# Final Review

- Study all material specified on the review sheets for Test 1, Test 2, and Test 3.
- Study the material on normalization from Chapter 6 and be prepared to solve a problem on normalization.
- Study the material on concurrency and transactions from Chapter 10 and be prepared to show examples or respond to questions on terminology.
- Study in detail the additional examples provided in the Review section of the Content area of D2L.

# Study all material specified on the review sheets for Test 1, Test 2, and Test 3

# Test Review 1

- Terms to know
    - table - A database view of information arranged in a grid of rows and columns
    - query - A question presented in a way that allows the database to process & generate specific from one or more tables
    - relationship - table or set of tables.
    - primary key - A Primary Key is a column or a combination of columns that uniquely identify a record. Only one Candidate Key can be Primary Key.
    - referential integrity - a property of data stating that all of its references are valid. A foreign key MUST reference some primary key.
    - Design View - Design view gives you more control over the query you create than the Query Wizard does.
    - Datasheet View - A web datasheet view in an Access app displays online data arranged in rows and columns in a web browser. It can allow data edits or not, as you prefer.
    - SQL View - Whenever you create a query in query design, Access automatically creates the SQL query for you. This actually retrieves data from the tables. To see how your query is created in sql when you create it in query design, let us open your database.
    - Concatenated Key - a combination of columns the creates a unique reference to a record/row
    - Row - a single record
    - Column - attributes of a record/table
    - Candidate Key - A Candidate Key can be any column or a combination of columns that can qualify as unique key in database. There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as Primary Key.
    - Entity set - An entity set is a set of entities of the same type (e.g., all persons having an account at a bank).
    - attribute domain - In computing, the attribute domain is the set of values allowed in an attribute. AKA the range of values/options a column can take on.
    - relation - relations are represented by tables, where each row of a table represents a single tuple, and where the values of each attribute form a column.

## Dangers of redudant data

- Possible results of uncontrolled data redundancy
  - Poor data security
  - Data inconsistency
  - Data-entry errors
  - Data integrity problem
- Unnecessarily storing the same data at different places
  - Islands of information (i.e., scattered data locations)
  - Increases the probability of having different versions of the same data
- Data Anomolies
  - Develop when not all of the required changes in the redundant data are made successfully
    - Update anomalies
    - Insertion anomalies
    - Deletion anomalie

# Test Review 2

Create a View - put simply, its a saved select query

```
CREATE VIEW <table name> AS
...<sql query>...
```

Syntax

```
CREATE VIEW <table name> AS
SELECT <colname1>, <colname2>, ...
FROM <tablename1>
WHERE Country = "Brazil";
```

Example

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = "Brazil";
```

## Count()

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

returs the number of records meeting the condition from the table

## AVG()

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

returns the average of the records in the col meeting the condition from the table

## SUM()

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

returns the sum of the records in the col meeting the condition from the table

## MAX()

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

returns the greatest value of a record in a table meeting the condition from the table

## MIN()

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

returns the greatest value of a record in a table meeting the condition from the table

## GROUP BY

- The GROUP BY clause groups records into summary rows.
- GROUP BY returns one records for each group.
- GROUP BY typically also involves aggregates: COUNT, MAX, SUM, AVG, etc.
- GROUP BY can group by one or more columns.

Syntax

```
SELECT column-names
FROM table-name
WHERE condition
GROUP BY column-names
```

Example

```
SELECT COUNT(Id), Country
FROM Customer
GROUP BY Country
```

## HAVING

The SQL HAVING Clause The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

## ORDER BY

ORDER BY keyword is used to sort the result-set in ascending or descending order.

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example

