# Overview of the Assignment:

In this exercise we will continue our review of advanced programming components, specifically looking at triggers and stored procedures and code-re-use. This exercise is based on the Advanced SQL Programming Netflix database.

# Preparing for the Assignment

# Before the exercises in this assignment can be completed, you should complete Advanced SQL Programming Part 1.

# It is strongly recommended that you review the Netflix schema and the questions before beginning the assignment. This includes inspecting the ERD schema provided with the Advanced Programming Assignments. The assignment simulates a real-world environment of walking into a working system and adding additional functionality. Code re-use as part of your solution is strongly encouraged, if you are able to solve the question in less steps feel free to do so

**Please use the submission template to submit your assignment.**

**Advanced Programming Part 2 Questions**

Continuing with the scenario from the Advanced Programming assignments, suppose that the small firm for which you designed the Netflix®-like DVD rental database has established an Internet presence and their sales are growing rapidly. You have been asked to make additional programming changes to the database and add new functionality. Your assignment is to implement the following additional requirements. Note that you may want to review your solution to Advanced Programming part 1 and reuse the components which you developed there.

1. Evaluate and implement (create) a **History** table for the Netflix database. The history table should keep track of **rental history**. Review the next few questions to see what you might want to include within the history table. Consider denormalization and what might be useful within the history table beyond just the rental information. Your rental history table could be an audit table, or a denormalized design, or a combination of both. Explain your design, the purpose of your history table and include reasoning for denormalization.

**For the next questions make sure to document and test your code:**

* Adding code comments will help us understand what you are logically trying to do, and is a great habit to get into for code readability, flexibility and reusability.
* In the case of triggers provide screen-shots of before and after of the data (i.e. what the history table looks like before the trigger is fired, and what it looks like after.)
* For question 5 and 6 please test your procedure by inserting a movie at the beginning, middle and the end of the queue. Your screenshots need to show the queue data before the insert/delete, the insert/delete itself, and the data in the queue after showing that the queue position has changed.

1. Implement a **trigger** for this new **rental history** table that prevents deletions from the table using error handling logic.
2. Implement a **trigger** that automatically updates the **rental history** when a DVD is shipped to a customer. Depending on your design of the rental history, this update may be an UPDATE and/or an INSERT.
3. Implement a **trigger** that automatically updates the rental history when a DVD is received from a customer.
4. Implement a **stored procedure** that adds a title to the customer’s movie list (the **Rental Queue** table). This procedure should take as IN parameters the customer ID and movie title ID as well as the location of where the movie is in the queue. The procedure should also make sure that no duplicate titles can be added. You will need to add some error handling in your code.

* You will also notice that there is no queue position attribute, so you will need to solve that issue.
* The stored procedure will need to do some queue position management of the existing movies in the queue. Queue position management allows the customer to rank the movies in their queue. For instance, the customer may want to rent “Jack Reacher” as their top rental choice, but before it’s shipped out, decides he’d really like to rent “Les Misérables” first instead. You will have to add this feature to the current schema design.

1. Write a **stored procedure** that deletes a title from a customer’s movie list (the **Rental Queue** table). This procedure should take as IN parameters the customer ID and movie title ID.

* Hint: The stored procedure will need to do some queue position management of the existing movies in the queue building on what you did for the previous requirement.