitPricing(manuf, prodItem, setUpCost, prodCostPerUnit):*** For manufacturing of *prodItem* by

manuf, the manufacturer base cost is computed as *setUpCost* plus the *prodPricePerUnit* times the *qty* of the produced *prodItem*.

8. ***shippingPricing(shipper, fromLoc, toLoc, minPackagePrice, pricePerLb, amt1, disc1, amt2, disc2):*** The shipping cost for a *shipper* from *fromLoc* to *toLoc* is computed as follows:
    a. determine the total weight of all items shipped from *fromLoc* to *toLoc* (by all senders at fromLoc to all recipients at toLoc)
    b. base cost: is computed based on total weight of shipment and *pricePerLb*
    c. discounted cost: then for amount between *amt1* and *amt2*, *disc1* is applied; and to the amount above *amt2*, *disc2* is applied
    d. total cost: the maximum of *minPackagePrice* and the discounted price.
9. ***customerDemand(customer, item, qty):*** The demand by customer is *qty* units of *item*; note that items may come from any combination of manufacturers and/or suppliers.
10. ***supplyOrders(item, supplier, qty):*** *qty* units of *item* were ordered from supplier
11. ***manufOrders(item, manuf, qty):*** *qty* units of *item* were ordered to be produced by manuf
12. ***shipOrders(item, shipper, sender, recipient, qty):*** *qty* units of *item* were requested to be shipped by *shipper* from *sender* to *recipient*. Note senders and recipients are business entities such as suppliers, manufacturers and customers.

B. Implement the following SQL views by populating the template "queries.sql". This SQL file must have your view definitions and nothing else. For Part 2 of the project (=HA3) submit ***"queries.sql"*** ONLY to Blackboard. Make sure that "queries.sql" compiles/runs w/out errors.

1. ***shippedVsCustDemand:*** For every (customer, item) pair in customerDemand, compute the total qty of this item shipped to this customer, along with the demand qty. Note that the items may come from manufacturers and/or suppliers. The resulting schema should be (customer, item, suppliedQty, demandQty). Order the result by customer, item.
2. ***totalManufItems:*** For every (product) item in manufOrders, compute the total qty of this item ordered from all manufacturers. The resulting schema should be (item, totalManufQty). Order the result by item.
3. ***matsUsedVsShipped:*** For every manuf in manufOrders, and matItem used by this manuf (i.e., manuf is ordered a prodItem that requires a matItem according to billOfMaterials) compute:
   - ***a.*** - the total qty of this matItem requiered to produce all (product) items ordered from this manuf,
   - ***b.*** - the total qty of this matItem shipped by all shippers to this manufacturer

   The resulting schema should be (manuf, matItem, requiredQty, shippedQty). Order the result by manuf, matItem.
4. ***producedVsShipped:*** For every (item, manuf) in manufOrders compute the total qty of this item shipped out from this manuf (by all shippers to any recipient), along with the total qty of this item ordered from this manufacturer (in manufOrders). The resulting schema should be (item, manuf, shippedOutQty, orderedQty). Order the result by item, manuf.
5. ***suppliedVsShipped:*** For every (item, supplier) in supplyOrders compute the total qty of this item shipped from this supplier (by all shippers to any recipient), along with the ordered qty of this item. The resulting schema should be (item, supplier, suppliedQty, shippedQty). Order the result by item, supplier.
6. ***perSupplierCost:*** For each supplier in supplierDiscounts, compute the total cost of items supplied by this supplier (according to supplyOrders). Clarification: to compute the perSupplierCost: (1) express the cost before discount (summation of ppu*qty over all items

ordered from that supplier); (2) apply the discount. For example, assume that CostBeforeDiscount for supplier 17 is $2500; and amt1=1000, disc1=0.1, amt2 = 2000, disc2 =0.2. The cost after discount will be (1000 + (2000- 1000)*(1-0.1) + (2500-2000)*(1-0.2) = 2300. The resulting schema should be (supplier, cost). Order the result by supplier.

7. ***perManufCost:*** For each manufacturer in manufDiscounts, compute the total manufacturing cost of all items produced by this manufacturer (according to manufOrders). The resulting schema should be (manuf, cost). Order the result by manuf.
8. ***perShipperCost:*** For each shipper in shippingPricing, compute the total shipping cost of this shipper. The resulting schema should be (shipper, cost). Order the result by shipper.
9. ***totalCostBreakdown:*** Compute the total supply cost, manufacturing cost, and shipping cost. The resulting schema should be (supplyCost, manufCost, shippingCost, totalCost).
10. ***customersWithUnsatisfiedDemand:*** Find customers whose demand is NOT satisfied, i.e., their demanded quantity is not satisfied by all the quantities of shipped items. The resulting schema should be (customer). Order the result by customer.
11. ***suppliersWithUnsentOrders:*** Find suppliers whose orders are not fully shipped out. The resulting schema should be (supplier). Order the result by supplier.
12. ***manufsWoutEnoughMats:*** Find manufacturers who do not have enough materials to produce ordered product quantities, i.e., not enough materials were shipped to them. The resulting schema should be (manuf). Order the result by manuf.
13. ***manufsWithUnsentOrders:*** Find manufacturers whose orders are not fully shipped out. The resulting schema should be (manuf). Order the result by manuf.

C. How to test and debug your views (and know your score when you're done)?

1. As described in "install_instructions.txt", install:
    a. python3
    b. cx_oracle Python module
    c. cx_oracle client
    d. recommended: ATOM studio (alternatively you can use any other IDE which has syntax binding for SQL and JSON.
2. Download and unpack the archive ***ha3_template.zip*** It has a number of files. If you use ATOM studio (which I recommend), under "File", choose "Add Project Folder" and select ***ha3_template***. If you don't use ATOM studio, you can use any IDE that has syntax binding for SQL and JSON (Java Script Object Notation). Fill in the file "credentials.py" with your Oracle DBMS credentials.
3. Consider the input database data captured in the JSON file "testDBs/sampleDB.json". The mapping of the input to relational tables is self-explanatory. To test/debug your SQL script in "queries.sql" on an input DB associated with "sampleDB.json", do the following (from command line):
    a. Duplicate file “queries_template.sql” to create "queries.sql" in the directory ***solution_sql***.
    b. In command line, make ***solution_sql*** your current directory.
    c. Run in command line: >> python3 test_queries_main.py (note some systems use "python" or “py” etc. rather than "python3".)
    d. "out.json" will contain the answers to all your views represented in JSON. The Python script "test_queries_main.py" generates tables, populate tables with data from "sampleDB.json". Then it runs your script "queries.sql" and extract output into "out.json" file.
    e. You can compare your result "out.json" with the correct data results in the file "sampleDBanswers.json", which is provided in the HA4 folder.
    f. You can play and modify "sampleDB.json", and see changes in the result.
4. You can also generate a report on correctness of your views (which is exactly the same report that will be used by the TA to grade your HA3 = Project Part 2) as follows:
    a. In command line, make ***solution_sql*** your current directory.
    b. Run in command line: >> python3 produce_answers_main_sql.py
    c. This generated the file "answers.json" in the current directory, which contains the result of your views for test databases "db1.json", "db2.json", ... in the directory ./testDBs
    d. Run in command line:

```
>> python3 report_unordered.py
```

e. This generated the file "report.json" that tells you how many correct views you have, and for each incorrect view, will list databases for which your answer is incorrect. You may want to inspect this file in ATOM studio, to find a db JSON file for which your views generated an incorrect answer.