```
SELECT * FROM Customers
ORDER BY Country;
```

Example, desc

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Example, many

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

## SQL Datatypes

7 basic data types:

1. Exact Numbers
2. Approximate Numbers
3. Unicode strings
4. Binary strings
5. Date and Time
6. Other Data types
7. Character Strings

- a `BigInt` = 64 0's and 1's = 2^64
- an `Int` = 32 0's and 1's = 2^32
- an `SmallInt` = 16 0's and 1's = 2^16
- a `TinyInt` = 8 0's and 1's = 2^8
- an `nchar` = 1 character
- an `nchar(20)` = 20 characters
- `nvarchar(max)` = 2GB of unicode
- `binary` = 1 character
- `binary(20)` = 20 characters
- `varbinary(max)` = up to 2GB of storage
- `varchar(max)`

| Exact Numbers | Approximate Numbers | Unicode strings | Binary strings | Date and Time | Character Strings |
|---|---|---|---|---|---|
| int(4) | float(8) | nchar | binary | char(140) | char(140) |

| Exact Numbers | Approximate Numbers | Unicode strings | Binary strings | Date and Time | Character Strings |
|---|---|---|---|---|---|
| Money(8) | real | nchar(20) | binary(20) | char | |
| | | nvarchar(max) | varbinary(max) | varchar(max) | |

# Test Review 3

# COMPUTER SCIENCE 4400 REVIEW FOR TEST 3

# 1. Be able to write a simple SQL trigger for a specified table.

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Example

```
-- Create a trigger that prevents the deposit amount in the Rental table from
being less than half of the deposit amount in the Rental table for the property
that is involved.

CREATE TRIGGER
  t7
  ON RENTAL
  FOR INSERT, UPDATE

  AS

  DECLARE @V1 FLOAT
  DECLARE @V2 FLOAT
  SELECT @V1 = INSERTED.RENT_DEPOSIT FROM INSERTED
  SELECT @V2 = DELETED.RENT_DEPOSIT FROM DELETED

  IF @V1 < (@V2 * 0.5) -- less than half
  BEGIN
    ROLLBACK TRANSACTION
    PRINT 'DEPOSIT MUST BE AT LEAST HALF OF RENT'
  END
```

# 2. Be able to write a simple SQL stored procedure to solve a specified problem. Be sure to study the use of parameters.

```
CREATE PROCEDURE procedure_name
  @variable_name type,
  @variable_name type
  .
  .
  .
AS
sql_statement
GO;
```

```
EXEC procedure_name;
```

# 3. Be able to write SQL statements to solve specified problems that involve creating tables, updates, inserts, and deletes.

1. Create Table

```
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .
  .
  .
);
```

Example

```
CREATE TABLE Persons (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
```

```
  City varchar(255)
);
```

2. Update

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
WHERE CustomerID = 1;
```

3. Insert

```
INSERT INTO table-name (column-names)
VALUES (values);
```

Example

```
INSERT INTO Customer (FirstName, LastName, City, Country, Phone)
VALUES ('Craig', 'Smith', 'New York', 'USA', 1-01-993 2800)
```

4. Delete

```
DELETE FROM table
WHERE
    condition;
```

```
DELETE FROM employees
WHERE employeeID = 3;
```

# 4. Be able to write SQL statements to modify table structure using the ALTER TABLE command.

1. ALTER ADD

```
ALTER TABLE "table_name"
ALTER COLUMN "column_name" "New Data Type";
```

Example

```
ALTER TABLE Customers
ADD Email varchar(255);
```

2. ALTER DROP

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example

```
ALTER TABLE Customers
DROP COLUMN Email;
```

3. ALTER MODIFY

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Example - change date type

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

# 5. Be able to write SQL code to construct views.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

# 6. Be familiar with the concept of a transaction as discussed in Chapter 10.

- Transaction Control - The following commands are used to control transactions.
    - COMMIT – to save the changes.
    - ROLLBACK – to roll back the changes.
    - SAVEPOINT – creates points within the groups of transactions in which to ROLLBACK.
    - SET TRANSACTION – Places a name on a transaction.

Example

```
DECLARE @TranName VARCHAR(20);
SELECT @TranName = 'MyTransaction';

BEGIN TRANSACTION @TranName;
USE AdventureWorks2012;
DELETE FROM AdventureWorks2012.HumanResources.JobCandidate
  WHERE JobCandidateID = 13;

COMMIT TRANSACTION @TranName;
GO
```

Example - rollback a delete

```
BEGIN TRANSACTION;
DELETE FROM HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
COMMIT;
```

# 7. Be able to list and describe the five properties of a transaction.

- Atomicity – ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.
- Consistency – ensures that the database properly changes states upon a successfully committed transaction.
- Isolation – enables transactions to operate independently of and transparent to each other.

