**Database Test 3 Review**

**Test is on Thur, Dec 1**

The two main topics on this class are more SQL and Transactions. As we did for the chapter 7 test, the SQL portion of the test will be open book. In all SQL questions, you are expected to know/use only the Sql Server form of the command. The Transactions part of the test will be closed book.

**SQL**

Chapter 7 – Review basic SQL syntax, but especially joins (section 7-7). One topic we did *not* cover previously was Recursive Joins (p. 303-304), so be sure to review recursive joins.

To handle these, select everything that you want. Do this using aliases. In your FROM clause, join the table to itself using aliases. Do this where the E.emp\_mgr = M.emp\_num, for example. You will have two different column names here since the table will have to have columns of different names.

Also, note what you have to group by in order to do this aggregate

SELECT brand\_id, Avg(prod\_price) AS avgprice

FROM lgproduct

GROUP BY brand\_id

Chapter 8

* Table 8.1: You need to know how to use the Old-Style inner join, JOIN ON, and LEFT, RIGHT, FULL JOIN syntax.
  + Just know this subset. The old-style is where you use a WHERE x = y type clause.
  + You use an outer join whenever you want all of the entries in one (or both) of the tables where things don’t match.
  + Use the inner join when you only want things that match.
  + These all use the join tables on condition syntax, which is exemplified below.

SELECT e.emp\_num, emp\_lname, emp\_fname, sal\_amount

FROM lgemployee e join lgsalary\_history s ON e.emp\_num = s.emp\_num

WHERE sal\_from = (SELECT Min(sal\_from)

FROM lgsalary\_history s2 WHERE e.emp\_num = s2.emp\_num)

ORDER BY e.emp\_num;

* 8-1a: skip
* 8-1b: skip
* 8-1c: skip
* 8-1d: e: review these sections
* 8-2: review all of this section. The completely new thing in Chapter 8 from this section is correlated subqueries.
  + Correlated subqueries are when you have subqueries that reference an attribute from the outer query via aliases. It is used almost always when you want an aggregate function result for every row of a table.
  + Subqueries are always on the right side of an operator.
  + If it returns a single value, it can be used wherever a single value is expected.
  + If it returns multiple values, consider using an IN statement. (IN subquery)

SELECT \*

FROM lgbrand

WHERE brand\_id NOT IN (SELECT brand\_id

FROM lgproduct);

* + A HAVING subquery can be used to restrict a GROUP BY. Pg 353. ANY and ALL are discussed too. This is an inequality statement as opposed to IN, which is an equality statement.
  + FROM subqueries must always have an alias for the name of the virtual table created. These are useful when you need to select from or join with a virtual table.

SELECT emp.emp\_num, emp\_fname, emp\_lname, emp\_email, total

FROM lgemployee emp JOIN

(SELECT employee\_id, Sum(line\_qty) AS total

FROM lginvoice i join lgline l ON i.inv\_num = l.inv\_num

JOIN lgproduct p ON l.prod\_sku = p.prod\_sku

JOIN lgbrand b ON b.brand\_id = p.brand\_id

WHERE brand\_name = 'BINDER PRIME'

AND INV\_DATE BETWEEN '01-NOV-13' AND '06-DEC-13'

GROUP BY employee\_id) sub

ON emp.emp\_num = sub.employee\_id

where total = (Select max(numSold) from (SELECT Sum(line\_qty) as NumSold

FROM lginvoice i join lgline l ON i.inv\_num = l.inv\_num

JOIN lgproduct p ON l.prod\_sku = p.prod\_sku

JOIN lgbrand b ON b.brand\_id = p.brand\_id

WHERE brand\_name = 'BINDER PRIME'

AND INV\_DATE BETWEEN '01-NOV-13' AND '05-DEC-13'

GROUP BY employee\_id) table1)

--What do you notice about using a FROM subquery? You will use it when you want the result of an aggregate function. So, you have to make a subquery and join with it so that you can get the Total column in the select statement.

--The nested subqueries in the WHERE requires you to get all of the totals again, and then select the max from that.

* + Attribute Subqueries are select queries in the select statements. These are useful when you want an aggregate function’s returned value to be displayed as an attribute. If you want the highest sales for each employee for example.

