

Cinemate: A Database for Movie Fans

Conor Klump

I. Ideation

Movies are one of humanity's most popular forms of entertainment. In 2018, the film industry in the United States alone grossed over \$32 billion, and several other countries tallied revenue of at least \$1 billion. (Filmed entertainment revenue in selected countries worldwide in 2018, n.d.).

For ardent fans, movies represent a rich source of debate and discussion, a body of facts and figures worthy of scholarly study. The purpose of this project is to develop such a tool for a group of movie enthusiasts to use to track their home movie watching habits.

Movies are complicated to produce. They are likewise complicated in their data makeup because so many people are involved in their production. Previous attempts have been made as part of previous submissions for this final project to store a much wider variety of facts and figures regarding movies. In fact, a database consisting of 13 tables was successfully produced, and interesting queries were run against it. For this final submission, the project will be simplified significantly to store a subset of the data originally envisioned and implemented.

This version of the project will store data about the movies watched by a group of movie fans at home. It will store the name, age, and gender of each movie fan; the title, year of release, box office total, and runtime and details about the rental itself, such as the date of the rental, the format, the company that furnished the rental, and the price. This data is currently stored in a spreadsheet. The goal of this project is store the data in a better way that is more scalable and less prone to insert, update, and delete anomalies.

II. Project Definition

A. Problems and concerns to be addressed

The current approach stores data in one spreadsheet, like so:

MovieRental (ViewerFirstName, ViewerLastName, ViewerAge, ViewerGender, MovieTitle, MovieYearReleased, MovieBoxOffice, MovieLength, RentalDate, RentalFormat, RentalCost, RentalCompany, RentalCompanyType)

The spreadsheet is shown in section II.B. The dataset has four themes: Viewer, Movie, Rental, and RentalCompany. Because one table tries to store data about four different things, redundancy is a problem. This leads to problems, which are described here:

Update Modification Problems:

- If Karen Klump, for example, changes her last name, that has to be changed in three rows.
- If it is found that Raiders of the Lost Ark had actually been released in 1982 instead of 1981, three rows would have to be changed.
- If Amazon Prime suddenly became a physical movie supplier instead of digital, three records would have to be changed to reflect the new company direction.

- If Amazon Prime changes its name to Amazon Video, again three records would have to be changed.

Insert Modification Problems:

- Null values for viewers without a rental. For example, Marissa Kmetty has not rented a movie, but she appears in this dataset because she is part of the movie fan group.
- Viewer names are duplicated. If Conor Klump rents yet another movie, it would be easy to accidentally mistype his name with two n's instead of one, and this would be inconsistent.
- Rental company names are duplicated. Any time a movie is rented from Amazon Prime, care must be taken to type it just like the previous entries. What if, for example, "Amazon Prime" was entered as "Amazon" in a future insert?
- The same issues exist for other fields, such as when another person rents Raiders of the Lost Ark. Movie titles, years of release, etc. all must be typed with care.

Delete Modification Problems:

- Data about a viewer could be lost if a movie is deleted. For example, if La La Land is deleted, Lauren Klump's presence in the movie fan group would be lost.
- If Conor Klump is removed from the fan group, data about four rentals will be lost. Whenever any viewer is removed, all data about the rentals that occurred are removed.
- Likewise, if a viewer is the only one to rent a movie, all data about that movie will disappear. For example, if Lauren Klump is deleted, all data about La La Land is purged.
- If a viewer who happens to be the only one to rent from a particular company, all memory of a rental being made from that company will disappear. This is the case with Lauren Klump; if she is removed from the group, information about RedBox disappears.