- Durability – ensures that the result or effect of a committed transaction persists in case of a system failure.
- Serializable - Ensures that the schedule for the concurrent execution of several transactions should yield consistent results.

# 8. Be able to list and give an example of the three concurrency control problems described in Chapter 10.

- Lost update
  - Occurs in two concurrent transactions when: -Same data element is updated-One of the updates is lost
- Uncommitted data
  - Occurs when:-Two transactions are executed concurrently-First transaction is rolled back after the second transaction has already accessed uncommitted data
- Inconsistent retrievals
  - Occurs when:-A transaction accesses data before and after one or more other transactions finish working with such dat

# 9. Be familiar with the date and time functions covered in class.

## CSCI_4400_NOTES_4_16_2019.md

## Date and Time stuff

LEARN THE ISO 8601 standard

```
SELECT SYSDATETIME(), SYSDATETIMEOFFSET(), SYSUTCDATETIME();
```

get the current system date and time of the computer

get the time of the computer running the DB but with an offset

get the current system date and time of the computer but set to UTC timezone.

```
/* Returned:
SYSDATETIME()      2007-04-30 13:10:02.0474381
SYSDATETIMEOFFSET()2007-04-30 13:10:02.0474381 -07:00
SYSUTCDATETIME()   2007-04-30 20:10:02.0474381
CURRENT_TIMESTAMP  2007-04-30 13:10:02.047
GETDATE()          2007-04-30 13:10:02.047
GETUTCDATE()       2007-04-30 20:10:02.047
*/
``
```

```sql
SELECT CURRENT_TIMESTAMP, GETDATE(), GETUTCDATE();
```

gets the current time, current date, get the utc date

```sql
SELECT DAY(GETDATE()), MONTH(GETDATE()), YEAR(GETDATE());
```

Gets the day, month, year of the current time

```sql
SELECT DAY('1970-12-25'), MONTH('1970-12-25'), YEAR('1970-12-25');
```

```sql
SELECT DATEPART(DAY, GETDATE()), DATEPART(MONTH, GETDATE()), DATEPART(YEAR,
GETDATE()),
```

Gets a part of the date based on the input args

```sql
SELECT DATEPART(QUARTER, GETDATE()), DATEPART(ISO_WEEK, GETDATE()), DATEPART(WEEK,
GETDATE()), DATEPART(NANOSECOND, GETDATE())
```

get the current quater (3mon) period

get iso week number

```sql
-- An ISO week-numbering year (also called ISO year informally) has 52 or 53 full
weeks. That is 364 or 371 days instead of the usual 365 or 366 days. The extra
week is sometimes referred to as a leap week, although ISO 8601 does not use this
term.
```

get current week number

get the nanosecond...why?

```sql
SELECT DATEMONTHPARTS(1970, 12, 25);
```

builds date object from 'parts'

# DATES THAT HAPPEN BEFORE OR AFTER

## DAYS TILL CHRISTMAS

```
SELECT DATEDIFF(DAY, GETDATE(), DATEFROMPARTS(YEAR(GETDATE()), 12, 25));
```

DAYS SINCE CHRISTMAS

```
SELECT DATEDIFF(DAY, GETDATE(), DATEFROMPARTS(YEAR(GETDATE())-1, 12, 25));
```

## DATE 2 WEEKS INTO THE FUTURE

```
SELECT DATEFROMPARTS(YEAR(GETDATE(), MONTH(GETDATE()), DAY(GETDATE())+21)
```

NOTE: THIS FAILS

ADDING DATES

```
SELECT DATEADD(DAY, 21, GETDATE());
```

## ISDATE

NOTE: SQL DOES NOT HAVE TRUE FALSE, they have 0 for true and 1 for false.

```
SELECT ISDATE(GETDATE()); -- 1
SELECT ISDATE('12-25-1970'); -- 1
SELECT ISDATE('Superman'); -- 0
SELECT ISDATE('1-1-1'); -- 0
```