SELECT DISTINCT B.brand\_name, B.brand\_type, --I thought of a different way to do this one

(SELECT AVG(prod\_price)

FROM lgbrand B2 JOIN lgproduct P2 ON B2.brand\_id = P2.brand\_id

GROUP BY B2.brand\_name

HAVING B2.brand\_name = B.brand\_name) AS Average\_Price,

(SELECT SUM(L.LINE\_QTY)

FROM lgline L JOIN lgproduct P3 ON L.prod\_sku = P3.prod\_sku

GROUP BY P3.BRAND\_ID

HAVING P3.BRAND\_ID = B.brand\_id) AS Units\_Sold

FROM lgbrand B JOIN lgproduct P ON B.brand\_id = P.brand\_id

ORDER BY B.brand\_name;

* 8-3: You are not responsible for every possible SQL function. Just make sure you know how to use the specific functions from the sections below. You only need to know the SQL Server form of these functions:
* 8-3a: Convert, Year, Month, Day, Date(), DateDiff (pg 362)
  + Convert(VARCHAR(x), date\_value, format code) = returns date according to format code
  + Year(date\_value) = returns the 4 digit year value.
  + Month(date\_value) = returns the two digit month number from the date
  + Day(dat\_value) = returns the number of the day.
  + GETDATE() = returns the current system date. –SQL Server
  + DATEDIFF(datepart, startdate, enddate) = subtracts two dates expressed in the datepart
* 8-3b: Round
  + ROUND(numeric\_value, precision) = rounds to a specified precision, meaning number of decimal places. 0 is an integer.
* 8-3c: Upper/Lower case conversion functions
  + UPPER(string\_type) = returns as upper case
  + LOWER(string\_type) = returns as lower case
* 8-4: Review all of the commands in this section
  + Note: these only work if the two results are union-compatible, which means they have to have the same number of attributes and these all need to be of compatible data types.
  + UNION = combines two queries without duplicate rows

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER

UNION

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER\_2;

* + UNION ALL = combines two queries with duplicate rows

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER

UNION ALL – yields duplicate that union gets rid of

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER\_2;

* + INTERSECT = combines two queries with only duplicate rows in the result set

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER

INTERSECT

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER\_2;

* + EXCEPT = combines two queries removing the from the first result those found in the second. So only the results not in the second set appear.

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER\_2

EXCEPT

SELECT CUST\_LNAME, CUST\_FNAME

FROM CUSTOMER

ORDER BY CUST\_LNAME DESC;

* 8-5: You need to know what a View is/does, but you are not responsible for using SQL to create or access a View.
  + A view is an object that you create in the database. When you do this, you give it a SQL statement that is going to automatically be run whenever that view is accessed. It is always up to date because the query is run every time you do it.
  + It’s nice because you can select \* from a view to make the application code look nicer.
  + You can use the views to set different levels of permissions for tables so that only people with a need to know can see certain data.
  + These can be used to make reports.
* 8-6: skip, but you should know how to use the IDENTITY attribute in a CREATE TABLE statement.
  + Sets up auto-increment for your tables.

CREATE TABLE CUSTOMER (

CUST\_NUM INT IDENTITY(1000, 1) PRIMARY KEY,

CUST\_LNAME VARCHAR(20),

CUST\_FNAME VARCHAR(20),

CUST\_BALANCE FLOAT(8)

);

* 8-7: You need to know the following terms: PSM, Transact-SQL
  + PSM: Persistent stored modules are blocks of code containing standard SQL statements and procedural extensions that is stored and executed at the DBMS server. This is an umbrella term for SQL that you are sticking into the Database.
  + Transact-SQL: is SQL Server’s implementation of PSMs. This is Procedural Language SQL, which makes it possible to use and store procedural code and SQL statements within the database and to merge SQL and traditional programming constructs.
* 8-7a: Be able to answer the following:
  + What is a trigger/how is a trigger related to PSM?
    - A trigger is procedural SQL code that is automatically invoked by the RDBMS upon the occurrence of a given data manipulation event.
    - It is related to the concept of Persistent Stored Module because it is stored in the DB and executes “for” or “after” a certain event. (they are the same)
  + Why would you use a trigger, what sort of general things are they used for?
    - A trigger is used to prevent data from become stale or redundant. For example, if something is a primary key, and you delete it. You can have it cascade through the database so that the foreign key that pointed to it is also removed. You can do the same thing for update. You can also use these when you need to keep mathematical interactions between tables in sync and accurate.
  + What is the difference between a trigger and a stored procedure?
    - A trigger is automatically executed upon a certain condition. A stored procedure must be called by the user/application.

