Homework 5
1.264, Fall 2006
Database implementation for chemical manufacturer
Due: Thursday, October 19, 2004

A. Overview

In this homework you will implement, in SQL Server Express 2005, the database that you designed in homework 4. You may use the homework 4 solutions as the guide for your database design. You may also use your homework 4 response; correct any errors in your data model before you begin building the SQL Server database. The dataset is on the course server; see the Web site for details.

B. Database implementation overview

The (logical) data model and (physical) database can be the same. If you make some changes between the data model and database implementation, explain them. The SQL Server database that you create must:

- Contain relationships among tables, to allow queries across all the data to be made conveniently.
- Contain referential integrity, to ensure correctness when records are added, deleted or modified.
- Contain appropriate keys, both primary and foreign.
- Be appropriately structured to support the operations required. The entities and attributes must be based on the data model from homework 4.
- Support appropriate cascading delete/update rules for entities with foreign keys.

You must load your database with the data from the dataset provided on the course Web site. You will need to write a set of SQL queries to load the tables in your database from the original datasets provided to you. The dataset contains 6 tables, but they are not in normal form, nor do they have relationships among them.

Do not build any data entry forms or menus. We will create these in the following homework sets. Also, you don't need to create indexes for performance in this homework

A set of suggested steps to develop the database are given below. You are free to approach it in other ways. The steps generally mirror those used to build the data model in homework 4.

C. Database implementation

NOTA BENE: Constructing a database in SQL Server is different than how Visual Paradigm builds the data model. While Visual Paradigm automatically inserted the foreign keys in child tables, SQL Server and other databases don't do that. You must build tables with all attributes, including foreign key attributes, and then set the relationships.

Also: You must REFRESH after adding each table or relationship in the SQL Server client. Otherwise the new table or relationship may not appear. You must SAVE each new table after each modification before continuing.

Last: SQL Server allows NULLs in columns by default. It's ok to accept the default in this homework, to save some time, but it's a bad idea to do this in general.

1. Chemical Product.
    a. Create the ChemicalProduct table by doing a SELECT DISTINCT column1, column2, … INTO ChemicalProduct from CHEMICAL_RAW. Use the client; type the SQL into the query window. (Hit 'New Query' on toolbar to get the window.) Select the correct columns from CHEMICAL_RAW; use your data model for guidance.
    b. Set the primary key. In the object explorer (left part of client), expand Tables, then right click on the ChemicalProduct table and select 'Modify'. In the window, select the attribute(s) to be the primary key, and right click and select 'Set Primary Key'. You can select more than one attribute; use the shift or control key. Save the table after setting the primary key.
    c. If SQL Server doesn't let you set the primary key, there are duplicates in your primary key column. Usually this means that you've forgotten the 'DISTINCT' keyword. We give you a SQL query to look for duplicates at the end of this homework to help you discover any errors.
    d. (If you prefer, you may write a CREATE TABLE statement that defines the primary and foreign keys, and then write an INSERT statement to load the data. This avoids having to use the client for this.)
2. Chemical Name.
    a. Create the ChemicalName table by doing the appropriate SELECT DISTINCT from CHEMICAL_RAW. Use the client.
    b. Set the primary key for this table, and for each table as you build it. Use the client. For composite keys, highlight all the attributes and then hit the key icon.
    c. Also, create relationships as you create each table, to make sure that you've created the keys and attributes correctly. Create the relationship between ChemicalName and ChemicalProduct now. Use the client; select ChemicalName, right click, select 'Relationships'. Click on the … after Tables and Column Specification; using the dropdown boxes, select UN Number as the primary key in ChemicalProduct to define the foreign key UN Number in ChemicalName. Hit 'Save' when done.
    d. See the final step in the homework for how to resolve problems that may occur. Also, create Diagrams in the client (right click on your database name, choose 'Database Diagrams', 'New Database Diagram', add the tables you want in the diagram, and the diagram will show you all the relationships). Do this frequently as you build the database. The key symbol is the '1' end of the 1-to-many relationship.

3. Create the ChemicalClass or HazClass table. Use SELECT DISTINCT again to get just the list of unique HazClass values (there are 20). Set the primary key and relationship to ChemicalProduct.
    a. You'll need to go to the ChemicalProduct table to set the relationship with ChemicalClass.
    b. You'll need to add the 'description' attribute manually, using 'Modify' on the ChemicalClass or HazClass table.
4. Create a VehicleType table, with passenger and cargo as the two rows. The SQL commands are:

```
CREATE TABLE VehicleType (
VehicleType NCHAR(10) NOT NULL PRIMARY KEY)

INSERT INTO VehicleType
VALUES('Passenger')

INSERT INTO VehicleType
VALUES('Cargo')
```