B. Examples of the current approach

An Excel spreadsheet that contains all the data consists of the following columns:

	A	B	C	D	E	F	G	H	I	J	K	L	M
	ViewerFirstName	ViewerLastName	ViewerAge	ViewerGender	MovieTitle	MovieYearReleased	MovieBoxOffice	MovieLength	RentalDate	RentalFormat	RentalCompany	RentalCompanyType	RentalCost
1	Conor	Klump	20	M	Raiders of the Lost Ark	1981	389.9	115	5/16/21	Streaming	Amazon Prime	Digital	3.99
2	Conor	Klump	20	M	Rain Man	1988	354.8	134	4/20/21	Streaming	Amazon Prime	Digital	2.99
3	Conor	Klump	20	M	Philadelphia	1993	206.7	126	4/2/21	DVD	Family Video	Physical	3.99
4	Conor	Klump	20	M	Kill Bill Vol. 1	1994	180.9	111	5/2/21	Streaming	Netflix	Digital	3.49
5	Lauren	Klump	18	F	La La Land	2018	448.9	128	4/30/21	DVD	RedBox	Physical	2.99
6	Karen	Klump	46	F	The Muppets	2011	165.2	103	5/5/21	Streaming	Hulu	Digital	3.29
7	Karen	Klump	46	F	Forrest Gump	1994	683.1	142	5/7/21	VHS	Family Video	Physical	2.99
8	Karen	Klump	46	F	The Devil Wears Prada	2006	326.7	109	5/1/21	Streaming	Amazon Prime	Digital	3.99
9	Evan	Klump	14	M	Kill Bill Vol. 1	2003	180.9	111	4/18/21	Streaming	Netflix	Digital	3.29
10	Evan	Klump	14	M	Sound of Metal	2020	0.176	120	5/4/21	Streaming	Netflix	Digital	3.29
11	Ray	Klump	50	M	Raiders of the Lost Ark	1981	389.9	115	5/5/21	DVD	Family Video	Physical	3.59
12	Cathy	Lazuka	45	F	Raiders of the Lost Ark	1981	389.9	115	4/19/21	Streaming	Netflix	Digital	3.29

C. The new approach

The new approach would separate the data in the Excel spreadsheet according to the four themes of Viewer, Movie, Rental, and RentalCompany.

Four entities result from this, with these possible fields:

- Viewer (ViewerFirstName, ViewerLastName, ViewerAge, Viewer Gender)
- Movie (MovieTitle, MovieYearRelease, MovieBoxOffice, MovieLength)
- Rental (Viewer, Movie, RentalDate, RentalFormat, RentalCompany, RentalCost)
- RentalCompany (RentalCompanyName, RentalCompanyType)

This separation of data by theme solves a number of problems:

- **Update Modification Problems:**
 - All viewer information is now stored in one table. So if, for example, Karen Klump changes her last name, that change needs to be made in only one place.
 - All movie information is now stored in one table. So if, for example, Raiders of the Lost Ark's year is changed to 1982, that change needs to be made in only one place.
 - All rental company information is now stored in one table. So if, for example, Amazon Prime changes to a physical store or changes its name to Amazon Video, those changes need to be made in only one place.
- **Insert Modification Problems:**
 - Viewers who have no rentals - i.e. they are members of the movie fan club but haven't yet rented a movie - no longer pose a problem related to null values. Marissa Kmetty, for example, can be stored in the Viewer table; she will have no related record in the rental table, but that is perfectly fine.
 - Because each viewer appears in only one place, when a new viewer is added, the way the data is typed the first time is what matters, because that is the only time the data will be first entered. When Conor Klump rents another movie, the existing Conor Klump record will remain and not be re-entered, thus avoiding inconsistencies due to typing mistakes. The same claim can be made for all other fields, such as the name of the rental company, as the name won't have to be re-entered when another video is rented from that source.
 - When Raiders of the Lost Ark is rented yet again, the data concerning the movie will not have to be re-entered, as it is already stored in the Movie table.
- **Delete Modification Problems:**
 - If La La Land is deleted, Lauren Klump will still exist in the Viewer table. Deletes from the Movie table will not imperil records in the Viewer table.
 - If Lauren Klump is deleted, the fact that she rented a movie would likewise go. This may or may not be wanted, and a database can be configured to restrict or cascade such changes. Regardless, La La Land will remain in the Movie table, even if Lauren is no longer there to rent it. Furthermore, Redbox will remain as a company that the database recognizes.
 - The same argument can be made for Conor Klump's four rentals. If Conor is deleted, the four movies Conor rented and the companies he rented them from will remain, regardless of whether the delete cascades to the Rental table or is restricted.