CREATE TRIGGER trg\_updatecustbalance ON INVOICE

FOR INSERT

AS

--declare variables to recieve values

DECLARE @cust\_num int;

DECLARE @inv\_amount float(8);

--Fill the variables with values

SELECT @inv\_amount = I.INV\_AMOUNT

FROM INVOICE I;

SELECT @cust\_num = I.CUST\_NUM

FROM INVOICE I;

--Update the customer table at the given cust\_num

--by adding the amount of the invoice to their balance

UPDATE CUSTOMER

SET CUST\_BALANCE = CUST\_BALANCE + @inv\_amount

WHERE CUST\_NUM = @cust\_num;

PRINT 'AFTER INSERT trigger fired.'

GO

* + You need to be able to read/explain trigger syntax of the form that we practiced on the homework. I will not ask you to write a trigger.
    - This will be closed book on paper. See the above Trigger example.
* 8-7b: Be able to answer the following:
  + What is a Stored Procedure/how is it related to PSM?
    - A stored procedure is a named collection of procedural and SQL statements. Like a trigger, these are related to the Persistent Stored Module concept because these are stored and executed at the DBMS server. These encapsulate business logic and lessen network traffic since they are stored at the server and compiled once. They are different from triggers because they must explicitly be called.
  + What are two advantages to using a stored procedure?
    - We can share the business logic between all applications because they are all stored in a database.
    - Store procedures are faster than the code executed from an application. You are missing the overhead of doing the front-end compiling on the SQL statement. Since you have already stored it, this compiling only happens once.
  + Why do some organizations *not* like to use stored procedures? (Hint: [here](http://www.seguetech.com/advantages-and-drawbacks-of-using-stored-procedures-for-processing-data/) is an overview of the tradeoff’s).
    - The pushback comes from two main sources. The first is: I don’t want to build my system so that I am tied down to a particular DB vendor.
    - The second is more of a philosophical issue: I don’t want half of my business logic in my source code and then the other half in the database. (The cool thing is that different applications can all access this stored procedure.)
  + Just as for triggers, I would not expect you to be able to write a stored procedure, but would expect that you could read a simple procedure of the type we used in the homework.
    - This will be closed book on paper.

USE Homework\_9 --Database name

GO

CREATE PROCEDURE prc\_invoice\_add @inv\_num int, @cust\_num int, @inv\_date date, @inv\_amount float(8)

AS

INSERT INTO INVOICE VALUES(@inv\_num, @cust\_num, @inv\_date, @inv\_amount)

GO

EXEC dbo.prc\_invoice\_add @inv\_num = 8006, @cust\_num = 1000, @inv\_date = '30-APR-16', @inv\_amount = 301.72

* 8-7c: skip this section
* 8-7d: skip this section
* 8-8: What is embedded SQL? What is Dynamic SQL? Which have we been using (via Dapper)?
  + - Embedded SQL: This is when the SQL is embedded in your code literally as if it were C# or something like that. This requires that the source code has to go to a SQL preprocessor that translates the SQL to the source language. Then, it is sent to the normal compiler. This was the standard way of doing it 15-20 years ago. (COBOL)
    - Dynamic SQL: When you are building up a string for a SQL call, this is considered dynamic SQL
    - We have been using Dynamic SQL via Dapper.
* Additional Sample SQL Practice Questions (Chapter 8): p. 427, problems 44-67
  + These are more reflective of what is going to be on the exam, so make sure that you do this. He is expecting us to be able to do bits and pieces of the hard ones. There is a diagram that describes this database. He has given us the SQL to create this database.
  + Make a mental note of the solutions used to solve the HW 10 problems.
* Misc: Know what the following constraints do: primary key, on update cascade, on delete cascade, foreign key constraint. Be able to give examples of operations that would violate each of these constraints.
  + Know what these are and why we would use them. This will be closed book as well.
  + Primary Key labels a particular attribute in a table as the Primary Key, which must be unique and cannot be null. It provides a name to the constraint and accepts one or more previously defined attributes.
  + The foreign key constraint provides a name to the constraint, accepts a foreign key as a parameter, and then tells what table and attribute in that entity are being referenced. The ON \* CASCADE statements are added after the reference.
  + On delete/update cascade is part of a foreign key constraint. This says that when there is a deletion or an update on the reference of the foreign key, the foreign key should also be deleted. This is good because foreign keys are not allowed to reference nulls.

