2015

Flix2You

Information Enrichment Database

IST 210
Team 02
8/5/2015

Table of Contents

Executive Summary 2
Project Overview 3
Project Management 5
- Project Resources and Budget 6
- Budget & Resources 7

User Analysis 8
Organization of Data 9
- Analysis of Flix2You's Existing Operational Database 9
- Normalized Design for a New Operational Database 11
- Read-Only OLAP Database, and MOLAP Cube 13

Database Administration 14
- Roles: User access levels 14
- Backup and Recovery 15
- Access and Security 16
- Data Load 17

Database Dashboard and Analytics 18
Legal Issues 25
References 28

Executive Summary

During the dozen years since the founding of Flix2You its business model has continuously evolved to adapt to the changing demands of its competitive and technological environment, and seize the opportunities presented by a rapidly expanding online market. But the company's internal IT resources, unfortunately, have failed to evolve to keep pace with the enterprise that they are meant to serve. The team has concluded that if Flix2You is to prevent its competitors from winning an insuperable advantage, the time to align its internal IT infrastructure with its business processes, goals and strategies is ripe, if not overdue.

The team here presents a plan for implementing a normalized redesign of the present operational database, which afterwards will continue to function mainly as a backend to the company's corporate website. Elimination of large-scale redundancies and more accurate conceptual modeling will quickly provide to customers a more responsive browsing experience, while at the same time increasing the efficiency of the insertions, updates and deletions with which the web application automatically processes routine rental and purchase transactions.

To enhance the continued ability of Flix2You's management to change course as proactively as it regularly has in the past, but with the greater speed and agility that will be required in the foreseeable future, the team's plan also designs and provides for the implementation of a read-only OLAP database, with associated structures and applications to facilitate analysis for better decision-making with timely and relevant business intelligence. Separation of the analytic and transactional databases will improve the performance of both.

To enable Flix2You to deploy an IT infrastructure that will be simultaneously ubiquitous, reliable, secure and easily scalable, the team's plan envisions resort to a cloud-based Microsoft Azure platform as a service that will include replicated webserver and database components. Azure was chosen because of its thoroughgoing integration with Microsoft technologies like SQL Server and ASP.NET that Flix2You already puts or will soon put to use.

Today a heightened public concern for the security and privacy of the data that has been entrusted to an enterprise is everywhere apparent, as are the risks to enterprise continuity that may be presented by vulnerabilities to anonymous online attacks. A cloud-based platform can be expected to afford some considerable part of the necessary protection, but the design of a policy-and-role-based administrative scheme of database access privileges is also essential, and the team's plan accordingly provides one. Other measures needed to guarantee the privacy/confidentiality and integrity of data are also elements of the plan, as are plans for storage redundancy, backup and recovery measures needed to guarantee its continued availability. Disclaimers, disclosures and waivers circumscribing the legal privacy rights of customers respecting their personal data have also been drafted for Flix2You.

The plan proposed by the team is complex, and project management is thus an essential component of the services that it is offering. Refer to the included Gantt chart depicting the timeline for plan execution, including the anticipated major milestones along its progress, and the budget of total estimated implementation costs.

Project Overview

From its beginnings in 2002 as a small video rental company with local outlets in Pittsburgh and Scranton, Pennsylvania, Flix2You grew to a regional enterprise that once operated 520 stores in the Eastern United States. In December 2013, however, it closed all of these brick-and-mortar stores, and today operates entirely online, where its website experiences a daily average of 20,000 unique hits from a total estimated base of 20,000,000 customers. Services offered to these customers from its website include mail-order distribution of Blu-Ray and DVD video discs and, ever more predominantly, HTTP video streaming in a multitude of high- and standard-definition formats.

The management of Flix2You, being unwilling to concede anything to online giants like Netflix, Apple and Amazon, entertains ambitions for their company to become the most comprehensive online video rental portal in the country. The company's present infrastructure, however, is woefully inadequate to these ambitions, and continues to betray the company's more modest origins: the corporate website, for example, is hosted entirely on a single rented Go-Daddy server, which is connected to a database server backend in the Flix2You headquarters in Scranton by a T-1 line that allows a maximum of only 1.5 Mbps throughput.

While Flix2You's management reports that its corporate database was developed from a packaged, off-the-shelf e-commerce solution, this database, too, reflects the company's now abandoned brick-and-mortar origins. Examination of its entities reveals a model in which the fundamental unit of saleable asset value is an individually packaged video disc or VCR tape (or perhaps "bins" of these on the shelves of local stores), rather than an intangible license that confers distribution rights to a single artistic production. Entities representing product distribution outlets are not the server clusters of an internet-based content distribution network, or the regional warehouses of an international mail-order system, but "video stores." Attributes of the entity representing a customer include none of those that are necessary to basic online account maintenance.

Predictably Flix2You is now suffering painful symptoms that it has outgrown its outmoded infrastructure. Management complains that it cannot easily retrieve the relevant and timely business intelligence that it needs to facilitate targeted marketing and intelligent product acquisition, or formulate a strategy for future growth. Meanwhile Flix2You's customers complain, during intervals when employees are querying the database for routine management purposes, that their online browsing experience is slowed or otherwise degraded.

These problems have prompted the Board of Flix2You to invite submission of proposed plans for the design and implementation of a solution for their practical mitigation. Although it is expected with good reason that the heart of any proposed solution will concern itself chiefly with the redesign of database structures, and although the Board of Flix2You naturally wishes to make fullest use of the company's substantial investment in existing IT resources, the Board affirms that it is willing to consider adopting a more robust infrastructure if the team shows it to be warranted by anticipated future business demands.

The team herein attempts to create for the Board of Directors of Flix2You a proposal for the re-design and implementation of customer, product and sales schema and related database structures that will (1) serve as the backend for an e-commerce application that provides customers with a pleasant, efficient and personalized experience when they browse the online Flix2You movie catalog, while (2) serving also as an accessible storehouse from which Flix2You management can easily retrieve relevant and timely business-intelligence information to facilitate targeted marketing, and intelligent product acquisition. The model for the redesigned database structures will be adapted as closely as possible to the present business processes and products of Flix2You, while retaining enough abstraction and flexibility to accommodate future evolution of the business.

Proposals for the redesign of existing database structures are documented at the conceptual modeling level by the creation of entity relationship diagrams. Sample SQL code is provided to illustrate the logical implementation of the model through the creation of necessary tables, relationships, attributes and constraints (and to validate the model by demonstrating its adequacy to the processing of everyday routine transactions) as well as to illustrate the subsequent querying of the implemented design for end-user data, either through the web application accessible to customers for browsing the movies catalog, or a dashboard that will be provided to management for graphical front-end access to data analytics.

Consideration of the needs of user classes like these and their required views serve as the premises for the design of a policy-and-role-based administrative scheme of database access privileges. Other measures needed to guarantee the privacy/confidentiality and integrity of data are also described, as are plans for storage redundancy, backup and recovery measures needed to guarantee its continued availability. Disclaimers, disclosures and waivers circumscribing the legal privacy rights of customers respecting their personal data have also been drafted for Flix2You.

A Gantt chart depicting the timeline for execution of the team's proposal, including the anticipated major milestones along its progress, is included, as also is a budget of estimated dollar implementation costs. Estimated cost parameters prominently include those associated with a cloud-based Microsoft Azure platform as a service, including replicated webserver and database components, that the team recommends to Flix2You, to enable it to deploy an IT structure that will be simultaneously ubiquitous, reliable, secure and easily scalable.

During project implementation the team may also suggest to the Board or participate in providing training programs that may be useful to Flix2You employees.

Project Management

Project Time Line	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Task Name	5/17 - 5/24	5/24 - 5/31	5/31 - 6/7	6/7 - 6/14	6/14 - 6/21	6/21 - 6/28	6/28 - 7/5	7/5 - 7/12	7/12 - 7/19	7/19 - 7/26	7/26 - 8/2	8/2 - 8/9
Introduction												
Project Overview												
Current Database Evaluation												
Establish Roles												
Project Planning												
Identifying Problems With Current Database												
Conducting Research on Softwares												
Ordering Equipment												
Design and Development												
ERD Outlines												
Developing a Dashboard and Final ERD												
Data Dictionary and Database Tables												
Begin SQL Coding												
Select a Single ERD												
Installation and Implementation												
Installation of All Equipment and Software Required												
Finalization of ERD												
Understand Scope of Implementation												
Perform Component Tests To Identify Problems												
Fix Any Lingering Issues or Bugs												
Completion of Assignment 2												
Create A Testing Base												
Configure Equipment and Software												
Testing												
Understand Database Procedures to be Tested												
Perform Sample Querries												
Analyze Results and Diagnostics												
Generate Reports and Gather Understanding												
Evaluation												
Understand Persisting Problems												
Fix Problems Which Persist												
Evaluate Results												
Ensure the Project Has Met It's Goal												
Training												
Develop A Training Program For Employees												
Train All Users												
Gather Feedback From Users												
Fix Any Lingering Issues												
Launch												
Launch Database on Large Scale												

- ***Project Resources and Budget***

Database Administrator:
This individual is responsible for managing, organizing, and controlling the centralized corporate database. The database administrator is also the manager of the database administration department. His/her responsibilities include but are not limited to:

- Database maintenance
- Database design
- Database accessibility
- Implementation of data models.

Project Manager:
This individual is responsible for keeping the group up to task weekly in order to meet project deadlines. He/she is also responsible for making sure the project is complete in a timely manner. The project manager's responsibilities include but are not limited to:

- Managing the overall project
- Identifying the appropriate scope of the project to proceed with
- Responsible for the completion of the project.

Database Analyst:
This individual is responsible for designing and evaluating the corporation's database. He/she is also responsible for constructing diagrams, conducting reviews, and making implementations on the database to assure conformance with user needs. The database analyst's responsibilities include but are not limited to:

- Writing the necessary codes needed to access a database
- Development and management of database policies
- Installation and implementation of databases.

Programmer:
This individual is responsible for writing and developing SQL code and scripts for the creation of the database. The SQL code is created to be user-friendly so it can be accessed and used with ease. The responsibilities of a programmer include but are not limited to:

- Developing SQL code and scripts
- Testing of SQL code to ensure it is user-friendly
- Performing maintenance checks on the SQL code that was created.

Quality Assurance:
This individual is responsible for developing test plans and creating test cases to ensure that the project completes the task it was required to perform. He/she is also responsible for making sure the implementation of the project succeeds under various test conditions.

❖ *Budget & Resources*

Job Title	Employee	Hourly Rate	Hours Worked	Salary
Project Manager	Aniruddh Guru	$50.00	250	$12,500.00
Database Administrator	Alfred Roebuck	$50.00	270	$13,500.00
Database Analyst	Austin Miller	$55.00	285	$15,675.00
Programmer	Matthew Krebs & Randall Miller	$60.00	300	$18,000.00
Quality Assurance	Stephen Ferstel	$50.00	220	$11,000.00
Total			1325	$70,675.00

Equipment	Cost
Computer	$4,000.00
RAM	$1,500.00
Internal & External Hard Drives	$1,000.00
Computer Upgrades	$150.00
Operating Systems and Network	$250.00
Microsoft Azure	$500.00
Total	$7,400.00

User Analysis

Based on the analysis of Flix2You's challenges and the proposed database structure there are four recommended roles each providing a different level of access. The Executive role is to be used by the upper level management and administrative support. The Management role will be utilized by mid-level management including the on-staff database administrator. Warehousing and logistics will access the database using the Logistics role. Finally there will be a role created to facilitate data entry when needed. These four roles will accommodate the needs of our proposed database for Flix2You.

The Executive tier will be assigned SELECT access rights to following tables:

- Customer
- MovieRating
- TransactedEvent

The 12 upper level management and administrative support users will utilize the User dashboard to view customer information gathered from the Customer table, the Movie dashboard to drive future film purchases based off information from the MovieRating table and the Company dashboard to view financial information.

The 18 mid-level management users utilizing the Management tier will possess full read/write access to all tables within the database. This will be accomplished by assigning the SELECT and UPDATE access roles to all tables. The users within this group will be responsible for making adjustments to high level information within the database, many times under the direction of executive leadership.

Logistics will be used by the 20 warehousing and logistics employees and possess read access to the following tables by assigning SELECT access rights:

- Customer
- Movie
- ViewableEncoding

The information provided will ensure that these users have the ability to provide timely delivery and service of Flix2You members who do not utilize a streaming device and request physical media.

Finally, we will create a DataEntry option for the inevitable need to manually update information for films. This role will have read/write access to the following tables:

- Actor
- Country
- Director
- Genre
- Movie
- MovieRole
- Studio

This will be completed by assigning INSERT, SELECT and UPDATE access rights to the tables above.

Organization of Data

❖ *Analysis of Flix2You's Existing Operational Database*

The entity-relationship diagram of a sound database design should faithfully mirror the operations, business processes and information flows of an organization. This is far from the case for the operations of Flix2You, and the schema of its current database that is reproduced in Appendix A.

In the upper left-hand corner of Appendix A there is depicted a Video_Stores entity class whose attributes include a name and physical address, and whose primary key is a foreign-key attribute of the adjacent Movies entity class. But Flix2You closed all 520 of its regional brick-and-mortar video outlets in December 2013, so that new rows inserted into the existing Movies table in 2014 and afterwards will all presumably have NULL values in the store_id column of this table.[1]

Flix2You instead now conducts all of its business on the worldwide web, and reports that it has a base of about 20,000,000 online customers. Although Flix2You's management also relates that its existing database was the canned product of a shrink-wrapped e-commerce package, there is little or nothing in the ERD of Appendix A to suggest the backend of a modern web application; to the contrary all of the business processes suggested by its entities, attributes and relationships recall the long-lost days of a neighborhood VCR rental store.

In the domain of the payment_method_description attribute of the Payment_Methods entity, for example, there is presented the illustrative value “Cash.” Currency does not physically change hands in an online consumer transaction, of course, since this evidently would be impractical, the vast majority of these being instead conducted with the use of customer credit-card accounts; so it is unclear that the Payment_Methods entity class now can serve any useful purpose in the Flix2You database at all. Since the Accounts entity class visible immediately above it in the ERD of Appendix A seems to be an associative one created only for the purpose of implementing a many-to-many relationship between the Payment_Methods and Customers entity classes, moreover, it seems that the Accounts entity class is another that no longer serves any useful business purpose.

In the Customers entity class itself there is a membership_number attribute, distinct from the customer_id attribute that serves as the primary key of the associated Customers table. The nature of this membership_number remains somewhat uncertain, but it most likely corresponds to a number on wallet-sized plastic membership cards like those that neighborhood Blockbuster outlets once issued. The equivalents of a plastic membership card that an online customer will instead present to the company today are his password and related online account credentials, which must be securely stored in the database for use by the web application in customer account validation. Flix2You

[1] It is also presumed here that the rows of the Video_Stores table were not deleted when the stores were closed, since doing so would have produced violations of referential integrity unless the deletions were configured to be cascading. And cascading deletions would have deleted all or most of the rows of the Movies table.

reports that its web portal is functioning, but customer online account credentials are nowhere visible in Appendix A.

Many of the remaining attributes of the entity classes visible in the ERD of Appendix A are related to a prototypical transaction consisting of the rental of a DVD or Blu-Ray disc for a term of days, with a physical checkout and return of the disc in person at a counter, and possibly the contemporaneous calculation and payment of late charges due. The online mail-order of digital movie discs survives as a business model in the single prominent example of Netflix, the purchaser of Blockbuster in bankruptcy having closed down its online DVD-by-mail rental business in November 2013, and only the customer clamor triggered by its attempt to spin off its own mail-order movie-disc business in the Qwikster debacle of September-October 2011 caused Netflix itself to desist from the course that it clearly preferred. Netflix' own subscription-based, n-discs-at-a-time model for the mail-order rental of digital movie discs, which differs significantly from the rental-for-a-term with late charges prototype envisioned in the ERD of Appendix A, may be the only model that is really feasible in the online context.

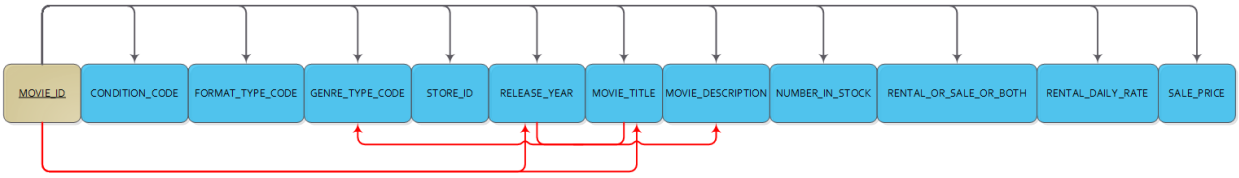
In any event Netflix' apparent belief that video streaming of movies and like video content is the wave of the future is an assessment shared by Flix2You's own management. To distribute a product by this means efficiently to customers, however, requires the existence of a content distribution network consisting of geographically dispersed clusters of web servers (within each of which there is available each video file, in a multitude of codecs and bitrates), just as the mail-order distribution of video discs requires a network of geographically dispersed mail distribution centers. Entity classes representing the nodes of this complex network— rather than the several hundred neighborhood brick-and-mortar video stores that are no longer in existence— are nowhere to be found in Appendix A.

Even when it better reflected the business model of its enterprise the existing operational database of Flix2You suffered from at least one glaring defect of its logical design. As was already pointed out at the beginning of this section, each row that represents a single entity occurrence of the Movies entity class contains a singled-valued store_attribute id, so that if some incarnation of the movie *Godfather Part 2* had formerly been available in all 520 of its regional brick-and-mortar video outlets, the Movies table would once have contained a minimum of 520 rows associated with this single artistic production. The granularity of the Format_Types entity class that participates in a one-to-many relationship with the Movies entity class is not entirely clear (as examples of the domain of its format_type_description Appendix A gives 'DVD' and 'Video', both extremely high-level descriptions), but it nevertheless seems evident that if separate Blu-Ray, DVD and VCR-tape incarnations of the movie *Godfather Part 2* had formerly been available in all 520 of its regional brick-and-mortar video outlets, the Movies table would once have contained a minimum of 3 * 520 or 1,560 rows associated with this single artistic production.

There is also depicted in Appendix A, in a one-to-many relationship with Movies, a Conditions entity class, whose condition_description clearly seems to logically

characterize an individual physical Blu-Ray or DVD disc, or an individual VCR tape.[2] If on the average a half-dozen copies of the movie *Godfather Part 2* had formerly been available in all 520 of its regional brick-and-mortar video outlets in each of the Blu-Ray, DVD and VCR-tape formats, the Movies table would then have contained a minimum of 6 *3 * 520 or 9,360 rows associated with this single artistic production. For a rather extreme example of an update anomaly, consider now the task confronting one who sought to enter all 69 of the credited members of the cast of *Godfather Part 2* into the Movie_Cast composite-entity table, created just to implement the M:N relationship between Movies and Actors: in the hypothetical under construction this simple task would require the insertion of 645,840 table rows.

The actors comprising the cast of a movie, like the movie's title, description, year of release and genre, [3] are all attributes that more immediately characterize a movie when it is considered as the single product of a process of artistic creation, to which Flix2You must purchase its distribution rights from the studio production company that owns them. In an abstract entity class of this kind the movie_title and release_date attributes might together serve as elements of a composite primary key, but in the existing database they are the nonprime elements of an undesirable transitive dependency:



This large redundancy is eliminated in the operational database that this team is recommending to replace the existing design. An entity-relationship diagram depicting the new recommended design is included here as this section's Appendix B.

❖ *Normalized Design for a New Operational Database*

The recommended design depicted in Appendix B replaces the single Movies entity in the existing operational database of Flix2You with a series of entities, of ever-decreasing abstraction:

1. An occurrence of the Movie entity class is an abstract artistic creation of the sort that a person can look up and read about at www.imdb.com. One and the same Movie can be seen at a neighborhood movie theatre, streamed over the internet to a home television, and reviewed in the *New York Times*. The residual intellectual property rights to a Movie are owned by a Studio.

---

[2] The Movies entity class also contains a number_in_stock attribute, which suggests in the alternative that each row in the Movies table represents the collection of discs or tapes in a specified condition and in a specified high-level format, of a particular copyrighted production, that are presently part of the inventory of a store at a particular physical location. This is not necessarily the case, however, since the attributes of a physical instance can also include the numerical size of the collection of which it is a member. And the gist of the present analysis remains unaffected in any case.

[3] The attributes that characterize a movie in this sense, including some additional ones like country of origin and director, are those likely to be of most interest to a customer browsing a movie catalog online, or to Flix2You managers who want to study and analyze customers' purchasing predilections.

2. When described by a consumer-friendly HighLevelFormat like “Blu-Ray” or “Standard-Definition Video” that corresponds to its potential embodiment in an optical disc or downloadable file, a Movie becomes a ViewableEncoding that Flix2You may make available for rent or purchase at a definite price, or that some consumer with a MemberAccount can order through the web site, rate for others, or place in his online movie queue.
3. When a Movie is actually embodied in a concrete file or optical disc its fine-grained format is much more specific than the HighLevelFormat which forms part of the composite key of a ViewableEncoding: Netflix reports, for example, that an average of 120 different combinations of video codec and bitrate are reposed on one or more individual servers at each regional CDN cluster for the purpose of dynamically streaming a single video to its subscribers. The completely specified digital embodiment of a Movie is an entity that in Appendix B is called an OutletResource, a hierarchical supertype that is there in turn decomposed into the mutually distinct hierarchical subtypes OpticalDisc and DownloadableFile. One and the same OutletResource can exist at a multitude of different distribution Outlets (mail centers or server clusters).
4. At the lowest level of abstraction, finally, are the DiscAtMailCenter and FileOnCluster entities, whose primary keys are composites of keys for an Outlet subtype with a definite geographic location, and for one of the two OutletResource subtypes already described above. The entity occurrences that correspond to the rows of these tables still do not correspond to identifiable individual physical objects of the kind that might be given a serial number, but are instead bins for fungible items like a “1/4-inch diameter hex bolt, 8 threads per inch, Alloy 50, zinc-plated”, for which a quantity on hand can be specified in a record's field.

The superannuated Video_Stores entity class is replaced with the Outlet hierarchy of Appendix B's ERD, even though team at present lacks the detailed knowledge of the business processes and rules that it needs to accomplish more than an entirely schematic modeling. The team has also sought to mirror the existing business operations of Flix2You by adding, for example, entity classes and attributes representing customer online account login credentials.

The increased design complexity that results from the measures just described is offset by a considerable simplification of the design used to model the business processes related to renting or purchasing a video disc or video download: the team has verified and is therefore confident that stored procedures making use of no more than standard procedural SQL will be able to efficiently handle all of these, while producing an adequate audit trail, using just the column values in the rows of the TransactedEvent, TransactionType, ViewableEncoding and Customer entity tables. Apart from the trivial and commonly recognized exceptions to the requirements of third normal form like calculated attributes or the transitive dependency associated with the determination of the value of city and state attributes by the value of a postal code, the design depicted in Appendix B is fully 3NF-normalized.

The team also knows that database normalization can often decrease the responsive performance of a database, because it invariably increases the number of necessary but computationally expensive table joins. The team believes, however, that in the

present instance the normalized design that it is recommending will probably, at least for the more common classes of queries, actually increase processing speeds, because of substantial reductions in (1) the number of hard disk sectors that must ordinarily be accessed, locked and read in order to handle the routine query, as well as (2) the size of temporary intermediate tables, containing the Cartesian products of the tables that are to be joined, that are populated before final filtering during processing of the query.

- ***Read-Only OLAP Database, and MOLAP Cube***

When managers of Flix2You query its existing operational database for business-intelligence information customers using the web site to browse the movie catalog are reported to often suffer a degradation of its performance,[4] and these managers themselves report that they typically find it difficult to structure their business-intelligence queries of the operational database in a manner that yields data useful to their analytic purposes. Accordingly the team's design solution also calls for the creation of a separate read-only relational OLAP database that will periodically pull data from the operational OLTP database used to process customer transactions[5] and organize them there in star-schema (or, more accurately, snowflake-schema) form. See the ERD depicting the relational model for this proposed new data repository in Appendix C.

The data that will be accumulated in the data-mart repository whose structure is shown in Appendix C will be immediately accessible by standard SQL and Microsoft Transact-SQL ROLAP extensions like GROUP BY ROLLUP( ) and GROUP BY CUBE( ), for plug-in access by a cube-based vendor dashboard and possibly other data-analytics tools to be developed by the team, or now available on the market. But the team has also taken advantage of the Analysis Services component included in Microsoft SQL Server to design and deploy, atop the ROLAP structure depicted in Appendix C, a proprietary Microsoft MOLAP data cube, whose many capabilities include the ability to export data to an Excel PowerPivot interface. To customize output in the PowerPivot interface a user need only drag available fields into the designated receptacles for row headings, column headings, table values or data filters; drop-down menus allow choice of hierarchical attributes from a list, and + or - buttons allow the user to drill down into or rollup data. The PowerPivot interface is now available to users on mobile devices like

---

[4] The team also recommends that the Flix2You's website be entirely rewritten upon the ASP.NET 4.5 web application platform, using C# code and .aspx pages in conjunction with either the Web Forms or Model-View-Controller APIs. For better security, code maintainability and site performance, database access code now scripted in PHP should be replaced by off-page C# code constructed within Microsoft's Entity Relationship Code First framework, an object relational tool (similar to the JavaEE Persistence API) that allows programmers to directly model database entities, attributes and relationships in instances of C# classes and retrieve or commit data with simplified queries written in its LINQ dialect.

[5] The team populated a sample from the operational database in Management Studio with simple INSERT INTO … FROM SELECT … WHERE … commands, augmented by the use of Transact-SQL functions; had the xml_survey attribute of the sample Customers table already been populated, these latter tools would also include the .query() operation, which makes use of Microsoft's XQUERY extension of the XPATH XML API.

In a production environment, where periodic data loads from the operational database pose the risk of temporarily impairing its essential use, it would be preferable to deploy and make use of a pre-coded, fully tested Microsoft Integration Services project, with controlled data flows implementing data validation, filtering, cleaning and transformation.

the iPad, and an Analysis Services instance operating in SharePoint mode can publish its MOLAP cube to a PowerPivot service hosted by an online SharePoint server.

Microsoft Analysis Services also provides a set of sophisticated algorithms for use in data mining a MOLAP cube, including algorithms capable of predictive as well as explanatory analytics. None of these capabilities will be implemented, however, by the team at the present time.

Database Administration

- ***Roles: User access levels***

The main user roles involved in the Database can be broken down into three categories; the Database Administrator, the Edit (or Read and Write) user, and the Read Only user. The users in each of these categories will have different responsibilities, required knowledge and skills, and levels of access to the database.

The Database Administrator(s), or DB Admin(s), will be responsible for building (implementing the designs), and managing the database. Managing the database will include Backup and Recovery, Access and Security, and Data Load. Each of these areas will be explained in more detail below.

The DB Admin will require a complete understanding of the database design and function, as well as how all data comes in and out of the database, and how the users will be interacting with the database.

The Edit User will be responsible for data entry and data edits. These edits will most likely be made through a web front end, or application, that ties into the database tables. This role will require a good understanding of how the database is used and the flow of data through the system. Since this user will be able to add or edit data within the tables, there is the potential for bad data to enter the system. Therefore, the level of access to write to the database should only be given to users who have a solid understanding of their role, and how the data they add or change will impact the entire database system. The DB Admin will review all web application code that accesses the database backend to verify that the code will not permit the integrity or security of data to be compromised to any degree (see below).

The Read Only role is for users who will only need to view the data contained in the database, but will not need to change the content of that data. Viewing the data will most likely be done via a web front end, or application that allows reporting from or searching of the database for information. Other than an understanding of how to retrieve data, no special knowledge is required for this level of access. This level of access cannot corrupt the data, since these users are only able to view data, but are not able to edit.

❖ *Backup and Recovery*

The servers in our proposed database will be hosted on Microsoft Azure, cloud based servers. Pricing will vary depending on the specs of the hardware and software that are chosen. We recommend Microsoft SQL Server Enterprise Edition, as the Enterprise edition allows for greater scalability, Business Intelligence features, and advanced security features. We also recommend that the SQL Servers be deployed in a cluster of at least two partner servers, to be used in load balancing and as a failover in case one node on the cluster should fail. In this way, there would not be a business interruption in the case of a single server failure.

The hardware specifications are subject to change based on the performance tests on the test Database which will be created. With the Azure cloud services it is very easy to adjust the CPU cores, RAM, and disk space allocated to the SQL server. We will create the server with a middle level processor core, and RAM allocation. Then, during our performance testing of the test database, we will scale the system up or down to match the expected server load. The best practice will be to go slightly above that load expectation so that there is a buffer in place for an unexpected increase. By keeping the specs close to the expected performance load on the server, we will keep the server costs down to the level the need to be for performance, without paying for more robust hardware than will actually be needed. Going forward, as the business demand increases, and the load on the server increases, the DB Admin will be able to monitor the server's workload and easily scale up the specifications to match the demand. This will allow you to only be paying for the hardware you need, as the business demand requires it.

Multiple types of backups should be run on the database. We recommend a weekly Full backup, which makes a complete backup of the database, as well as a daily Differential backup, which backup all of the changes made since the last Full backup. Along with these backups, there will also be a backup of the transaction log that runs every 15 mins. Backing up the transaction log will allow for a point in time restore up to the most recent backup, as well as truncating the transaction logs so that they do not grow too large.

SQL Server backups will be done on a set time schedule determined by the lowest points of activity. These backups will be stored on another partition on the server. The backup files will then be backed up to an "off-site" storage, to ensure that the data is safe in case of any issues with the Azure Cloud servers. This offsite storage location can be a different cloud storage provider, or it could be something as simple as a data storage device (e.g. a Network Attached Storage or NAS) that is located in the Flix2You office (provided that the internet connection the storage device is connected to, has sufficient bandwidth for transmission of the backup files). Backup of transaction log could be done on a more real time basis, but doing so might impact performance.

There are a number of 3rd party backup software/service options that can be selected for the file backup process. The main criteria needed for the file backup would be encryption, compression, and low cpu/ram usage (so that it will not interfere with the server performance).

The recovery process would depend on the specific recovery need. In cases of total server failure (True Disaster Recovery); a new server or servers would be created on the cloud server. The backed up data files on the “off site” storage would be transmitted to the new server. Then an SQL database Restore would be performed, using the recovered backup files. In this way, all data would be recovered up to the point of the last offsite transmission.

In the case of a less drastic failure, the SQL backups that are stored on the secondary partition of the SQL server itself, would be available to use for a restore. In this way, the database could very rapidly be restored (without waiting for the transmission of the offsite data files) to a previous state. With a transaction level backup, the database restore can be done on a very granular level.

- ***Access and Security***

It is recommended that the Microsoft SQL database servers be running in a Windows Domain environment. In a Windows Domain, all user accounts and groups will be managed by the Domain Controller. When a user logs into a workstation or application, that system will perform the authentication through the Domain Controller. In this way, all access can be centrally managed from the Domain Controller. Users will be placed into groups corresponding the the roles listed above (i.e. DB Admins, Edit Users, and Read Only Users). Within SQL Management, access to the databases will be set based on the User Groups managed by the Domain Controller.

Furthermore, the data will be accessed via a web front end or application. More granular access of the data can be set on the application that is used to view the data. In this way, users will only be able to view data that is required for their job function. This is a necessary limiting put in place to preserve customer privacy.

For example, a Sales Associate who is responsible for marketing movies to customers based on gender may have access to customer demographic information connected to gender and viewing history. But that same associate would not have access to that customer’s name, or address information.