III. Entity-Relationship/Data Modeling

A. Conceptual model

The conceptual model is described in Chapter 4 of *Database Concepts* (Kroenke, D. M., Auer, D. J., Vandenberg, S., Yoder, R.C., 2020). It focuses on the entities and the possible fields that will be assigned to those entities. For this model, surrogate keys will be introduced to uniquely identify the instances of each of the proposed entities.

Here are the four entities and their possible fields:

- Viewer (ViewerID, ViewerFirstName, ViewerLastName, ViewerAge, ViewerGender)
- Movie (MovieID, MovieTitle, MovieYearReleased, MovieBoxOffice, MovieLength)
- Rental (RentalID, RentalDate, RentalFormat, RentalCost, RentalCompany, Viewer, Movie)
- RentalCompany (RentalCompanyID, RentalCompanyName, RentalCompanyType)

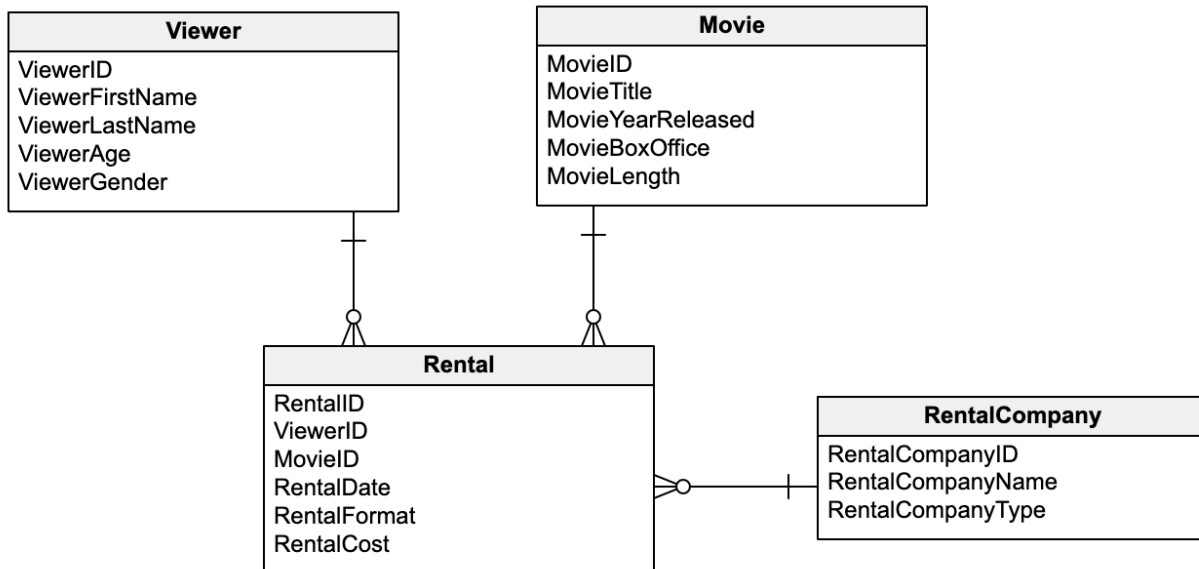
The model employs some assumptions and decisions about the relationships among the entities.

- A viewer may have zero rentals, one rental, or many rentals. Each rental is made by one viewer.
- A movie may have zero rentals, one rental, or many rentals. Each rental is made for one movie.
- A rental company may have sourced the rental for no rentals, one rental, or many rentals. Each rental is sourced by one rental company.
- The ViewerID, MovieID, RentalID, and RentalCompanyID fields are surrogate keys and could be implemented as unique sequential numbers or even auto-generated integer sequences in their respective tables.

This table summarizes the relationships.