# 10. Be familiar with the string functions covered in class.

```
SELECT LEN('George Washington'), REVERSE('George Washington')
```

returns the length of the string

returns the reverse of the characters of the string

```
-- SELECT LTRIM('   Hi!   '), RTRIM('   Hi!   '), TRIM('   Hi!   ');
SELECT LTRIM('   Hi!   '), RTRIM('   Hi!   '), LTRIM(RTRIM('   Hi!   '));
```

### UPPERCASE AND LOWERCASE

```
SELECT UPPER('aBcDeF'), LOWER('aBcDeF');
```

### RIGHT AND LEFT

```
SELECT LEFT('abcdefghij', 3), RIGHT('abcdefghij', 3);
```

### SUBSTRING

```
-- SELECT SUBSTRING(string, start, length);
SELECT SUBSTRING('abcdefghij', 3, 6);
```

## 11. Be familiar with the CAST and CONVERT operations.

CAST IS AN SQL SYNTAX to convert CONVERT IS A FUNCTION to convert

```
SELECT 9.5 AS ORIGINAL, CAST(9.5 as int) AS inte, CAST(9.5 as decimal(6, 4)) AS
deci;
```

### CONVERT

```
SELECT 9.5 AS ORIGINAL, CONVERT(int, 9.5) AS INTE, CONVERT(decimal(6, 4), 9.5) AS
DECI;
```

## Study the material on normalization from Chapter 6 and be prepared to solve a problem on normalization

### Normalization

Database Tables and Normalization

- Normalization: evaluating and correcting table structures to minimize data redundancies

    - Reduces data anomalies
    - Assigns attributes to tables based on determination
- Normal forms
    - First normal form (1NF)
    - Second normal form (2NF)
    - Third normal form (3NF)
- Structural point of view of normal forms
    - Higher normal forms are better than lower normal forms
    - Properly designed 3NF structures meet the requirement of fourth normal form (4NF)
- Denormalization: produces a lower normal form
    - Results in increased **performance** and greater **data redundancy**

## The Need for Normalization

- Used while designing a new database structure
    - Analyzes the relationship among the attributes within each entity
    - Determines if the structure can be improved through normalization
    - Improves the existing data structure and creates an appropriate database design

## Quick View or Normal Forms

- First normal form (1NF); Table format, no repeating groups, and PK identified; 6-3a
- Second normal form (2NF); 1NF and no partial dependencies; 6-3b
- Third normal form (3NF); 2NF and no transitive dependencies; 6-3c
- Boyce-Codd normal form (BCNF) Every determinant is a candidate key (special case of 3NF); 6-6a
- Fourth normal form (4NF); 3NF and no independent multivalued dependencies; 6-6b

## Partial dependency

- functional dependence in which the determinant is only part of the primary key
    - Assumption: one candidate key
    - Straight forward
    - Easy to identify

## Transitive dependency

- attribute is dependent on another attribute that is not part of the primary key
    - More difficult to identify among a set of data
    - Occur only when a functional dependence exists among nonprime attribute

## The Boyce-Codd Normal Form (1 of 4)

- Every determinant in the table should be a candidate key
    - Candidate key: same characteristics as primary key but not chosen to be the primary key
    - Equivalent to 3NF when the table contains only one candidate key
    - Violated only when the table contains more than one candidate key
    - Considered to be a special case of 3N

## Fourth Normal Form (4NF)

- Rules
    - All attributes must be dependent on the primary key, but they must be independent of each other
    - No row may contain two or more multivalued facts about an entity
- Table is in 4NF when it:
    - Is in 3NF
    - Has no multivalued dependencies

### Designing with Normalization in mind

- Normalization should be part of the design process
    - Proposed entities must meet required the normal form before table structures are created
- Principles and normalization procedures to be understood to redesign and modify databases
    - ERD is created through an iterative process
    - Normalization focuses on the characteristics of specific entitie