All TCP-protocol internet traffic destined to or originating from the company’s database servers, as well as the web servers that house its web application and streamed media files, will be encrypted using the most current version of the Transport Layer Security protocol. Web application API that access the database backend will be periodically reviewed by a security professional with CISSP-equivalent qualifications to ensure that the code is not vulnerable to known SQL-injection, cross-site-scripting or similar exploits. Customer online account passwords will be stored in the database as Base-64-encoded PDKDF2 hash values, salted with SHA-256 pseudo random numbers. Flix2You will not store sensitive customer credit-card information in its database, but will instead make use of reliable, established third-party payment processors.

All Customer data will be kept private, for details on this, please see the privacy policy detailed in Section X. It is also recommended that all employees be required to sign a legal disclaimer agreeing to keep all customer information private.

❖ *Data Load*

Loading the data from the existing production database into the new database will be a multi-step process. Initially a test version of the new database will be constructed. Then a stored procedure* will be run to pull the data from the current production database into the new test system. This process will be monitored for any errors or anomalies that may occur, and it will serve as a baseline to determine the time duration needed to migrate the data. An ETL package with multiple data flows that incorporate necessary data filtering or transformation logic and provide error handling can be created using the capabilities of Microsoft Integration Services. https://msdn.microsoft.com/en-us/library/jj720568.aspx

Once a copy of the production data has been pulled into the test system, a full series of tests will be run on the database and applications to ensure that all systems are functioning as expected, that all data mapped correctly to the tables in the test system, and that all data is accurate. Any adjustments to database design, or programming that are needed will be made during this phase.

The new database system will also be tested for performance. Based on the results, adjustments can easily be made to the cloud based hardware and networking specifications as needed.

Once the test database performs exactly as required, a clean set of data from the old production server can be pulled into the system, overwriting the test data (which may have been manipulated and become unreliable during the testing phase). This will be the preliminary data seeding of the new database.

The final stage is to interrupt data entry for a brief maintenance window, during which the old production database can be set into a read only state and no new data will be added. Customers will experience a brief interruption during this window. During this brief window, the data changes that have occurred since the initial data seeding of the new database, will be copied over. The majority of the data will have already been imported, so the amount of data for those few changes will be small. This will result in a fast transfer of those changes. As soon as the changes are copied over, all web applications and customer interfaces that pointed to the old database system can be repointed to the new system.

All connections will be briefly tested, and following a successful test, the maintenance window can be ended and production can continue on the new database system. In this way, the maintenance window will be very small, and business interruptions will be slight.

Database Dashboard and Analytics

The business-intelligence dashboard capability that the team will design and implement is twofold. Both elements of the planned user interface will present data gleaned predominantly (although not exclusively) from linked read-only OLAP data marts that have been organized by subject-matter.

The first and more static of these user interfaces will allow the user to select from among three separate screens for his viewing, one each for movies, Flix2You's user base, and the company itself. All screens share the same initial options like shortcuts and settings, and each of them allows the user to export data as an Excel file or pdf.

A contextual menu for each of the three screens displays a preconfigured textual list of key performance indicators, trend or correlation visualization aids, and other similar data summaries, each associated with a query that will be sent to the datasource to populate an underlying result-set data model; by dragging items from this list onto some region of the surface of the screen, the user can choose to have the current data returned by the database query visually presented upon the screen every time that he initially selects or afterwards refreshes it. Presentation code in the application transforms the model data into the graphical form in which it can be most effectively displayed, either numerically in tabular form or more vividly as charts and graphs. To remove an item from the screen that no longer interests him the user merely selects and drags it from the window with his mouse.

The first screen of this dashboard component is the movie screen. Selectable options in the contextual menu for this screen provide unit and dollar statistics for a movie that can be viewed by month, quarter, or year. The inclusion of genre, actor cast and director and query attributes makes it easy to draw conclusions about the popularity of each.

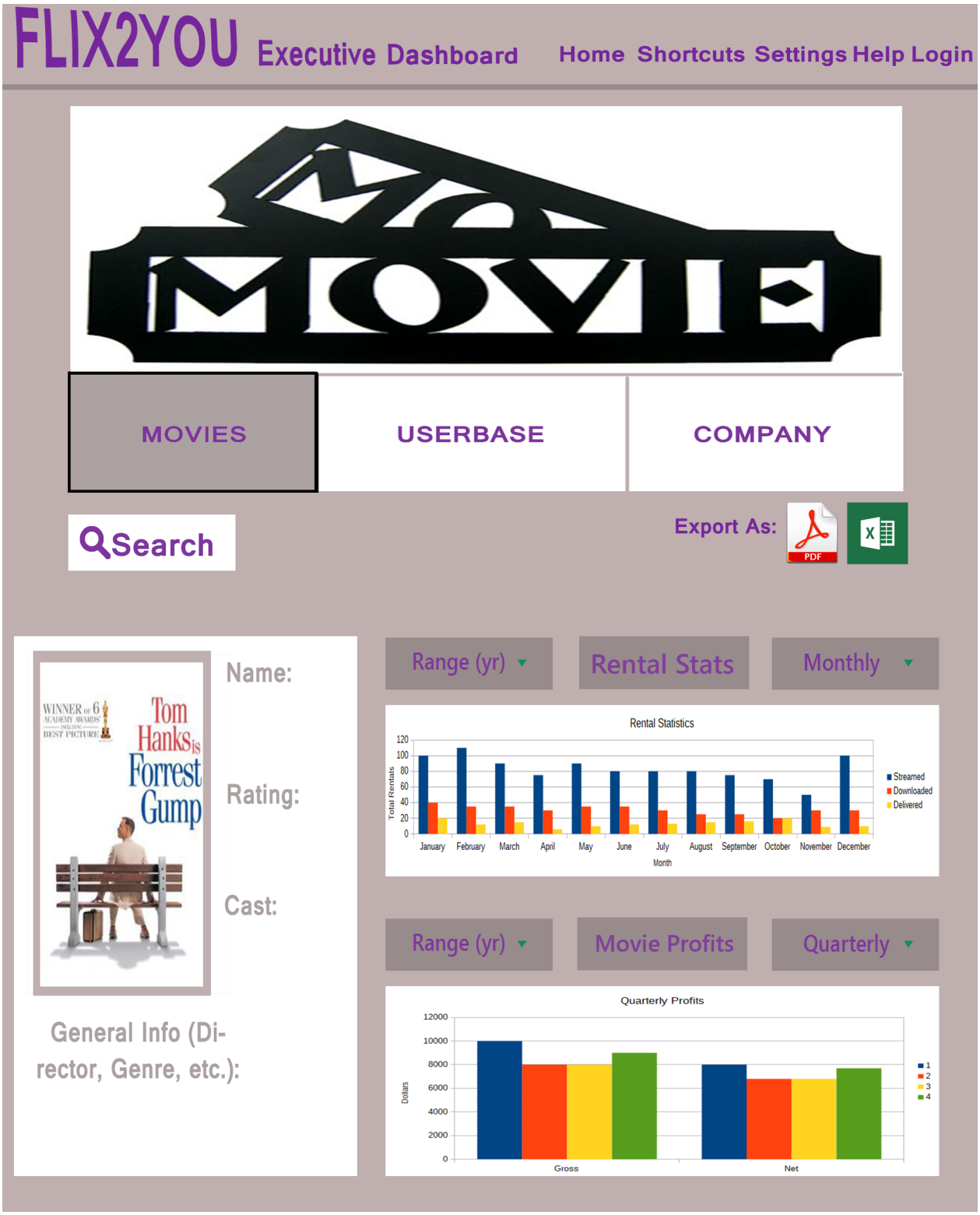
The second dashboard screen displays user-base information. This section provides data on user-base growth, movie rentals, and certain demographic characteristics like age, gender and marital status that are interesting in themselves, but can also be combined with other data (chiefly movie rental data) to reveal correlations. As in the last section all data can be displayed in monthly, quarterly, or yearly distributions, but in this section data can also be displayed in hour-of-the-day and day-of-the-week characteristic distributions, to facilitate inferences about user-base viewing habits and predilections.

The final screen of the preconfigured dashboard interface displays company information. This is the smallest screen, since the information that it can display is concerned mainly with profits. Data viewable in this section include actual and projected company profits and information about new Flix2You members, which resembles the data viewable in the previous section but is also included here to allow conclusions to be drawn about user-base and revenue growth and their probable effects upon profitability.

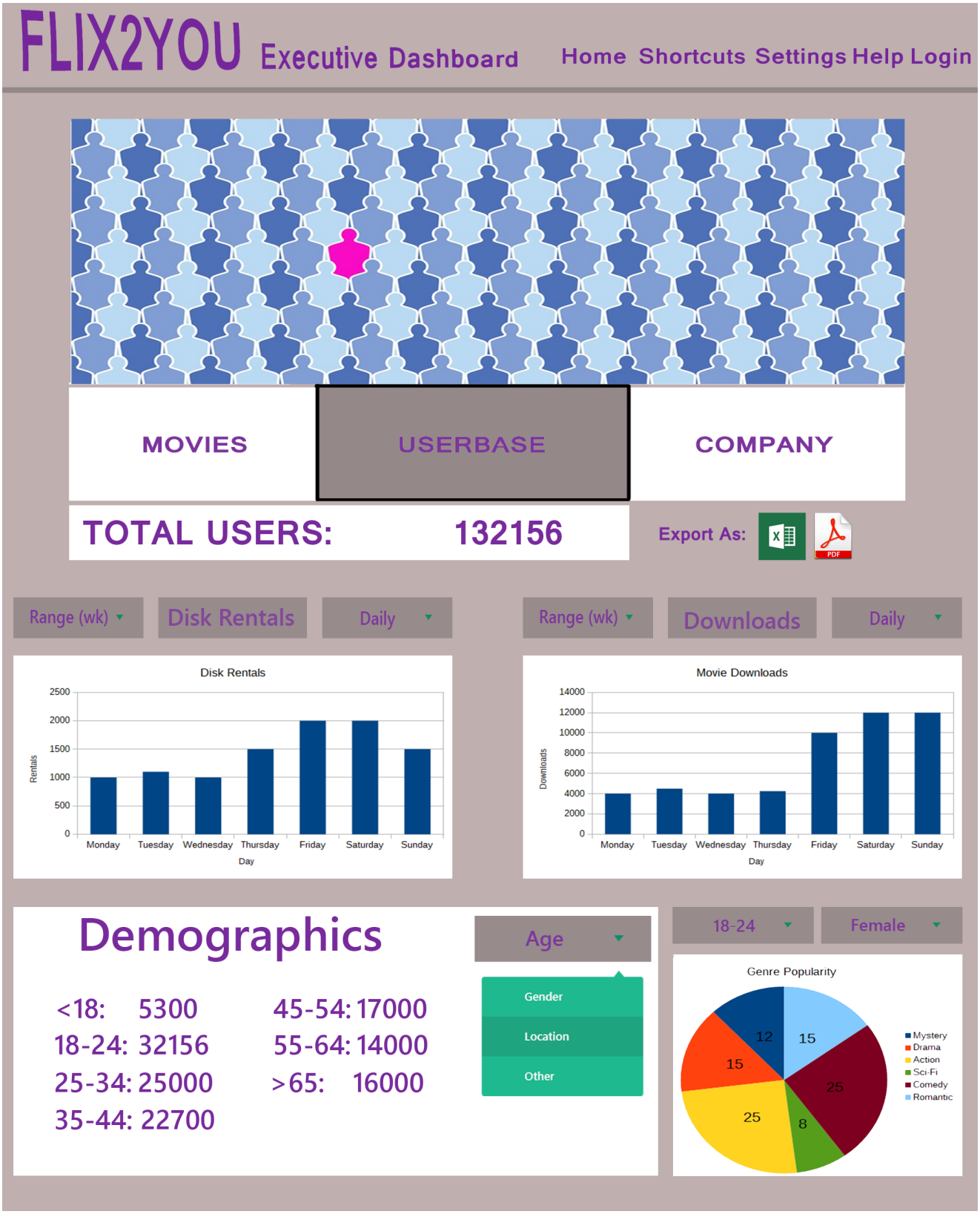
It must be here noted that according to the limited scope of this team's undertaking it has only concerned itself with design of the database schema directly related to sales. Business processes and rules related to product acquisition and distribution, and therefore to direct product costs and company gross profits, are not part of the subject matter of its present engagement, and thus the contribution that the team can make to

the development of the content of this last-described screen will be limited mainly to the subject of revenues.

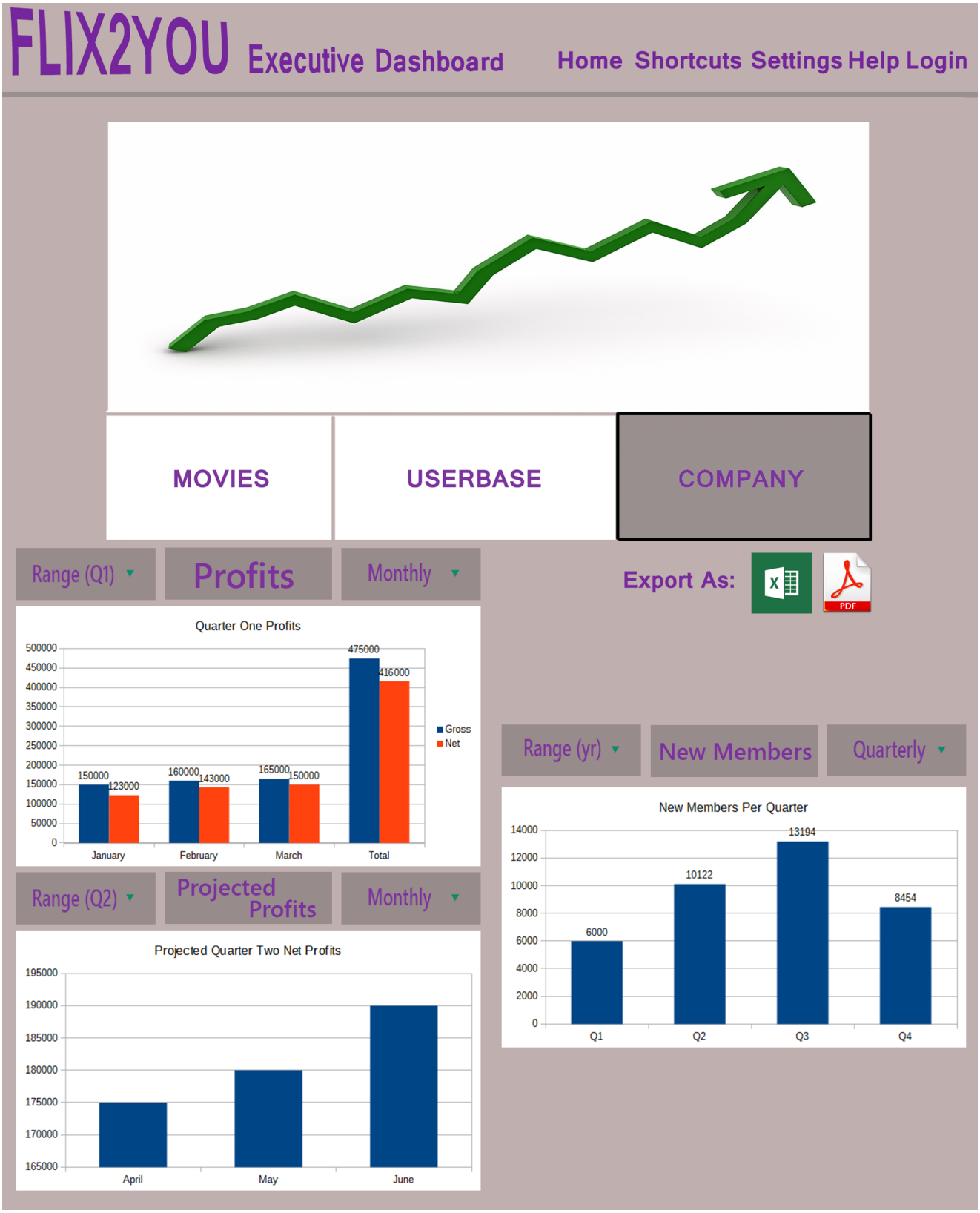
Movie screen:



User-base screen:

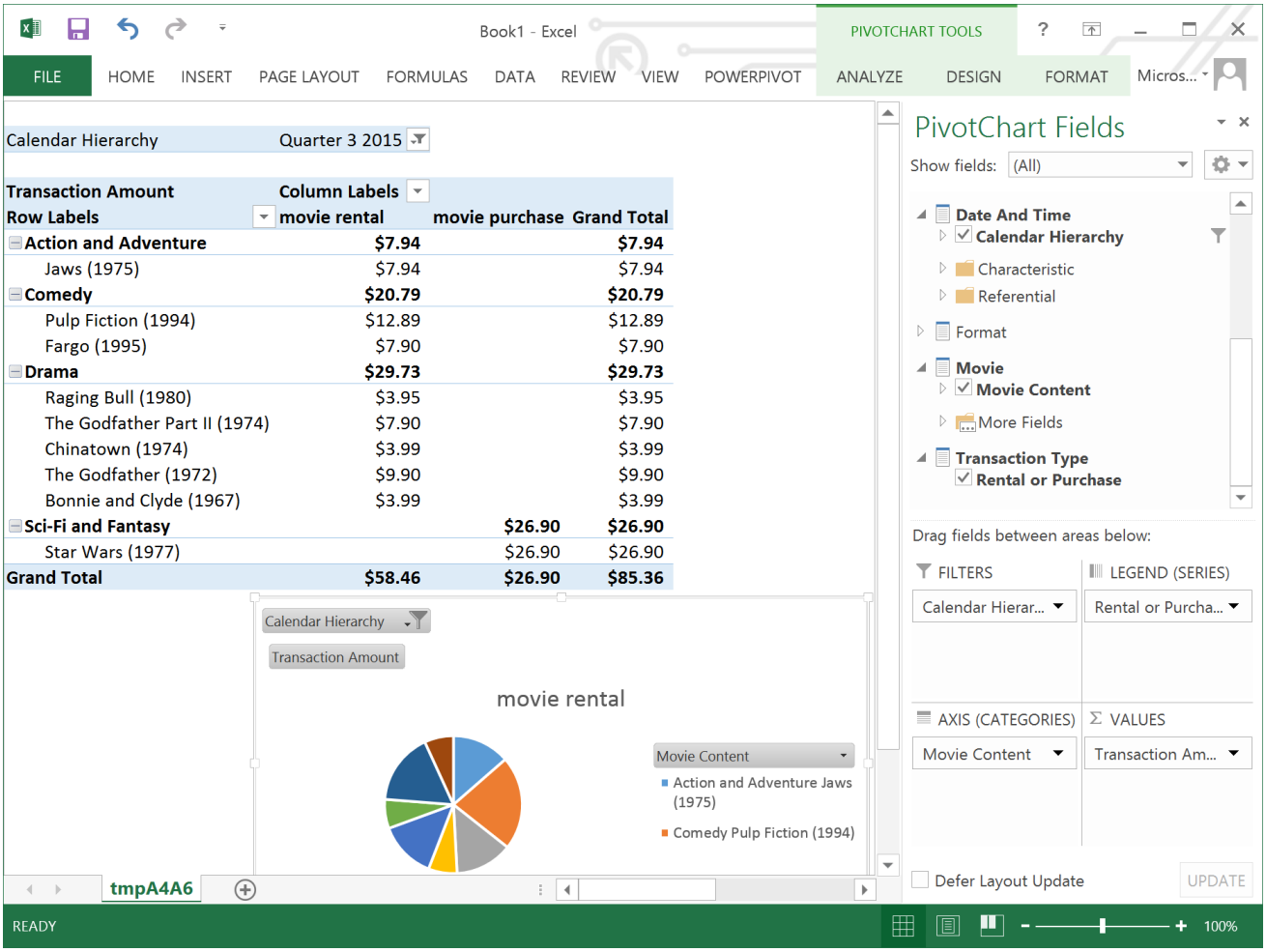


Company screen:

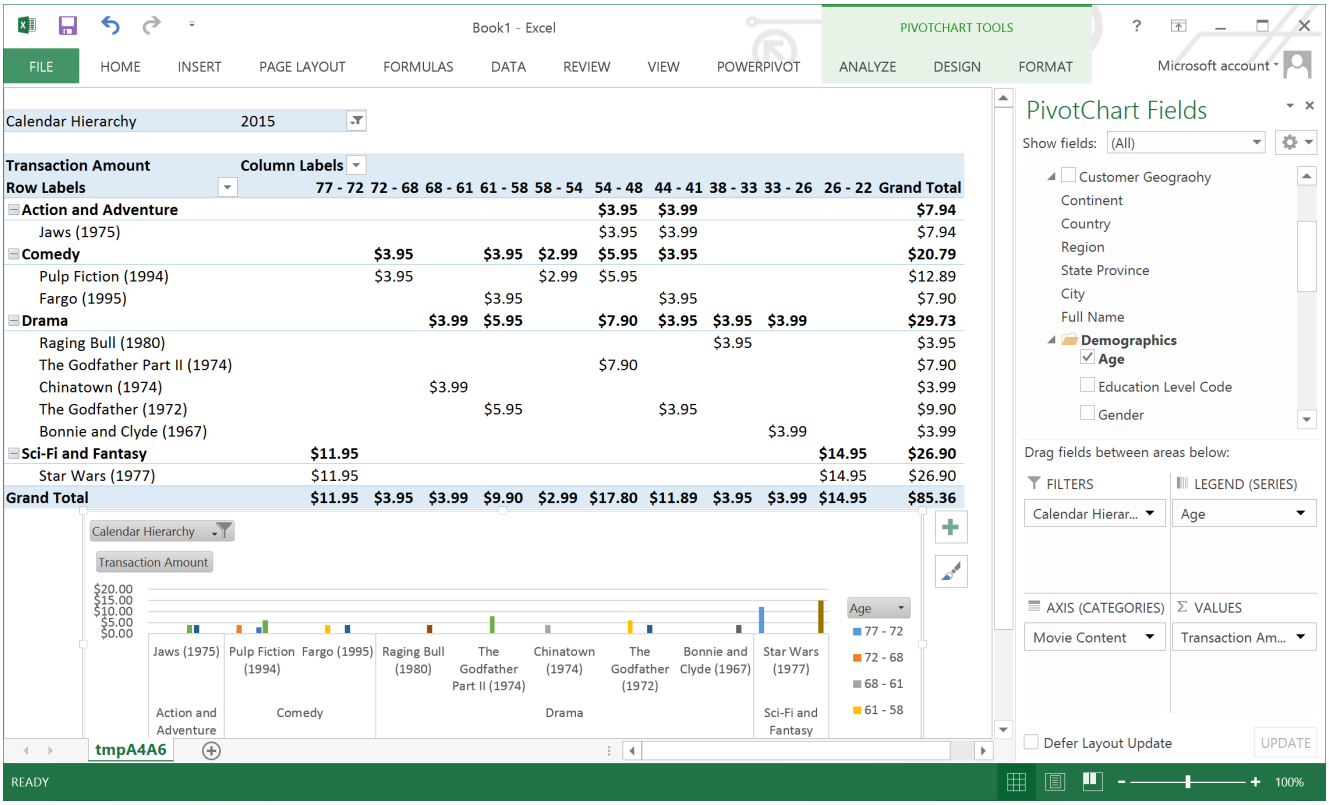


The user interface of second principal component of the dashboard capability that the team proposes to implement for Flix2You is less polished and easy to use, but at the same time more flexible, dynamic, and adaptable, especially for sophisticated management analysts who enjoy a longstanding familiarity with Microsoft Excel as a basic tool of their daily trade. Based upon a MOLAP cube designed and implemented in Microsoft Analysis Services, this interface allows an analyst to quickly construct a sophisticated *ad hoc* query himself, by "slicing and dicing" the underlying sales measure data upon any selection of a myriad of simultaneous dimensions along which the measure data has been classified in the underlying data structure.

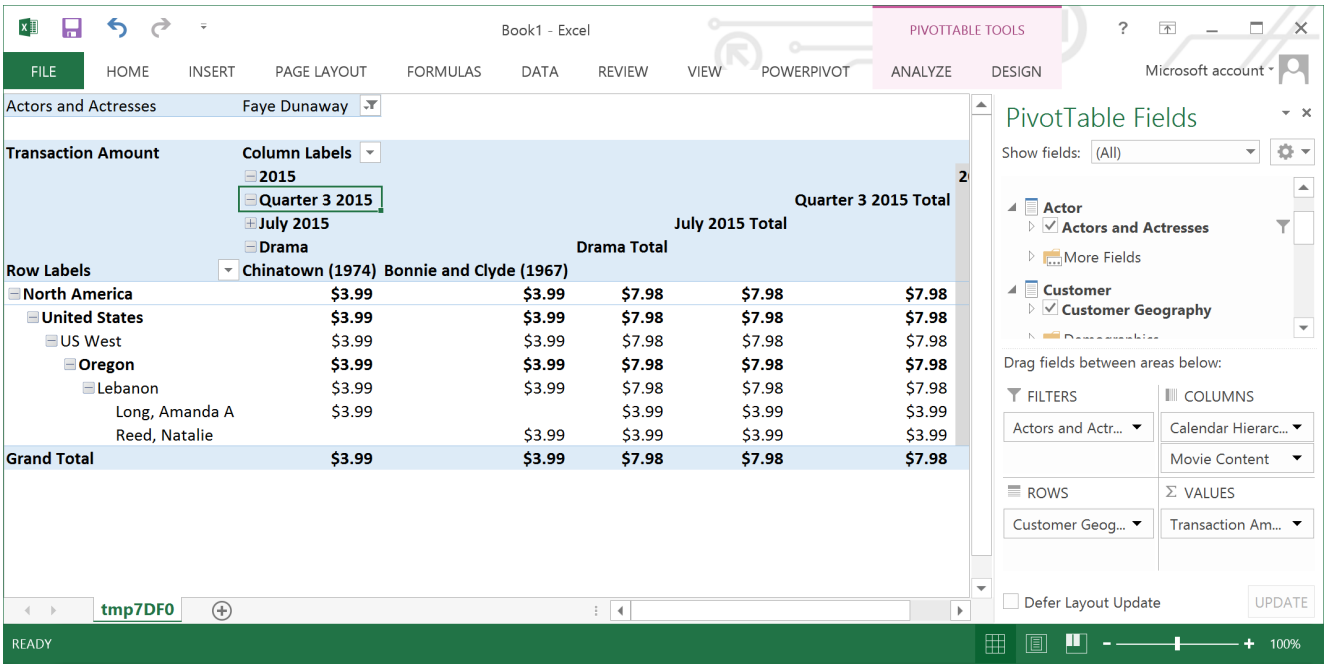
To customize output in the PowerPivot interface exposed by the Analysis Services cube a user need only drag available fields into the designated receptacles for row headings, column headings, table values or data filters; drop-down menus allow choice of hierarchical attributes from a list, and + or - buttons allow the user to drill down into or rollup data. A simple classification of sample movie sales data along the simultaneous dimensions of genre and transaction type (rental or purchase), filtered by calendar hierarchy to select the third fiscal quarter of calendar 2015, looks for example like this:



Substituting a demographic customer age stratum for the transaction-type dimension used as the single column-heading dimension of the output constructed above, but otherwise leaving its analysis parameters unchanged (in the limited sample data set used to test and present this section there is no difference between a filter that limits data to all of calendar 2015, rather than to its third quarter only) modifies the above output to look like this:



A query that classifies movie sales in its columns by the simultaneous dimensions of movie content (genre/title) and calendar hierarchies, and in its rows by the attributes included in the customer geography hierarchy, but which is filtered to limit output to movies in which Faye Dunaway is a member of the cast, looks in PowerPivot like this:



(See Appendix P for two more queries). The PowerPivot interface exposed by the Analysis Services cube can be published to a Microsoft SharePoint server in the cloud, from which it will be available to Flix2You managers on mobile devices like smart phones and tablets. The ability to illustrate the true analytic power of the interface that it creates is unfortunately limited by the small size of the sample data set that is so far available.

The SQL code used to build and load the ROLAP database that these business-intelligence interfaces will query, like the code used to build the OLTP database from which its content will be periodically loaded, is included in this document as an appendix. So too is sample SQL code (including Transact-SQL extensions like GROUP BY CUBE()) illustrating business-intelligence queries of the sort that can be used to populate the data models underlying the pre-formulated queries of the first component of the dashboard. But the construction of the Analysis Services MOLAP cube can only be viewed in the Microsoft Visual Studio IDE, by loading the solution file associated with its SSAS project; the queries that it executes "under the hood" are formulated by Analysis Services in MDX, a proprietary Microsoft language introduced in 1997 as part of its OLE DB for OLAP specification.

Legal Issues

1. **Privacy Policy:**
    a. **Collection of Information:** We receive and store information about you such as:
    - Information you provide us: We collect information you provide to us. Examples may include:
        - your name, email address, address or postal code, payment method and telephone number. We collect this information in a number of ways, including when you enter it while using our service, interact with our customer service, or participate in surveys or marketing promotions; and
        - information collected when you choose to provide reviews or ratings, set preferences in Your Account or otherwise provide information to us through our service or elsewhere.
    - Information we collect automatically: We collect information regarding you and your interactions with us and our advertising, your use of our service, applications, sites, tools, and customer service, as well as information regarding your computer or other device used to access our service (such as gaming systems, smart TVs, mobile devices, and set top boxes). This information may include:
        - your activity on the Flix2You service, such as title selections, watch history and search queries;
        - details regarding your interactions with customer service, such as the date, time and reason for contacting us, transcripts of any chat conversations, and if you call us, your phone number;
        - device IDs or unique identifiers, device and software characteristics (such as type and configuration), connection information, statistics on page views, referral URLs, ad data, IP address and standard web log information;
        - information collected via the use of cookies, web beacons and other technologies.

    b. **Use of Information:** We use the information we collect to provide, analyze, administer, enhance and personalize our services and marketing efforts for you and others, including to process your registration, your orders and your payments, and to communicate with you on these and other topics.

    c. **Disclosure of Information:** Except for name and other contact information, we may share information collected from or about you with our business partners, including advertisers, content licensors, distributors, and other companies that are not affiliated with Flix2You. We may share the information collected from or about you in aggregated or de-identified forms with advertisers and service providers that perform advertising-related services for us and our business partners in order to tailor advertisements, measure advertising effectiveness, and enable other enhancements. When you choose to share information with social networking services about your activities on Flix2You, including shows you

watch or like on Flix2You, Flix2You may share information about you and your activities with that social network.

d. **Cookies and Internet Advertising:** We and our service providers use cookies and other technologies (such as web beacons) for various reasons. For example, we use cookies and other technologies to make it easy to access our services by remembering you when you return, to provide and analyze our services, to learn more about our users and their likely interests, and to deliver and tailor marketing or advertising. We want you to be informed about our use of these technologies, so this notice explains the types of technologies we use, what they do and your choices regarding their use.