RELATIONSHIP			CARDINALITY	
PARENT	CHILD	TYPE	MAX	MIN
Viewer	Rental	ID-Dependent	1:N	M-O
Movie	Rental	ID-Dependent	1:N	M-O
RentalCompany	Rental	ID-Dependent	1:N	M-O

- i. The following Vertabelo diagram illustrates the conceptual model.



B. Logical model

The logical model is described in chapter 5 of *Database Concepts* (Kroenke, *et. al.*, 2020).

Primary keys are included in the logical model (although they were also included in the preceding section's conceptual model). In this listing of entities and their fields, the primary keys are underlined.

- Viewer (ViewerID, ViewerFirstName, ViewerLastName, ViewerAge, ViewerGender)
- Movie (MovieID, MovieTitle, MovieYearReleased, MovieBoxOffice, MovieLength)
- Rental (RentalID, RentalDate, RentalFormat, RentalCost, RentalCompany, Viewer, Movie)
- RentalCompany (RentalCompanyID, RentalCompanyName, RentalCompanyType)

For all of these entities, the surrogate key is a candidate key. For RentalCompany, RentalCompanyName can also be a candidate key, as no two companies have the same name.

i. Functional dependencies and the normalized tables

It is useful to consider the normalization process that gave rise to this logical model. The original dataset had this structure:

MovieRental (ViewerFirstName, ViewerLastName, ViewerAge, ViewerGender, MovieTitle, MovieYearReleased, MovieBoxOffice, MovieLength, RentalDate, RentalFormat, RentalCost, RentalCompany, RentalCompanyType)

Consider these assumptions:

- two or more viewers cannot have the same first and last names.
- two movies cannot have the same title.
- No more than one person can rent a particular movie from a particular store on a particular day.

With these assumptions, the following functional dependencies arise:

(ViewerFirstName, ViewerLastName) → ViewerAge, ViewerGender

MovieTitle → MovieYearReleased, MovieLength, MovieBoxOffice

RentalCompanyName → RentalCompanyType

(ViewerFirstName, ViewerLastName, MovieTitle, RentalCompanyName) → RentalDate, RentalFormat, RentalCost

ii. Constraints, assumptions, and relations

Rather than use combinations of fields as the keys, the database will use surrogate keys in each table. These surrogate keys become the basis of the foreign keys that implement the relationships among the tables. This results in the following normalized entities which, when implemented, become the database's tables. (In this listing, the primary keys are underlined, and the foreign keys are italicized).

- Viewer (ViewerID, ViewerFirstName, ViewerLastName, ViewerAge, ViewerGender)
- Movie (MovieID, MovieTitle, MovieYearReleased, MovieBoxOffice, MovieLength)
- Rental (RentalID, RentalDate, RentalFormat, RentalCost, *RentalCompanyID*, *ViewerID*, *MovieID*)
- RentalCompany (RentalCompanyID, RentalCompanyName, RentalCompanyType)

The relationships are the same as they were for the conceptual model. This table repeats them.

RELATIONSHIP			CARDINALITY	
PARENT	CHILD	TYPE	MAX	MIN
Viewer	Rental	ID-Dependent	1:N	M-O
Movie	Rental	ID-Dependent	1:N	M-O
RentalCompany	Rental	ID-Dependent	1:N	M-O

The relationships impose the following constraints.

Relationship		Referential Integrity Constraint	Cascading Behavior	
Parent	Child		On Update	On Delete

Viewer	Rental	ViewerID in Rental must exist in ViewerID in Viewer	No	No
Movie	Rental	MovieID in Rental must exist in MovieID in Movie	No	No
RentalCompany	Rental	RentalCompanyID in Rental must exist in RentalCompanyID in RentalCompany	No	No

Now identify the tables, their fields, and their data types.