5. Create a QuantityLimit table by doing a "SELECT DISTINCT .. INTO" from CHEMICAL_RAW. Create the table by selecting the passenger quantities in this step. Use the client.
    a. Then use the client 'Modify' to add a VehicleType column to QuantityLimit. Make it NCHAR(10) to match the VehicleType table. Set the vehicle type to be passenger in these rows by an UPDATE statement in the client.
    b. Rename the quantity column QuantityLimit instead of PASS_QTY_1, since it will hold both passenger and cargo quantities. Use the client 'Modify'
    c. Don't set a primary key yet; wait until step 6.
6. Next, insert the cargo quantities into the QuantityLimit table. Write an INSERT statement from CHEMICAL_RAW, and then set the vehicle type to be cargo for those rows. The INSERT statement is:

```
INSERT INTO QuantityLimit(UN_NUMBER, QuantityLimit,
VehicleType)

SELECT DISTINCT CHEMICAL_RAW.UN_NUMBER,
CHEMICAL_RAW.CARG_QTY_1

FROM CHEMICAL_RAW
```

    a. Set the primary key and relationship for the QuantityLimit table. Move the columns for the primary key together in 'Design Table' by grabbing the columns at their left hand edge and dragging them to change their order.
    b. Remember to create Diagrams periodically to ensure that your relationships are correct.
7. Create the EmergencyResponse table from EMERGENCY_DETAIL_RAW, using SELECT DISTINCT. Set the primary key. Create relationship to ChemicalProduct.

a. First you'll need to create a temporary EmergencyResponse table with UN numbers and guide numbers. All pairs of UN numbers and guide numbers will be distinct, but there will be duplicate guide numbers.

b. You'll need to add a guide number attribute to ChemicalProduct, then write an UPDATE statement (on a join with EmergencyResponse) to put the guide number into the ChemicalProduct table. Make sure the data type and length (nvarchar(4) or int, for example) is the same in both tables.

c. Then build a list of distinct guide numbers as the final EmergencyResponse table; the guide numbers can now be the primary key. Select them from EMERGENCY_DETAIL_RAW, not EMERGENCY_RESPONSE_RAW, since there are some instructions not currently used.

d. Then, in EmergencyResponse, set the guide number as the primary key, build the relationship, draw the diagram, as usual.

8. The EmergencyDetail table can use the 'raw' table; copy or rename the 'raw' table in the client and set the primary key.
9. Create the Isolation table from SPILL_RAW, using SELECT DISTINCT. Set the primary key and relationship.
10. Create the Mode table. There are 4 modes (Highway, Rail, Air, Water); you can enter them by hand. These 4 entries must match the names you use in step 13.
11. Rename or copy the CARRIER_RAW table as the Carrier table. Put a primary key on the Carrier table.
12. Create the CarrierMode table. This is the associative table between Carrier and Mode that lists all the modes a carrier provides. This may take a little effort.

    a. First, create 4 attributes in addition to the single Mode attribute in the Carrier table; call them HighwayMode, RailMode, WaterMode and AirMode.

    b. Use wildcard matching to convert, for example, '1___' to a 'Highway' in HighwayMode. And '_1__' to a 'Rail' in RailMode. Etc. Use the LIKE keyword with the wildcard.

    c. Now, you have 4 mode columns in the Carrier table, with the mode values in them for the carriers indicating the modes they offer.

    d. You can now create the CarrierMode table. This may also take a little effort. You may SELECT DISTINCT from the HighwayMode attribute of the Carrier table into the CarrierMode table. You will then need to INSERT INTO CarrierMode SELECT DISTINCT… for RailMode,WaterMode and AirMode into the CarrierMode table. (This is similar to what you did with Quantity Limits.) This table has a composite primary key. You should name the column Mode, not HighwayMode, in this table.

    e. Delete the original Mode column if you wish, or keep it to check correctness while you do the homework and delete it at the end. Be sure to specify the composite primary key to the CarrierMode table.