2. **Terms and Conditions for Use:**
    a. **Overview:** Flix2You is a subscription service that provides our members with access to motion pictures, television and other audio-visual entertainment ("movies & TV shows") streamed over the Internet to certain Internet-connected TV's, computers and other devices. These Terms of Use govern your use of our service. As used in these Terms of Use, "Flix2You service," "our service" or "the service" means the service provided by Flix2You for discovering and watching movies & TV shows, including all features and functionalities, website, and user interfaces, as well as all content and software associated with our service.
    b. **Acceptance of Terms of Use:** These Terms of Use, which include our Privacy Policy, govern your use of the Flix2You service. By using, visiting, or browsing the Flix2You service, you accept and agree to these Terms of Use. If you do not agree to these Terms of Use, do not use the Flix2You service.
    c. **Changes to Terms of Use:** Flix2You may, from time to time, change these Terms of Use, including the Privacy Policy. Such revisions shall be effective immediately; provided however, for existing members, such revisions shall, unless otherwise stated, be effective 30 days after posting
    d. **Communication Preferences:** By using the Flix2You service, you consent to receiving electronic communications from Flix2You relating to your account. These communications may involve sending emails to your email address provided during registration, or posting communications on the Flix2You service, or in the "Your Account" page and will include notices about your account (e.g., payment authorizations, change in password or Payment Method, confirmation e-mails and other transactional information) and are part of your relationship with Flix2You.
    e. **Membership, Billing and Cancellation:** Your Flix2You membership will continue month-to-month unless and until you cancel your membership or we terminate it. You must have Internet access and provide us with a current, valid, accepted method of payment (as such may be updated from time to time, "Payment Method") to use the Flix2You service. We will bill the monthly membership fee to your Payment Method. We reserve the right to adjust pricing for our service or any components thereof in any

manner and at any time as we may determine in our sole and absolute discretion. Except as otherwise expressly provided for in these Terms of Use, any price changes to your service will take effect following email notice to you.You may cancel your Flix2You membership at any time, and you will continue to have access to the Flix2You service through the end of your monthly billing period.

f. **Flix2You Service:** You must be 18 years of age, or the age of majority in your province, territory or country, to become a member of the Flix2You service. Individuals under the age of 18, or applicable age of majority, may utilize the service only with the involvement of a parent or legal guardian, under such person's account and otherwise subject to these Terms of Use. The Flix2You service, and any content viewed through our service, are for your personal and non-commercial use only. You agree to use the Flix2You service, including all features and functionalities associated therewith, in accordance with all applicable laws, rules and regulations, or other restrictions on use of the service or content therein.
g. **Passwords and Account Access:** The member who created the Flix2You account and whose Payment Method is charged is referred to here as the Account Owner. The Account Owner has access and control over the Flix2You account. The Account Owner's control is exercised through use of the Account Owner's password and therefore to maintain exclusive control, the Account Owner should not reveal the password to anyone.
h. **Disclaimers of Warranties and Limitations on Liability:** THE FLIX2YOU SERVICE AND ALL CONTENT AND SOFTWARE ASSOCIATED THEREWITH, OR ANY OTHER FEATURES OR FUNCTIONALITIES ASSOCIATED WITH THE FLIX2YOU SERVICE, ARE PROVIDED "AS IS" AND "AS AVAILABLE" WITH ALL FAULTS AND WITHOUT WARRANTY OF ANY KIND. FLIX2YOU DOES NOT GUARANTEE, REPRESENT, OR WARRANT THAT YOUR USE OF THE FLIX2YOU SERVICE WILL BE UNINTERRUPTED OR ERROR-FREE.
i. **Intellectual Property:** The Flix2You service, including all content provided on the Flix2You service, is protected by copyright, trade secret or other intellectual property laws and treaties. Flix2You is a trademark of Flix2You, Inc.
j. **Governing Law:** These Terms of Use shall be governed by and construed in accordance with the laws of the state of Pennsylvania, U.S.A. without regard to conflict of laws provisions. You may also be entitled to certain consumer protection rights under the laws of your local jurisdiction.

References

Advanced Television Systems Committee Standards. (n. d.). In *Wikipedia*. Retrieved July 29, 2015, from https://en.wikipedia.org/wiki/Advanced_Television_Systems_Committee_standards.

Blu-ray. (n. d.). In *Wikipedia*. Retrieved July 29, 2015, from https://en.wikipedia.org/wiki/Blu-ray.

DVD. (n. d.). In *Wikipedia*. Retrieved July 29, 2015, from https://en.wikipedia.org/wiki/DVD.

Ganz, Jr, C. G. (2010, July 20). *Migrating Data from Database to Database*. Retrieved 07 20, 2015, from SQL Server Pro: http://sqlmag.com/sql-server/migrating-data-database-database

Hulu, Inc. (2015, July 14) Privacy Policy and Terms of Use. Retrieved July 14, 2015 from: http://www.hulu.com/terms and http://www.hulu.com/privacy

Kurose, J. F., & Ross, K. W. (2013). *Computer networking: a top-down approach, 6th edition* (chapter 7, multimedia networking, section 7.2, streaming stored video). Saddle River, NJ: Addison-Wesley.

Microsoft Corporation. (2014, September 8). Getting started with ASP.NET 4.5 Web Forms and Visual Studio 2013. Retrieved July 29, 2015, from https://www.asp.net/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/introduction-and-overview.

Microsoft Corporation. (2015, July 29). Analysis Services tutorials, multidimensional modeling (Adventure Works Tutorial). In *Microsoft Developer Network Library, servers and enterprise development, SQL Server, SQL Server 2012, product documentation, tutorials for SQL Server 2012.* Retrieved July 29, 2015, from https://msdn.microsoft.com/en-us/library/ms170208(v=sql.110).aspx.

Microsoft Corporation. (2015, July 29). Integration Services tutorials, SSIS Tutorial: creating a simple ETL package. In *Microsoft Developer Network Library, servers and enterprise development, SQL Server, SQL Server 2014, product documentation, tutorials for SQL Server 2014.* Retrieved July 29, 2015, from https://msdn.microsoft.com/en-US/library/ms169917(v=sql.120).aspx.

Microsoft Corporation. (2015, July 29). Transact-SQL language reference (database engine). In *Microsoft Developer Network Library, servers and enterprise development, SQL Server, Microsoft SQL Server language reference.* Retrieved July 29, 2015, from https://msdn.microsoft.com/en-us/library/bb510741.aspx.

Microsoft Corporation. (2015). *SQL Server Editions*. Retrieved July 20, 2015, from Microsoft.com: https://www.microsoft.com/en-us/server-cloud/products/sql-server-editions/default.aspx

Moreau, T. (2007, March). *Top Tips for SQL Server Clustering*. Retrieved July 20, 2015, from TechNet Magazine: https://technet.microsoft.com/en-us/magazine/2007.03.sqlclusters.aspx.

Netflix, Inc. (2015, July 14) Privacy Policy and Terms of Use. Retrieved July 14, 2015 from: https://www.netflix.com/PrivacyPolicy and https://www.netflix.com/TermsOfUse

Ozer, J. (2014, January 15). How to produce high-quality H.264 video files. *Streaming Media, January/February 2014.* Retrieved July 29, 2015, from http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/How-to-Produce-High-Quality-H.264-Video-Files-94216.aspx.

Randal, P. S. (2009, July). *Understanding SQL Server Backups*. Retrieved July 20, 2015, from TechNet Magazine: https://technet.microsoft.com/en-us/magazine/2009.07.sqlbackup.aspx.

Roku, Inc. (2015, July 29) Encoding guide. In *Roku Dashboard, Roku Developer Documentation Wiki, Roku SDK documentation, component reference*. Retrieved July 29, 2015, from http: Retrieved July 29, 20125, from //sdkdocs.roku.com/display/sdkdoc/Encoding+Guide - EncodingGuide-21RokuStreamingPlayerRequirements.

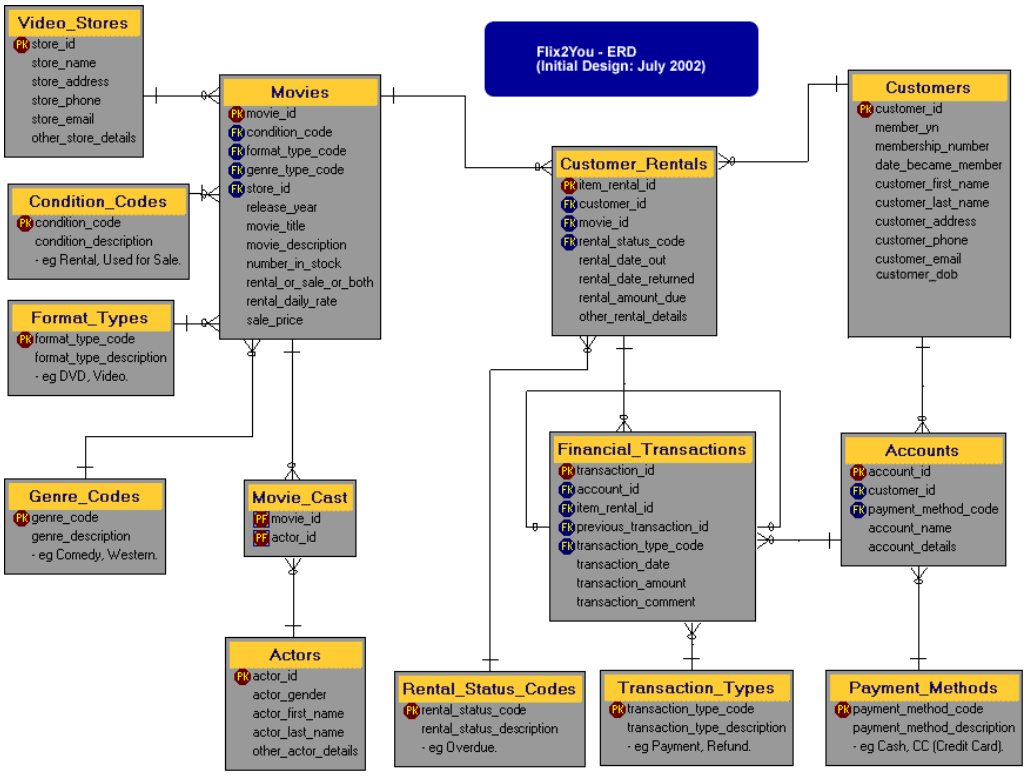
APPENDICES

ERD of Existing Database ..... Appendix A
ERD of Proposed Sales Schema, Operational Database ..... Appendix B
ERD for Snowflake Schema, Proposed ROLAP Data Mart ..... Appendix C
Data Dictionary, OLTP Sales Schema ..... Appendix D
SQL code:
- DDL build of OLTP database structures ..... Appendix E
- DML load of sample OLTP data ..... Appendix F
- DDL build of OLAP snowflake schema structure ..... Appendix G
- DML load from OLTP to OLAP database ..... Appendix H
- Procedural SQL for transaction processing ..... Appendix I
- Execution of stored procedures to process rentals/purchases ..... Appendix J
- SQL for use in web catalog-browsing code ..... Appendix K
- SQL for use in business-intelligence applications ..... Appendix L

SQL output:
- From code for routine business transaction processing ..... Appendix M
- From code for online movie catalog browsing ..... Appendix N
- From code for business-intelligence applications ..... Appendix O

Analysis Services MOLAPCube/PowerPivot output ..... Appendix P

Current Database Schema:

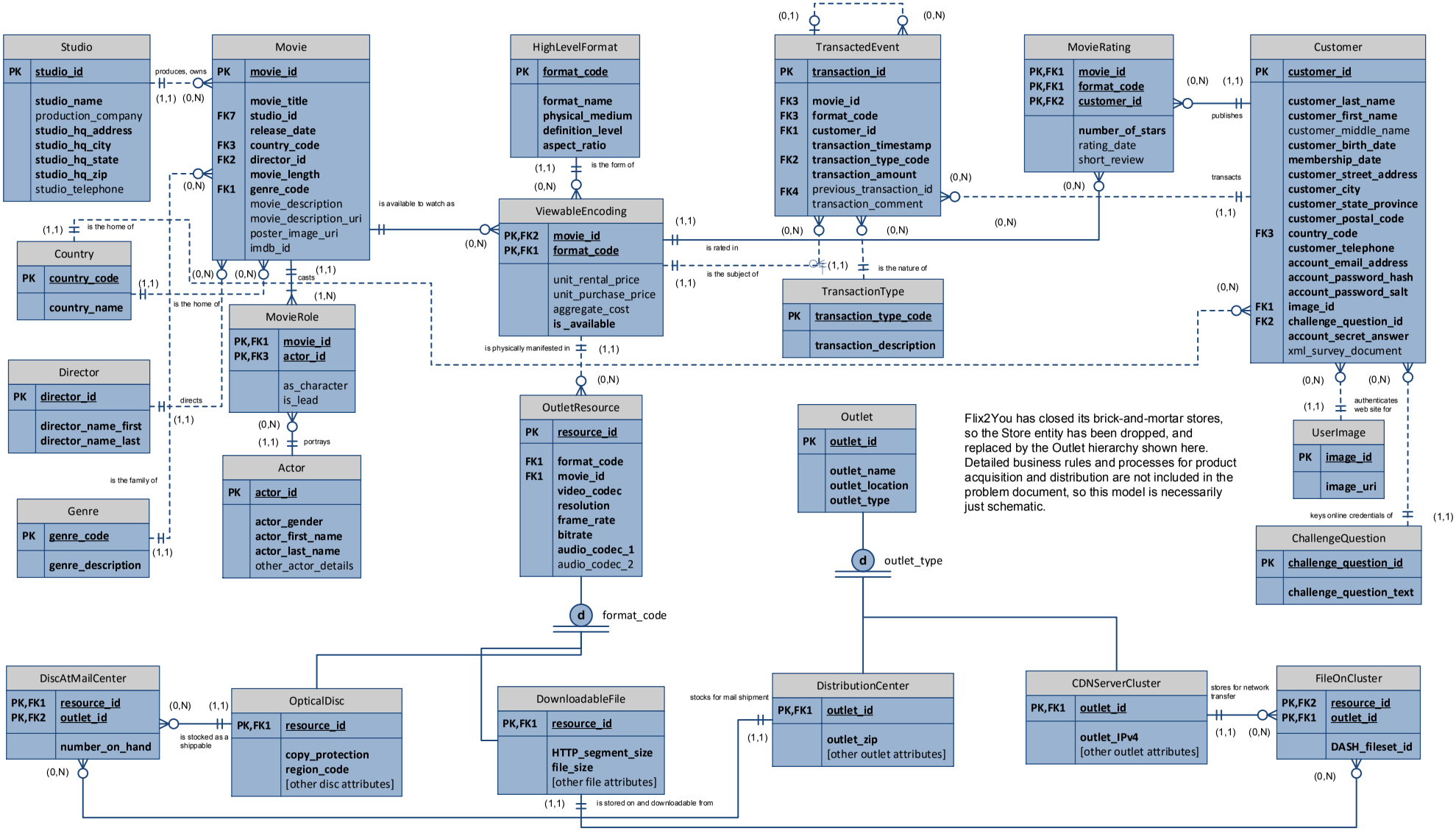


© Copyright 1999-2014 The Penn State State University World Campus
College of Information Sciences and Technology

APPENDIX B

Flix2You: Proposed ERD for Sales-Related Operational Database Schema
June 13, 2015

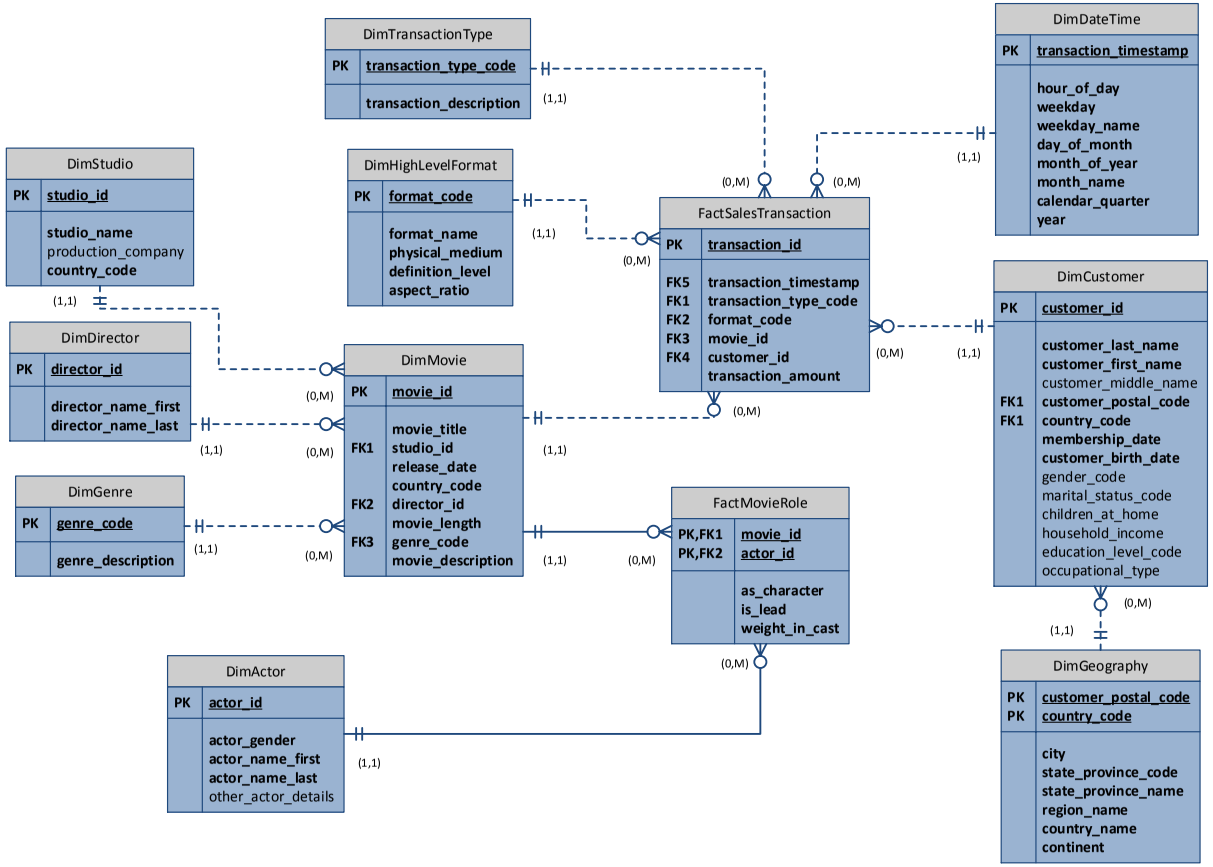
NOTES. Simplification of the right-hand side of this ERD has resulted from the following choices and/or assumptions:
1. All transactions are online, via credit card, so there is no need for a payment-type entity. Terms are schedule shipment/prepare to upload upon credit-card authorization, and collect credit-card payment upon shipment/successful download, so there are no open account balances.
2. For better security Flix2You's website will use reputable third-party payment processors, and will not itself store sensitive credit-card data.
3. Separate member IDs seemed redundant, and the purpose of the Account entity (more than one to a customer) was unclear. These were jettisoned.



APPENDIX C

Flix2You

Snowflake Schema for ROLAP Data Mart



Attribute hierarchies:

year, calendar_quarter, month, day_of_month

continent, country_name, region_name, state_province, city, postal_code

genre_code, movie_code

Flix2You
Data Dictionary for Sales Schema of Proposed Operational Database

Table	Column	Comment	Data Type	Format	Constraint	Required	Key	Reference
Sales.Movie	movie_id	EIDR, an industry-standard identifier like a book ISBN, minus a redundant movies prefix. See http://eidr.org	char(26)	'([0-9A-F]{4, 4}-){5, 5}[0-9A-Z]'		x	PK	
	movie_title		nvarchar(50)			x		
	studio_id		char(9)			x	FK	Sales.Studio(studio_id)
	release_date		date			x		
	country_code		nchar(2)			x	FK	Sales.Country(country_code)
	director_id		smallint			x	FK	Sales.Director(director_id)
	movie_length	measured in minutes	smallint			x		
	genre_code		tinyint			x	FK	Sales.Genre(genre_code)
	movie_description		nvarchar(max)					
	movie_description_uri	link usable by the Flix2You website	nvarchar(100)					
	movie_poster_uri	link usable by the Flix2You website	nvarchar(100)					
	imdb_id	appended as a path suffix to the web URL of the Internet Movie Database, this string, of the form "tt9999999", displays all or links to all of the information about a movie that is available at that site. It is readily available (in the EIDR.org record, for example), and may serve as the parameter of a RESTful web service.	nchar(9) Text					
Sales.Country	country_code		nchar(2)			x	PK	
	country_name		nvarchar(35)		UNIQUE	x		
Sales.Studio	studio_id	another standard EIDR identifier	char(9)	'[0-9A-F]{4, 4}-[0-9A-F]{4, 4}'		x	PK	
	studio_name		nvarchar(50)			x		
	production_company	studios often operate through several	nvarchar(50)					
	studio_hq_address		nvarchar(35)			x		
	studio_hq_city		nvarchar(35)			x		
	studio_hq_state		nchar(2)			x		
	studio_hq_postal_code		nchar(10)			x		
	country_code		nchar(2)			x	FK	Sales.Country(country_code)
	studio_telephone		char(14)	'(999) 999-9999'		x		
Sales.Director	director_id		smallint		IDENTITY(1,1)	x		
	director_name_first		nvarchar(35)			x		
	director_name_last		nvarchar(35)			x		
Sales.Genre	genre_code	the 1:M relationship of Genre to Movie may prove unworkable	tinyint			x	PK	
	genre_description		nvarchar(35)		UNIQUE	x		
Sales.Actor	actor_id		smallint		IDENTITY(1,1)	x	PK	
	actor_gender		char(1)		IN ('M', 'F')	x		
	actor_name_first		nvarchar(35)			x		
	actor_name_last		nvarchar(35)			x		
	other_actor_details		nvarchar(max)					
Sales.MovieRole	movie_id		char(26)			x	PK1, FK	Sales.Movie(movie_id),
	actor_id		smallint			x	PK2, FK	Sales.Actor(actor_id)
	as_character	names the fictional person portrayed	nvarchar(50)					
	is_lead	sometimes hard to pick just 1 per sex	bit					
Sales.HighLevelFormat	format_code	*e.g.* , 'DVD-1'	char(6)			x	PK	
	format_name	*e.g.* , 'High-Definition Stream'	varchar(35)		UNIQUE	x		
	physical_medium	'disk' or 'file'	char(4)		UNIQUE combination;	x		
	definition_level	*e.g.* , 'high', 'standard'	varchar(8)		physical_medium IN ('disk',	x		
	aspect_ratio	*e.g.* , 'widescreen', 'letterbox'	varchar(15)		'file')	x		

Flix2You
Data Dictionary for Sales Schema of Proposed Operational Database

Table	Column	Comment	Data Type	Format	Constraint	Required	Key	Reference
Sales.ViewableEncoding	movie_id		char(26)			x	PK1, FK	Sales.Movie(movie_id)
	format_code		char(6)			x	PK2, FK	Sales.HighLevelFormat(format_code)
	rental_price		money					
	purchase_price		money					
	aggregate_cost	intended to include the cost of the license to distribute the movie, and of one reproducible encoded set	money					
	is_available	in this form, from an accessible outlet	bit			x		
Sales.OutletResource	resource_id	as child of parent HighLevelEncoding	int		IDENTITY(1, 1)	x	PK	
	movie_id		char(26)			x	FK	Sales.Movie(movie_id)
	format_code	discriminator for its two subclasses	char(6)			x	FK	Sales.HighLevelFormat(format_code)
	video_codec	*e.g.* , 'MPEG-2/H.262',' MPEG-4/H.264'	varchar(8)			x		
	resolution	*e.g* ., '1920i', '1080p'	varchar(10)			x		
	frame_rate	fps: 24 and 60 are common	smallint			x		
	bitrate	kbps: 768-62,500 current range	int			x		
	audio_codec_1	*e.g.* , 'AC-3', 'DTS', 'LPCM', 'MP-2'	varchar(8)			x		
	audio_codec_2	*e.g.* , 'AC-3', 'DTS', 'LPCM', 'MP-2'	varchar(8)					
Sales.OpticalDisc	resource_id	ID of its hierarchical superclass	int			x	PK, FK	Sales.OutletResource(resource_id)
	copy_protection	'CSS', 'Macrovision',...	varchar(8)					
	region_code	officially, 0-9	tinyint					
Sales.DownloadableFile	resource_id	ID of its hierarchical superclass	int			x	PK, FK	Sales.OutletResource(resource_id)
	HTTP_segment_size	in bytes	smallint			x		
	file_size	in bytes	bigint			x		
Sales.Outlet	outlet_id		smallint		IDENTITY(1, 1)	x	PK	
	outlet_name		nvarchar(35)			x		
	outlet_location		nvarchar(50)			x		
	outlet_type	discriminator for its two subclasses	char(1)		IN ('M', 'F')	x		
Sales.DistributionCenter	outlet_id	ID of its hierarchical superclass	smallint			x	PK, FK	Sales.Outlet(outlet_id)
	outlet_postal_code		char(10)			x		
Sales.CDNCluster	outlet_id	ID of its hierarchical superclass	smallint			x	PK, FK	Sales.Outlet(outlet_id)
	outlet_IPv4		varchar(11)			x		
Sales.DiscAtMailCenter	outlet_id		smallint			x	PK1, FK	Sales.Outlet(outlet_id)
	resource_id		int			x	PK2, FK	Sales.OutletResource(resource_id)
	number_on_hand	an inventory count of physical items	smallint			x		
Sales.FileOnCluster	outlet_id		smallint			x	PK1, FK	
	resource_id		int			x	PK2, FK	
	DASH_fileset	Dynamic Adaptive Streaming over HTTP	smallint			x		
Sales.Customer	customer_id		int		IDENTITY(1,1)	x	PK	
	customer_last_name		nvarchar(35)			x		
	customer_first_name		nvarchar(35)			x		
	customer_middle_name		nvarchar(35)					
	customer_birth_date		date		DATEDIFF(year, customer_birth_date, GETDATE()) BETWEEN 16 AND 95	x		
	membership_date		date		BETWEEN '1995-01-01' AND GETDATE()	x		
	customer_street_address		nvarchar(35)			x		
	customer_city		nvarchar(35)			x		
	customer_state_province	assumed international operation	nchar(2)			x		
	customer_postal_code	Canadian cities may share one	nchar(10)			x		
	country_code	assumed international operation	nchar(2)		DEFAULT 'US',	x	FK	Sales.Country(country_code)
	customer_telephone		char(14)	'(999) 999-9999'		x		
	account_email_address		nvarchar(100)	can't validate syntax in SQL		x		

Flix2You

Data Dictionary for Sales Schema of Proposed Operational Database

Table	Column	Comment	Data Type	Format	Constraint	Required	Key	Reference
	account_password_hash	Base-64 encoding of a PBKDF2 hash of the user's salted password	char(48)			x		
	account_password_salt	Base-64 encoding of an HMAC-1 SHA-256 pseudo random number	char(48)			x		
	image_id		smallint			x	FK	Sales.UserImage(image_id)
	challenge_question_id		smallint			x	FK	Sales.ChallengeQuestion(challenge_question_id)
	account_secret_answer	*e.g.* , "O'Hallihan" (see below)	nvarchar(250)			x		
	xml_survey_document	to store demographic data in an XML format. Extractible , *e.g.* , by SELECT xml_survey_document.query('/IndividualSurvey/MaritalStatus'). An XML schema is available in the MSDN AdventureWorks tutorial.	xml					
Sales.UserImage	image_id	image will authenticate the web site	smallint		IDENTITY(1,1)	x	PK	
	image_uri	(into a directory on the web server)	nvarchar(100)		UNIQUE	x		
Sales.ChallengeQuestion	challenge_question_id		smallint		IDENTITY(1,1)	x	PK	
	challenge_question_text	*e.g.* , "mother's maiden name?"	nvarchar(100)		UNIQUE	x		
Sales.TransactionType	transaction_type_code		tinyint			x	PK	
	transaction_description	see the constraint. In the redesigned operational database schema separate but linked related transactions are used to maintain state for an ongoing active rental or purchase.	varchar(50)		UNIQUE; IN('movie rental', 'movie purchase', 'credit-card authorization', 'credit-card payment', 'credit-card refund', 'file transfer', 'disk shipment', 'disc return', 'other')	x		
Sales.TransactedEvent	transaction_id		bigint	*from:*	Sales.TransactionSequence	x	PK	
	movie_id		char(26)			x	FK	Sales.Movie(movie_id),
	format_code		char(6)			x	FK	Sales.HighLevelFormat(format_code)
	customer_id		int			x	FK	Sales.Customer(customer_id
	transaction_timestamp		datetime		DEFAULT GETDATE()	x		
	transaction_type_code	see the expanded entity semantics	tinyint			x	FK	Sales.TransactionType(transaction_type_code)
	transaction_amount		money		DEFAULT 0.00	x		
	previous_transaction_id	much use is made is made of this attribute in the redesign, to replace the Customer_Rentals entity with a unary relationship. NULL only for the original rental or purchase.	bigint				FK	Sales.TransactedEvent(transaction_id)
	transaction_comment		varchar(300)					
					UNIQUE(movie_id, format_code, customer_id, transaction_timestamp, transaction_type_code)			
SEQUENCE Sales.TransactionSequence AS bigint								
	START WITH 1							
	INCREMENT BY 1							
	MINVALUE 1							
	MAXVALUE 9223372036854775807;							