Table Name: VIEWER

Col Name	Datatype	PK	NotNull	M
ViewerID	int	x	x	x
ViewerLastName	varchar(25)		x	x
ViewerFirstName	varchar(25)		x	x
ViewerGender	char(1)		x	x
ViewerAge	int		x	x

Table Name: MOVIE

Col Name	Datatype	PK	NotNull	M
MovieID	int	x	x	x
MovieTitle	varchar(80)		x	x
MovieYearReleased	int		x	x
MovieBoxOffice	float		x	x
MovieLength	int		x	x

Table Name: RENTALCOMPANY

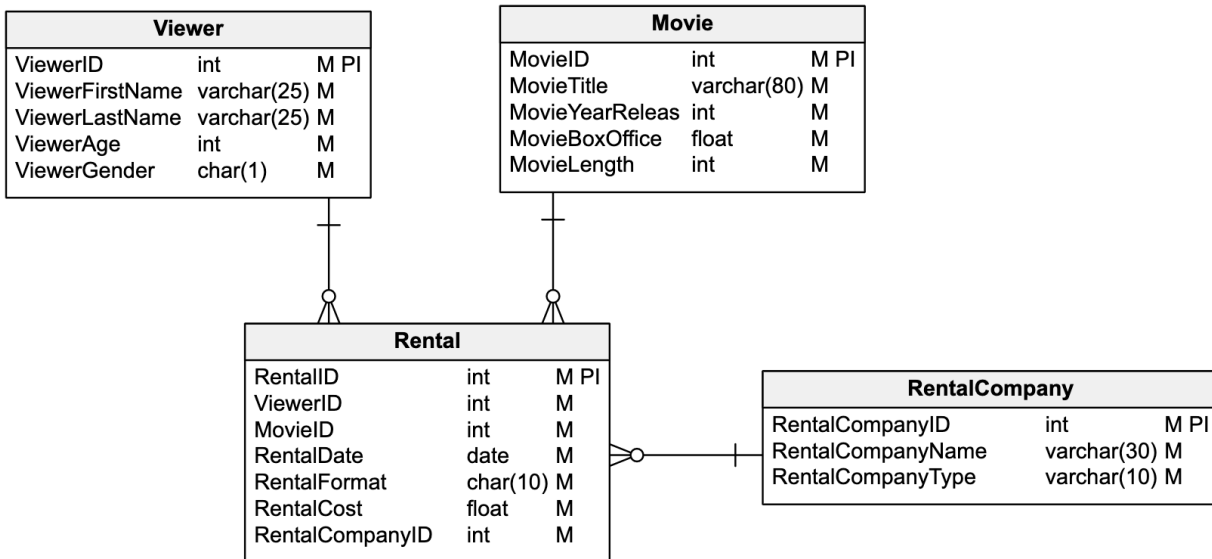
Col Name	Datatype	PK	NotNull	M
RentalCompanyID	int	x	x	x
RentalCompanyName	varchar(30)		x	x
RentalCompanyType	varchar(10)		x	x

Table Name: RENTAL

Col Name	Datatype	PK	NotNull	M
RentalID	int	x	x	x
MovieID	int		x	x
ViewerID	int		x	x
RentalCompanyID	int		x	x
RentalDate	date		x	x
RentalFormat	varchar(10)		x	x
RentalCost	float		x	x

iii. The model

Here is the logical model as built in Vertabello.



C. Physical model for implementation in Access

NOTE TO INSTRUCTOR: The model presented in the template requests that the functional dependencies, normalized tables, relationships, and constraints all be listed in this section. However, the Space Trip case example presents them in the Logical Model section, and the instructor's comments in the latest graded assignment also suggested moving them from the Physical Model section to the Logical Model section. So, this section deviates from the assignment template, because of both the Space Trip example and the instructor's advice. Please take that into account when grading.

i. Functional dependencies and normalized tables.