CONSTRAINT PK\_LGPRODUCT PRIMARY KEY (PROD\_SKU),

CONSTRAINT FK\_LGBRAND\_LGPRODUCT FOREIGN KEY (BRAND\_ID)

REFERENCES LGBRAND (BRAND\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE

**Chapter 10 – Transactions**

1. P. 511-512, Review Questions: 1-13, 16, 17 part B (if given the isolation level, identify the type of ‘read’ allowed).
2. P. 512-514: Problems 1, 5, 6a, 6b, 7

General Questions

1. Which SQL statements are typically used within a transaction?
   * Select, update, insert, delete
2. In SQL Server, what statements are used to begin/end a transaction?
   * Begin Transaction
   * Commit or rollback;
3. If a program crashes in the middle of a transaction, what happens to the transaction in the database? Explain how the database would accomplish this.
   * If it crashes in the middle of a transaction, the transaction is automatically rolled back. When a transaction updates data, it first updates the transaction log. Then, it updates the database in memory only (in a buffer). Normally, it would then commit, and all of those changes will be written to the database at the next checkpoint (deferred).
   * When your program crashes, the OS will set the socket connection in such a way that the DBMS will know that there is nothing on the other end of the socket. Since It is dead, it knows that the transaction will never be committed. Then the DBMS can go through and clean things up.
   * It will use the transaction log to undo the changes in RAM.
   * Release the locks
   * Mark the transaction as rolled back.
   * This is in the example of a transient failure, as opposed to hardware.
4. What is concurrency control, exactly?
   1. Coordination of the simultaneous transactions execution in a multiuser database system.
5. Why do we want to execute transactions simultaneously, given the problems it can cause.
   1. Because doing so makes everything run SOOO much faster.
6. Explain a specific scenario that illustrates a Lost Update.
   1. This is where two different transactions are trying to update the same data. If they were allowed to execute such that both transactions read the data first, then one of the updates is not reflected in the result after both transactions are finished.
7. Explain a specific scenario that illustrates the Uncommitted data problem.
   1. This is when a transaction reads data that another transaction has updated but not yet committed. The problem here is that the first transaction could be rolled back. This will lead to results that do not reflect the rollback from transaction 1.
8. Explain a specific scenario that illustrates the inconsistent retrievals problem.
   1. This occurs when another transaction completes and changes the values of the first transaction as it is running. Multiple queries on the same thing are executed and they return different results. (Phantom read and Nonrepeatable read)
9. Explain the term “serializable schedule”
   1. Interleaved execution of transactions yields the same results as the serial execution of the transactions.
10. Assume pessimistic locking for the following:
    1. Describe the tradeoff’s involved in lock granularity
       1. The higher level of locking, the less likely we are to run into serializability issues and deadlocks, but there is a huge speed issue because transactions are always waiting on others to finish.
       2. Having a lower level of locking costs in memory and speed as well. There is a large amount of overhead associated with having locks on each level or field.
    2. Which locks are “compatible” with each other (i.e., can be held at the same time).
       1. Shared locks can be held at the same time. These are given when a transaction is attempting to read something from the database. There is no writing involved. If there were, an exclusive lock would be required.
       2. There is also the sense that a transaction could be setting a bunch of exclusive locks or shared locks on different sections of the database during the growing phase. They will all be of like type.
    3. What happens when a transaction attempts to gain access to a resource that is already locked?
       1. It has to wait. When the lock is released, then it will be collected by the next transaction.
       2. There are the wait/die and wound/wait schemes with time stamping.
    4. In what scenario are exclusive locks used? What about shared locks?
       1. Exclusive locks are used when something is being written.
       2. Shared locks are used when something is being read, and only read.
    5. True/False: 2 phase locking means that a transaction can never obtain a lock after releasing a lock.
       1. True. If we are in the shrinking phase, we cannot get new locks.
    6. How exactly does two-phase locking guarantee serializability?
       1. Since all transactions have to get all locks before they can do anything, everything will have to behave as if things were done in a one-after-another fashion. It is the fact that all locks are required before the action can be executed that makes this ensure serializability.