Flix2You

Data Dictionary for Sales Schema of Proposed Operational Database

Table	Column	Comment	Data Type	Format	Constraint	Required	Key	Reference
INDEX idxTransaction_Customer_Date ON Sales.TransactedEvent(customer_id, transaction_timestamp DESC);								
INDEX idxTransaction_Movie ON Sales.TransactedEvent(movie_id);								
Sales.QueuedMovie	movie_id	the ability to queue and review movies might be used to induce the customer to surrender demographic data in the above survey	char(26)			x	PK1, FK	Sales.Movie(movie_id)
	format_code		char(6)			x	PK2, FK	Sales.HighLevelFormat(format_code)
	customer_id		int			x	PK3, FK	Sales.Customer(customer_id)
	date_queued		date			x		
	rank_in_queue		tinyint			x		
Sales.MovieRating	movie_id	see above	char(26)			x	PK1, FK	Sales.Movie(movie_id)
	format_code		char(6)			x	PK2, FK	Sales.HighLevelFormat(format_code)
	customer_id		int			x	PK3, FK	Sales.Customer(customer_id)
	number_of_stars		tinyint		BETWEEN 0 AND 5	x		
	rating_date		date					
	short_review		nvarchar(4000)					
Sales.PostalCodeAssignment	country_code		nchar(2)			x	PK1, FK	Sales.Country(country_code)
	postal_code		nchar(10)			x	PK2	
	outlet_id_1	primary outlet	smallint			x		
	outlet_id_2	backup outlet	smallint			x		
	outlet_id_3	backup outlet	smallint					

```sql
/*Flix2YouBuildv2.sql
 *SQL DDL code to build the proposed Flix2You OLTP database
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015
*/


USE master;
GO

--Delete the Flix2You database if it exists.
IF EXISTS(SELECT * from sys.databases WHERE name='Flix2You')
BEGIN
    DROP DATABASE Flix2You;
END;
GO

--Create a new database called Flix2You.
CREATE DATABASE Flix2You;
GO

USE Flix2You;
GO

--Create the schema Sales for the Flix2You database if it does not exist.
IF (SCHEMA_ID('Sales') IS NULL)
BEGIN
    EXECUTE('CREATE SCHEMA Sales AUTHORIZATION dbo');
END;
GO

--Drop the Movie table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'Movie'))
BEGIN
   DROP TABLE Sales.Movie;
END;
GO

--Create the Movie table (without foreign-key constraints, for now)
CREATE TABLE Sales.Movie
(
    movie_id                char(26)        NOT NULL    PRIMARY KEY,
    movie_title             nvarchar(50)    NOT NULL,
    studio_id               char(9)         NOT NULL,
    release_date            date            NOT NULL,
    country_code            nchar(2)        NOT NULL,
    director_id             smallint        NOT NULL,
    movie_length            smallint        NOT NULL,
    genre_code              tinyint         NOT NULL,
    movie_description       nvarchar(max),
    movie_description_uri   nvarchar(100),
    movie_poster_uri        nvarchar(100),
    imdb_id                 nchar(9),
    CONSTRAINT  check_EIDR  CHECK(movie_id LIKE     '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-Z]')
);
GO

--Drop the Country table if it exists.
```

```sql
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'Country'))
BEGIN
   DROP TABLE Sales.Country;
END;
GO

--Create the Country table
CREATE TABLE Sales.Country
(
    country_code    nchar(2)        NOT NULL    PRIMARY KEY,
    country_name    nvarchar(35)    NOT NULL    UNIQUE,
);
GO

--Drop the Studio table if it exists.
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'Studio'))
BEGIN
   DROP TABLE Sales.Studio;
END;
GO

--Create the Studio table
CREATE TABLE Sales.Studio
(
    studio_id                   char(9)         NOT NULL    PRIMARY KEY,
    studio_name                 nvarchar(50)    NOT NULL,
    production_company          nvarchar(50),
    studio_hq_address           nvarchar(35)    NOT NULL,
    studio_hq_city              nvarchar(35)    NOT NULL,
    studio_hq_state             nchar(2)        NOT NULL,
    studio_hq_postal_code       nchar(10)       NOT NULL,
    country_code                nchar(2)        NOT NULL,
    studio_telephone            char(14)        NOT NULL,
    CONSTRAINT  check_studio_id     CHECK(studio_id LIKE    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                            '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]'),
    CONSTRAINT  check_studio_phone  CHECK(studio_telephone  LIKE    '([0-9][0-9][0-9])' +
                                                                    ' [0-9][0-9][0-9]' +
                                                                    '-[0-9][0-9][0-9][0-9]'),
    CONSTRAINT  fk_studio_1 FOREIGN KEY(country_code) REFERENCES Sales.Country(country_code)
);
GO

--Drop the Director table if it exists.
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'Director'))
BEGIN
   DROP TABLE Sales.Director;
END;
GO

--Create the Director table
CREATE TABLE Sales.Director
(
    director_id             smallint        NOT NULL    IDENTITY(1,1)   PRIMARY KEY,
    director_name_first     nvarchar(35)    NOT NULL,
    director_name_last      nvarchar(35)    NOT NULL
);
GO
```

```sql

--Drop the Genre table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'Genre'))
BEGIN
   DROP TABLE Sales.Genre;
END;
GO

--Create the Genre table
CREATE TABLE Sales.Genre
(
    genre_code              tinyint         NOT NULL    PRIMARY KEY,
    genre_description       nvarchar(35)    NOT NULL    UNIQUE
);
GO

--Drop the Actor table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'Actor'))
BEGIN
   DROP TABLE Sales.Actor;
END;
GO

--Create the Actor table
CREATE TABLE Sales.Actor
(
    actor_id            smallint        NOT NULL    IDENTITY(1,1)   PRIMARY KEY,
    actor_gender        char(1)         NOT NULL,
    actor_name_first    nvarchar(35)    NOT NULL,
    actor_name_last     nvarchar(35)    NOT NULL,
    other_actor_details nvarchar(max),
    CONSTRAINT  check_actor_gender  check(actor_gender = 'M' OR actor_gender = 'F')
);
GO

--Drop the MovieRole table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'MovieRole'))
BEGIN
   DROP TABLE Sales.MovieRole;
END;
GO

--Create the MovieRole table
CREATE TABLE Sales.MovieRole
(
    movie_id        char(26)        NOT NULL,
    actor_id        smallint        NOT NULL,
    as_character    nvarchar(50),
    is_lead         bit,
    CONSTRAINT  fk_movierole_1  FOREIGN KEY(movie_id)   REFERENCES Sales.Movie(movie_id),
    CONSTRAINT  fk_movierole_2  FOREIGN KEY(actor_id)   REFERENCES Sales.Actor(actor_id),
    CONSTRAINT  pk_movie_role   PRIMARY KEY(movie_id, actor_id)
);
GO

--Add the foreign-key constraints to the Movie table
ALTER TABLE Sales.Movie ADD
```

```sql
    CONSTRAINT  fk_movie_1  FOREIGN KEY(studio_id)      REFERENCES Sales.Studio(studio_id),
    CONSTRAINT  fk_movie_2  FOREIGN KEY(director_id)    REFERENCES Sales.Director(director_id),
    CONSTRAINT  fk_movie_3  FOREIGN KEY(genre_code)     REFERENCES Sales.Genre(genre_code),
    CONSTRAINT  fk_movie_4  FOREIGN KEY(country_code)   REFERENCES Sales.Country(country_code);
GO

--Drop the HighLevelFormat table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'HighLevelFormat'))
BEGIN
   DROP TABLE Sales.HighLevelFormat;
END;
GO

--Create the HighLevelFormat table
CREATE TABLE Sales.HighLevelFormat
(
    format_code         char(6)         NOT NULL    PRIMARY KEY,
    format_name         varchar(35)     NOT NULL    UNIQUE,
    physical_medium     char(4)         NOT NULL,
    definition_level    varchar(8)      NOT NULL,
    aspect_ratio        varchar(15),
    CONSTRAINT ck_medium    CHECK(physical_medium = 'disc' OR physical_medium = 'file'),
    CONSTRAINT uk_format    UNIQUE(physical_medium, definition_level, aspect_ratio)
);
GO

--Drop the ViewableEncoding table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'ViewableEncoding'))
BEGIN
   DROP TABLE Sales.ViewableEncoding;
END;
GO

--Create the ViewableEncoding table
CREATE TABLE Sales.ViewableEncoding
(
    movie_id        char(26)        NOT NULL,
    format_code     char(6)         NOT NULL,
    rental_price    money,
    purchase_price  money,
    aggregate_cost  money,
    is_available    bit,
    CONSTRAINT  fk_encoding_1   FOREIGN KEY(movie_id)
                                    REFERENCES Sales.Movie(movie_id),
    CONSTRAINT  fk_encoding_2   FOREIGN KEY(format_code)
                                    REFERENCES Sales.HighLevelFormat(format_code),
    CONSTRAINT  pk_encoding     PRIMARY KEY(movie_id, format_code)
);
GO

--Drop the OutletResource table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'OutletResource'))
BEGIN
   DROP TABLE Sales.OutletResource;
END;
GO
```

```sql

--Create the OutletResource table
CREATE TABLE Sales.OutletResource
(
    resource_id     int         NOT NULL    IDENTITY(1, 1)  PRIMARY KEY,
    movie_id        char(26)    NOT NULL,
    format_code     char(6)     NOT NULL,
    video_codec     varchar(8)  NOT NULL,
    resolution      varchar(10) NOT NULL,
    frame_rate      smallint    NOT NULL,
    bitrate         int         NOT NULL,
    audio_codec_1   varchar(8)  NOT NULL,
    audio_codec_2   varchar(8),
    CONSTRAINT  fk_outlet_resource_1    FOREIGN KEY(movie_id)
                                            REFERENCES Sales.Movie(movie_id),
    CONSTRAINT  fk_outlet_resource_2    FOREIGN KEY(format_code)
                                            REFERENCES Sales.HighLevelFormat(format_code)
);
GO

--Drop the OpticalDisc table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'OpticalDisc'))
BEGIN
   DROP TABLE Sales.OpticalDisc;
END;
GO

--Create the OpticalDisc table
CREATE TABLE Sales.OpticalDisc
(
    resource_id     int         NOT NULL    PRIMARY KEY,
    copy_protection varchar(8),
    region_code     tinyint,
    CONSTRAINT  fk_disc     FOREIGN KEY(resource_id)
                                REFERENCES Sales.OutletResource(resource_id),
);
GO

--Drop the DownloadableFile table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'DownloadableFile'))
BEGIN
   DROP TABLE Sales.DownloadableFile;
END;
GO

--Create the DownloadableFile table
CREATE TABLE Sales.DownloadableFile
(
    resource_id         int         NOT NULL    PRIMARY KEY,
    HTTP_segment_size   smallint    NOT NULL,
    file_size           bigint      NOT NULL,
    CONSTRAINT  fk_file     FOREIGN KEY(resource_id)
                                REFERENCES Sales.OutletResource(resource_id),
);
GO

--Drop the Outlet table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
```

```sql
                 AND    TABLE_NAME = 'Outlet'))
BEGIN
   DROP TABLE Sales.Outlet;
END;
GO

--Create the Outlet table
CREATE TABLE Sales.Outlet
(
    outlet_id           smallint        NOT NULL    IDENTITY(1, 1)  PRIMARY KEY,
    outlet_name         nvarchar(35)    NOT NULL,
    outlet_location     nvarchar(50)    NOT NULL,
    outlet_type         char(1)         NOT NULL,
    CONSTRAINT  ck_outlet_discriminator CHECK(outlet_type IN ('M', 'F'))
);
GO

--Drop the DistributionCenter table if it exists.
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'DistributionCenter'))
BEGIN
   DROP TABLE Sales.DistributionCenter;
END;
GO

--Create the DistributionCenter table
CREATE TABLE Sales.DistributionCenter
(
    outlet_id           smallint    NOT NULL PRIMARY KEY,
    outlet_postal_code  char(10)    NOT NULL,
    CONSTRAINT  fk_distribution_center  FOREIGN KEY(outlet_id)
                                            REFERENCES Sales.Outlet(outlet_id)
);
GO

--Drop the CDNCluster table if it exists.
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'CDNCluster'))
BEGIN
   DROP TABLE Sales.CDNCluster;
END;
GO

--Create the CDNCluster table
CREATE TABLE Sales.CDNCluster
(
    outlet_id           smallint    NOT NULL PRIMARY KEY,
    outlet_IPv4         varchar(11) NOT NULL,
    CONSTRAINT  fk_cdn_cluster  FOREIGN KEY(outlet_id)
                                    REFERENCES Sales.Outlet(outlet_id)
);
GO

--Drop the DiscAtMailCenter table if it exists.
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'DiscAtMailCenter'))
BEGIN
   DROP TABLE Sales.DiscAtMailCenter;
END;
GO
```

```sql

--Create the DiskAtMailCenter table
CREATE TABLE Sales.DiscAtMailCenter
(
    outlet_id           smallint    NOT NULL,
    resource_id         int         NOT NULL,
    number_on_hand      smallint    NOT NULL,
    CONSTRAINT  fk_disc_at_mail_center_1    FOREIGN KEY(outlet_id)
                                                REFERENCES Sales.Outlet(outlet_id),
    CONSTRAINT  fk_disc_at_mail_center_2    FOREIGN KEY(resource_id)
                                                REFERENCES Sales.OutletResource(resource_id),
    CONSTRAINT  pk_disc_at_mail_center      PRIMARY KEY(outlet_id, resource_id)
);
GO

--Drop the FileOnCluster table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'FileOnCluster'))
BEGIN
   DROP TABLE Sales.FileOnCluster;
END;
GO

--Create the FileOnCluster table
CREATE TABLE Sales.FileOnCluster
(
    outlet_id           smallint    NOT NULL,
    resource_id         int         NOT NULL,
    DASH_fileset        smallint    NOT NULL,
    CONSTRAINT  fk_file_on_cluster_1    FOREIGN KEY(outlet_id)
                                            REFERENCES Sales.Outlet(outlet_id),
    CONSTRAINT  fk_file_on_cluster_2    FOREIGN KEY(resource_id)
                                            REFERENCES Sales.OutletResource(resource_id),
    CONSTRAINT  pk_file_on_cluster      PRIMARY KEY(outlet_id, resource_id)
);
GO

--Drop the Customer table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'Customer'))
BEGIN
   DROP TABLE Sales.Customer;
END;
GO

--Create the Customer table (without foreign-key constraints, for the moment)
CREATE TABLE Sales.Customer
(
    customer_id                 int             NOT NULL    IDENTITY(1,1)   PRIMARY KEY,
    customer_last_name          nvarchar(35)    NOT NULL,
    customer_first_name         nvarchar(35)    NOT NULL,
    customer_middle_name        nvarchar(35),
    customer_birth_date         date            NOT NULL,
    membership_date             date            NOT NULL,
    customer_street_address     nvarchar(35)    NOT NULL,
    customer_city               nvarchar(35)    NOT NULL,
    customer_state_province     nchar(2)        NOT NULL,
    customer_postal_code        nchar(10)       NOT NULL,
    country_code                nchar(2)        NOT NULL DEFAULT 'US',
    customer_telephone          char(14)        NOT NULL,
    account_email_address       nvarchar(100)   NOT NULL,
```

```sql
    account_password_hash       char(48)        NOT NULL,
    account_password_salt       char(48)        NOT NULL,
    image_id                    smallint        NOT NULL,
    challenge_question_id       smallint        NOT NULL,
    account_secret_answer       nvarchar(250)   NOT NULL,
    xml_survey_document         xml,
    CONSTRAINT  check_customer_dob      CHECK(DATEDIFF(year, customer_birth_date, GETDATE())
                                                                                BETWEEN 16 AND 95),
    CONSTRAINT  check_membership_date   CHECK(membership_date BETWEEN '1995-01-01' AND GETDATE()),
    CONSTRAINT  check_customer_phone    CHECK(customer_telephone LIKE   '([0-9][0-9][0-9])' +
                                                                        ' [0-9][0-9][0-9]'  +
                                                                        '-[0-9][0-9][0-9][0-9]')
);
GO

--Drop the UserImage table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'UserImage'))
BEGIN
   DROP TABLE Sales.UserImage;
END;
GO

--Create the UserImage table
CREATE TABLE Sales.UserImage
(
    image_id    smallint        NOT NULL    IDENTITY(1,1)   PRIMARY KEY,
    image_uri   nvarchar(100)   NOT NULL    UNIQUE

);
GO

--Drop the ChallengeQuestion table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'ChallengeQuestion'))
BEGIN
   DROP TABLE Sales.ChallengeQuestion;
END;
GO

--Create the ChallengeQuestion table
CREATE TABLE Sales.ChallengeQuestion
(
    challenge_question_id   smallint        NOT NULL    IDENTITY(1,1)   PRIMARY KEY,
    challenge_question_text nvarchar(100)   NOT NULL    UNIQUE

);
GO

--Add the foreign-key constraints to the Customer table
ALTER TABLE Sales.Customer ADD
    CONSTRAINT  fk_member_account_1     FOREIGN KEY(country_code)
                                            REFERENCES Sales.Country(country_code),
    CONSTRAINT  fk_member_account_2     FOREIGN KEY(image_id)
                                            REFERENCES Sales.UserImage(image_id),
    CONSTRAINT  fk_member_account_3     FOREIGN KEY(challenge_question_id)
                                            REFERENCES Sales.ChallengeQuestion(challenge_question_id);
GO

--Drop the TransactionType table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION SCHEMA.TABLES
```

```sql
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME =   'TransactionType'))
BEGIN
   DROP TABLE Sales.TransactionType;
END;
GO

--Create the TransactionType table
CREATE TABLE Sales.TransactionType
(
    transaction_type_code       tinyint     NOT NULL    PRIMARY KEY,
    transaction_description     varchar(50) NOT NULL    UNIQUE,
--the following intentionally makes it difficult to alter the last attribute:
    CONSTRAINT  ck_description  CHECK(transaction_description IN(
                                            'movie rental',
                                            'movie purchase',
                                            'credit-card authorization',
                                            'credit-card payment',
                                            'credit-card refund',
                                            'file transfer',
                                            'disk shipment',
                                            'disc return',
                                            'other'))
);
GO

--Drop the TransactionSequence if it exists
IF EXISTS(SELECT * FROM sys.objects
          WHERE object_id = OBJECT_ID(N'Sales.TransactionSequence') AND type = 'SO')
BEGIN
    DROP SEQUENCE Sales.TransactionSequence;
END;

--Create a TransactionSequence (for generating primary keys)
CREATE SEQUENCE Sales.TransactionSequence
    AS bigint
    START WITH 1        --[or the successor to any existing transaction number]
    INCREMENT BY 1
    MINVALUE 1
    MAXVALUE 9223372036854775807;
GO

--Drop the TransactedEvent table if it exists.
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE  TABLE_SCHEMA = 'Sales'
                 AND    TABLE_NAME = 'TransactedEvent'))
BEGIN
   DROP TABLE Sales.TransactedEvent;
END;
GO

--Create the TransactedEvent table
CREATE TABLE Sales.TransactedEvent
(
    transaction_id          bigint      NOT NULL    PRIMARY KEY,
    movie_id                char(26)    NOT NULL,
    format_code             char(6)     NOT NULL,
    customer_id             int         NOT NULL,
    transaction_timestamp   datetime    NOT NULL    DEFAULT GETDATE(),
    transaction_type_code   tinyint     NOT NULL,
    transaction_amount      money       NOT NULL    DEFAULT 0.00,
    previous_transaction_id bigint,
    transaction_comment     varchar(300),
    CONSTRAINT  fk_transaction_1    FOREIGN KEY(movie_id)
```

```sql
                                        REFERENCES Sales.Movie(movie_id),
    CONSTRAINT  fk_transaction_2    FOREIGN KEY(format_code)
                                        REFERENCES Sales.HighLevelFormat(format_code),
    CONSTRAINT  fk_transaction_3    FOREIGN KEY(customer_id)
                                        REFERENCES Sales.Customer(customer_id),
    CONSTRAINT  fk_transaction_4    FOREIGN KEY(transaction_type_code)
                                        REFERENCES Sales.TransactionType(transaction_type_code),
    CONSTRAINT  fk_transaction_5    FOREIGN KEY(previous_transaction_id)
                                        REFERENCES Sales.TransactedEvent(transaction_id),
    CONSTRAINT  unique_transaction   UNIQUE(movie_id,
                                            format_code,
                                            customer_id,
                                            transaction_timestamp,
                                            transaction_type_code)
);
GO

--Check for the existence of a named Customer/Date index on the TransactedEvent table
IF EXISTS (SELECT *  FROM sys.indexes  WHERE name='idxTransaction_Customer_Date'
    AND object_id = OBJECT_ID('Sales.TransactedEvent'))
BEGIN
    DROP INDEX idxTransaction_Customer_Date ON Sales.TransactedEvent;
END

--Create a named Customer/Date index on the TransactedEvent table
CREATE INDEX idxTransaction_Customer_Date ON
                        Sales.TransactedEvent(customer_id, transaction_timestamp DESC);

--Check for the existence of a named Movie index on the TransactedEvent table
IF EXISTS (SELECT *  FROM sys.indexes  WHERE name='idxTransaction_Movie'
    AND object_id = OBJECT_ID('Sales.TransactedEvent'))
BEGIN
    DROP INDEX idxTransaction_Movie ON Sales.TransactedEvent;
END

--Create a named Customer/Date index on the TransactedEvent table
CREATE INDEX idxTransaction_Movie ON Sales.TransactedEvent(movie_id);

--Drop the QueuedMovie table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'QueuedMovie'))
BEGIN
   DROP TABLE Sales.QueuedMovie;
END;
GO

--Create the QueuedMovie table
CREATE TABLE Sales.QueuedMovie
(
    movie_id                char(26)    NOT NULL,
    format_code             char(6)     NOT NULL,
    customer_id             int         NOT NULL,
    date_queued             date        NOT NULL,
    rank_in_queue           tinyint     NOT NULL,
    CONSTRAINT  fk_queued_movie_1   FOREIGN KEY(movie_id)
                                        REFERENCES Sales.Movie(movie_id),
    CONSTRAINT  fk_queued_movie_2   FOREIGN KEY(format_code)
                                        REFERENCES Sales.HighLevelFormat(format_code),
    CONSTRAINT  fk_queued_movie_3   FOREIGN KEY(customer_id)
                                        REFERENCES Sales.Customer(customer_id),
    CONSTRAINT  pk_queued_movie     PRIMARY KEY(movie_id, format_code, customer_id)
);
GO
```

```sql

--Drop the MovieRating table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'MovieRating'))
BEGIN
   DROP TABLE Sales.MovieRating;
END;
GO

--Create the MovieRating table
CREATE TABLE Sales.MovieRating
(
    movie_id                char(26)     NOT NULL,
    format_code             char(6)      NOT NULL,
    customer_id             int          NOT NULL,
    number_of_stars         tinyint      NOT NULL,
    rating_date             date,
    short_review            nvarchar(4000),
    CONSTRAINT  fk_movie_rating_1   FOREIGN KEY(movie_id)
                                        REFERENCES Sales.Movie(movie_id),
    CONSTRAINT  fk_movie_rating_2   FOREIGN KEY(format_code)
                                        REFERENCES Sales.HighLevelFormat(format_code),
    CONSTRAINT  fk_movie_rating_3   FOREIGN KEY(customer_id)
                                        REFERENCES Sales.Customer(customer_id),
    CONSTRAINT  pk_movie_rating     PRIMARY KEY(movie_id, format_code, customer_id),
    CONSTRAINT  check_rating        CHECK(number_of_stars >= 0 AND  number_of_stars <= 5)
);
GO

--Drop the PostalCodeAssignment table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'PostalCodeAssignment'))
BEGIN
   DROP TABLE Sales.PostalCodeAssignment;
END;
GO

--Create the PostalCodeAssignment table
CREATE TABLE Sales.PostalCodeAssignment
(
    country_code    nchar(2)    NOT NULL,
    postal_code     nchar(10)   NOT NULL,
    outlet_id_1     smallint    NOT NULL,
    outlet_id_2     smallint    NOT NULL,
    outlet_id_3     smallint
    CONSTRAINT  pk_zip_assignment   PRIMARY KEY(country_code, postal_code),
    CONSTRAINT  fk_zip_assignment   FOREIGN KEY(country_code)
                                        REFERENCES Sales.Country(country_code)
);
GO