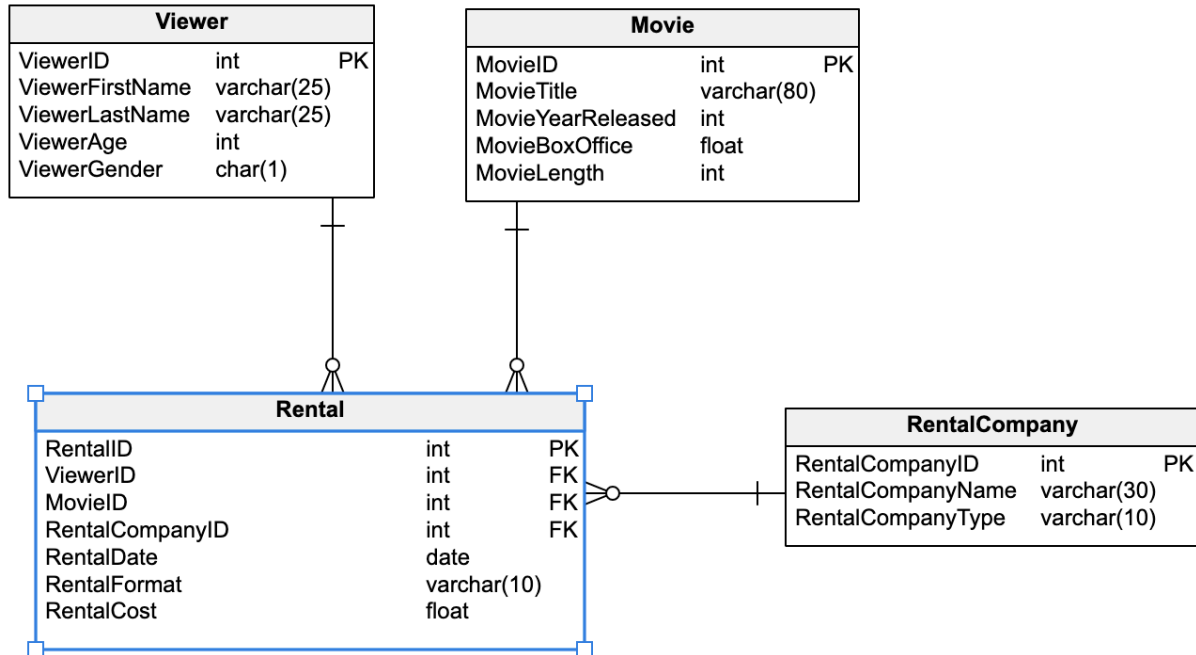
Please see the preceding section, as advised by the instructor and demonstrated in the Space Trip example.

ii. Constraints, assumptions, and relations.

Please see the preceding section, as advised by the instructor and demonstrated in the Space Trip example.

iii. Physical model and SQL

All the preceding work culminates in the physical model. A Vertabello diagram of the physical model appears below. Note that it includes specifics about the data type for each field.



From this model, Vertabelo's automatic SQL generator yields the following Data Definition Language (DDL) script. The target platform is SQL Server. This would have to be changed to use it with Access.

```

-- Created by Vertabelo (http://vertabelo.com)
-- Last modification date: 2021-06-08 02:16:41.809

-- tables
-- Table: Movie
CREATE TABLE Movie (
    MovieID int NOT NULL,
    MovieTitle varchar(80) NOT NULL,
    MovieYearReleased int NOT NULL,
    MovieBoxOffice float NOT NULL,
    MovieLength int NOT NULL,
    CONSTRAINT Movie_pk PRIMARY KEY (MovieID)
);

-- Table: Rental
CREATE TABLE Rental (
    RentalID int NOT NULL,
    ViewerID int NOT NULL,
    MovieID int NOT NULL,
    RentalCompanyID int NOT NULL,
    RentalDate date NOT NULL,
    RentalFormat varchar(10) NOT NULL,
    RentalCost float NOT NULL,
    CONSTRAINT Rental_pk PRIMARY KEY (RentalID)
  
