# 5.1. Database implementation

## 5.1.1. Concurrency

Concurrency is the execution of multiple transactions, which may access the same database rows during overlapping time periods. Such simultaneous accesses (called collisions) may result in errors or inconsistencies in data if not handled properly, causing the following phenomena to arise:

* Lost Update- a transaction is rendering previous one obsolete.
* Dirty Read- a transaction can read data from a row that has been modified by another running transaction and which is not yet committed.
* Non-Repeatable Read- during a transaction, a row is retrieved twice and the values within the row differs between reads.
* Phantom Read- while a transaction, new rows are added or removed by another transaction to the records being read.

To deal with the already mentioned issues there are two commonly used approaches, with each of them tackling the problem in different ways:

* Pessimistic locking: used when a collision is anticipated, the transactions violating synchronization are simply blocked. May cause deadlocks if not properly implemented.
* Optimistic locking: costs less to do operation especially when many collisions are not expected, but if the collision occurs the transaction should abort, losing the progress.

For our solution we opted to use pessimistic locking (explained further in section transactions). However, the optimistic locking might have been better solution for tasks like borrowing and returning book. The decision to use pessimistic locking everywhere was made, since for the intended amount of operations, that this application would experience during a day, no noticeable change should not be seen and group was on tight schedule to deliver the working product before deadline, but in future if need be it will not be hard to update the code implementing optimistic approach.

## 5.1.2. Isolation levels

Transactions specify an isolation level that defines the degree to which one transaction must be isolated, from resource or data modifications made, by other transactions. A transaction always gets an exclusive lock on any data it modifies and holds that lock until the transaction completes, regardless of the isolation level set for that transaction. A lower isolation level increases the ability of many users to access data at the same time, but increases the number of concurrency effects, such as dirty reads or lost updates, that users might encounter. Conversely, a higher isolation level reduces the types of concurrency effects that users might encounter but requires more system resources and increases the chances that one transaction will block another. Choosing the appropriate isolation level depends on balancing the data integrity requirements of the application against the overhead of each isolation level.

The following table (Table 1) shows the concurrency side effects allowed by the different isolation levels.

Table 1

|  |  |  |  |
| --- | --- | --- | --- |
| Isolation Level | Dirty Read | Non-Repeatable Read | Phantom |
| Read uncommitted | Yes | Yes | Yes |
| Read committed | No | Yes | Yes |
| Repeatable read | No | No | Yes |
| Snapshot | No | No | No |
| Serializable | No | No | No |

## 5.1.3. Transactions

A transaction is a package consisting of single or more commands, who are executed as one unit of work. If a failure occurs at one point in the transaction, all of the updates can be rolled back to their pre-transaction state. Transactions that involve multiple resources can lower concurrency, if locks are held too long, therefore, it is advisable that transactions are kept as short as possible. Our wish to have the data consistent and reliable meant that while developing the application we have made use of transactions in places where they were necessary (shown in Figure 1), with each of them using isolation levels we determined would best fit the requirements of the task.

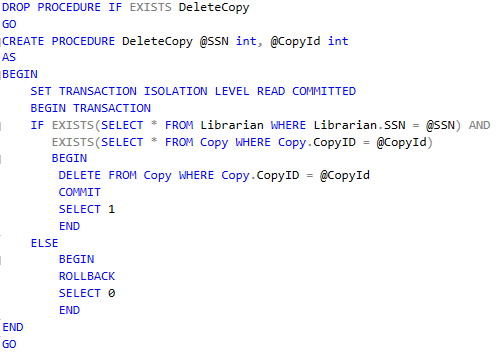


Figure 1

## 5.1.4. Security

A SQL Server instance contains a hierarchical collection of entities, starting with the server. Each server contains multiple databases, and each database contains a collection of securable objects (Tables, Views etc). Every SQL Server securable has associated permissions that can be granted to an individual, group or process. The SQL Server security framework manages access to securable entities through authentication and authorization.

* Authentication establishes the identity of the user or process being authenticated.
* Authorization is the process of determining which resources and operations is the authenticated user allowed to use.

Every securable object has permissions that can be granted to a user using permission statements. Granting permissions to roles rather than to users simplifies security administration, as permission sets that are assigned to roles are inherited by all members of the role. When assigning permissions to database users you should always follow the principle of least privilege (Grant the minimum permissions necessary to a user or role to accomplish a given task). Currently our database does not use any permissions or roles, as we were still developing the solution at the time of writing this document and planned focusing more on security after we had established some basic application satisfying the needs of GTL.

### Encryption

SQL Server provides functions to encrypt and decrypt data using a certificate, asymmetric key, or symmetric key. However, it was decided to not implement it, as all expected communication, at the moment, would be performed through our application, which is located on the same machine, and this would result only in degraded performance with little to no improvement to security.