```

```sql
/*Flix2YouLoadv2.sql
 *SQL DML code to load the proposed Flix2You OLTP database with test data
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015*/

USE Flix2You;

INSERT INTO Sales.Country(country_code,country_name)
VALUES
    ('US','United States'),
    ('FR','France'),
    ('CA', 'Canada');
GO

INSERT INTO Sales.Studio
  (studio_id,studio_name,production_company,studio_hq_address,studio_hq_city,studio_hq_state,studio_hq_postal_code,country_code,studio_telephone)
VALUES
    ('CFC0-F4BC','Paramount Pictures Corporation',NULL,'425 Meadowlands Pkwy.','Secaucus','NJ','07094','US','(201) 867-1541'),
    ('35C5-3F4D','Warner Brothers Entertainment, Inc.','Tatira-Hiller Productions','4000 Warner Blvd.','Burbank','CA','91522','US','(818) 954-6000'),
    ('9591-0774','Miramax, LLC',NULL,'1601 Cloverfield Blvd.','Santa Monica','CA','90404','US','(310) 409-4321'),
    ('46E3-BC0D','Universal Studios','Gramercy Pictures','100 Universal City Plaza ','Universal City','CA','91608','US','(818) 777-1000'),
    ('2FE2-24F2','Twentieth Century Fox',NULL,'10201 West Pico Boulevard','Los Angeles','CA','90035','US','(310) 277-2211'),
    ('D893-0AC2','United Artists',NULL,'10250 Constellation Blvd #19','Los Angeles','CA','90067','US','(310) 449-3000'),
    ('46B6-F1ED','Universal Studios','Zanuck/Brown Productions','100 Universal City Plaza','Universal City','CA','91608','US','(818) 777-1000');
GO

INSERT INTO Sales.Director(director_name_first,director_name_last)
VALUES
    ('Francis Ford','Coppola'),
    ('Roman','Polanski'),
    ('Arthur','Penn'),
    ('Quentin','Tarantino'),
    ('Joel','Coen'),
    ('George','Lucas'),
    ('Martin','Scorsese'),
    ('Steven','Spielberg');
GO

INSERT INTO Sales.Genre(genre_code,genre_description)
VALUES
    (1, 'Action and Adventure'),
    (2, 'Comedy'),
    (3, 'Documentary'),
    (4, 'Drama'),
    (5, 'Horror'),
    (6, 'Kids and Family'),
    (7, 'Musical'),
    (8, 'Romance'),
    (9, 'Sci-Fi and Fantasy'),
    (10,'Suspense and Thriller');
GO

INSERT INTO Sales.Movie
  (movie_id,movie_title,studio_id,release_date,country_code,director_id,movie_length,genre_code,movie_description,movie_description_uri,movie_poster_uri,imdb_id)
VALUES
    ('843E-C238-7F61-DDFC-F427-9','The Godfather','CFC0-F4BC','1972-03-24','US',1,175,4,NULL,NULL,NULL,'tt0068646'),
    ('2331-775F-399A-937C-54C2-0','The Godfather Part II','CFC0-F4BC','1974-12-12','US',1,200,4,NULL,NULL,NULL,'tt0071562'),
    ('2411-6FF0-31E2-D49D-870C-7','Chinatown','CFC0-F4BC','1974-06-20','US',2,150,4,NULL,NULL,NULL,'tt007131'),
    ('B193-2EDD-9478-C964-FEF9-K','Bonnie and Clyde','35C5-3F4D','1967-08-13','US',3,111,4,NULL,NULL,NULL,'tt0061418'),
    ('4EB2-DFF8-6B78-660F-C61B-D','Pulp Fiction','9591-0774','1994-10-14','US',4,154,2,NULL,NULL,NULL,'tt0110912'),
    ('9824-4212-BADD-A847-068B-1','Fargo','46E3-BC0D','1995-04-05','US',5,98,2,NULL,NULL,NULL,'tt0116282'),
    ('5868-409E-7BFB-536A-6067-E','Star Wars','2FE2-24F2','1977-05-25','US',6,121,9,NULL,NULL,NULL,'tt0076759'),
    ('0A89-4017-2ED5-D199-0491-H','Raging Bull','D893-0AC2','1980-12-19','US',7,129,4,NULL,NULL,NULL,'tt0081398'),
    ('36E3-649C-57F2-A251-72B1-E','Jaws','46B6-F1ED','1975-06-20','US',8,124,1,NULL,NULL,NULL,'tt0073195');
GO

INSERT INTO Sales.Actor(actor_gender,actor_name_first,actor_name_last)
VALUES
    ('M','Marlon','Brando'),
    ('M','Al','Pacino'),
    ('M','James','Caan'),
    ('M','Robert','Duvall'),
    ('F','Diane','Keaton'),
    ('F','Talia','Shire'),
    ('M','John','Cazale'),
    ('M','Richard S.','Castellano'),
    ('M','Abe','Vigoda'),
    ('M','Gianni','Russo'),
    ('M','Robert','De Niro'),
    ('M','Lee','Strasberg'),
    ('M','Michael V.','Gazzo'),
    ('M','Jack','Nicholson'),
```

```sql
    ('F','Faye','Dunaway'),
    ('M','John','Huston'),
    ('M','Warren','Beatty'),
    ('M','Gene','Hackman'),
    ('M','Michael J.','Pollard'),
    ('F','Estelle','Parsons'),
    ('M','John','Travolta'),
    ('M','Samuel L.','Jackson'),
    ('M','Bruce','Willis'),
    ('M','Ving','Rhames'),
    ('F','Uma','Thurman'),
    ('M','Harvey','Keitel'),
    ('M','Steve','Buscemi'),
    ('M','William H.','Macy'),
    ('M','Peter','Stormare'),
    ('F','Frances','McDormand'),
    ('M','Mark','Hamill'),
    ('M','Harrison','Ford'),
    ('F','Carrie','Fisher'),
    ('M','Alec','Guiness'),
    ('F','Cathy','Moriarty'),
    ('M','Joe','Pesci'),
    ('M','Frank','Vincent'),
    ('M','Roy','Scheider'),
    ('M','Robert','Shaw'),
    ('M','Richard','Dreyfuss');
GO

INSERT INTO Sales.MovieRole(movie_id,actor_id,as_character,is_lead)
VALUES
    ('843E-C238-7F61-DDFC-F427-9',1, 'Don Vito Corleone',1),
    ('843E-C238-7F61-DDFC-F427-9',2, 'Michael Corleone',0),
    ('843E-C238-7F61-DDFC-F427-9',3, 'Sonny Corleone',0),
    ('843E-C238-7F61-DDFC-F427-9',4, 'Tom Hagen',0),
    ('843E-C238-7F61-DDFC-F427-9',5, 'Kay Adams',0),
    ('843E-C238-7F61-DDFC-F427-9',6, 'Connie Corleone',0),
    ('843E-C238-7F61-DDFC-F427-9',7, 'Fredo Corleone',0),
    ('843E-C238-7F61-DDFC-F427-9',8, 'Clemenza',0),
    ('843E-C238-7F61-DDFC-F427-9',9, 'Tessio',0),
    ('843E-C238-7F61-DDFC-F427-9',10, 'Carlo',0),
    ('2331-775F-399A-937C-54C2-0',2, 'Michael Corleone',1),
    ('2331-775F-399A-937C-54C2-0',4, 'Tom Hagen',0),
    ('2331-775F-399A-937C-54C2-0',5, 'Kay',1),
    ('2331-775F-399A-937C-54C2-0',11,'Vito Corleone',0),
    ('2331-775F-399A-937C-54C2-0',7, 'Fredo Corleone',0),
    ('2331-775F-399A-937C-54C2-0',6, 'Connie Corleone',0),
    ('2331-775F-399A-937C-54C2-0',12,'Hyman Roth',0),
    ('2331-775F-399A-937C-54C2-0',13,'Frankie Pentangeli',0),
    ('2411-6FF0-31E2-D49D-870C-7',14,'J.J. Gittes',1),
    ('2411-6FF0-31E2-D49D-870C-7',15,'Evelyn Mulwray',1),
    ('2411-6FF0-31E2-D49D-870C-7',16,'Noah Cross',0),
    ('B193-2EDD-9478-C964-FEF9-K',17,'Clyde Barrow',1),
    ('B193-2EDD-9478-C964-FEF9-K',15,'Bonnie Parker',1),
    ('B193-2EDD-9478-C964-FEF9-K',18,'Buck Barrow',0),
    ('B193-2EDD-9478-C964-FEF9-K',19,'C.W. Moss',0),
    ('B193-2EDD-9478-C964-FEF9-K',20,'Estelle',0),
    ('4EB2-DFF8-6B78-660F-C61B-D',21,'Vincent Vega',1),
    ('4EB2-DFF8-6B78-660F-C61B-D',22,'Jules Winnfield',1),
    ('4EB2-DFF8-6B78-660F-C61B-D',23,'Butch Coolidge',0),
    ('4EB2-DFF8-6B78-660F-C61B-D',24,'Marsellus Wallace',0),
    ('4EB2-DFF8-6B78-660F-C61B-D',25,'Mia Wallace',0),
    ('4EB2-DFF8-6B78-660F-C61B-D',26,'The Wolf',0),
    ('9824-4212-BADD-A847-068B-1',27,'Carl Showalter',0),
    ('9824-4212-BADD-A847-068B-1',28,'Jerry Lundergaard',0),
    ('9824-4212-BADD-A847-068B-1',29,'Gaear Grimsrud',0),
    ('9824-4212-BADD-A847-068B-1',30,'Marge Gunderson',1),
    ('5868-409E-7BFB-536A-6067-E',31,'Luke Skywalker',1),
    ('5868-409E-7BFB-536A-6067-E',32,'Han Solo',1),
    ('5868-409E-7BFB-536A-6067-E',33,'Princess Leia Organa',0),
    ('5868-409E-7BFB-536A-6067-E',34,'Ben Obi-Wan Kenobi',0),
    ('0A89-4017-2ED5-D199-0491-H',11,'Jake La Motta',1),
    ('0A89-4017-2ED5-D199-0491-H',35,'Vickie La Motta',0),
    ('0A89-4017-2ED5-D199-0491-H',36,'Joey',0),
    ('0A89-4017-2ED5-D199-0491-H',37,'Salvy',0),
    ('36E3-649C-57F2-A251-72B1-E',38,'Brody',1),
    ('36E3-649C-57F2-A251-72B1-E',39,'Quint',0),
    ('36E3-649C-57F2-A251-72B1-E',40,'Hooper',0);
GO

INSERT INTO Sales.HighLevelFormat(format_code,format_name,physical_medium,definition_level,aspect_ratio)
VALUES
    ('HD-1',    'High-Definition Stream',       'file',     'high',         'widescreen'),
    ('STD-1',   'Standard-Definition Stream',   'file',     'standard',     'widescreen'),
```

```sql
    ('BR-1',    'Blu-ray',                      'disc',     'high',         'widescreen'),
    ('DVD-1',   'DVD',                          'disc',     'standard',     'widescreen');
GO


INSERT INTO Sales.ViewableEncoding(movie_id,format_code,rental_price,purchase_price,aggregate_cost,is_available)
VALUES
    ('843E-C238-7F61-DDFC-F427-9','HD-1',3.99,14.99,NULL,1),
    ('843E-C238-7F61-DDFC-F427-9','STD-1',2.99,9.99,NULL,1),
    ('843E-C238-7F61-DDFC-F427-9','BR-1',5.95,14.95,NULL,1),
    ('843E-C238-7F61-DDFC-F427-9','DVD-1',3.95,11.95,NULL,1),
    ('2331-775F-399A-937C-54C2-0','HD-1',3.99,14.99,NULL,1),
    ('2331-775F-399A-937C-54C2-0','STD-1',2.99,9.99,NULL,1),
    ('2331-775F-399A-937C-54C2-0','BR-1',5.95,14.95,NULL,1),
    ('2331-775F-399A-937C-54C2-0','DVD-1',3.95,11.95,NULL,1),
    ('2411-6FF0-31E2-D49D-870C-7','HD-1',3.99,14.99,NULL,1),
    ('2411-6FF0-31E2-D49D-870C-7','STD-1',2.99,9.99,NULL,1),
    ('2411-6FF0-31E2-D49D-870C-7','BR-1',5.95,14.95,NULL,1),
    ('2411-6FF0-31E2-D49D-870C-7','DVD-1',3.95,11.95,NULL,1),
    ('B193-2EDD-9478-C964-FEF9-K','HD-1',3.99,14.99,NULL,1),
    ('B193-2EDD-9478-C964-FEF9-K','STD-1',2.99,9.99,NULL,1),
    ('B193-2EDD-9478-C964-FEF9-K','BR-1',5.95,14.95,NULL,1),
    ('B193-2EDD-9478-C964-FEF9-K','DVD-1',3.95,11.95,NULL,1),
    ('4EB2-DFF8-6B78-660F-C61B-D','HD-1',3.99,14.99,NULL,1),
    ('4EB2-DFF8-6B78-660F-C61B-D','STD-1',2.99,9.99,NULL,1),
    ('4EB2-DFF8-6B78-660F-C61B-D','BR-1',5.95,14.95,NULL,1),
    ('4EB2-DFF8-6B78-660F-C61B-D','DVD-1',3.95,11.95,NULL,1),
    ('9824-4212-BADD-A847-068B-1','HD-1',3.99,14.99,NULL,1),
    ('9824-4212-BADD-A847-068B-1','STD-1',2.99,9.99,NULL,1),
    ('9824-4212-BADD-A847-068B-1','BR-1',5.95,14.95,NULL,1),
    ('9824-4212-BADD-A847-068B-1','DVD-1',3.95,11.95,NULL,1),
    ('5868-409E-7BFB-536A-6067-E','HD-1',3.99,14.99,NULL,1),
    ('5868-409E-7BFB-536A-6067-E','STD-1',2.99,9.99,NULL,1),
    ('5868-409E-7BFB-536A-6067-E','BR-1',5.95,14.95,NULL,1),
    ('5868-409E-7BFB-536A-6067-E','DVD-1',3.95,11.95,NULL,1),
    ('0A89-4017-2ED5-D199-0491-H','HD-1',3.99,14.99,NULL,1),
    ('0A89-4017-2ED5-D199-0491-H','STD-1',2.99,9.99,NULL,1),
    ('0A89-4017-2ED5-D199-0491-H','BR-1',5.95,14.95,NULL,1),
    ('0A89-4017-2ED5-D199-0491-H','DVD-1',3.95,11.95,NULL,1),
    ('36E3-649C-57F2-A251-72B1-E','HD-1',3.99,14.99,NULL,1),
    ('36E3-649C-57F2-A251-72B1-E','STD-1',2.99,9.99,NULL,1),
    ('36E3-649C-57F2-A251-72B1-E','BR-1',5.95,14.95,NULL,1),
    ('36E3-649C-57F2-A251-72B1-E','DVD-1',3.95,11.95,NULL,1);
GO

INSERT INTO Sales.ChallengeQuestion(challenge_question_text)
VALUES
    (N'Who is the person that you most admire?'),
    (N'Who is your favorite artist, in any medium?'),
    (N'What place in all the world would you most like to visit?'),
    (N'When and where in history would it have been best to be alive?');
GO

INSERT INTO Sales.UserImage(image_uri)
VALUES
    ('images/8ball.png'),
    ('images/Baseball.png'),
    ('images/Basketball.png'),
    ('images/Bowling.png'),
    ('images/Cactus.png'),
    ('images/Chalk.png'),
    ('images/Dahlia.png'),
    ('images/Dandelion.png'),
    ('images/Drum.png'),
    ('images/Eagle.png'),
    ('images/Earth.png'),
    ('images/Flower.png'),
    ('images/Football.png'),
    ('images/Fortune Cookie.png'),
    ('images/Gingerbread Man.png'),
    ('images/Golf.png'),
    ('images/Guitar.png'),
    ('images/Hockey.png'),
    ('images/Leaf.png'),
    ('images/Lightning.png'),
    ('images/Lotus.png'),
    ('images/Medal.png'),
    ('images/Nest.png'),
    ('images/Owl.png'),
    ('images/Parrot.png'),
    ('images/Penguin.png'),
    ('images/Piano.png'),
    ('images/Poppy.png'),
```

```sql
    ('images/Sandollar.png'),
    ('images/Smack.png'),
    ('images/Snowflake.png'),
    ('images/Soccer.png'),
    ('images/Sunflower.png'),
    ('images/Target.png'),
    ('images/Tennis.png'),
    ('images/Turntable.png'),
    ('images/Violin.png'),
    ('images/Whiterose.png'),
    ('images/Yellow Daisy.png'),
    ('images/Ying Yang.png'),
    ('images/Zebra.png'),
    ('images/Zen.png');
GO

/*to shorten this document the INSERT of Customer records has been here attenuated to include for purposes of referential integrity only those fifteen
for whom TransactedEvents are recorded in BusinessTransactions.sql */

SET IDENTITY_INSERT Sales.Customer ON;

INSERT INTO Sales.Customer(customer_id,customer_last_name,customer_first_name,customer_middle_name,customer_birth_date,membership_date,customer_street_address,customer_city,customer_state_province,customer_postal_code,country_code,customer_telephone,account_email_address,account_password_hash,account_password_salt,image_id,challenge_question_id,account_secret_answer)
VALUES
    (250,'Long','Amanda','A','1950-08-03','2012-09-01','10 Napa Ct.','Lebanon','OR','97355','US','(328) 555-0124','amanda31@adventure-works.com','0wx26ePqZmQIUxrCCi0DXnDrmnZWZG9PftHfXpz/H8A=','XK7n/Jw=',33,3,''),
    (258,'Barnes','Sarah','J','1978-04-28','2002-06-29','1011 Green St.','Bellingham','WA','98225','US','(271) 555-0194','sarah28@adventure-works.com','EhYOJWS8qekbBgQFdDTkV79HBn9GMAlRKpb8YgpUPe4=','3D/eKJQ=',20,3,''),
    (266,'Reed','Natalie',NULL,'1984-06-15','2005-07-09','1023 Hawkins Street','Lebanon','OR','97355','US','(833) 555-0176','natalie4@adventure-works.com','VxZp3PUBHmzahGfHq35qFNWJZLfACsHx6jO6yyrOKWo=','MqthIEk=',23,2,''),
    (275,'Collins','Adam','A','1974-09-19','2005-05-20','104 Hilltop Dr.','Concord','CA','94519','US','(502) 555-0184','adam32@adventure-works.com','8PCW1ZCOMmAG613RDBfFLl2XJWHiEUDsHOS/Onc6maw=','kY+pL2E=',15,2,''),
    (298,'Baker','Amanda',NULL,'1972-09-11','2004-11-15','1077 Willow Court','Imperial Beach','CA','91932','US','(991) 555-0143','amanda59@adventure-works.com','P4ar8Mc90B90HekFvfEC9QayiH6gsUvyn4QSXg5AyVs=','etyn2oY=',10,3,''),
    (302,'Baker','Lucas',NULL,'1962-09-27','2007-09-26','1086 Ash Lane','National City','CA','91950','US','(787) 555-0141','lucas48@adventure-works.com','vmblANdd38GVHJD+ETzOQaxvPhjV4ZTxuRGlOaufdRM=','mRIRGlY=',12,2,''),
    (322,'Howard','Seth',NULL,'1963-11-22','2014-11-01','1119 Elderwood Dr.','Lake Oswego','OR','97034','US','(309) 555-0116','seth82@adventure-works.com','2XOVGN8FDq7tjyHoPWN1xPGdeWowA+JwklNr8O1r65w=','z6lb8Co=',2,2,''),
    (335,'Wood','Kayla','L','1946-06-02','2014-12-08','1142 Firestone Dr.','Downey','CA','90241','US','(310) 555-0119','kayla26@adventure-works.com','W886lBaNRpOgUN2/J0IGIM0A9jiV7tbcTfD4IR16JQs=','nxmYL8E=',30,2,''),
    (344,'Hughes','Hailey','R','1956-07-31','2008-12-19','1159 LaCrosse Ave','San Francisco','CA','94109','US','(843) 555-0191','hailey31@adventure-works.com','Xt2WUuD07L4KF3H66zl9+ByF+bQwmuq9Sb7jrI/cT70=','LOiyJcc=',16,2,''),
    (416,'Hall','Xavier',NULL,'1943-03-06','2013-05-09','1293 F Street','La Jolla','CA','92806','US','(780) 555-0196','xavier21@adventure-works.com','+jlVEsZaFBHtPIkqeY89N5zG8Iqu3dkfeJ50Zr9WXBg=','saSs3v4=',34,3,''),
    (419,'Hill','Amber','W','1955-06-18','2012-10-18','1299 Carpetta Circle','Grossmont','CA','91941','US','(320) 555-0155','amber13@adventure-works.com','F6KJnGLKUXTXd/1xXSp0DgoxlpDjco4C1Nh8YcM8P+w=','wmUtl7Q=',1,3,''),
    (425,'Perry','Logan',NULL,'1957-10-29','2012-06-23','1308 Mt. Hood Circle','Chula Vista','CA','91910','US','(988) 555-0121','logan10@adventure-works.com','kWKOTJgie3gR2zLBV64+yz8pN2folu6e8bFlH1r54NY=','Vg8dTpc=',37,1,''),
    (437,'Brooks','Kaitlyn','S','1965-04-25','2009-02-17','1342 Isla Bonita','Woodland Hills','CA','91364','US','(674) 555-0174','kaitlyn65@adventure-works.com','vp6GjCyWJRL9LfThk+cA9ypHoJvbjup4h1lP/eSzyMU=','QGST5hY=',40,1,''),
    (483,'Ross','Samantha','R','1992-11-04','2010-12-08','1433 Manila Avenue','Burlingame','CA','94010','US','(956) 555-0151','samantha29@adventure-works.com','FEw1QA798LlRF6yhRzoY2AEb0VPO7uyBYP4/IKK8ceE=','v6Yyhi8=',25,2,''),
    (484,'Arun','Manuel',NULL,'1962-10-22','2010-12-15','1437 Donaleen Cr','Beverly Hills','CA','90210','US','(773) 555-0132','manuel5@adventure-works.com','tNyA2GfDejPX0oUt4e9l2rt0Crrd8XK9x5wlPkYrE/Q=','e9j1yow=',28,3,'');

SET IDENTITY_INSERT Sales.Customer OFF;
DBCC CHECKIDENT('Sales.Customer', RESEED, 500);

GO


INSERT INTO Sales.TransactionType(transaction_type_code,transaction_description)
VALUES
    (1,'movie rental'),
    (2,'movie purchase'),
    (3,'credit-card authorization'),
    (4,'credit-card payment'),
    (5,'credit-card refund'),
    (6,'file transfer'),
    (7,'disc shipment'),
    (8,'disc return'),
    (9,'other');
GO

--create some mail center distribution center data so that stored procedures in TransactionProcessing.sql can be tested

INSERT INTO Sales.Outlet
    (outlet_name,           outlet_location,    outlet_type)
VALUES
    ('New England',         'Boston, MA',       'M'),
    ('Mid-Atlantic 1',      'Newark, NJ',       'M'),
    ('Mid-Atlantic 2',      'Pittsburgh, PA',   'M'),
```

```sql
    ('Midwest North',       'Milwaukee, WI',    'M'),
    ('Midwest Central',     'St. Louis, MO',    'M'),
    ('South 1',             'Atlanta, GA',      'M'),
    ('South 2',             'Baton Rouge, LA',  'M'),
    ('Florida',             'Orlando, FL',      'M'),
    ('Pacific Northwest',   'Spokane, WA',      'M'),
    ('Southern California', 'Santa Barbara, CA','M');
GO

INSERT INTO Sales.DistributionCenter
    (outlet_id, outlet_postal_code)
VALUES
    (1,         '02133'),
    (2,         '07104'),
    (3,         '15206'),
    (4,         '53202'),
    (5,         '63116'),
    (6,         '30326'),
    (7,         '70825'),
    (8,         '32805'),
    (9,         '99205'),
    (10,        '93101');
GO

INSERT INTO Sales.PostalCodeAssignment
    (country_code,  postal_code,    outlet_id_1,    outlet_id_2,    outlet_id_3)
VALUES
    ('US',          '98225',        9,              10,             NULL),
    ('US',          '91950',        10,             9,              NULL),
    ('US',          '97034',        9,              10,             NULL),
    ('US',          '91364',        10,             9,              NULL),
    ('US',          '90210',        10,             9,              NULL),
    ('US',          '90241',        10,             9,              NULL),
    ('US',          '94010',        9,              10,             NULL),
    ('US',          '92806',        10,             9,              NULL),
    ('US',          '94109',        9,              10,             NULL),
    ('US',          '94519',        9,              10,             NULL),
    ('US',          '91932',        10,             9,              NULL),
    ('US',          '91941',        10,             9,              NULL),
    ('US',          '20151',        6,              8,              NULL),
    ('US',          '45202',        3,              5,              NULL),
    ('US',          '77840',        7,              5,              NULL);
GO
```

```sql
/*StarSchemaBuild.sql
 *SQL DDL code to build the proposed Flix2You OLAP database
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015 */

USE master;
GO

--Delete the Flix2YouDW database if it exists.
IF EXISTS(SELECT * from sys.databases WHERE name='Flix2YouDW')
BEGIN
    DROP DATABASE Flix2YouDW;
END;
GO

--Create a new database called Flix2YouDW.
CREATE DATABASE Flix2YouDW;
GO

USE Flix2YouDW;
GO

--Create the schema Sales for the Flix2YouDW database if it does not exist.
IF (SCHEMA_ID('Sales') IS NULL)
BEGIN
    EXECUTE('CREATE SCHEMA Sales AUTHORIZATION dbo');
END;
GO

--Drop the DimMovie table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'DimMovie'))
BEGIN
   DROP TABLE Sales.DimMovie;
END;
GO

--Create the DimMovie table (without foreign-key constraints, for now)
CREATE TABLE Sales.DimMovie
(
    movie_id                char(26)        NOT NULL    PRIMARY KEY,
    movie_title             nvarchar(50)    NOT NULL,
    studio_id               char(9)         NOT NULL,
    release_date            date            NOT NULL,
    country_code            nchar(2)        NOT NULL,
    director_id             smallint        NOT NULL,
    movie_length            smallint        NOT NULL,
    genre_code              tinyint         NOT NULL,
    movie_description       nvarchar(max),
    CONSTRAINT  check_EIDR  CHECK(movie_id LIKE     '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                    '[0-9A-Z]')
);
GO

--Drop the DimStudio table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
```

```sql
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'DimStudio'))
BEGIN
   DROP TABLE Sales.DimStudio;
END;
GO

--Create the DimStudio table
CREATE TABLE Sales.DimStudio
(
    studio_id                   char(9)         NOT NULL    PRIMARY KEY,
    studio_name                 nvarchar(50)    NOT NULL,
    production_company          nvarchar(50),
    country_code                nchar(2)        NOT NULL,
    CONSTRAINT  check_studio_id     CHECK(studio_id LIKE    '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                                                            '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]')
);
GO

--Drop the DimDirector table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'DimDirector'))
BEGIN
   DROP TABLE Sales.DimDirector;
END;
GO

--Create the DimDirector table
CREATE TABLE Sales.DimDirector
(
    director_id             smallint        NOT NULL    PRIMARY KEY,
    director_name_first     nvarchar(35)    NOT NULL,
    director_name_last      nvarchar(35)    NOT NULL
);
GO

--Drop the DimGenre table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'DimGenre'))
BEGIN
   DROP TABLE Sales.DimGenre;
END;
GO

--Create the DimGenre table
CREATE TABLE Sales.DimGenre
(
    genre_code              tinyint         NOT NULL    PRIMARY KEY,
    genre_description       nvarchar(35)    NOT NULL    UNIQUE
);
GO

--Drop the DimActor table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'DimActor'))
BEGIN
```

```sql
   DROP TABLE Sales.DimActor;
END;
GO

--Create the DimActor table
CREATE TABLE Sales.DimActor
(
    actor_id            smallint        NOT NULL    PRIMARY KEY,
    actor_gender        char(1)         NOT NULL,
    actor_name_first    nvarchar(35)    NOT NULL,
    actor_name_last     nvarchar(35)    NOT NULL,
    other_actor_details nvarchar(max),
    CONSTRAINT  check_actor_gender  check(actor_gender = 'M' OR actor_gender = 'F')
);
GO

--Drop the FactMovieRole table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'FactMovieRole'))
BEGIN
   DROP TABLE Sales.FactMovieRole;
END;
GO

--Create the FactMovieRole table
CREATE TABLE Sales.FactMovieRole
(
    movie_id        char(26)        NOT NULL,
    actor_id        smallint        NOT NULL,
    as_character    nvarchar(50),
    is_lead         bit,
    weight_in_cast  decimal
    CONSTRAINT  fk_movierole_1  FOREIGN KEY(movie_id)   REFERENCES Sales.DimMovie(movie_id),
    CONSTRAINT  fk_movierole_2  FOREIGN KEY(actor_id)   REFERENCES Sales.DimActor(actor_id),
    CONSTRAINT  pk_movie_role   PRIMARY KEY(movie_id, actor_id)
);
GO

--Add the foreign-key constraints to the DimMovie table
ALTER TABLE Sales.DimMovie ADD
    CONSTRAINT  fk_movie_1  FOREIGN KEY(studio_id)      REFERENCES Sales.DimStudio(studio_id),
    CONSTRAINT  fk_movie_2  FOREIGN KEY(director_id)    REFERENCES Sales.DimDirector(director_id),
    CONSTRAINT  fk_movie_3  FOREIGN KEY(genre_code)     REFERENCES Sales.DimGenre(genre_code);
GO

--Drop the DimHighLevelFormat table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'DimHighLevelFormat'))
BEGIN
   DROP TABLE Sales.DimHighLevelFormat;
END;
GO

--Create the DimHighLevelFormat table
CREATE TABLE Sales.DimHighLevelFormat
(
    format_code         char(6)         NOT NULL    PRIMARY KEY,
    format_name         varchar(35)     NOT NULL,
    physical_medium     char(4)         NOT NULL,
```

```sql
    definition_level    varchar(8)      NOT NULL,
    aspect_ratio        varchar(15),
    CONSTRAINT ck_medium    CHECK(physical_medium = 'disc' OR physical_medium = 'file')
);
GO

--Drop the DimTransactionType table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'DimTransactionType'))
BEGIN
   DROP TABLE Sales.DimTransactionType;
END;
GO

--Create the DimTransactionType table
CREATE TABLE Sales.DimTransactionType
(
    transaction_type_code       tinyint     NOT NULL    PRIMARY KEY,
    transaction_description     varchar(50) NOT NULL    UNIQUE
);
GO

--Drop the DimGeography table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'DimGeography'))
BEGIN
   DROP TABLE Sales.DimGeography;
END;
GO

--Create the DimGeography table
CREATE TABLE Sales.DimGeography
(
    customer_postal_code        nchar(10)       NOT NULL,
    country_code                nchar(2)        NOT NULL,
    city                        nvarchar(35)    NOT NULL,
    state_province_code         nchar(2)        NOT NULL,
    state_province_name         nvarchar(35),   --nullable for loading convenience
    region_name                 nvarchar(35),   --nullable for loading convenience
    country_name                nvarchar(35)    NOT NULL,
    continent                   nvarchar(15)    --nullable for loading convenience
    CONSTRAINT  pk_dimgeography     PRIMARY KEY(country_code, customer_postal_code, city)
);
GO

--Drop the DimCustomer table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'DimCustomer'))
BEGIN
   DROP TABLE Sales.DimCustomer;
END;
GO

--Create the DimCustomer table (without foreign-key constraints, for the moment)
CREATE TABLE Sales.DimCustomer
(
    customer_id                 int             NOT NULL    PRIMARY KEY,
```

```sql
    customer_last_name          nvarchar(35)    NOT NULL,
    customer_first_name         nvarchar(35)    NOT NULL,
    customer_middle_name        nvarchar(35),
    customer_postal_code        nchar(10)       NOT NULL,
    customer_city               nvarchar(35)    NOT NULL,
    country_code                nchar(2)        NOT NULL,
    membership_date             date            NOT NULL,
    customer_birth_date         date            NOT NULL,
    gender_code                 char(1),
    marital_status_code         char(1),
    children_at_home            tinyint,
    household_income            money,
    education_level_code        tinyint,
    occupational_type           varchar(15),
    CONSTRAINT  check_customer_dob      CHECK(DATEDIFF(year, customer_birth_date, GETDATE())
                                                                        BETWEEN 16 AND 95),
    CONSTRAINT  check_membership_date   CHECK(membership_date BETWEEN '1995-01-01' AND GETDATE()),
    CONSTRAINT  fk_dimcustomer_1        FOREIGN KEY(country_code, customer_postal_code, customer_city)
                                            REFERENCES  Sales.DimGeography(country_code,
                                                customer_postal_code, city)
                                                --(Canadian postal codes contain multiple cities)
);
GO

--Drop the DimDateTime table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME =   'DimDateTime'))
BEGIN
   DROP TABLE Sales.DimDateTime;
END;
GO

--Create the DimDateTime table
CREATE TABLE Sales.DimDateTime
(
    transaction_timestamp   datetime    NOT NULL    PRIMARY KEY,
/*NOTE: The following fields lack a NOT NULL constraine for convenience only,
  to facilitate the data load process.*/
    hour_of_day             int,
    day_of_week             int,
    weekday_name            varchar(10),
    day_of_month            int,
    month_of_year           int,
    month_name              varchar(10),
    calendar_quarter        int,
    calendar_year           int
);
GO

--Drop the FactSalesTransaction table if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE  TABLE_SCHEMA = 'Sales'
                AND    TABLE_NAME = 'FactSalesTransaction'))
BEGIN
   DROP TABLE Sales.FactSalesTransaction;
END;
GO

--Create the FactSalesTransaction table
```

```sql
CREATE TABLE Sales.FactSalesTransaction
(
    transaction_id              bigint      NOT NULL    PRIMARY KEY,
    transaction_timestamp       datetime    NOT NULL,
    transaction_type_code       tinyint     NOT NULL,
    format_code                 char(6)     NOT NULL,
    movie_id                    char(26)    NOT NULL,
    customer_id                 int         NOT NULL,
    transaction_amount          money       NOT NULL
    CONSTRAINT  fk_transaction_1    FOREIGN KEY(transaction_timestamp)
                                        REFERENCES Sales.DimDateTime(transaction_timestamp),
    CONSTRAINT  fk_transaction_2    FOREIGN KEY(transaction_type_code)
                                        REFERENCES Sales.DimTransactionType(transaction_type_code),
    CONSTRAINT  fk_transaction_3    FOREIGN KEY(format_code)
                                        REFERENCES Sales.DimHighLevelFormat(format_code),
    CONSTRAINT  fk_transaction_4    FOREIGN KEY(movie_id)
                                        REFERENCES Sales.DimMovie(movie_id),
    CONSTRAINT  fk_transaction_5    FOREIGN KEY(customer_id)
                                        REFERENCES Sales.DimCustomer(customer_id),
    CONSTRAINT  unique_transaction  UNIQUE(transaction_timestamp, transaction_type_code,
                                                                  format_code,movie_id, customer_id)
);
GO

--Check for the existence of a named DimMovie index on the FactSalesTransaction table
IF EXISTS (SELECT *  FROM sys.indexes  WHERE name='idxTransaction_Movie'
    AND object_id = OBJECT_ID('Sales.FactSalesTransaction'))
BEGIN
    DROP INDEX idxTransaction_Movie ON Sales.FactSalesTransaction;
END;

--Create a named DimMovie index on the FactSalesTransaction table
CREATE INDEX idxTransaction_Movie ON Sales.FactSalesTransaction(movie_id);

--Check for the existence of a named DimCustomer index on the FactSalesTransaction table
IF EXISTS (SELECT *  FROM sys.indexes  WHERE name='idxTransaction_Customer'
    AND object_id = OBJECT_ID('Sales.FactSalesTransaction'))
BEGIN
    DROP INDEX idxTransaction_Customer ON Sales.FactSalesTransaction;
END;

--Create a named DimCustomer index on the FactSalesTransaction table
CREATE INDEX idxTransaction_Customer ON Sales.FactSalesTransaction(customer_id);
GO
```

```sql

/*StarScehmaLoad.sql
 *SQL DML code to load the proposed Flix2You OLAP database with OLTP data
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015*/

INSERT INTO Flix2YouDW.Sales.DimStudio
(
    studio_id,
    studio_name,
    production_company,
    country_code
)
SELECT
    studio_id,
    studio_name,
    production_company,
    country_code
FROM Flix2You.Sales.Studio;
GO

INSERT INTO Flix2YouDW.Sales.DimDirector
(
    director_id,
    director_name_first,
    director_name_last
)
SELECT
    director_id,
    director_name_first,
    director_name_last
FROM Flix2You.Sales.Director;
GO

INSERT INTO Flix2YouDW.Sales.DimGenre
(
    genre_code,
    genre_description
)
SELECT
    genre_code,
    genre_description
FROM Flix2You.Sales.Genre;

INSERT INTO Flix2YouDW.Sales.DimMovie
(
    movie_id,
    movie_title,
    studio_id,
    release_date,
    country_code,
    director_id,
    movie_length,
    genre_code,
    movie_description
)
SELECT
    movie_id,
    movie_title,
    studio_id,
```

```
    release_date,
    country_code,
    director_id,
    movie_length,
    genre_code,
    movie_description
FROM Flix2You.Sales.Movie;
GO

INSERT INTO Flix2YouDW.Sales.DimActor
(
    actor_id,
    actor_gender,
    actor_name_first,
    actor_name_last,
    other_actor_details
)
SELECT
    actor_id,
    actor_gender,
    actor_name_first,
    actor_name_last,
    other_actor_details
FROM Flix2You.Sales.Actor;
GO

INSERT INTO Flix2YouDW.Sales.FactMovieRole
(
    movie_id,
    actor_id,
    as_character,
    is_lead
)
SELECT
    movie_id,
    actor_id,
    as_character,
    is_lead
FROM Flix2You.Sales.MovieRole;
GO

INSERT INTO Flix2YouDW.Sales.DimHighLevelFormat
(
    format_code,
    format_name,
    physical_medium,
    definition_level,
    aspect_ratio
)
SELECT
    format_code,
    format_name,
    physical_medium,
    definition_level,
    aspect_ratio
FROM Flix2You.Sales.HighLevelFormat;
GO

INSERT INTO Flix2YouDW.Sales.DimTransactionType
(
```

```sql
    transaction_type_code,
    transaction_description

)
SELECT
    transaction_type_code,
    transaction_description
FROM Flix2You.Sales.TransactionType
WHERE transaction_type_code < 3;
GO

INSERT INTO Flix2YouDW.Sales.DimGeography
(
    customer_postal_code,
    country_code,
    city,
    state_province_code,
    country_name

)
SELECT
    DISTINCT    a.customer_postal_code,
                a.country_code,
                a.customer_city,
                a.customer_state_province,
                country_name
FROM Flix2You.Sales.Customer AS a
JOIN Flix2You.Sales.Country  AS b
    ON a.country_code = b.country_code
ORDER BY a.country_code, a.customer_postal_code
GO

INSERT INTO Flix2YouDW.Sales.DimCustomer
(
    customer_id,
    customer_last_name,
    customer_first_name,
    customer_middle_name,
    customer_postal_code,
    customer_city,
    country_code,
    membership_date,
    customer_birth_date

)
SELECT
    customer_id,
    customer_last_name,
    customer_first_name,
    customer_middle_name,
    customer_postal_code,
    customer_city,
    country_code,
    membership_date,
    customer_birth_date
FROM Flix2You.Sales.Customer;
GO

INSERT INTO Flix2YouDW.Sales.DimDateTime
(
    transaction_timestamp,
    hour_of_day,
    day_of_week,
```

```sql
    weekday_name,
    day_of_month,
    month_of_year,
    month_name,
    calendar_quarter,
    calendar_year
)
SELECT
    transaction_timestamp,
    DATEPART(hh, transaction_timestamp),
    DATEPART(dw, transaction_timestamp),
    DATENAME(dw, transaction_timestamp),
    DATEPART(dd, transaction_timestamp),
    DATEPART(mm, transaction_timestamp),
    DATENAME(mm, transaction_timestamp),
    CASE
        WHEN DATEPART(mm, transaction_timestamp) BETWEEN 1 AND 3 THEN 1
        WHEN DATEPART(mm, transaction_timestamp) BETWEEN 4 AND 6 THEN 2
        WHEN DATEPART(mm, transaction_timestamp) BETWEEN 7 AND 9 THEN 3
        ELSE 4
    END,
    DATEPART(yyyy, transaction_timestamp)
FROM Flix2You.Sales.TransactedEvent
WHERE transaction_type_code < 3
ORDER BY transaction_timestamp ASC;
GO

INSERT INTO Flix2YouDW.Sales.FactSalesTransaction
(
    transaction_id,
    transaction_timestamp,
    transaction_type_code,
    format_code,
    movie_id,
    customer_id,
    transaction_amount
)
SELECT
    transaction_id,
    transaction_timestamp,
    transaction_type_code,
    format_code,
    movie_id,
    customer_id,
    transaction_amount
FROM Flix2You.Sales.TransactedEvent
WHERE transaction_type_code < 3
ORDER BY transaction_id ASC;
GO

UPDATE Flix2YouDW.Sales.DimGeography
SET continent = 'North America'
WHERE country_code IN ('CA', 'US');
GO

UPDATE Flix2YouDW.Sales.DimGeography
SET region_name = (
    CASE
        WHEN state_province_code IN ('CA', 'OR', 'WA')  THEN 'US West'
        WHEN state_province_code IN ('TX')              THEN 'US Southwest'
```

```sql
        WHEN state_province_code IN ('OH')              THEN 'US Midwest'
        WHEN state_province_code IN ('VA')              THEN 'US South'
    END)
WHERE state_province_code IN ('CA', 'OH', 'OR', 'TX', 'VA', 'WA');
GO

UPDATE Flix2YouDW.Sales.DimGeography
SET region_name = 'Canada West'
WHERE state_province_code IN ('BC');
GO

UPDATE Flix2YouDW.Sales.DimGeography
SET state_province_name = (
CASE state_province_code
    WHEN 'WA' THEN 'Washington'
    WHEN 'BC' THEN 'British Columbia'
    WHEN 'CA' THEN 'California'
    WHEN 'OH' THEN 'Ohio'
    WHEN 'OR' THEN 'Oregon'
    WHEN 'TX' THEN 'Texas'
    WHEN 'VA' THEN 'Virginia'
    WHEN 'WA' THEN 'Washington'
END)
WHERE state_province_code IN ('BC', 'CA', 'OH', 'OR', 'TX', 'VA', 'WA');
GO
```

```sql
/*TransactionProcessing.sql
 *Example procedural SQL code to process routine business transacions of Flix2You;
 *written and run to verify the adequacy of the proposed new OLTP model
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015*/


--(1) Stored procedure to record the purchase/rental of a video disc or file by a customer:

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspRecordSale', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspRecordSale;
GO

CREATE PROCEDURE Sales.uspRecordSale    @customer_ID            int,
                                        @movie_id               char(26),
                                        @format_code            char(6),
                                        @transaction_type_code  tinyint     AS
BEGIN
SET NOCOUNT ON;

DECLARE @is_available           bit,
        @transaction_amount     money,
        @base_transaction_id    bigint;

--abort with printed message if the object is not available
SET @is_available = (
    SELECT  is_available FROM Sales.ViewableEncoding
    WHERE   movie_id =      @movie_id
    AND     format_code =   @format_code )
IF  @is_available = 0
BEGIN
    --not the ideal way to handle the situation, but this could be replaced later with a call to another procedure
    PRINT N'The movie ' + @movie_id + N' is not available as a ' + @format_code +
                                        N' for customer ' + @customer_ID + N', ' + GETDATE();
    RETURN; --a predefined value value could also be returned here
END

--the price might probably also be more easily passed in a parameter by the caller too. But with this design:
IF      @transaction_type_code =
        (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'movie rental')
BEGIN
    SET @transaction_amount = (
        SELECT  rental_price FROM Sales.ViewableEncoding
        WHERE   movie_id =  @movie_id
        AND     format_code =   @format_code );
END;
ELSE IF @transaction_type_code =
        (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'movie purchase')
BEGIN
    SET @transaction_amount = (
        SELECT  purchase_price FROM Sales.ViewableEncoding
        WHERE   movie_id  = @movie_id
        AND     format_code =   @format_code );
END;
ELSE
    THROW 50000, 'Invalid procedure parameter value.', 1;

SET @base_transaction_id = NEXT VALUE FOR Sales.TransactionSequence;

INSERT INTO Sales.TransactedEvent (
                transaction_id,
                movie_id,
                format_code,
                customer_id,
                transaction_timestamp,
                transaction_type_code,
                transaction_amount,
                previous_transaction_id )
VALUES
--record the rental or purchase of a file or disc
(
    @base_transaction_id,
    @movie_id,
    @format_code,
    @customer_id,
    GETDATE(),
```

```sql
    @transaction_type_code,
    @transaction_amount,
    NULL),
--record the necessary credit-card authorization
(
    NEXT VALUE FOR Sales.TransactionSequence,
    @movie_id,
    @format_code,
    @customer_id,
    GETDATE(),
    (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'credit-card authorization'),
    @transaction_amount,
    @base_transaction_id);

SET NOCOUNT OFF;
END;
GO

/*Some procedure with the logical effect of the code just given above should be executed by the company's web application
as soon as the third-party payment processor returns to the web application a confirmation that the customer's bank has
authorized the credit-card transaction that will pay for the movie rental or purchase.

In a streaming media transaction the web application will then bring about the display in the customer's web browser of a
link that will permit the customer to navigate to a CDN server cluster from which he can download the movie. In one
elegant solution using what is called IP anycast, the same IP address is assigned to all of the CDN clusters serving
movies, each of which then advertises this IP through the Border Gateway Protocol; the  normal mechanism of BGP is then
relied upon to select the route from the customer to a server cluster that requires the fewest link hops.

If the content distribution network used to stream video files is owned and operated by Flix2You, the transaction_id generated
and associated with the sale transaction in the foregoing procedure can be appended as a parameter to the connection URL returned
to the customer, for whom it will then serve as his "admission ticket" to the movie server. In a secured scenario:
a) a hash of the concatenation of the transaction_id with a random salt (to serve as a nonce and prevent replay attacks) is tacked
onto the connection URL as a parameter, together with the cleartext random salt;
b) the cleartext random salt is broadcast by the web application to all of its CDN clusters, where is stored for a limited period
determined by a fixed timeout value; and
c) when a cluster receives a request for a movie download from a Flix2You customer, it compares the random salt attached to the
connection URL with its list of unexpired values and, if a match is found, concatenates the salt with the transaction_ids of all
recent sales in the operational database and calculates a hash of each. Any hash that matches the one attached to the
customer's connection URL will validate his "admission ticket" to the movie, format, and transaction type of that TransactedEvent.

A hashing process like the one described above may be necessary despite the fact that the TCP/IP connection of the customer
to the web server was maintained throughout with TLS, because of the initially unsecured handoff to the CDN server cluster.

The corresponding procedures used by the mail distribution centers that process disc rentals and purchases will of course be
much different. Some of the database-accessing SQL that these may employ will resemble code given below.

In any case when the download of a file begins, or a disc is placed in the mail for delivery, a procedure with logic equivalent
to that of the stored procedure shown below must be executed by the distribution outlet, whose business responsibility it
will also be to then initiate collection against the credit-card authorization that preceded the sale.*/

--(2) Stored procedure to record the shipment of a video disc or transfer of a file to a customer,
--and the accompanying credit-card payment:

/*NOTE: This procedure does not reduce the number_on_hand attribute for the DiscAtMailCenter
entity occurrence that references some disc that is an instance of the rented or purchased
ViewableEncoding, and that is actually mailed to a customer. This is a more specialized
business process than the general transaction type handled in this procedure. See (6)
below, by which a mail DistributionCenter determines whether it has the necessary discs
on hand to fulfill pending orders.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspRecordTransfer', 'P' ) IS NOT NULL
   DROP PROCEDURE Sales.uspRecordTransfer;
GO

/*the previous_transaction_id  for the movie rental or purchase should be available as a
 matter of business-process routine when this procedure is called*/

CREATE PROCEDURE Sales.uspRecordTransfer    @previous_transaction_id    bigint  AS

BEGIN
SET NOCOUNT ON;

DECLARE @customer_ID            int,
        @movie_id               char(26),
        @format_code            char(6),
        @transaction_amount     money,
        @physical_medium        char(4);
```

```sql

DECLARE sale_cursor CURSOR FOR

    SELECT  customer_ID,
            movie_id,
            format_code,
            transaction_amount
    FROM Sales.TransactedEvent
    WHERE transaction_id = @previous_transaction_id;

OPEN sale_cursor;
FETCH sale_cursor INTO
    @customer_ID,
    @movie_id,
    @format_code,
    @transaction_amount;

IF  @@FETCH_STATUS <> 0
BEGIN
    PRINT N'A rental or purchase transaction with the ID ' + @previous_transaction_id + N' could not be located in the database.';
    RETURN; --a predefined error value might also be returned here
END;

SET @physical_medium = (SELECT physical_medium FROM Sales.HighLevelFormat WHERE format_code = @format_code);

INSERT INTO Sales.TransactedEvent (
                transaction_id,
                movie_id,
                format_code,
                customer_id,
                transaction_timestamp,
                transaction_type_code,
                transaction_amount,
                previous_transaction_id)
VALUES
--record the disc shipment or file transfer transaction
(
    NEXT VALUE FOR Sales.TransactionSequence,
    @movie_id,
    @format_code,
    @customer_id,
    GETDATE(),
    CASE @physical_medium
        WHEN 'disc' THEN (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'disc shipment')
        WHEN 'file' THEN (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'file transfer')
    END,
    @transaction_amount, --might also use NULL for the value of this attribute here; but vendors might be entitled to a percentage
    @previous_transaction_id
),
--record the credit-card payment concurrently collected
(
    NEXT VALUE FOR Sales.TransactionSequence,
    @movie_id,
    @format_code,
    @customer_id,
    GETDATE(),
    (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'credit-card payment'),
    @transaction_amount,
    @previous_transaction_id
);

CLOSE sale_cursor;
DEALLOCATE sale_cursor;
SET NOCOUNT OFF;

END;
GO

--(3) Code to create a current view of all recent (up to one week old) unfilled disc orders awaiting mail shipment:

USE Flix2You;
GO

--Drop the vwUnfilledDiscOrderview if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.VIEWS
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'vwUnfilledDiscOrder'))
BEGIN
   DROP VIEW Sales.vwUnfilledDiscOrder;
```

```sql
END;
GO

CREATE VIEW Sales.vwUnfilledDiscOrder AS
SELECT
    --get the designated primary mail distribution center
    p.outlet_id_1,
    --get the movie order reference number, and date and time
    t1.transaction_id, t1.transaction_timestamp,
    --get the customer id and mailing information
    c.customer_id, c.customer_first_name, c.customer_last_name,
    c.customer_street_address, c.customer_city, c.customer_state_province, c.customer_postal_code, c.country_code,
    --get the name and format of the ordered movie
    t1.movie_id, t1.format_code
FROM
    Sales.Customer AS c
INNER JOIN
    Sales.TransactedEvent AS t1
    ON  c.customer_id = t1.customer_id
INNER JOIN
    Sales.PostalCodeAssignment AS p
    ON  c.country_code = p.country_code
    AND c.customer_postal_code = p.postal_code
WHERE
    --apply a moving time-based search limitation
    DATEDIFF(second, GETDATE(), t1.transaction_timestamp) <= 7 * 24 * 3600
AND
    --ignore orders for streaming video
    t1.format_code  IN (SELECT format_code FROM Sales.HighLevelFormat
                        WHERE physical_medium = 'disc')
AND
    --get rental or purchase transactions only
    t1.transaction_type_code  IN (SELECT transaction_type_code FROM Sales.TransactionType
                                  WHERE transaction_description IN ('movie rental', 'movie purchase'))
AND
    --eliminate orders that have already been shipped
    NOT EXISTS (SELECT t2.transaction_id FROM Sales.TransactedEvent t2
                WHERE t2.previous_transaction_id = t1.transaction_id
                AND t2.transaction_type_code = (SELECT transaction_type_code FROM Sales.TransactionType
                                                WHERE transaction_description = 'disc shipment'));
GO

--(4) Stored procedure to return a cursor for a single mail outlet's pending orders from the above view

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetPendingDiscOrderCursor', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetPendingDiscOrderCursor;
GO

CREATE PROCEDURE Sales.uspGetPendingDiscOrderCursor
                @PendingDiscOrderCursor CURSOR VARYING OUTPUT,
                @outlet_id  smallint
                AS
BEGIN
SET NOCOUNT ON;

SET @PendingDiscOrderCursor = CURSOR SCROLL DYNAMIC FOR
    SELECT  transaction_id, transaction_timestamp,
            movie_id, format_code, customer_id,
            customer_first_name, customer_last_name, customer_street_address, customer_city, customer_state_province,
              customer_postal_code, country_code
    FROM Sales.vwUnfilledDiscOrder
    WHERE outlet_id_1 = @outlet_id
    ORDER BY transaction_timestamp;

OPEN @PendingDiscOrderCursor;

SET NOCOUNT OFF;
END;
GO

--(5) Stored procedure to print a mail outlet's pending unshipped orders using a cursor selected from a view

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspPrintUnshippedDiscOrders', 'P' ) IS NOT NULL
```

```sql
    DROP PROCEDURE Sales.uspPrintUnshippedDiscOrders;
GO

CREATE PROCEDURE Sales.uspPrintUnshippedDiscOrders  @of_outlet_id   smallint AS

BEGIN
SET NOCOUNT ON;

DECLARE @ProcessingCursor CURSOR;
DECLARE @transaction_id             bigint,
        @transaction_timestamp      datetime,
        @movie_id                   char(26),
        @format_code                char(6),
        @customer_id                int,
        @customer_first_name        nvarchar(35),
        @customer_last_name         nvarchar(35),
        @customer_street_address    nvarchar(35),
        @customer_city              nvarchar(35),
        @customer_state_province    nchar(2),
        @customer_postal_code       nchar(10),
        @country_code               nchar(2);

EXECUTE Sales.uspGetPendingDiscOrderCursor  @PendingDiscOrderCursor = @ProcessingCursor OUTPUT,
                                            @outlet_id = @of_outlet_id;


PRINT N'Unshipped Disc Orders for Mail Distribution Outlet #' + CAST(@of_outlet_id AS varchar(4)) + CHAR(13) + CHAR(10);
PRINT CHAR(13) + CHAR(10);
FETCH NEXT FROM @ProcessingCursor INTO
    @transaction_id,
    @transaction_timestamp,
    @movie_id,
    @format_code,
    @customer_id,
    @customer_first_name,
    @customer_last_name,
    @customer_street_address,
    @customer_city,
    @customer_state_province,
    @customer_postal_code,
    @country_code;

WHILE (@@FETCH_STATUS = 0)
BEGIN;

    PRINT N'Transaction ID: ' + CAST(@transaction_id AS varchar(15)) + CHAR(13) + CHAR(10);
    PRINT N'Transaction Timestamp: ' + CONVERT(varchar(35), @transaction_timestamp, 100) + CHAR(13) + CHAR(10);
    PRINT N'Customer ID: ' + CAST(@customer_id AS varchar(10)) + CHAR(13) + CHAR(10);
    PRINT N'Customer Name and Address:' + CHAR(13) + CHAR(10);
    PRINT @customer_first_name + ' ' + @customer_last_name + CHAR(13) + CHAR(10);
    PRINT @customer_street_address + CHAR(13) + CHAR(10);
    PRINT @customer_city + ', ' + @customer_state_province + ' ' + @customer_postal_code + CHAR(13) + CHAR(10);
    PRINT N'Country: ' + @country_code + CHAR(13) + CHAR(10);
    PRINT N'Movie ID: ' + @movie_id + CHAR(13) + CHAR(10);
    PRINT N'Format Code: ' + @format_code + CHAR(13) + CHAR(10);
    PRINT CHAR(13) + CHAR(10);

    FETCH NEXT FROM @ProcessingCursor INTO
    @transaction_id,
    @transaction_timestamp,
    @movie_id,
    @format_code,
    @customer_id,
    @customer_first_name,
    @customer_last_name,
    @customer_street_address,
    @customer_city,
    @customer_state_province,
    @customer_postal_code,
    @country_code;

END;

CLOSE @ProcessingCursor;
DEALLOCATE @ProcessingCursor;

SET NOCOUNT OFF;
END;
GO
```

```sql

--(6) Stored procedure to create a table showing the present disc shortages at a mail center (here, the center with outlet_id = 10)

/*NOTE: No SQL code is provided here to illustrate how the OutletResource entity occurrences that stand in a child-to-parent
relationship with a given HighLevelFormat may be martialed by an Outlet for file transfer or shipment in response to a
particular customer order. Association of an OutletResource with a HighlevelFormat can involve a complicated process of
online negotiation between the company's and customer's hosts, in which information about the customer's country, device
type and available bandwidth- in addition to the definition-level and aspect-ratio attribute values belonging to the named
HighLevelFormat value instance- will be exchanged. The process might well require the creation of a number of associative
tables in the database to implement M:N relationships.

The process is less complicated in the case of discs, but the customer's country of residence must also at a minimum also be
considered for determining format.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspDiscShortagesAt10', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspDiscShortagesAt10;
GO

CREATE PROCEDURE Sales.uspDiscShortagesAt10 AS
BEGIN
SET NOCOUNT ON;

IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE TABLE_SCHEMA = 'Sales'
                 AND  TABLE_NAME = 'DiscInventoryShortageAtCenter10'))
BEGIN
   DROP TABLE Sales.DiscInventoryShortageAtCenter10;
END;


SELECT * INTO Sales.DiscInventoryShortageAtCenter10
FROM (
SELECT
    v.movie_id,
    v.format_code,
    COUNT(v.transaction_id) AS Needed,
    0 AS On_Hand,
    COUNT(v.transaction_id) AS Deficit
FROM
    Sales.vwUnfilledDiscOrder AS v
WHERE NOT EXISTS   (SELECT d1.resource_id
                    FROM Sales.DiscAtMailCenter AS d1
                    WHERE d1.outlet_id = 10
                    AND   d1.resource_id IN
                        (SELECT r1.resource_id FROM Sales.OutletResource AS r1
                         WHERE  r1.movie_id = v.movie_id
                         AND    r1.format_code = v.format_code))
GROUP BY
    v.movie_id, v.format_code

UNION

SELECT
    v.movie_id,
    v.format_code,
    COUNT(v.transaction_id) AS Needed,
   (SELECT SUM(d2.number_on_hand)
    FROM Sales.DiscAtMailCenter AS d2
    WHERE d2.outlet_id =  10 --@outlet_id
    AND   d2.resource_id IN
       (SELECT r2.resource_id FROM Sales.OutletResource AS r2
        WHERE  r2.movie_id = v.movie_id
        AND    r2.format_code = v.format_code)) AS On_Hand,
   (COUNT(v.transaction_id) -
   (SELECT SUM(d2.number_on_hand)
    FROM Sales.DiscAtMailCenter AS d2
    WHERE d2.outlet_id =  10 --@outlet_id
    AND   d2.resource_id IN
       (SELECT r2.resource_id FROM Sales.OutletResource AS r2
        WHERE  r2.movie_id = v.movie_id
        AND    r2.format_code = v.format_code))) AS Deficit
FROM
    Sales.vwUnfilledDiscOrder AS v
WHERE EXISTS   (SELECT d1.resource_id
                FROM Sales.DiscAtMailCenter AS d1
```

```sql
                WHERE d1.outlet_id = 10 --@outlet_id
                AND   d1.resource_id IN
                    (SELECT r1.resource_id FROM Sales.OutletResource AS r1
                     WHERE  r1.movie_id = v.movie_id
                     AND    r1.format_code = v.format_code))
GROUP BY
    v.movie_id, v.format_code
HAVING
    COUNT(v.transaction_id) >
   (SELECT SUM(d2.number_on_hand)
    FROM Sales.DiscAtMailCenter AS d2
    WHERE d2.outlet_id =  10 --@outlet_id
    AND   d2.resource_id IN
       (SELECT r2.resource_id FROM Sales.OutletResource AS r2
        WHERE  r2.movie_id = v.movie_id
        AND    r2.format_code = v.format_code))

) AS TemporaryUnion;

SET NOCOUNT OFF;
END;
GO

/**(7) (placeholder) code to create a list of unfillable orders to send to the designated secondary mail distribution
center for fulfilment: Code would use a cursor to successively fetch rows from the table created in (6). The variables filled
with the values from each row would then be used to perform a SELECT TOP (@Deficit) ... FROM Sales.wwUnfilledDiscOrder
WHERE outlet_id = XX AND movie_id = @movie_id AND format_code = @format_code ORDER BY transaction_timestamp DESC,
with the orders selected like this then being sent to the secondary mail center associated with each customer's postal code.*/


--(8) Stored procedure to query all unreturned and overdue disc rentals for the customer identified by @customer_id,
--where the rental term is specified by @days_rented:

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspFindAllDiscsOverdueFromCustomer', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspFindAllDiscsOverdueFromCustomer;
GO

CREATE PROCEDURE Sales.uspFindAllDiscsOverdueFromCustomer   @customer_id    int,
                                                            @days_rented    int AS
BEGIN
SET NOCOUNT ON;

SELECT  Table1.rental_id,
        Table1.rental_date,
        Table2.return_id,
        Table2.return_date
FROM (
    SELECT  transaction_id          AS rental_id,
            transaction_timestamp   AS rental_date
    FROM Sales.TransactedEvent
    WHERE   customer_id = @customer_id
    AND     format_code IN
        (SELECT format_code FROM Sales.HighLevelFormat WHERE physical_medium = 'disc')
    AND     transaction_type_code =
        (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'movie rental') )  AS Table1
LEFT OUTER JOIN (
    SELECT  transaction_id          AS return_id,
            transaction_timestamp   AS return_date,
            previous_transaction_id AS rental_id
    FROM Sales.TransactedEvent
    WHERE   customer_id = @customer_id
    AND     format_code IN
        (SELECT format_code FROM Sales.HighLevelFormat WHERE physical_medium = 'disc')
    AND     transaction_type_code =
        (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'disc return') )   AS Table2
ON Table1.rental_id = Table2.rental_id
WHERE   Table2.return_id IS NULL
AND     DATEDIFF(day, Table1.rental_date, GETDATE()) > @days_rented;
SET NOCOUNT OFF;
END;
GO

--(9) Stored procedure to get the date and time of the last disc rental of the customer identified by @customer_id:

USE Flix2You;
GO
```

```sql

IF OBJECT_ID ( 'Sales.uspWhenCustomerLastRentedDisc', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspWhenCustomerLastRentedDisc;
GO

CREATE PROCEDURE Sales.uspWhenCustomerLastRentedDisc        @customer_id    int,
                                                            @when_rented    datetime    OUTPUT AS
BEGIN


SELECT @when_rented = MAX(transaction_timestamp) FROM Sales.TransactedEvent
WHERE   customer_id = @customer_id
AND format_code IN (SELECT format_code FROM Sales.HighLevelFormat WHERE physical_medium = 'disc')
AND transaction_type_code =  (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'movie rental');

SET NOCOUNT OFF;
END;
GO


--(10) Combine (8) and (9) to query whether the last disc rental of the customer identified by @customer_id is overdue

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetLastRentedDiscIfOverdue', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetLastRentedDiscIfOverdue;
GO

CREATE PROCEDURE Sales.uspGetLastRentedDiscIfOverdue        @this_customer_id   int,
                                                            @days_rented        int AS
BEGIN
DECLARE @last_rental_datetime   datetime;

EXECUTE Sales.uspWhenCustomerLastRentedDisc     @customer_id = @this_customer_id,
                                                @when_rented = @last_rental_datetime OUTPUT;
SELECT  Table1.rental_id,
        Table1.rental_date,
        Table2.return_id,
        Table2.return_date
FROM (
    SELECT  transaction_id          AS rental_id,
            transaction_timestamp   AS rental_date
    FROM Sales.TransactedEvent
    WHERE   customer_id = @this_customer_id
    AND     format_code IN
        (SELECT format_code FROM Sales.HighLevelFormat WHERE physical_medium = 'disc')
    AND     transaction_type_code =
        (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'movie rental')
    AND     transaction_timestamp = @last_rental_datetime /*(
        SELECT MAX(transaction_timestamp) FROM Sales.TransactedEvent
        WHERE   customer_id = 32567 --@customer_id
        AND     format_code IN
            (SELECT format_code FROM Sales.HighLevelFormat WHERE physical_medium = 'disc')
        AND     transaction_type_code =
            (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'movie rental') )*/
    )   AS Table1
LEFT OUTER JOIN (
    SELECT  transaction_id          AS return_id,
            transaction_timestamp   AS return_date,
            previous_transaction_id AS rental_id
    FROM Sales.TransactedEvent
    WHERE   customer_id = @this_customer_id
    AND     format_code IN
        (SELECT format_code FROM Sales.HighLevelFormat WHERE physical_medium = 'disc')
    AND     transaction_type_code =
        (SELECT transaction_type_code FROM Sales.TransactionType WHERE transaction_description = 'disc return')
    )   AS Table2
    ON Table1.rental_id = Table2.rental_id
WHERE   Table2.return_id IS NULL
AND     DATEDIFF(day, Table1.rental_date, GETDATE()) > @days_rented;

SET NOCOUNT OFF;
END;
GO

 /*(11) Create a trigger on Sales.TransactedEvent to promote the integrity of its
 required unary relationship*/

USE Flix2You;
```

```sql
GO

IF OBJECT_ID ('Sales.trgInsertOrUpdateTransactedEvent', 'TR') IS NOT NULL
   DROP TRIGGER Sales.trgInsertOrUpdateTransactedEvent;
GO

CREATE TRIGGER Sales.trgInsertOrUpdateTransactedEvent ON Sales.TransactedEvent
AFTER INSERT, UPDATE
AS

DECLARE    @this_transaction_type_code tinyint,
           @last_transaction_type_code tinyint,
           @previous_transaction_id    bigint,
           @movie_id                   char(26),
           @format_code                char(9),
           @customer_id                int;

DECLARE new_transactions_cursor CURSOR FOR
   SELECT  movie_id,
           format_code,
           customer_id,
           transaction_type_code,
           previous_transaction_id
   FROM inserted;

OPEN new_transactions_cursor;
FETCH NEXT FROM new_transactions_cursor INTO   @movie_id,
                                               @format_code,
                                               @customer_id,
                                               @this_transaction_type_code,
                                               @previous_transaction_id;

WHILE (@@FETCH_STATUS = 0)
BEGIN
   IF @this_transaction_type_code IN  (SELECT transaction_type_code FROM Sales.TransactionType
                                      WHERE transaction_description IN ('movie rental',
                                                                        'movie purchase'))
   BEGIN
       IF @previous_transaction_id IS NOT NULL
       BEGIN
           RAISERROR('A movie rental or purchase is related to no previous transaction.', 15, 1);
           ROLLBACK TRANSACTION;
           RETURN;
       END;
   END;
   ELSE
   BEGIN
       IF @previous_transaction_id IS NULL
       BEGIN
           RAISERROR(
           'A previous transaction must be specified for any transaction other than a rental or purchase.'
           , 15, 1);
           ROLLBACK TRANSACTION;
           RETURN;
       END;
       ELSE
       BEGIN
           SET @last_transaction_type_code =  (SELECT transaction_type_code FROM Sales.TransactedEvent
                                              WHERE transaction_id = @previous_transaction_id);
           IF @last_transaction_type_code NOT IN  (SELECT transaction_type_code FROM Sales.TransactionType
                                                  WHERE transaction_description IN ('movie rental',
                                                                                    'movie purchase'))
           BEGIN
               RAISERROR(
                   'A previous related transaction must be either a movie rental or purchase.'
                   , 15, 1);
               ROLLBACK TRANSACTION;
               RETURN;
           END;
           IF (@movie_id       <>  (SELECT movie_id    FROM Sales.TransactedEvent
                                    WHERE transaction_id = @previous_transaction_id) OR
               @format_code    <>  (SELECT format_code FROM Sales.TransactedEvent
                                    WHERE transaction_id = @previous_transaction_id) OR
               @customer_id    <>  (SELECT customer_id FROM Sales.TransactedEvent
                                    WHERE transaction_id = @previous_transaction_id))
           BEGIN
               RAISERROR(
                   'Critical fields in a new and related previous transaction fail to correspond.'
```

```
                    , 15, 1);
                ROLLBACK TRANSACTION;
                RETURN;
            END
        END;
    END;

    FETCH NEXT FROM new_transactions_cursor  INTO   @movie_id,
                                                    @format_code,
                                                    @customer_id,
                                                    @this_transaction_type_code,
                                                    @previous_transaction_id;
END;

CLOSE new_transactions_cursor;
DEALLOCATE new_transactions_cursor;
GO
```

```sql
/*BusinessTransactions.sql
 *Records sample transactions using the procedural SQL of TransactionProcessing.sql
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015*/


EXECUTE Sales.uspRecordSale @customer_id = 250,
                            @movie_id = '2411-6FF0-31E2-D49D-870C-7',
                            @format_code = 'HD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 5;
GO

EXECUTE Sales.uspRecordSale @customer_id = 298,
                            @movie_id = '9824-4212-BADD-A847-068B-1',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 11;
GO

EXECUTE Sales.uspRecordSale @customer_id = 425,
                            @movie_id = '4EB2-DFF8-6B78-660F-C61B-D',
                            @format_code = 'STD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 15;
GO

EXECUTE Sales.uspRecordSale @customer_id = 483,
                            @movie_id = '5868-409E-7BFB-536A-6067-E',
                            @format_code = 'BR-1',
                            @transaction_type_code = 2;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 19;
GO

EXECUTE Sales.uspRecordSale @customer_id = 266,
                            @movie_id = 'B193-2EDD-9478-C964-FEF9-K',
                            @format_code = 'HD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 23;
GO

EXECUTE Sales.uspRecordSale @customer_id = 275,
                            @movie_id = '36E3-649C-57F2-A251-72B1-E',
                            @format_code = 'HD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 27;
GO

EXECUTE Sales.uspRecordSale @customer_id = 275,
                            @movie_id = '843E-C238-7F61-DDFC-F427-9',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 258,
```

```sql
                            @movie_id = '0A89-4017-2ED5-D199-0491-H',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 302,
                            @movie_id = '2331-775F-399A-937C-54C2-0',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 344,
                            @movie_id = '843E-C238-7F61-DDFC-F427-9',
                            @format_code = 'BR-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 437,
                            @movie_id = '36E3-649C-57F2-A251-72B1-E',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 484,
                            @movie_id = '4EB2-DFF8-6B78-660F-C61B-D',
                            @format_code = 'BR-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 322,
                            @movie_id = '2331-775F-399A-937C-54C2-0',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 335,
                            @movie_id = '4EB2-DFF8-6B78-660F-C61B-D',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordSale @customer_id = 416,
                            @movie_id = '5868-409E-7BFB-536A-6067-E',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 2;



EXECUTE Sales.uspRecordSale @customer_id = 419,
                            @movie_id = '9824-4212-BADD-A847-068B-1',
                            @format_code = 'DVD-1',
                            @transaction_type_code = 1;
GO

EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 31;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 33;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 35;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 37;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 39;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 41;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 43;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 45;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 47;
EXECUTE Sales.uspRecordTransfer @previous_transaction_id = 49;
GO
```

```sql
/*WebCatalogBrowsing.sql
 *IST 210 Section 1 Team 2
 *Last modified August 2, 2015*/


/*Sample queries given below illustrate the retrieval of data from the Flix2You operational database
to populate fields displayed on the XHTML pages of its web application.

The programming language(s) of the development platform on which Flix2You's web application is built
will themselves contain API that-- in a  process similar to but easier than using the API of the older
Embedded SQL standard illustrated in our textbook-- allow the queries used below to generate cursors
on the SQL Server to be formulated instead as text within the programming language itself, and incorporated
in native calls that return scrollable result sets directly accessible to the application. Thus the
cursors generated by the illustrative code below need not themselves be generated and manipulated by the
code of the web application.

As a matter of fact both the Microsoft .NET and JavaEE modern web application server platforms allow
the developer to directly model entities with their attributes and relationships as object classes in
C# or Java code itself, after which the interactions with a database connection designated as a datasource
within the application that are needed to effect the select queries, insertions, updates and deletions
associated with instances of these classes are handled more or less transparently by a manager object. In
an ASP.NET web application server this might be accomplished by the Entity Framework Code First framework,
while a web application server like Glassfish or JBoss that works with JavaEE might instead make
use of the Java Persistence API for the same purpose. In any event the logic of the underlying SQL code
will nevertheless still be operative.*/

/*(1) stored procedure to return a cursor for the high-level encodings of a specified movie that are available
for rental or purchase from Flix2You, and the prices of each.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetAvailableFormsOfMovieCursor', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetAvailableFormsOfMovieCursor;
GO

CREATE PROCEDURE Sales.uspGetAvailableFormsOfMovieCursor    @FormsAndPricesCursor   CURSOR VARYING OUTPUT,
                                                            @movie_id               char(26)
                                    AS
BEGIN
SET NOCOUNT ON;

--check the format of the supplied parameter:
IF @movie_id NOT LIKE   '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-Z]'
BEGIN
    PRINT   N'The movie id supplied does not have the format of an EIDR movie suffix.';
    RETURN;
END;

--define the cursor:
SET @FormsAndPricesCursor = CURSOR SCROLL STATIC FOR
    SELECT  movie_title,
            release_date,
            format_name,
            rental_price,
            purchase_price
    FROM Sales.ViewableEncoding AS e
    JOIN Sales.HighLevelFormat  AS f
        ON e.format_code =  f.format_code
    JOIN Sales.Movie            AS m
        ON e.movie_id =     m.movie_id
    WHERE e.movie_id = @movie_id AND is_available = 1
    ORDER BY e.format_code;
```

```sql

--populate the cursor:
OPEN @FormsAndPricesCursor;

SET NOCOUNT OFF;
END;
GO

/*(2) stored procedure to return a cursor for the movies in the database corresponding to the search text
supplied by a user for a movie title.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetMovieCursorFromTitleText', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetMovieCursorFromTitleText;
GO

CREATE PROCEDURE Sales.uspGetMovieCursorFromTitleText   @MoviesFromTitleCursor  CURSOR VARYING OUTPUT,
                                                        @title_search_text      nvarchar(50)
                                    AS
BEGIN
SET NOCOUNT ON;

--The following could be improved, but at least the thought is there:
IF DATALENGTH(REPLACE(@title_search_text, ' ', '')) < 3
BEGIN
    PRINT   N'The title search text must contain at least 3 non-blank characters';
    RETURN;
END;

--define the cursor:
SET @MoviesFromTitleCursor = CURSOR SCROLL STATIC FOR
    SELECT  movie_id,
            movie_title,
            studio_id,
            release_date,
            country_code,
            director_id,
            movie_length,
            genre_code,
            movie_description,
            movie_description_uri,
            movie_poster_uri,
            imdb_id
    FROM Sales.Movie
    WHERE movie_title LIKE ('%' + @title_search_text + '%')
    ORDER BY movie_title ASC, release_date DESC;

--populate the cursor:
OPEN @MoviesFromTitleCursor;

SET NOCOUNT OFF;
END;
GO

/*(3) stored procedure to return a cursor for the movies in the database that were released within
the specified last @number_months:*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetRecentReleasesMovieCursor', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetRecentReleasesMovieCursor;
GO

CREATE PROCEDURE Sales.uspGetRecentReleasesMovieCursor  @RecentReleasesCursor   CURSOR VARYING OUTPUT,
                                                        @number_of_months       tinyint
                                    AS
```

```sql
BEGIN
SET NOCOUNT ON;

--Limit the size of the data set returned:
IF @number_of_months > 36
BEGIN
    PRINT   N'This procedure cannot return movie released more than three years ago';
    RETURN;
END;

--define the cursor:
SET @RecentReleasesCursor = CURSOR SCROLL STATIC FOR
    SELECT  movie_id,
            movie_title,
            studio_id,
            release_date,
            country_code,
            director_id,
            movie_length,
            genre_code,
            movie_description,
            movie_description_uri,
            movie_poster_uri,
            imdb_id
    FROM Sales.Movie
    WHERE DATEDIFF(month, release_date, GETDATE()) <= @number_of_months
    ORDER BY release_date DESC, movie_title ASC;

--populate the cursor:
OPEN @RecentReleasesCursor;

SET NOCOUNT OFF;
END;
GO

/*(4) stored procedure to return a cursor for the movies in the database that are instances of the
specified genre.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetMovieCursorFromGenreCode', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetMovieCursorFromGenreCode;
GO

CREATE PROCEDURE Sales.uspGetMovieCursorFromGenreCode   @MoviesFromGenreCode    CURSOR VARYING OUTPUT,
                                                        @genre_code             tinyint
                                    AS
BEGIN
SET NOCOUNT ON;

--check for valid genre code:
IF @genre_code  NOT IN (SELECT genre_code FROM Sales.Genre)
BEGIN
    PRINT   N'The supplied genre code is invalid';
    RETURN;
END;

--define the cursor:
SET @MoviesFromGenreCode = CURSOR SCROLL STATIC FOR
    SELECT  movie_id,
            movie_title,
            studio_id,
            release_date,
            country_code,
            director_id,
            movie_length,
            genre_code,
```

```sql
            movie_description,
            movie_description_uri,
            movie_poster_uri,
            imdb_id
    FROM Sales.Movie
    WHERE genre_code = @genre_code
    ORDER BY movie_title ASC, release_date DESC;

--populate the cursor:
OPEN @MoviesFromGenreCode;

SET NOCOUNT OFF;
END;
GO


/*(5) stored procedure to return a cursor for the movies in the database in which an actor/actress
with the supplied last name appears as a member of the cast.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetMovieCursorFromActorName', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetMovieCursorFromActorName;
GO

CREATE PROCEDURE Sales.uspGetMovieCursorFromActorName   @MoviesFromActorCursor  CURSOR VARYING OUTPUT,
                                                        @actor_name_last        nvarchar(35)
                                    AS
BEGIN
SET NOCOUNT ON;

--The following could be improved, but at least the thought is there:
IF DATALENGTH(REPLACE(@actor_name_last, ' ', '')) < 2
BEGIN
    PRINT   N'The actor last name text must contain at least 2 non-blank characters';
    RETURN;
END;

--define the cursor:
SET @MoviesFromActorCursor = CURSOR SCROLL STATIC FOR
    SELECT  m.movie_id,
            movie_title,
            studio_id,
            release_date,
            country_code,
            director_id,
            movie_length,
            genre_code,
            movie_description,
            movie_description_uri,
            movie_poster_uri,
            imdb_id,
            a.actor_id,
            actor_gender,
            actor_name_first,
            actor_name_last,
            other_actor_details,
            as_character,
            is_lead
    FROM Sales.Movie        AS m
    JOIN Sales.MovieRole    AS r
        ON m.movie_id =     r.movie_id
    JOIN Sales.Actor        AS a
        ON  r.actor_id =    a.actor_id
    WHERE actor_name_last = @actor_name_last
    ORDER BY release_date DESC, movie_title ASC;