### Normalization Summary

- Normalization is a technique used to design tables in which data redundancies are minimized
- A table is in 1NF when all key attributes are defined and all remaining attributes are dependent on the primary key
- A table is in 2NF when it is in 1NF and contains no partial dependencies
- A table is in 3NF when it is in 2NF and contains no transitive dependencies
- A table that is not in 3NF may be split into new tables until all of the tables meet the 3NF requirements
- Normalization is an important part—but only a part—of the design process
- A table in 3NF might contain multivalued dependencies that produce either numerous null values or redundant dat
- The larger the number of tables, the more additional I/O operations and processing logic you need to join them
- The data-modeling checklist provides a way for the designer to check that the ERD meets a set of minimum requiremen

---

# Study the material on concurrency and transactions from Chapter 10 and be prepared to show examples or respond to questions on terminology

---

## Material on Concurrency

### Problems with Concurrency

- Lost update
    - Occurs in two concurrent transactions when:
        - Same data element is updated
        - One of the updates is lost
- Uncommitted data

    - Occurs when:
        - Two transactions are executed concurrently
        - First transaction is rolled back after the second transaction has already accessed uncommitted data
- Inconsistent retrievals
    - Occurs when:
        - A transaction accesses data before and after one or more other transactions finish working with such data

## The Scheduler

- Establishes the order in which the operations are executed within concurrent transactions
    - Interleaves the execution of database operations to ensure serializability and isolation of transactions
- Bases actions on concurrent control algorithms
    - Determines appropriate order
- Creates serialization schedule
    - Serializable schedule: interleaved execution of transactions yields the same results as the serial execution of the transactions

## Concurrency Control with Locking Methods

- Locking methods facilitate isolation of data items used in concurrently executing transactions
    - Lock: guarantees exclusive use of a data item to a current transaction
    - Pessimistic locking: use of locks based on the assumption that conflict between transactions is likely
    - Lock manager: responsible for assigning and policing the locks used by the transaction

## Lock Granularity

- Indicates the level of lock use
    - Database-level lock
    - Table-level lock
    - Page-level lock
        - Page or diskpage: directly addressable section of a disk
    - Row-level lock
    - Field-level lock

## Lock Types

- Binary lock
    - Two states: locked (1) and unlocked (0)
        - If an object is locked by a transaction, no other transaction can use that object-
        - If an object is unlocked, any transaction can lock the object for its use
- Exclusive lock
    - Access is reserved for the transaction that locked the object
- Shared lock
    - Concurrent transactions are granted read access on the basis of a common

- Problems using locks
    - Resulting transaction schedule might not be serializable
    - Schedule might create deadlock

## Two-Phase Locking to Ensure Serializability

- Defines how transactions acquire and relinquish locks
    - Guarantees serializability but does not prevent deadlocks
- Phases
    - Growing phase: transaction acquires all required locks without unlocking any data
    - Shrinking phase: transaction releases all locks and cannot obtain any new
- Governing rules
    - Two transactions cannot have conflicting locks
    - No unlock operation can precede a lock operation in the same transaction
    - No data are affected until all locks are obtained

## Deadlocks

- Occur when two transactions wait indefinitely for each other to unlock data
    - Also known as deadly embrace
- Control techniques
    - Deadlock prevention
    - Deadlock detection
    - Deadlock avoidance
- Choice of deadlock control method depends on database environment

## Concurrency Control with Time Stamping Methods

- Time stamping assigns global, unique time stamp to each transaction
    - Produces explicit order in which transactions are submitted to DBMS
- Properties
    - Uniqueness: ensures no equal time stamp values exist
    - Monotonicity: ensures time stamp values always increases
- Disadvantages
    - Each value stored in the database requires two additional stamp fields
    - Increases memory needs
    - Increases the database's processing overhead
    - Demands a lot of system resources

## Wait/Die and Wound/Wait Schemes

- Wait/die
    - A concurrency control scheme in which an older transaction must wait for the younger transaction to complete and release the locks before requesting the locks itself
        - Otherwise, the newer transaction dies and is rescheduled
- Wound/wait
    - A concurrency control scheme in which an older transaction can request the lock, preempt the younger transaction, and reschedule it