    7. Deadlocks:
       1. Describe a specific scenario between two transactions that illustrates a deadlock.
          1. Say transaction A accesses resource A then B.
          2. Transaction B then accesses resource B then A.
          3. These are happening concurrently. We now have a deadlock.
       2. What exactly is the difference between deadlock prevention and deadlock detection.
          1. Deadlock prevention refers to the practice of checking to see if a deadlock “could” happen before allowing any of the operations to be executed.
          2. Deadlock detection allows deadlocks to happen. Then it will kill them if a certain timeout is reached.
       3. Why do most production databases use detection instead of prevention?
          1. It is the cheapest option of the two. Prevention requires more resources to detect whether or not a deadlock will happen.
       4. What does the database do when it detects a deadlock?
          1. The DBMS chooses a victim, rolls that transaction back, and reschedules it.
11. Assume optimistic approach to concurrency control:
    1. True/False: optimistic approaches use no locking.
       1. True. It is not required.
    2. What technique does optimistic concurrency control use in lieu of locking?
       1. It uses an extra validation phase to ensure that the changes, which are written to a private copy of the database, are going to leave the DB in a consistent state before it writes the changes.
    3. In what scenario could optimistic methods be better than pessimistic, and why?
       1. It works well when using a read or query database system where there are few update transactions. If there are very few updates, it is much faster to not have to worry about locking while just reading data.
12. Consider the Transaction Isolation Levels (Table 10.15):
    1. What is a non-repeatable read? What is a phantom read?
       1. A non-repeatable read happens within a single transaction. A query is executed, and then executed again, both on a single row. When this happens, the returned values are not consistent between the two reads.
       2. Phantom reads are the same thing, but it affects multiple rows.
    2. What is the difference between Read Uncommitted, Read Committed, Repeatable Read, and Serializable?
       1. Read Uncommitted allows dirty reads, nonrepeatable reads, and phantom reads.
       2. Read Committed allows nonrepeatable reads and phantom reads
       3. Repeatable Read allows phantom reads
       4. Serializable allows none of these.
13. Database Recovery
    1. How do buffers, checkpoints, and deferred write techniques interact to ensure that recovery is possible?
       1. Buffers are located in memory. After changes are written to the transaction log, they are written into buffers. At a certain point, a checkpoint is reached and recorded in the log. When that happens, the deferred write technique kicks in and all of the writes in the buffers are written to disk. Deferred writing essentially means that things are not written to disk until a checkpoint is reached.
    2. Describe the recovery process for the deferred-write technique.
       1. The recovery process of the deferred-write technique is to go into the transaction log and see where the last checkpoint is. Then it scans to find a commit. If it does, then it executes all of the actions in the transaction log to that commit. It then repeats this process.
          1. For a write-through database, the process is essentially the same except for Rollbacks have to be done.
14. Describe four guidelines/practices that programmers can follow to minimize deadlocks.
    1. Team members should design their transactions so that tables are accessed in a consistent order
       1. This means we won’t get in a deadlock over any of the things we are doing in order.
    2. Don’t start a transaction then do user I/O or network I/O in the middle of it
       1. We don’t have any control over how long this might take.
    3. Keep transactions as short as possible
    4. If possible, lower isolation levels.
       1. This will reduce the number of locks that are maintained. It will speed things up and fewer things are locked.
15. Be able to read/explain Dapper code that uses transactions.
    1. We won’t write any of the code on the test. We have to be able to write and explain it.
    2. There is a using clause for the connection
    3. The connection then has to be opened
    4. There is a using clause for tran = conn.BeginTransaction()
    5. Then, in a try catch block, add the conn.Execute statements with the sql and transaction: tran as a named parameter. Put all of these in the try block.
    6. End the try block with a tran.Commit()
    7. In the catch block, you should write out your error message, and have another try catch block.
    8. This second try catch block holds the tran.rollback() statement in the try. It should catch the error if rollback has already happen with a //silent fail.