```

```sql
--populate the cursor:
OPEN @MoviesFromActorCursor;

SET NOCOUNT OFF;
END;
GO

/*(6) stored procedure to return a cursor for the actors/actresses
in the cast of a specified movie.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetMovieCastCursor', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetMovieCastCursor;
GO

CREATE PROCEDURE Sales.uspGetMovieCastCursor    @MovieCastCursor    CURSOR VARYING OUTPUT,
                                                @movie_id           char(26)
                                      AS
BEGIN
SET NOCOUNT ON;

--check the format of the supplied parameter:
IF @movie_id NOT LIKE   '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-F][0-9A-F][0-9A-F][0-9A-F]-' +
                        '[0-9A-Z]'
BEGIN
    PRINT   N'The movie id supplied does not have the format of an EIDR movie suffix.';
    RETURN;
END;

--define the cursor:
SET @MovieCastCursor = CURSOR SCROLL STATIC FOR
    SELECT  a.actor_id,
            actor_gender,
            actor_name_first,
            actor_name_last,
            other_actor_details,
            as_character,
            is_lead
    FROM Sales.MovieRole    AS r
    JOIN Sales.Actor        AS a
        ON  r.actor_id =    a.actor_id
    WHERE movie_id = @movie_id
  --ORDER BY [should add an attribute to MovieRole to reflect studio's billing rank of actor's role]

--populate the cursor:
OPEN @MovieCastCursor;

SET NOCOUNT OFF;
END;
GO

/*(7) stored procedure to return a cursor for the movies in the database directed
by the director with the supplied last name.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetMovieCursorFromDirectorName', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetMovieCursorFromDirectorName;
GO

CREATE PROCEDURE Sales.uspGetMovieCursorFromDirectorName    @MoviesFromDirectorCursor   CURSOR VARYING OUTPUT,
                                                            @director name last         nvarchar(35)
```

```sql
                                    AS
BEGIN
SET NOCOUNT ON;

--The following could be improved, but at least the thought is there:
IF DATALENGTH(REPLACE(@director_name_last, ' ', '')) < 2
BEGIN
    PRINT   N'The director last name text must contain at least 2 non-blank characters';
    RETURN;
END;

--define the cursor:
SET @MoviesFromDirectorCursor = CURSOR SCROLL STATIC FOR
    SELECT  m.movie_id,
            movie_title,
            studio_id,
            release_date,
            country_code,
            movie_length,
            genre_code,
            movie_description,
            movie_description_uri,
            movie_poster_uri,
            imdb_id,
            d.director_id,
            director_name_first,
            director_name_last
    FROM Sales.Movie        AS m
    JOIN Sales.Director     AS d
        ON m.director_id =  d.director_id
    WHERE director_name_last = @director_name_last
    ORDER BY release_date DESC, movie_title ASC;

--populate the cursor:
OPEN @MoviesFromDirectorCursor;

SET NOCOUNT OFF;
END;
GO

/*(8) stored procedure to return a cursor containing the movies placed by a specified customer in his
queue, with associated movie, price and availability data for each.*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetCustomerQueueCursor', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetCustomerQueueCursor;
GO

CREATE PROCEDURE Sales.uspGetCustomerQueueCursor    @MovieQueueCursor   CURSOR VARYING OUTPUT,
                                                    @customer_id        int
                                    AS
BEGIN
SET NOCOUNT ON;


--define the cursor:
SET @MovieQueueCursor = CURSOR SCROLL STATIC FOR
    SELECT  q.movie_id,
            rank_in_queue,
            date_queued,
            movie_title,
            studio_id,
            release_date,
            country_code,
            director_id,
            movie_length,
```

```sql
            genre_code,
            movie_description,
            movie_description_uri,
            movie_poster_uri,
            imdb_id,
            format_name,
            rental_price,
            purchase_price
    FROM Sales.QueuedMovie      AS q
    JOIN Sales.ViewableEncoding AS e
        ON  q.movie_id =    e.movie_id
        AND q.format_code = e.format_code
    JOIN Sales.HighLevelFormat  AS f
        ON  e.format_code = f.format_code
    JOIN Sales.Movie            AS m
        ON  e.movie_id =        m.movie_id
    WHERE customer_id = @customer_id
    ORDER BY rank_in_queue;

--populate the cursor:
OPEN @MovieQueueCursor;

SET NOCOUNT OFF;
END;
GO

/*(9) stored procedure to return in a variable the average user rating of a movie in a designated parameter*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspGetAverageMovieRating', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspGetAverageMovieRating;
GO

CREATE PROCEDURE Sales.uspGetAverageMovieRating @movie_id       char(56),
                                                @average_rating numeric(2,1) OUTPUT AS
BEGIN
SET NOCOUNT ON;

SET @average_rating = (
    SELECT AVG(CAST(number_of_stars AS numeric(2,1)))
    FROM Sales.MovieRating
    WHERE movie_id = @movie_id);

SET NOCOUNT OFF;
END;
GO
```

```sql
/*BusinessIntelligence.sql
 *Example procedural SQL code to supply business intelligence to Flix2You
 *IST 210 Section 1 Team 2
 *Last modified August 3, 2015*/


/*(1) stored procedure to show ranked total revenue production by movie during a designated period of time.

NOTE: This procedure doesn't distinguish between sales or rentals, or between discs and files; to create
these it is only necessary to add the necessary parameters to the signature of the stored procedure, and
corresponding conditions to the WHERE clause of the SELECT commands.

This procedure might create a misleading aggregate if two movies with the same title are both producing
revenue within the interval and their combined revenue "makes the cut" (not a remote  possibility: I think
of "The Thomas Crown Affair", for example.). In designing a MOLAP cube in Analysis Services it is easy
to avoid such problems by separately defining a key, value and name field for the dimension attribute.
The only satisfactory way that I can think of to deal with the problem in the procedure below is to
replace the movie_title GROUP BY attribute with another that is a (textual) function of this
attribute and the release_date or studio_name attributes.*/

USE Flix2YouDW;
GO

IF OBJECT_ID ( 'Sales.uspQueryTopGrossingMovies', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspQueryTopGrossingMovies;
GO

CREATE PROCEDURE Sales.uspQueryTopGrossingMovies    @datetime1 datetime,    --beginning of range
                                                    @datetime2 datetime,    --end of range
                                                    @how_many  smallint     --number of movies to show
                                        AS
BEGIN
SET NOCOUNT ON;

DECLARE @total_revenue money = (
    SELECT SUM(transaction_amount) FROM Sales.FactSalesTransaction
    WHERE transaction_timestamp BETWEEN @datetime1 AND @datetime2 );

SELECT TOP(@how_many)   LEFT(movie_title, 35)                   AS 'Movie',
                        LEFT(FORMAT(SUM(transaction_amount),
                            'C', 'en-US'), 15)                  AS 'Revenue',
                        LEFT(FORMAT(SUM(transaction_amount)/@total_revenue, 'P1', 'en-US'), 15)
                                                                AS 'Per Cent of Total'
FROM Sales.FactSalesTransaction AS t
JOIN Sales.DimMovie             AS m
    ON t.movie_id = m.movie_id
WHERE transaction_timestamp BETWEEN @datetime1 AND @datetime2
GROUP BY movie_title
ORDER BY SUM(transaction_amount) DESC;

SET NOCOUNT OFF;
END;
GO


/*(2) stored procedure to show most popular movie genres in a given customer age bracket. NOTE: To ensure
that totals in queries like this will add up to no more than 100% the database has been set up with a 1:M
relationship between Genre and Movie (and the genre descriptions have been chosen to minimize potential
overlap). But in many cases it is still difficult to assign a given movie to a single definite genre.*/

USE Flix2YouDW;
GO

IF OBJECT_ID ( 'Sales.uspRankGenrePopularityInAgeGroup', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspRankGenrePopularityInAgeGroup;
```

```sql
GO

CREATE PROCEDURE Sales.uspRankGenrePopularityInAgeGroup @datetime1  datetime,   --beginning of date range
                                                        @datetime2  datetime,   --end of date range
                                                        @age1       tinyint,    --beginning of age range
                                                        @age2       tinyint     --end of age range
                                        AS
BEGIN
SET NOCOUNT ON;

DECLARE @total_orders int = (
    SELECT COUNT(transaction_id) FROM Sales.FactSalesTransaction
    WHERE   transaction_timestamp BETWEEN @datetime1 AND @datetime2 );

SELECT  LEFT(genre_description, 20) AS  Genre,
        LEFT(movie_title, 35)       AS  Movie,
        COUNT(transaction_id)       AS  Orders,
        LEFT(FORMAT(CAST(COUNT(transaction_id) AS decimal)/CAST(@total_orders AS decimal), 'P1', 'en-US'), 18)
                                    AS 'Per Cent of Total'
FROM Sales.FactSalesTransaction AS s
JOIN Sales.DimMovie             AS m
    ON s.movie_id = m.movie_id
JOIN Sales.DimGenre             AS g
    ON m.genre_code = g.genre_code
JOIN Sales.DimCustomer          AS c
    ON s.customer_id = c.customer_id
WHERE DATEDIFF(year, c.customer_birth_date, GETDATE())
        BETWEEN @age1 AND @age2
  AND transaction_timestamp BETWEEN @datetime1 AND @datetime2
GROUP BY ROLLUP(genre_description, movie_title)
ORDER BY genre_description, movie_title;

SET NOCOUNT OFF;
END;
GO

/*(3) stored procedure to show customer movie rentals and purchases by day of the week and hour
of the day. NOTE: a procedure to usefully accomplish the true purpose of this one would
probably need to be more complex, and adjust display of the transaction timestamp associated
in the database with each sale to correlate it with the customer's home time zone. In the
individual case this adjustment would not be too difficult, but in the calculation of aggregates
as in the procedure below this could prove to be very complex indeed.*/

USE Flix2YouDW;
GO

IF OBJECT_ID ( 'Sales.uspShowSalesByHourAndWeekday', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspShowSalesByHourAndWeekday;
GO

CREATE PROCEDURE Sales.uspShowSalesByHourAndWeekday @datetime1  datetime,   --beginning of date range
                                                    @datetime2  datetime    --end of date range
                                        AS
BEGIN
SET NOCOUNT ON;

DECLARE @total_orders int = (
    SELECT COUNT(transaction_id) FROM Sales.FactSalesTransaction
    WHERE   transaction_timestamp BETWEEN @datetime1 AND @datetime2 );

SELECT  weekday_name                AS 'Weekday',
        hour_of_day                 AS 'Hour (1-24)',
        COUNT(transaction_id)       AS  Orders,
        FORMAT(CAST(COUNT(transaction_id) AS decimal)/CAST(@total_orders AS decimal), 'P1', 'en-US')
                                    AS 'Per Cent of Total'
```

```sql
FROM Sales.FactSalesTransaction AS s
JOIN Sales.DimDateTime          AS d
    ON s.transaction_timestamp = d.transaction_timestamp
WHERE s.transaction_timestamp BETWEEN @datetime1 AND @datetime2
GROUP BY ROLLUP(weekday_name, hour_of_day)
ORDER BY weekday_name, hour_of_day;

SET NOCOUNT OFF;
END;
GO

/*(4) Stored procedure to show number of customers by geographic region and month of original membership,
after a specified specific beginning date of interest:*/

USE Flix2YouDW;
GO

IF OBJECT_ID ( 'Sales.uspShowCustomerCountByMembershipDate', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspShowCustomerCountByMembershipDate;
GO

CREATE PROCEDURE Sales.uspShowCustomerCountByMembershipDate @after_date datetime
                                AS
BEGIN
SET NOCOUNT ON;

SELECT  LEFT(DATENAME(yyyy, membership_date) + '-' +
        FORMAT(DATEPART(mm, membership_date), '00'), 7)
                                    AS 'Month Joined',
        region_name                 AS 'Geographic Region',
        COUNT(customer_id)          AS 'Number Added'
FROM DimCustomer    AS c
JOIN DimGeography   AS g
    ON  c.country_code = g.country_code
    AND c.customer_postal_code = g.customer_postal_code
    AND c.customer_city = g.city
WHERE membership_date >= @after_date
GROUP BY CUBE(DATENAME(yyyy, membership_date) + '-' +
              FORMAT(DATEPART(mm, membership_date), '00'), region_name)
ORDER BY DATENAME(yyyy, membership_date) + '-' +
              FORMAT(DATEPART(mm, membership_date), '00'), region_name

SET NOCOUNT OFF;
END;
GO

/*(5) create view to show the cumulative net cash flow realized upon each ViewableEncoding.
NOTE: As the problem document supplies no detailed information about Flix2You's business processes
related to the acquisition and delivery of its product no model proving for the accretion of costs
on a transaction-by-transaction basis has been created in the Sales schema; instead there has only
been created an aggregate_cost attribute in the ViewableEncoding entity class, in which it is envisaged
that Flix2You will accumulate the dollar total of the payments made to Studios for distribution rights
to each movie master, the costs of encoding filesets for each master in various combinations of codec,
bit rate, etc., the in-house cost of burning its stock of DVD and Blu-Ray discs, and so on.

The unit cost of an individual disc is probably what consumer buyers of retail Blu-Ray and DVD movie
discs are most likely to first think of as the principal non-revenue determinant of gross profit, but
an enterprise like Flix2You can use old commodity Pentium desktops to burn encodings onto blanks
right at the mail distribution outlets for a direct marginal unit cost of cents per copy. Insofar as
there may be a variable direct unit cost of a rental that doesn't result from amortization of the
cost of acquiring the intellectual property rights to a master and processing the original set of
reproducible encodings, it is more likely to be per-viewing royalty that is payable to a studio.

Thus the view created below allows only for the calculation of a cumulative cash flow. The result
is called cash flow rather than profit because the cumulative investment in each ViewableEncoding is
```

```sql
not being written off on a pro-rata basis against revenues, based upon an some estimate of the total
number of viewings that will be eventually be purchased by customers during the economic lifetime of
the investment. Thus the cumulative cash flow will always begin as a negative value.*/

USE Flix2You;
GO

--Drop the vwCumulativeProfitPerViewableEncoding if it exists.
IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.VIEWS
                WHERE TABLE_SCHEMA = 'Sales'
                AND  TABLE_NAME = 'vwCumulativeProfitPerViewableEncoding'))
BEGIN
   DROP VIEW Sales.vwCumulativeProfitPerViewableEncoding;
END;
GO

CREATE VIEW Sales.vwCumulativeProfitPerViewableEncoding AS

SELECT  p.movie_id,
        p.format_code,
        SUM(transaction_amount)     AS revenue,
        AVG(aggregate_cost)         AS cost,
        SUM(transaction_amount) -
        AVG(aggregate_cost)         AS net_cash_flow
FROM    Sales.TransactedEvent       AS t
JOIN    Sales.ViewableEncoding      AS p
    ON  t.movie_id = p.movie_id AND t.format_code = p.format_code
WHERE
    transaction_type_code IN (
        SELECT transaction_type_code FROM Sales.TransactionType
        WHERE
            transaction_description = 'movie purchase' OR
            transaction_description = 'movie rental')
GROUP BY p.format_code, p.movie_id;
GO

/*(6) procedure to calculate cumulative cash flow by HighLevelFormat type from the
view just created above*/

USE Flix2You;
GO

IF OBJECT_ID ( 'Sales.uspQueryCashFlowByHighLevelFormat', 'P' ) IS NOT NULL
    DROP PROCEDURE Sales.uspQueryCashFlowByHighLevelFormat;
GO

CREATE PROCEDURE Sales.uspQueryCashFlowByHighLevelFormat AS
BEGIN
SET NOCOUNT ON;

SELECT  format_name         AS 'Product Type',
        SUM(revenue)        AS 'Total Revenue',
        SUM(cost)           AS 'Total Cost',
        SUM(revenue) -
        SUM(cost)           AS 'Net Cash Flow'
FROM Sales.vwCumulativeProfitPerViewableEncoding    AS v
JOIN Sales.HighLevelFormat                          AS f
    ON v.format_code = f.format_code
GROUP BY f.format_name;

SET NOCOUNT OFF;
END;
GO