        - Otherwise, the newer transaction waits until the older transaction finished

### Concurrency Control with Optimistic Methods

- Optimistic approach: based on the assumption that the majority of database operations do not conflict
    - Does not require locking or time stamping techniques
    - Transaction is executed without restrictions until it is committed
- Phases of optimistic approach
    - Read
    - Validation
    - Write
- Read phase
    - Transaction:
        - Reads the database
        - Executes the needed computations
        - Makes the updates to a private copy of the database values
- Validation phase
    - Transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database
- Write phase
    - Changes are permanently applied to the databa

## MATERIAL FOR TRANSACTIONS

### ANSI Levels of Transaction Isolation

- The ANSI SQL standard (1992) defines transaction management based on transaction isolation levels
    - Transaction isolation levels refer to the degree to which transaction data is “protected or isolated” from other concurrent transactions
- Transaction isolation levels are described by the type of “reads” that a transaction allows or not
    - Dirty read: transaction can read data that is not yet committed
    - Nonrepeatable read: transaction reads a given row at time t1, and then it reads the same row at time t2, yielding different results-The original row may have been updated or deleted
    - Phantom read: transaction executes a query at time t1, and then it runs the same query at time t2, yielding additional rows that satisfy the query
- Read Uncommitted will read uncommitted data from other transactions
    - Increases transaction performance but at the cost of data consistency
- Read Committed forces transactions to read only committed data
    - Default mode of operation for most databases
- Repeatable Read isolation level ensures that queries return consistent results
    - Uses shared locks to ensure other transactions do not update a row after the original query reads it
- Serializable isolation level is the most restrictive level defined by the ANSI SQL standard
    - Deadlocks are still always possibe

### Database Recovery Management

- Database recovery: restores database from a given state to a previously consistent state

- Recovery transactions are based on the atomic transaction property
    - All portions of a transaction must be treated as a single logical unit of work-If transaction operation cannot be completed:
        - Transaction must be aborted
        - Changes to database must be rolled back

## Transaction Recovery

- Concepts that affect the recovery process
    - Write-ahead log protocol
        - Ensures that transaction logs are always written before the data are updated
    - Redundant transaction logs
        - Ensure that a physical disk failure will not impair the DBMS's ability to recover data
    - Buffers
        - Temporary storage areas in a primary memory used to speed up disk operations
    - Checkpoints-Allows DBMS to write all its updated buffers in memory to dis
- Techniques used in transaction recovery procedures
    - Deferred-write technique or deferred update
        - Transaction operations do not immediately update the physical database-Only transaction log is updated
    - Write-through technique or immediate update
        - Database is immediately updated by transaction operations during transaction's execution
- Recovery process steps
    - Identify the last check point in the transaction log
        - If transaction was committed before the last check point nothing needs to be done
        - If transaction was committed after the last check point the transaction log is used to redo the transaction
        - If transaction had a ROLLBACK operation after the last check point the DBMS uses the transaction log records to ROLLBACK or undo the operations, using the "before" values in the transaction log

## Transaction Summary

- A transaction is a sequence of database operations that access the database
- Transactions have four main properties: atomicity, consistency, isolation, and durability
- SQL provides support for transactions through the use of two statements: COMMIT, which saves changes to disk, and ROLLBACK, which restores the previous database state
- Concurrency control coordinates the simultaneous execution of transactions
- A lock guarantees unique access to a data item by a transaction
- Serializability of schedules is guaranteed through the use of two-phase locking
- Concurrency control with time stamping methods assigns a unique time stamp to each transaction and schedules the execution of conflicting transactions in time stamp order
- Concurrency control with optimistic methods assumes that the majority of database transactions do not conflict and that transactions are executed concurrently, using private, temporary copies of the data
- Database recovery restores the database from a given state to a previous consistent state

---

# Study in detail the additional examples provided in the Review section of the Content area of D2L.

## REVIEW PROBLEMS ON STORED PROCEDURES AND TRIGGERS

### Number 1

```
-- Create a stored procedure that accepts a property ID and a deposit amount as
parameters and assigns the deposit amount as the new deposit amount for that
property.

CREATE PROC
  num1 -- name
  @propID int, -- property ID
  @depAmt FLOAT -- deposit Amount

  AS

  UPDATE Property -- table
  SET Property.PROP_DEPOSIT = @depAmt -- set this to this
  WHERE Property.PROP_ID = propID; -- but only when the id mathes (one record)

GO;

-- THEN FIRE IT

EXEC num1 1, 123.45;
```