13. Create the ChemicalGroup table by hand. Enter groups A, B, C, D and E and a brief description of each (A= radioactive, B= explosive, C= fluids, D= flammables, poisons, E= other). Set the primary key.
14. Add a Chemical Group column to the ChemClass or HazClass table that shows which chemical classes are in which chemical group; enter the data by hand. Right click on HazClass or ChemClass, select Open->Return all rows, and you can enter the data by hand in the grid that displays. See homework 2 for the mapping. (1.x=B,2.x=C, 3=C, 4.x=D, 5.x=D, 6.x=D, 7=A, 8=C, 9=E)
15. In a similar fashion to the CarrierMode step above, create the CarrierGroup table that indicates the chemical groups that each carrier can handle.
16. Create a Customer table by renaming or copying the CUSTOMER_RAW table
    a. You do not need to create CustomerMode or CustomerGroup tables, since this is tedious. Rename or copy the CUSTOMER_RAW table to be the Customer table and set its primary key.
    b. Our Web-database application will not implement the customer checking requirements, because it would duplicate the carrier checking.
17. Create the OrderHeader table. It will initially contain no data. Set its primary key. (Don't call this table Order, since 'Order' is a SQL reserved word, as in 'ORDER BY'.) Create foreign keys (by creating relationships) for the distributor certificate number (distributor ID), carrier certificate number (carrier ID) and mode in the OrderHeader table.
    a. You must make the data types and lengths of foreign keys the same as they are in the parent table (or you won't be able to create the relationships).
18. Create the OrderDetail table. It will initially contain no data. Set its composite primary key. Set a foreign key for chemical product (UN number). Again, make the data types and lengths of foreign keys the same as they are in the parent table.
19. After you've built all the tables, you need to verify the relationships between them. Build the relationships as you go.
    a. Follow the data model from homework 4. Build each relationship shown in that model.
    b. If a relationship fails to build, check the following items:
        i. Does each table have the correct primary keys?
        ii. Are the data types and lengths the same?
        iii. Have you selected the correct foreign key for the relationship?
        iv. If you cannot set the primary key for a table, use the Find duplicates SQL code below to see if there are duplicates. There shouldn't be any (we've fixed the original data, which did have problems), so this means that you've either built the table incorrectly (perhaps forgetting the DISTINCT keyword) or that you've defined the primary key incorrectly (perhaps not setting it as a composite key). Primary keys cannot have NULLs.

v. If you cannot set a relationship:

1. Check that the data types are the same in both tables.
2. If SQL Server says it cannot build the relationship, use the Find unmatched SQL code below to find the records that have no entry in the parent table. This can help decipher what is wrong.
3. If there is a composite key, have the keys in the same order in both tables.

This completes your database. Please leave the original 'raw' tables and any temporary tables you've created in the database so that we can check your queries. Normally you would delete the 'raw' and temporary tables at this point, of course, and just use the new database you've created. We will delete the 'raw' and temporary tables before using the database in later homework sets.

D. Queries

Now that you've built the database, implement 4 queries to return data from it. You cannot use the original datasets; use the tables and relationships you have built.

1. Display the name of all carriers that can handle chemical group A by air.
2. Display the count of carriers by mode (highway, rail, water, air) that can handle group D.
3. Display the cargo packaging quantity and UN number, for all UN numbers that are shipping hazards or water reactive (from the Isolation table).
4. Display the chemicals (UN numbers and all names) with no quantity limits in passenger vehicles.

E. Assignment

1. Turn in a Word file briefly describing any issues or comments in building your database and queries. This should contain and issues with any tables that you built in part C of this homework, including temporary tables. It can be very brief, less than a page.
2. Turn in the queries that you wrote to create the database, saved in the client as a .sql or .txt file. When you save your queries, please number them from 1 to 18, to match the steps in part C of this homework. If you need more than one query for a step, number them 1a, 1b, 1c, etc. Thus, when you write a SQL statement, please comment it as '--7 Create Emergency Response table', for example. You don't need to write queries for tables you created from scratch, such as OrderHeader, etc. If you use your own sequence of queries, please number them sequentially even if they don't match the homework sequence.
3. Turn in the queries from part D in the assignment. Please number them D1 through D4 and place them in a .txt or .sql file. They can be in the same file as your earlier queries.

4. Copy the actual database file (the .mdf file that SQL Server creates) to the 1.264 course Web server. The file will be quite large, and the 'Shrink' utility appears suspect, so use it at your peril. See recitation notes and the TAs for details.

5. Place the answers to parts 1, 2 and 3 above in a single zip file and upload it to the the MIT Server Web site. See the TAs if you need help creating a zip file. Copy the .mdf database file to the course Web server; see recitation notes and the TAs for details.

F. SQL Code Examples

1. Find duplicates. To find duplicate certificate numbers in the carrier table:

   ```
   SELECT Carrier.CERT_NO, Carrier.COMPANY
   FROM Carrier
   WHERE (Carrier.CERT_NO In (SELECT CERT_NO FROM Carrier As Tmp
   GROUP BY CERT_NO HAVING Count(*)>1 ))
   ORDER BY Carrier.CERT_NO;
   ```

   Modify this for any table in which you wish to check the intended primary key for duplicates.

2. Find unmatched rows between a parent and child table. For example, find rows in order detail for which there is no row in the order table (e.g., we have order detail for order 733 but there is no row 733 in the order header table):

   ```
   SELECT ManufOrderDetail.OrderNbr
   FROM ManufOrderDetail LEFT JOIN ManufOrder ON ManufOrderDetail.OrderNbr
   = ManufOrder.OrderNbr
   WHERE (ManufOrder.OrderNbr Is Null);
   ```

   Modify this to find rows in any child table for which there is no corresponding row in the parent table.