```

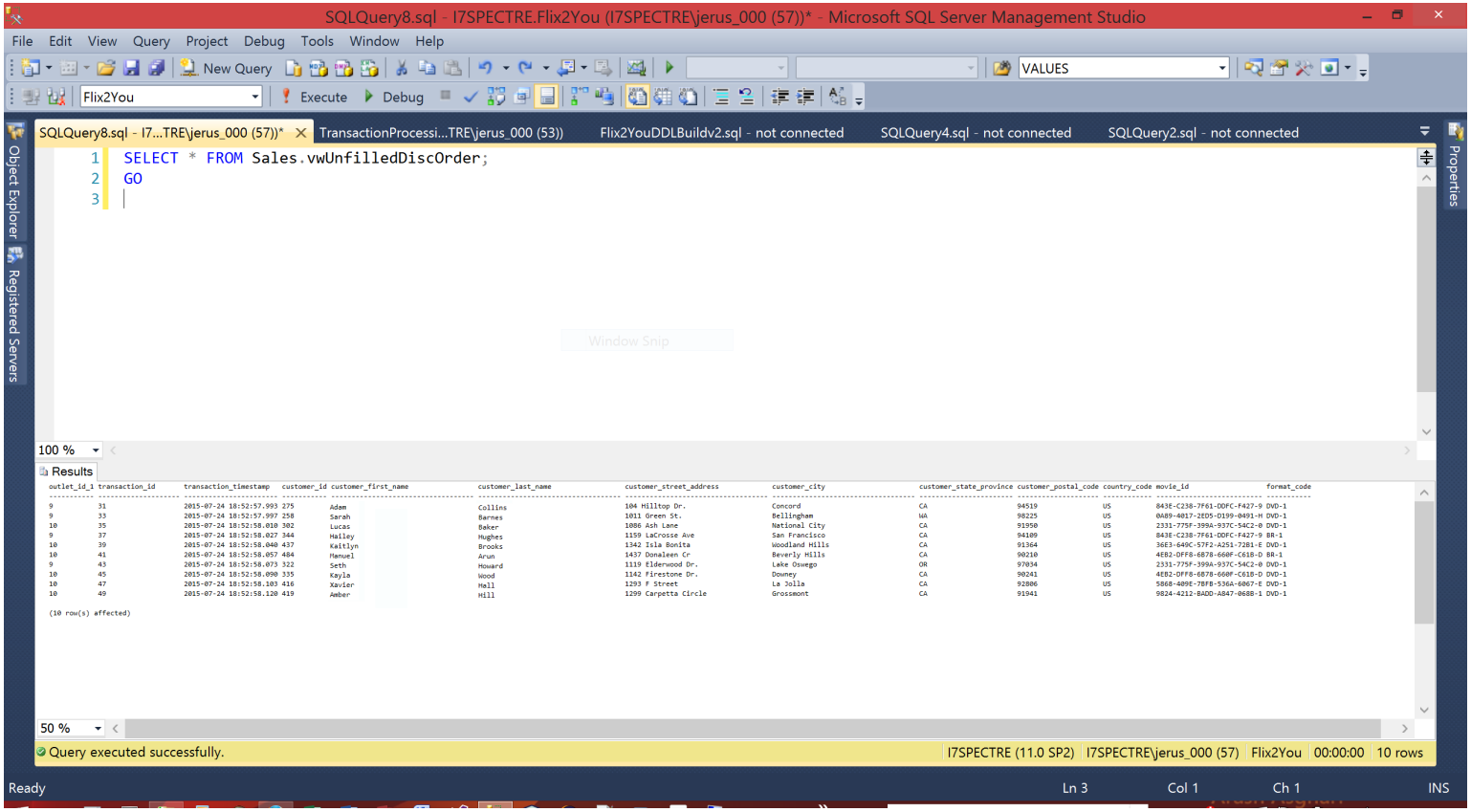
TransactionProcessingOutput.docx
IST 210 Section 1 Team 2
Output produced with use of the code created in TransactionProcessing.sql:

(1) and (2). The Sales.TransactedEvent table resulting from multiple execution of the Sales.uspRecordSale and Sales.uspRecordTransfer stored procedures (see BusinessTransactions.sql):

	transaction_id	movie_id	format_code	customer_id	transaction_timestamp	transaction_type_co...	transaction_amount	previous_transaction...	transaction_comment
1	5	2411-6FF0-31E2-D49D-870C-7	HD-1	250	2015-07-24 14:27:14.417	1	3.99	NULL	NULL
2	6	2411-6FF0-31E2-D49D-870C-7	HD-1	250	2015-07-24 14:27:14.417	3	3.99	5	NULL
3	7	2411-6FF0-31E2-D49D-870C-7	HD-1	250	2015-07-24 14:55:45.270	6	3.99	5	NULL
4	8	2411-6FF0-31E2-D49D-870C-7	HD-1	250	2015-07-24 14:55:45.270	4	3.99	5	NULL
5	11	9824-4212-BADD-A847-068B-1	DVD-1	298	2015-07-24 15:05:36.573	1	3.95	NULL	NULL
6	12	9824-4212-BADD-A847-068B-1	DVD-1	298	2015-07-24 15:05:36.573	3	3.95	11	NULL
7	13	9824-4212-BADD-A847-068B-1	DVD-1	298	2015-07-24 15:07:57.770	7	3.95	11	NULL
8	14	9824-4212-BADD-A847-068B-1	DVD-1	298	2015-07-24 15:07:57.770	4	3.95	11	NULL
9	15	4EB2-DFF8-6B78-660F-C61B-D	STD-1	425	2015-07-24 15:10:43.507	1	2.99	NULL	NULL
10	16	4EB2-DFF8-6B78-660F-C61B-D	STD-1	425	2015-07-24 15:10:43.507	3	2.99	15	NULL
11	17	4EB2-DFF8-6B78-660F-C61B-D	STD-1	425	2015-07-24 15:13:14.847	6	2.99	15	NULL
12	18	4EB2-DFF8-6B78-660F-C61B-D	STD-1	425	2015-07-24 15:13:14.847	4	2.99	15	NULL
13	19	5868-409E-7BFB-536A-6067-E	BR-1	483	2015-07-24 15:16:31.277	2	14.95	NULL	NULL
14	20	5868-409E-7BFB-536A-6067-E	BR-1	483	2015-07-24 15:16:31.277	3	14.95	19	NULL
15	21	5868-409E-7BFB-536A-6067-E	BR-1	483	2015-07-24 15:18:20.727	7	14.95	19	NULL
16	22	5868-409E-7BFB-536A-6067-E	BR-1	483	2015-07-24 15:18:20.727	4	14.95	19	NULL
17	23	B193-2EDD-9478-C964-FEF9-K	HD-1	266	2015-07-24 15:21:05.297	1	3.99	NULL	NULL
18	24	B193-2EDD-9478-C964-FEF9-K	HD-1	266	2015-07-24 15:21:05.297	3	3.99	23	NULL
19	25	B193-2EDD-9478-C964-FEF9-K	HD-1	266	2015-07-24 15:22:53.280	6	3.99	23	NULL
20	26	B193-2EDD-9478-C964-FEF9-K	HD-1	266	2015-07-24 15:22:53.280	4	3.99	23	NULL
21	27	36E3-649C-57F2-A251-72B1-E	HD-1	275	2015-07-24 15:24:57.493	1	3.99	NULL	NULL
22	28	36E3-649C-57F2-A251-72B1-E	HD-1	275	2015-07-24 15:24:57.493	3	3.99	27	NULL
23	29	36E3-649C-57F2-A251-72B1-E	HD-1	275	2015-07-24 15:26:08.073	6	3.99	27	NULL
24	30	36E3-649C-57F2-A251-72B1-E	HD-1	275	2015-07-24 15:26:08.073	4	3.99	27	NULL
25	31	843E-C238-7F61-DDFC-F427-9	DVD-1	275	2015-07-24 18:52:57.993	1	3.95	NULL	NULL
26	32	843E-C238-7F61-DDFC-F427-9	DVD-1	275	2015-07-24 18:52:57.993	3	3.95	31	NULL
27	33	0A89-4017-2ED5-D199-0491-H	DVD-1	258	2015-07-24 18:52:57.997	1	3.95	NULL	NULL
28	34	0A89-4017-2ED5-D199-0491-H	DVD-1	258	2015-07-24 18:52:57.997	3	3.95	33	NULL
29	35	2331-775F-399A-937C-54C2-0	DVD-1	302	2015-07-24 18:52:58.010	1	3.95	NULL	NULL
30	36	2331-775F-399A-937C-54C2-0	DVD-1	302	2015-07-24 18:52:58.010	3	3.95	35	NULL

	transaction_id	movie_id	format_code	customer_id	transaction_timestamp	transaction_type_co...	transaction_amount	previous_transaction...	transaction_comment
31	37	843E-C238-7F61-DDFC-F427-9	BR-1	344	2015-07-24 18:52:58.027	1	5.95	NULL	NULL
32	38	843E-C238-7F61-DDFC-F427-9	BR-1	344	2015-07-24 18:52:58.027	3	5.95	37	NULL
33	39	36E3-649C-57F2-A251-72B1-E	DVD-1	437	2015-07-24 18:52:58.040	1	3.95	NULL	NULL
34	40	36E3-649C-57F2-A251-72B1-E	DVD-1	437	2015-07-24 18:52:58.040	3	3.95	39	NULL
35	41	4EB2-DFF8-6B78-660F-C61B-D	BR-1	484	2015-07-24 18:52:58.057	1	5.95	NULL	NULL
36	42	4EB2-DFF8-6B78-660F-C61B-D	BR-1	484	2015-07-24 18:52:58.057	3	5.95	41	NULL
37	43	2331-775F-399A-937C-54C2-0	DVD-1	322	2015-07-24 18:52:58.073	1	3.95	NULL	NULL
38	44	2331-775F-399A-937C-54C2-0	DVD-1	322	2015-07-24 18:52:58.073	3	3.95	43	NULL
39	45	4EB2-DFF8-6B78-660F-C61B-D	DVD-1	335	2015-07-24 18:52:58.090	1	3.95	NULL	NULL
40	46	4EB2-DFF8-6B78-660F-C61B-D	DVD-1	335	2015-07-24 18:52:58.090	3	3.95	45	NULL
41	47	5868-409E-7BFB-536A-6067-E	DVD-1	416	2015-07-24 18:52:58.103	2	11.95	NULL	NULL
42	48	5868-409E-7BFB-536A-6067-E	DVD-1	416	2015-07-24 18:52:58.103	3	11.95	47	NULL
43	49	9824-4212-BADD-A847-068B-1	DVD-1	419	2015-07-24 18:52:58.120	1	3.95	NULL	NULL
44	50	9824-4212-BADD-A847-068B-1	DVD-1	419	2015-07-24 18:52:58.120	3	3.95	49	NULL
45	51	843E-C238-7F61-DDFC-F427-9	DVD-1	275	2015-07-25 10:39:11.390	7	3.95	31	NULL
46	52	843E-C238-7F61-DDFC-F427-9	DVD-1	275	2015-07-25 10:39:11.390	4	3.95	31	NULL
47	53	0A89-4017-2ED5-D199-0491-H	DVD-1	258	2015-07-25 10:39:11.393	7	3.95	33	NULL
48	54	0A89-4017-2ED5-D199-0491-H	DVD-1	258	2015-07-25 10:39:11.393	4	3.95	33	NULL
49	55	2331-775F-399A-937C-54C2-0	DVD-1	302	2015-07-25 10:39:11.393	7	3.95	35	NULL
50	56	2331-775F-399A-937C-54C2-0	DVD-1	302	2015-07-25 10:39:11.393	4	3.95	35	NULL
51	57	843E-C238-7F61-DDFC-F427-9	BR-1	344	2015-07-25 10:39:11.393	7	5.95	37	NULL
52	58	843E-C238-7F61-DDFC-F427-9	BR-1	344	2015-07-25 10:39:11.393	4	5.95	37	NULL
53	59	36E3-649C-57F2-A251-72B1-E	DVD-1	437	2015-07-25 10:39:11.397	7	3.95	39	NULL
54	60	36E3-649C-57F2-A251-72B1-E	DVD-1	437	2015-07-25 10:39:11.397	4	3.95	39	NULL
55	61	4EB2-DFF8-6B78-660F-C61B-D	BR-1	484	2015-07-25 10:39:11.397	7	5.95	41	NULL
56	62	4EB2-DFF8-6B78-660F-C61B-D	BR-1	484	2015-07-25 10:39:11.397	4	5.95	41	NULL
57	63	2331-775F-399A-937C-54C2-0	DVD-1	322	2015-07-25 10:39:11.397	7	3.95	43	NULL
58	64	2331-775F-399A-937C-54C2-0	DVD-1	322	2015-07-25 10:39:11.397	4	3.95	43	NULL
59	65	4EB2-DFF8-6B78-660F-C61B-D	DVD-1	335	2015-07-25 10:39:11.397	7	3.95	45	NULL
60	66	4EB2-DFF8-6B78-660F-C61B-D	DVD-1	335	2015-07-25 10:39:11.397	4	3.95	45	NULL
61	67	5868-409E-7BFB-536A-6067-E	DVD-1	416	2015-07-25 10:39:11.397	7	11.95	47	NULL
62	68	5868-409E-7BFB-536A-6067-E	DVD-1	416	2015-07-25 10:39:11.397	4	11.95	47	NULL
63	69	9824-4212-BADD-A847-068B-1	DVD-1	419	2015-07-25 10:39:11.397	7	3.95	49	NULL
64	70	9824-4212-BADD-A847-068B-1	DVD-1	419	2015-07-25 10:39:11.397	4	3.95	49	NULL