### SQL Injection

Applications frequently take external input and perform actions based on the information provided, if the information is not handled properly it could be used, to inject malicious code which could make the application perform actions that it was not designed for. These attacks are called SQL injections and they are most commonly used for website but can be used for any other application using relational databases (Example in Figure 2). SQL injection attacks, allow attackers to spoof identity, tamper with existing data, cause refusal issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

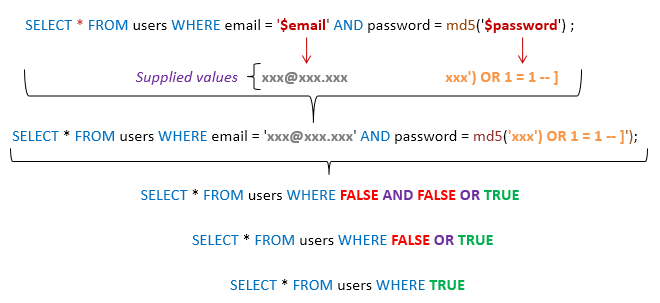


Figure 2

The most basic way of defending yourself against simple attacks like these is to parameterize the results passed to database, that way the information passed will not be considered as a part of the SQL statement and will not be executed. Since we used entity framework LINQ for communication with the database, where information is automatically passed as parameter, thus making it not susceptible to traditional SQL injection attacks. Furthermore, we never used Entity SQL with input supplied by the user, thus following the steps ensuring safety provided by Microsoft when using entity framework.

## 5.1.5. Procedures

### a) Stored Procedures

A stored procedure is a prepared SQL query that you can save, so it can be reused multiple times. Stored procedures resemble constructs found also in other languages, by allowing input parameters and return values; they contain programming statements that perform various operations and return a status value to a calling program to indicate success or failure (and the reason for failure). This made it attractive to our solution, where we wished to do as much as possible in the database itself. The use of stored procedure not only reduced the duplication of code which could have arisen, but also let the solution to be easier to maintain.

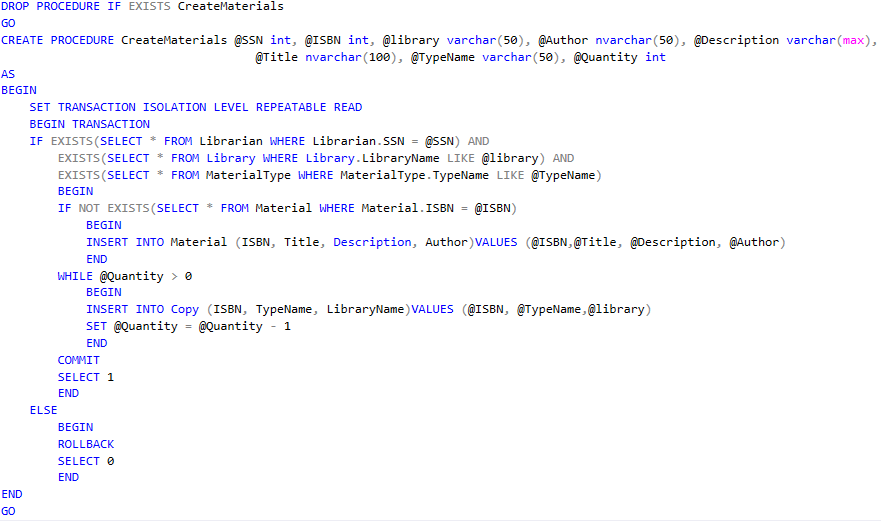


Figure 3

FIgure 3 shows one of the main procidures found in our solution, that being creation of materials. This procedure has 8 paramaters which have to be provided in order to perform the inclosed statement. One of the first actions that are performed, when calling the procedure is establishing transaction (since we wanted to be sure each call to database is atomic), folowing that the intended operations are performed on the data, ending with changes being either commited or undone.

### Computed column

A computed column is a virtual column that is not physically stored in the table, unless the column is marked “PERSISTED”. A computed column expression can use data from other columns in the table, to calculate a value for the column to which it belongs.

The idea to use computed columns was suggested and considered, but due to its limitations it couldn’t be implemented as the developers envisioned the solution.

## 5.1.6. Scheduling

Scheduling and execution of jobs, which would be needed to be performed at certain time periods, was done using SQL Server Agent. The use of SQL server agent however was limited to just running single SQL script once every day (Figure 4), this would in return allow librarians to know what books have been borrowed and not retuned before the end of the grace period, thus making it simpler for them to manage it and send notices to involved members. It could have been possible to automate this even further and automatically send email messages to the members in question, but due to limitation in information required to become a user of the library system this was not yet possible to implement.

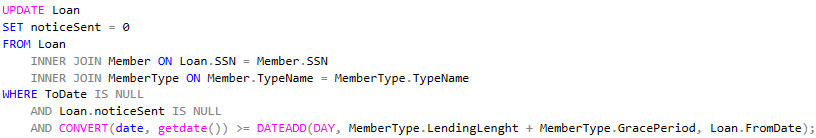


Figure 4

Used:

<https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-2017>

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/overview-of-sql-server-security>

<https://en.wikipedia.org/wiki/Isolation_%28database_systems%29#Isolation_levels>

<https://docs.microsoft.com/en-us/sql/relational-databases/tables/specify-computed-columns-in-a-table?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-2017>

<https://crate.io/docs/sql-99/en/latest/chapters/37.html>

<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations>

<https://en.wikipedia.org/wiki/SQL_injection>

<https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-2017>