```

```
);

-- Table: RentalCompany
CREATE TABLE RentalCompany (
    RentalCompanyID int NOT NULL,
    RentalCompanyName varchar(30) NOT NULL,
    RentalCompanyType varchar(10) NOT NULL,
    CONSTRAINT RentalCompany_pk PRIMARY KEY (RentalCompanyID)
);

-- Table: Viewer
CREATE TABLE Viewer (
    ViewerID int NOT NULL,
    ViewerFirstName varchar(25) NOT NULL,
    ViewerLastName varchar(25) NOT NULL,
    ViewerAge int NOT NULL,
    ViewerGender char(1) NOT NULL,
    CONSTRAINT Viewer_pk PRIMARY KEY (ViewerID)
);

-- foreign keys
-- Reference: Rental_Movie (table: Rental)
ALTER TABLE Rental ADD CONSTRAINT Rental_Movie
    FOREIGN KEY (MovieID)
    REFERENCES Movie (MovieID);

-- Reference: Rental_RentalCompany (table: Rental)
ALTER TABLE Rental ADD CONSTRAINT Rental_RentalCompany
    FOREIGN KEY (RentalCompanyID)
    REFERENCES RentalCompany (RentalCompanyID);

-- Reference: Rental_Viewer (table: Rental)
ALTER TABLE Rental ADD CONSTRAINT Rental_Viewer
    FOREIGN KEY (ViewerID)
    REFERENCES Viewer (ViewerID);

-- End of file.
```

IV. Database Prototype

A. Table Structures

The database was submitted along with this document. In the database, you will find queries that begin with the word "create". These queries were adapted from the SQL code Vertabello generated automatically to comply with Access's variations on SQL.

Here are screenshots of each of the tables.

Movie					
MovieID	MovieTitle	MovieYearR	MovieBoxOf	MovieLength	
1	Raiders of the I	1981	389.9	115	
2	Rain Man	1988	354.8	134	
3	Philadelphia	1993	206.7	126	
4	Kill Bill Vol. 1	1994	180.9	111	
5	La La Land	2018	448.9	128	
6	The Muppets	2011	165.2	103	
7	Forrest Gump	1994	683.1	142	
8	The Devil Wea	2006	326.7	109	
9	Sound of Meta	2020	0.176	120	

Rental						
RentalID	ViewerID	MovieID	RentalComp	RentalDate	RentalForma	RentalCost
1	1	1	1	05/16/2021	Streaming	3.99
2	1	2	1	04/20/2021	Streaming	2.99
3	1	3	2	04/02/2021	DVD	3.99
4	1	4	3	05/02/2021	Streaming	3.49
5	2	5	4	04/30/2021	DVD	2.99
6	4	6	5	05/05/2021	Streaming	3.29
7	4	7	2	05/07/2021	VHS	2.99
8	4	8	1	05/01/2021	Streaming	3.99
9	3	4	3	04/18/2021	Streaming	3.29
10	3	9	3	05/04/2021	Streaming	3.29
11	5	1	2	05/05/2021	DVD	3.59
12	7	1	3	04/19/2021	Streaming	3.29

RentalCompany		
RentalComp	RentalComp	RentalComp
1	Amazon Prime	Digital
2	Family Video	Physical
3	Netflix	Digital
4	RedBox	Physical
5	Hulu	Digital

Viewer				
ViewerID	ViewerFirstI	ViewerLastI	ViewerAge	ViewerGenc
1	Conor	Klump	20	M
2	Lauren	Klump	18	F
3	Evan	Klump	14	F
4	Karen	Klump	46	F
5	Ray	Klump	50	M
6	Marissa	Kmetty	21	F
7	Cathy	Lazuka	45	F

B. Sample Data

The preceding section included screenshots that showed all four tables and the data in them. This data was compiled from online sources like IMDB and Wikipedia as well as information about personal movie purchases. This data was entered into an Excel spreadsheet (shown earlier) and, through the database design process outlined in this document, transformed into the database attached here.

C. SQL Test Queries

Three useful queries were written to test the usefulness of the data model and the soundness of the completed prototype.