### Number 2

```
-- Create a stored procedure that accepts values for a Repair Category ID and
Repair Category Name as parameters and inserts a new row into the Repair Category
table.

CREATE PROC
  p2
  @repCatID int,
  @repCatName varchar(255)

  AS

  INSERT INTO [Repair Category]
    ([Repair Category].Repair_Category_ID,
     [Repair Category].Repair_Category_Name)
  VALUES
    (@repCatID, @repCatName);

GO;
```

```
-- THEN FIRE IT

EXEC p2 100, 'Cool stuff';
```

## Number 3

```
-- Create a stored procedure that accepts a tenant Social Security Number as a
parameter and displays the address of all properties that have been rented by that
tenant.  Print a message if the Social Security Number is not the number of a
person who has ever rented a property.

CREATE PROC
  p3
  @T_SSN varchar(20)

  AS

  IF (0 = (SELECT COUNT(TENANT_SSNO) FROM TENANT WHERE TENANT_SSNO = @T_SSN))
    PRINT 'INVALID SOCIAL SECURITY NUMBER'
  ELSE
    SELECT PROP_ADDRESS
    FROM (TENANT INNER JOIN PROPERTY
      ON TENANT.TENANT_ID = PROPERTY.TENANT)
    INNER JOIN RENTAL
      ON PROPERTY.PROP_ID = RENTAL.PROP_ID
    WHERE LAND_ID = @ID;
GO;

-- THEN FIRE IT

EXEC p3 '123-45-6789';
```

## Number 4

```
-- Create as stored procedure that accepts a Landlord ID as a parameter and
increases the rent amount for all properties owned by that landlord by $35.00.

CREATE PROC
  p4
  @landlordID int

  AS

  UPDATE Property
  SET PROP_RENT = PROP_RENT + 35.00
  WHERE Property.LAND_ID = @landlordID

GO;
```

## Number 5

```
-- Create a stored procedure that displays the property ID and address of all
properties that had a tenant move out during 2012.  Hint:  The RENT_END_DATE must
be on or after January 1, 2012.

CREATE PROC
  p5

  AS

  SELECT
    PROPERTY_ID,
    PROP_ADDRESS
  FROM PROPERTY INNER JOIN RENTAL
    ON PROPERTY.PROP_ID = RENTAL.PROP_ID
  WHERE
    RENTAL.RENT_END_DATE >= '01-01-2012' AND
    RENTAL.RENT_END_DATE < '01-01-2013';

GO;
```

## Number 6

```
-- Create a trigger that prevents the deposit amount in the Rental table from
being lowered.

CREATE TRIGGER t6
  ON RENTAL
  FOR INSERT, UPDATE

  AS

  DECLARE @V1 money
  DECLARE @V2 money
  SELECT @V1 = INSERTED.RENT_DEPOSIT FROM INSERTED
  SELECT @V2 = DELETED.RENT_DEPOSIT FROM DELETED

  IF @V1 < @V2
  BEGIN
    ROLLBACK TRANSACTION
    PRINT 'DEPOSIT AMOUT MAY NOT BE DECREASED'
  END
```

## Number 7

```
-- Create a trigger that prevents the deposit amount in the Rental table from
being less than half of the deposit amount in the Rental table for the property
that is involved.

CREATE TRIGGER
  t7
  ON RENTAL
  FOR INSERT, UPDATE

  AS

  DECLARE @V1 FLOAT
  DECLARE @V2 FLOAT
  SELECT @V1 = INSERTED.RENT_DEPOSIT FROM INSERTED
  SELECT @V2 = DELETED.RENT_DEPOSIT FROM DELETED

  IF @V1 < (@V2 * 0.5) -- less than half
  BEGIN
    ROLLBACK TRANSACTION
    PRINT 'DEPOSIT MUST BE AT LEAST HALF OF RENT'
  END
```