(3) Query and result set to SELECT from the Sales.vwUnfilledDiscOrders view (executed 07-24-2015, before any recording any transaction with an ID number greater than 50):



(4) and (5). Hardcopy printed list of pending unshipped disc orders at the mail distribution outlet with the id 10 (Santa Barbara, CA), generated from the view depicted above (but after execution of the additional transactions 51-58) with the use of the procedures Sales.uspGetPendingDiscOrdersCursor and Sales.uspPrintPendingDiscOrders:

```
EXECUTE Sales.uspPrintUnshippedDiscOrders   @of_outlet_id  = 10;
GO
```

Unshipped Disc Orders for Mail Distribution Outlet #10

Transaction ID: 35
Transaction Timestamp: Jul 24 2015 6:52PM
Customer ID: 302
Customer Name and Address:
Lucas Baker
1086 Ash Lane
National City, CA 91950
Country: US
Movie ID: 2331-775F-399A-937C-54C2-0
Format Code: DVD-1

Transaction ID: 39
Transaction Timestamp: Jul 24 2015 6:52PM
Customer ID: 437
Customer Name and Address:
Kaitlyn Brooks
1342 Isla Bonita
Woodland Hills, CA 91364
Country: US
Movie ID: 36E3-649C-57F2-A251-72B1-E
Format Code: DVD-1

Transaction ID: 41
Transaction Timestamp: Jul 24 2015 6:52PM
Customer ID: 484
Customer Name and Address:
Manuel Arun
1437 Donaleen Cr
Beverly Hills, CA 90210
Country: US
Movie ID: 4EB2-DFF8-6B78-660F-C61B-D
Format Code: BR-1

Transaction ID: 45
Transaction Timestamp: Jul 24 2015 6:52PM
Customer ID: 335
Customer Name and Address:
Kayla Wood
1142 Firestone Dr.
Downey, CA 90241
Country: US
Movie ID: 4EB2-DFF8-6B78-660F-C61B-D
Format Code: DVD-1

Transaction ID: 47
Transaction Timestamp: Jul 24 2015 6:52PM
Customer ID: 416
Customer Name and Address:
Xavier Hall
1293 F Street
La Jolla, CA 92806
Country: US
Movie ID: 5868-409E-7BFB-536A-6067-E
Format Code: DVD-1

Transaction ID: 49
Transaction Timestamp: Jul 24 2015 6:52PM
Customer ID: 419
Customer Name and Address:
Amber Hill
1299 Carpetta Circle
Grossmont, CA 91941
Country: US
Movie ID: 9824-4212-BADD-A847-068B-1
Format Code: DVD-1

(6) query and result set to display the content of the temporary table created by the SELECT INTO… FROM (SELECT… UNION SELECT…) command of Sales.DiscInventoryShortgageAtCenter10. This query is contemporary with the view output in (3) above, and reflects the circumstance that at the time of its execution the Sales.DiscAtMailCenter tables remained unpopulated with inventory quantities:

```
SELECT * FROM Sales.DiscInventoryShortageAtCenter10;
GO
```

results:

```
movie_id                    format_code Needed      On_Hand     Deficit
--------------------------- ----------- ----------- ----------- -----------
0A89-4017-2ED5-D199-0491-H DVD-1       1           0           1
2331-775F-399A-937C-54C2-0 DVD-1       2           0           2
36E3-649C-57F2-A251-72B1-E DVD-1       1           0           1
4EB2-DFF8-6B78-660F-C61B-D BR-1        1           0           1
4EB2-DFF8-6B78-660F-C61B-D DVD-1       1           0           1
5868-409E-7BFB-536A-6067-E DVD-1       1           0           1
843E-C238-7F61-DDFC-F427-9 BR-1        1           0           1
843E-C238-7F61-DDFC-F427-9 DVD-1       1           0           1
9824-4212-BADD-A847-068B-1 DVD-1       1           0           1

(9 row(s) affected)
```

(7) still unimplemented

(8) Use Sales.FindAllOverdueDiscsFromCustomer to find all discs rented by the customer with ID 322 that are overdue at 9:30 AM on July 30, 2015, if the rental term from order date is only 5 days:

```
EXECUTE Sales.uspFindAllDiscsOverdueFromCustomer @customer_id = 322,
                                                 @days_rented = 5;
GO
```

result:

```
rental_id            rental_date             return_id            return_date
-------------------- ----------------------- -------------------- -----------------------
43                   2015-07-24 18:52:58.073 NULL                 NULL
```

(9) Use Sales.WhenCustomerLastRentedDisc to get the last disc rented by the customer with ID 322:

```
DECLARE @last_date datetime;
EXECUTE Sales.uspWhenCustomerLastRentedDisc      @customer_id = 322,
                                                 @when_rented = @last_date OUTPUT;
SELECT @last_date;
GO
```

result:

```
-----------------------
2015-07-24 18:52:58.073

(1 row(s) affected)
```

(10) Use the combination of (8) and (9) to find the last disc rented by the customer with ID 322 is overdue at 9:30 AM on July 30, 2015, if the rental term from order date is only 5 days:

```
EXECUTE Sales.uspWGetLastRentedDiscIfOverdue          @this_customer_id = 322,
                                                      @days_rented = 5;
GO
```

Result:

```
rental_id            rental_date             return_id            return_date
-------------------- ----------------------- -------------------- -----------------------
43                   2015-07-24 18:52:58.073 NULL                 NULL

(1 row(s) affected)
```

(11) test the trigger created on Sales.TransactedEvent to promote the integrity of its unary relationship-

(a) try to insert a movie purchase with a previous_transaction_id value:

```
INSERT INTO Sales.TransactedEvent
(
     transaction_id,
     movie_id,
     format_code,
     customer_id,
     transaction_timestamp,
     transaction_type_code,
     transaction_amount,
     previous_transaction_id
)
VALUES
(
     NEXT VALUE FOR Sales.TransactionSequence,
     '5868-409E-7BFB-536A-6067-E',
     'DVD-1',
     399,
     GETDATE(),
     (SELECT transaction_type_code FROM Sales.TransactionType
      WHERE transaction_description = 'movie purchase'),
```

```
     11.95,
     50
);
GO
```

Result:

```
Msg 50000, Level 15, State 1, Procedure trgInsertOrUpdateTransactedEvent, Line 27
A movie rental or purchase is related to no previous transaction.
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

(a) try to insert a DVD disc return with no previous_transaction_id value:

```
INSERT INTO Sales.TransactedEvent
(
     transaction_id,
     movie_id,
     format_code,
     customer_id,
     transaction_timestamp,
     transaction_type_code,
     transaction_amount,
     previous_transaction_id
)
VALUES
(
     NEXT VALUE FOR Sales.TransactionSequence,
     '5868-409E-7BFB-536A-6067-E',
     'DVD-1',
     399,
     GETDATE(),
     (SELECT transaction_type_code FROM Sales.TransactionType
      WHERE transaction_description = 'disc return'),
```

```
     11.95,
     NULL
);
GO
```

Result:

```
Msg 50000, Level 15, State 1, Procedure trgInsertOrUpdateTransactedEvent, Line 36
A previous transaction must be specified for any transaction other than a rental or purchase.
Msg 3609, Level 16, State 1, Line 4
The transaction ended in the trigger. The batch has been aborted.
```

(c) Try to specify a previous related transaction other than a rental or sale:

```
INSERT INTO Sales.TransactedEvent
(
     transaction_id,
     movie_id,
     format_code,
     customer_id,
     transaction_timestamp,
     transaction_type_code,
     transaction_amount,
     previous_transaction_id
)
VALUES
(
     NEXT VALUE FOR Sales.TransactionSequence,
     '9824-4212-BADD-A847-068B-1',
     'DVD-1',
     419,
     GETDATE(),
     (SELECT transaction_type_code FROM Sales.TransactionType
      WHERE transaction_description = 'disc return'),
```

```
     11.95,
     70
);
GO
```

Result:

```
Msg 50000, Level 15, State 1, Procedure trgInsertOrUpdateTransactedEvent, Line 59
A previous related transaction must be either a movie rental or purchase.
Msg 3609, Level 16, State 1, Line 4
The transaction ended in the trigger. The batch has been aborted.
```

(d) Try to relate a new transaction to a previous rental or purchase by a different customer:

```
INSERT INTO Sales.TransactedEvent
(
     transaction_id,
     movie_id,
     format_code,
     customer_id,
     transaction_timestamp,
     transaction_type_code,
     transaction_amount,
     previous_transaction_id
)
VALUES
(
     NEXT VALUE FOR Sales.TransactionSequence,
     '9824-4212-BADD-A847-068B-1',
     'DVD-1',
     416,
     GETDATE(),
     (SELECT transaction_type_code FROM Sales.TransactionType
      WHERE transaction_description = 'disc return'),
```

```
     11.95,
     49
);
GO
```

Result:

```
Msg 50000, Level 15, State 1, Procedure trgInsertOrUpdateTransactedEvent, Line 72
Critical fields in a new and related previous transaction fail to correspond.
Msg 3609, Level 16, State 1, Line 4
The transaction ended in the trigger. The batch has been aborted.
```

WebCatalogBrowsingOutput.docx
IST 210 Section 1 Team 2
Sample result sets produced by the SELECT queries that populate the output cursors of the web-catalog browsing procedures:

(1) Sales.uspGetAvailableFormsOfMovieCursor,
with @movie_id = '2331-775F-399A-937C-54C2-0'

	movie_title	release_date	format_name	rental_price	purchase_price
1	The Godfather Part II	1974-12-12	Blu-ray	5.95	14.95
2	The Godfather Part II	1974-12-12	DVD	3.95	11.95
3	The Godfather Part II	1974-12-12	High-Definition Stream	3.99	14.99
4	The Godfather Part II	1974-12-12	Standard-Definition Stream	2.99	9.99

(2) Sales.uspGetMovieCursorFromTitleText,
with @title_search_text = 'Godfather'

	movie_id	movie_title	studio_id	release_date	country_co...	director_id	movie_length	genre_code	movie_description	movie_description_uri	movie_poster_uri	imdb_id
1	843E-C238-7F61-DDFC-F427-9	The Godfather	CFC0-F4BC	1972-03-24	US	1	175	4	NULL	NULL	NULL	tt0068646
2	2331-775F-399A-937C-54C2-0	The Godfather Part II	CFC0-F4BC	1974-12-12	US	1	200	4	NULL	NULL	NULL	tt0071562

(3) Sales.uspGetRecentReleasesMovieCursor,
with @number_of_months = 250

	movie_id	movie_title	studio_id	release_date	country_co...	director_id	movie_length	genre_code	movie_description	movie_description_uri	movie_poster_uri	imdb_id
1	9824-4212-BADD-A847-068B-1	Fargo	46E3-BC0D	1995-04-05	US	5	98	2	NULL	NULL	NULL	tt0116282
2	4EB2-DFF8-6B78-660F-C61B-D	Pulp Fiction	9591-0774	1994-10-14	US	4	154	2	NULL	NULL	NULL	tt0110912

(4) Sales.uspGetMovieCursorFromGenreCode, with @genre_code = 4

	movie_id	movie_title	studio_id	release_date	country_co...	director_id	movie_length	genre_code	movie_description	movie_description_uri	movie_poster_uri	imdb_id
1	B193-2EDD-9478-C964-FEF9-K	Bonnie and Clyde	35C5-3F4D	1967-08-13	US	3	111	4	NULL	NULL	NULL	tt0061418
2	2411-6FF0-31E2-D49D-870C-7	Chinatown	CFC0-F4BC	1974-06-20	US	2	150	4	NULL	NULL	NULL	tt007131
3	0A89-4017-2ED5-D199-0491-H	Raging Bull	D893-0AC2	1980-12-19	US	7	129	4	NULL	NULL	NULL	tt0081398
4	843E-C238-7F61-DDFC-F427-9	The Godfather	CFC0-F4BC	1972-03-24	US	1	175	4	NULL	NULL	NULL	tt0068646
5	2331-775F-399A-937C-54C2-0	The Godfather Part II	CFC0-F4BC	1974-12-12	US	1	200	4	NULL	NULL	NULL	tt0071562

(5) Sales.uspGetMovieCursorFromActorName,
with @actor_name_last = 'De Niro'

	movie_id	movie_title	studio_id	release_date	country_code	director_id	movie_length	genre_code	movie_description	movie_description_uri	movie_poster_uri
1	0A89-4017-2ED5-D199-0491-H	Raging Bull	D893-0AC2	1980-12-19	US	7	129	4	NULL	NULL	NULL
2	2331-775F-399A-937C-54C2-0	The Godfather Part II	CFC0-F4BC	1974-12-12	US	1	200	4	NULL	NULL	NULL

imdb_id	actor...	actor_gender	actor_name_first	actor_name_last	other_actor_details	as_character	is_lead
tt0081398	11	M	Robert	De Niro	NULL	Jake La Motta	1
tt0071562	11	M	Robert	De Niro	NULL	Vito Corleone	0

(6) Sales.uspGetMovieCastCursor,
with @movie_id = '2411-6FF0-31E2-D49D-870C-7'

	actor_id	actor_gender	actor_name_first	actor_name_last	other_actor_details	as_character	is_lead
1	14	M	Jack	Nicholson	NULL	J.J. Gittes	1
2	15	F	Faye	Dunaway	NULL	Evelyn Mulwray	1
3	16	M	John	Huston	NULL	Noah Cross	0

(7) Sales.uspGetMovieCursorFromDirectorName,
with @director_name_last = 'Coppola'

	movie_id	movie_title	studio_id	release_date	country_co...	movie_length	genre_code	movie_description	movie_description_uri
1	2331-775F-399A-937C-54C2-0	The Godfather Part II	CFC0-F4BC	1974-12-12	US	200	4	NULL	NULL
2	843E-C238-7F61-DDFC-F427-9	The Godfather	CFC0-F4BC	1972-03-24	US	175	4	NULL	NULL

movie_poster_uri	imdb_id	director_id	director_name_first	director_name_last
NULL	tt0071562	1	Francis Ford	Coppola
NULL	tt0068646	1	Francis Ford	Coppola

(8) Sales.uspGetCustomerQueueCursor, with @customer_id = ?

(no data so far loaded)

(9) Sales.uspGetAverageMovieRating, with @movie_id = ?

(no data so far loaded)

BusinessIntelligenceOutput.docx
IST 210 Section 1 Team 2
Output produced with use of the code created in BusinessIntelligence.sql:

(1) Show the 5 top-grossing movies in July 2015:

```
USE Flix2YouDW;
GO

EXECUTE Sales.uspQueryTopGrossingMovies     @datetime1 = '2015-07-01',
                                            @datetime2 = '2015-07-31',
                                            @how_many = 5;
GO
```

Movie	Revenue	Per Cent of Total
Star Wars	$26.90	31.5 %
Pulp Fiction	$12.89	15.1 %
The Godfather	$9.90	11.6 %
Jaws	$7.94	9.3 %
Fargo	$7.90	9.3 %

(2) Rank the popularity of movie genres and titles during the month of July 2015, for the age group including all viewers between 18 and 95 years old;

```
USE Flix2YouDW;
GO

EXECUTE Sales.uspRankGenrePopularityInAgeGroup   @datetime1 = '2015-07-01',
                                                 @datetime2 = '2015-07-31',
                                                 @age1      = 18,
                                                 @age2      = 95;
GO
```

```
Genre                Movie                                Orders      Per Cent of Total
-------------------- ------------------------------------ ----------- ------------------
NULL                 NULL                                 16          100.0 %
Action and Adventure NULL                                 2           12.5 %
Action and Adventure Jaws                                 2           12.5 %
Comedy               NULL                                 5           31.3 %
Comedy               Fargo                                2           12.5 %
Comedy               Pulp Fiction                         3           18.8 %
Drama                NULL                                 7           43.8 %
Drama                Bonnie and Clyde                     1           6.3 %
Drama                Chinatown                            1           6.3 %
Drama                Raging Bull                          1           6.3 %
Drama                The Godfather                        2           12.5 %
Drama                The Godfather Part II                2           12.5 %
Sci-Fi and Fantasy   NULL                                 2           12.5 %
Sci-Fi and Fantasy   Star Wars                            2           12.5 %
```

(3) Rank the popularity of hours of the day a days of the weeks as movie viewing times for all customers, during the month of July 2015 (see note in code):

```
USE Flix2YouDW;
GO

EXECUTE Sales.uspShowSalesByHourAndWeekday  @datetime1 = '2015-07-01',
                                            @datetime2 = '2015-07-31';
GO

Weekday    Hour (1-24) Orders      Per Cent of Total
-------    ----------- ------      -----------------
NULL       NULL        16          100.0 %
Friday     NULL        16          100.0 %
Friday     14          1           6.3 %
Friday     15          5           31.3 %
Friday     18          10          62.5 %
```

(4) Show total customers added by month and geographic region on and after January 1, 2015:

```
USE Flix2YouDW;
GO

EXECUTE  Sales.uspShowCustomerCountByMembershipDate    @after_date = '2015-01-01';
GO
```

Month Joined	Geographic Region	Number Added
NULL	NULL	9
NULL	Canada West	2
NULL	US West	7
2015-01	NULL	2
2015-01	US West	2
2015-03	NULL	4
2015-03	US West	4
2015-04	NULL	1
2015-04	Canada West	1
2015-05	NULL	2
2015-05	Canada West	1
2015-05	US West	1

(5) and (6) Query the cumulative aggregate new cash flow produced by revenue-producing movies in all four categories of high-level encoding format, after updating the ViewableEncoding entity to set an aggregate cost of $10,000 for each of its occurrences (see the note in the code for this procedure):

```
USE Flix2You;
GO
```

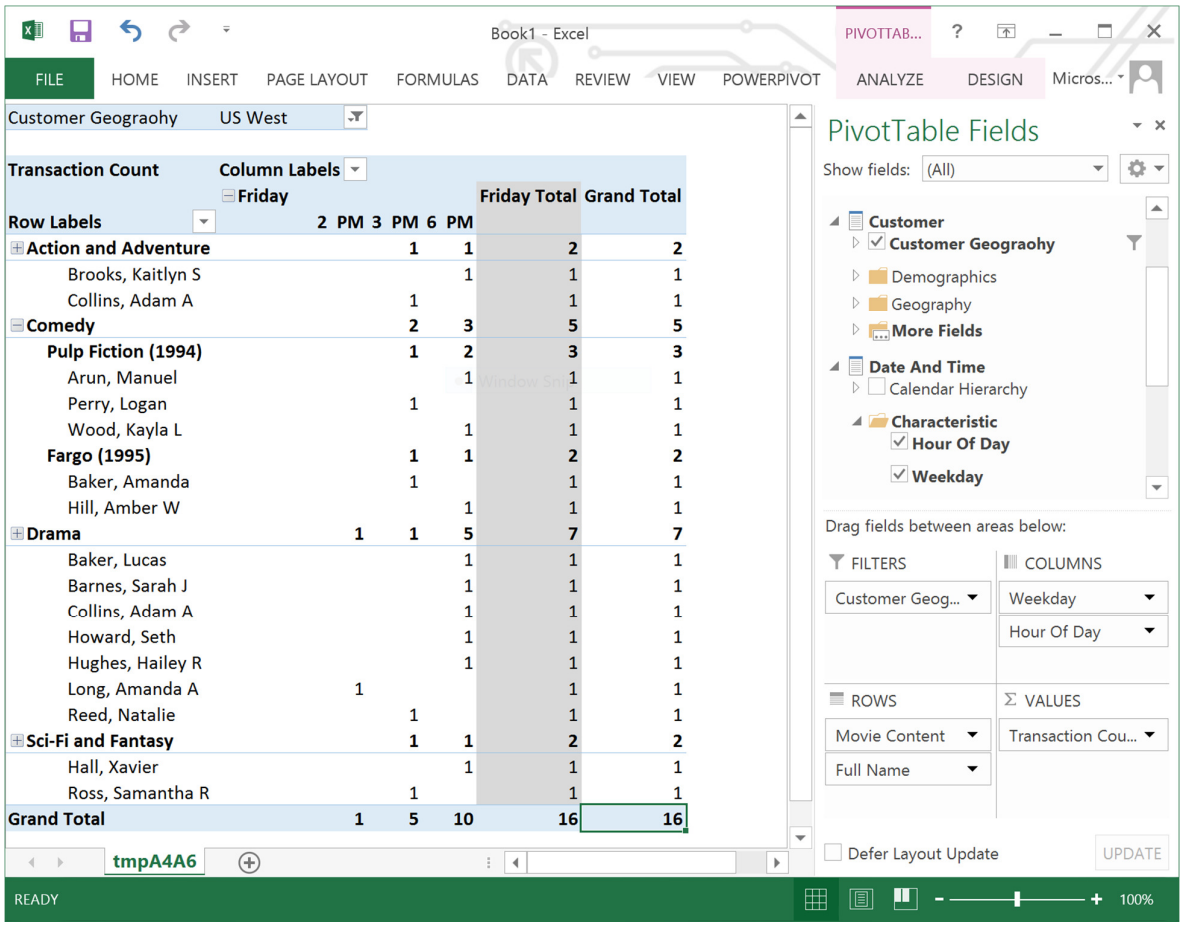
```
UPDATE Sales.ViewableEncoding
SET aggregate_cost = 10000;
GO

EXECUTE Sales.uspQueryCashFlowByHighLevelFormat
GO

(36 row(s) affected)
Product Type                        Total Revenue         Total Cost            Net Cash Flow
--------------------------          --------------        ----------            --------------
Blu-ray                             26.85                 30000.00              -29973.15
DVD                                 43.55                 70000.00              -69956.45
High-Definition Stream              11.97                 30000.00              -29988.03
Standard-Definition Stream          2.99                  10000.00              -9997.01
```

PowerPivotMOLAPOutput.docx
IST 210 Section 1 Team 2
Additional PowerPivot queries on the Analysis Services MOLAP cube

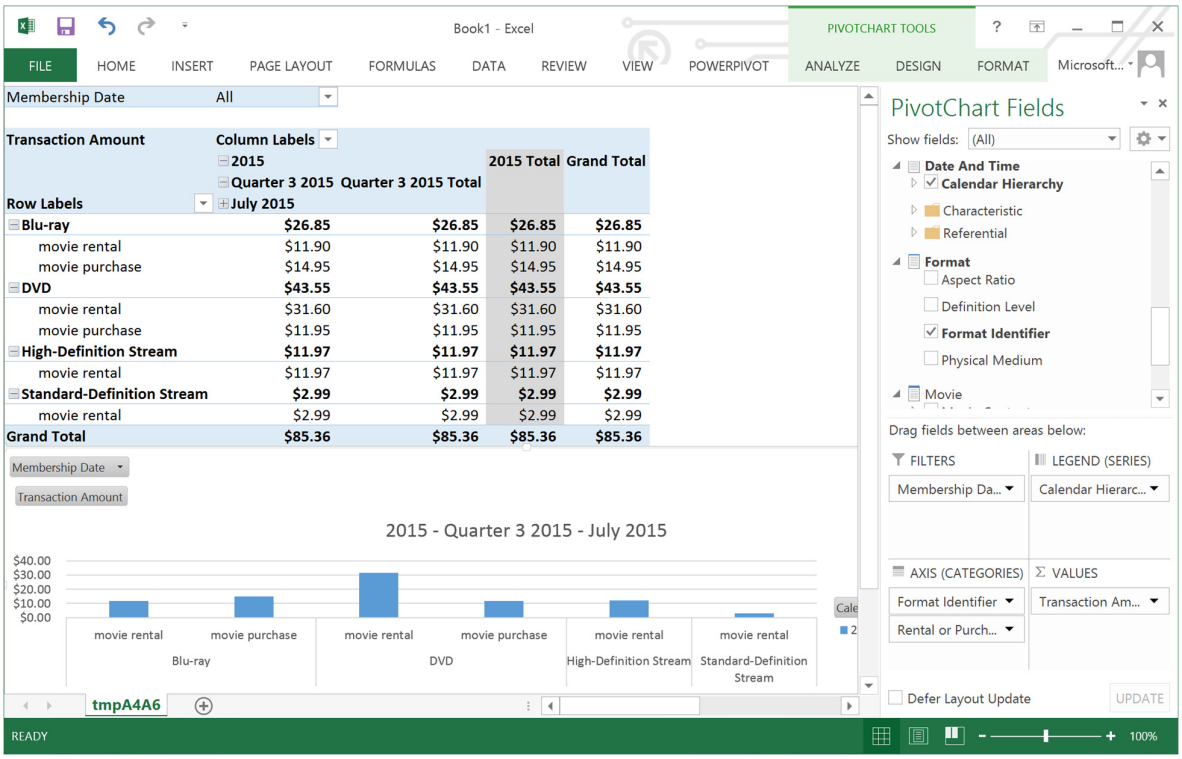
(1) Genres, movies and the Flix2You customers who watched them (ordered by last name ascending), by weekday and hour of the day rented or purchased, and by calendar month, quarter and year, for the US West market region:



Customer Geograohy: US West

Transaction Count	Column Labels				
	Friday				
Row Labels	2 PM	3 PM	6 PM	Friday Total	Grand Total
Action and Adventure		1	1	2	2
Brooks, Kaitlyn S			1	1	1
Collins, Adam A		1		1	1
Comedy		2	3	5	5
Pulp Fiction (1994)		1	2	3	3
Arun, Manuel			1	1	1
Perry, Logan		1		1	1
Wood, Kayla L			1	1	1
Fargo (1995)		1	1	2	2
Baker, Amanda		1		1	1
Hill, Amber W			1	1	1
Drama	1	1	5	7	7
Baker, Lucas			1	1	1
Barnes, Sarah J			1	1	1
Collins, Adam A			1	1	1
Howard, Seth			1	1	1
Hughes, Hailey R			1	1	1
Long, Amanda A	1			1	1
Reed, Natalie		1		1	1
Sci-Fi and Fantasy		1	1	2	2
Hall, Xavier			1	1	1
Ross, Samantha R		1		1	1
Grand Total	1	5	10	16	16

(2) Revenue by year, calendar quarter and month (there have only been loaded a few movie rentals and purchases, all of them taking place in July 2015), categorized by high-level movie format and revenue transaction type (rental or purchase), for customers having any membership date:


Book1 - Excel
PIVOTCHART TOOLS
FILE
HOME
INSERT
PAGE LAYOUT
FORMULAS
DATA
REVIEW
VIEW
POWERPIVOT
ANALYZE
DESIGN
FORMAT
Microsoft...
Membership Date All
Transaction Amount
Column Labels
2015
Quarter 3 2015
Quarter 3 2015 Total
2015 Total
Grand Total
Row Labels
July 2015
Blu-ray $26.85 $26.85 $26.85 $26.85
movie rental $11.90 $11.90 $11.90 $11.90
movie purchase $14.95 $14.95 $14.95 $14.95
DVD $43.55 $43.55 $43.55 $43.55
movie rental $31.60 $31.60 $31.60 $31.60
movie purchase $11.95 $11.95 $11.95 $11.95
High-Definition Stream $11.97 $11.97 $11.97 $11.97
movie rental $11.97 $11.97 $11.97 $11.97
Standard-Definition Stream $2.99 $2.99 $2.99 $2.99
movie rental $2.99 $2.99 $2.99 $2.99
Grand Total $85.36 $85.36 $85.36 $85.36
Membership Date
Transaction Amount
2015 - Quarter 3 2015 - July 2015
$40.00
$30.00
$20.00
$10.00
$0.00
movie rental
movie purchase
movie rental
movie purchase
movie rental
movie rental
Blu-ray
DVD
High-Definition Stream
Standard-Definition Stream
tmpA4A6
READY
100%
PivotChart Fields
Show fields: (All)
Date And Time
Calendar Hierarchy
Characteristic
Referential
Format
Aspect Ratio
Definition Level
Format Identifier
Physical Medium
Movie
Drag fields between areas below:
FILTERS
LEGEND (SERIES)
Membership Da...
Calendar Hierarc...
AXIS (CATEGORIES)
VALUES
Format Identifier
Transaction Am...
Rental or Purch...
Defer Layout Update
UPDATE