The first query, RentalsByViewer, tabulates the number of rentals made by each viewer, sorted by the number of rentals each made. The list includes the viewer who made no rentals thanks to the use of a LEFT JOIN. The query is this:

```
SELECT ViewerFirstName, ViewerLastName, count(RentalID)
FROM Viewer LEFT JOIN Rental ON Viewer.ViewerID = Rental.ViewerID
GROUP BY ViewerFirstName, ViewerLastName
ORDER BY count(RentalID) desc;
```

The results are this:

ViewerFirstI	ViewerLastI	Expr1002
Conor	Klump	4
Karen	Klump	3
Evan	Klump	2
Ray	Klump	1
Lauren	Klump	1
Cathy	Lazuka	1
Marissa	Kmetty	0

The second query, TotalSpentAtEachStore, lists the rental companies and how much money was spent there on rentals by the movie fan group. It is sorted in descending order so that the highest earning rental company appears at the top.

Here is the query:

```
SELECT RentalCompanyName, sum(RentalCost)
FROM RentalCompany LEFT JOIN Rental
ON RentalCompany.RentalCompanyID = Rental.RentalCompanyID
GROUP BY RentalCompanyName
ORDER by sum(RentalCost) desc;
```

Here are the results:

RentalComp	Expr1001
Netflix	13.36
Amazon Prime	10.97
Family Video	10.57
Hulu	3.29
RedBox	2.99

The third query, which is called YouthViewingHabits, attempts to capture the rental habits of people under the age of 21. It counts the number of streaming, DVD, and VHS rentals made by viewers who are under the age of 21.

Here is the query:

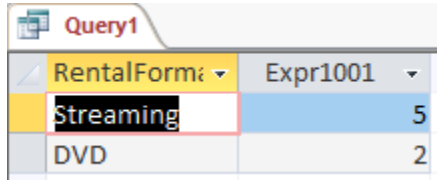
```
SELECT RentalFormat, count(Rental.MovieID)
FROM (Rental INNER JOIN Movie
ON Rental.MovieID = Movie.MovieID)
```

```

INNER JOIN Viewer on Rental.ViewerID = Viewer.ViewerID
WHERE ViewerAge < 21
GROUP BY RentalFormat
ORDER BY count(Rental.MovieID) desc;

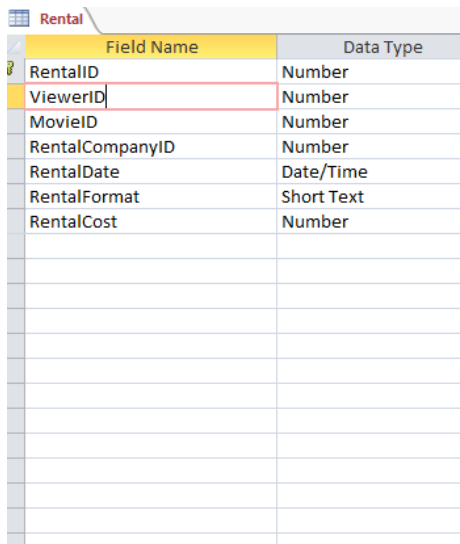
```

Here are the results:

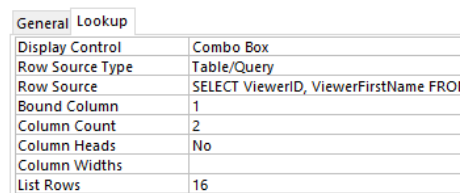


RentalFormat	Expr1001
Streaming	5
DVD	2

Based on the results of the query, the prototype seems to be working as desired. Moreover, by using Access's ability to define fields as "lookups", entering values for the foreign key fields is easier than it otherwise would be. Lookup fields appear as dropdown lists in Access's Datasheet view. They made entering the data for the foreign key fields, all of which link to surrogate keys, a lot easier and less error-prone. Such lookup fields were added to the Rental table, which has foreign keys to the other tables and thus could benefit from this lookup feature. An example of defining the ViewerID field in the Rental table as a lookup field is shown here:



Field Name	Data Type
RentalID	Number
ViewerID	Number
MovieID	Number
RentalCompanyID	Number
RentalDate	Date/Time
RentalFormat	Short Text
RentalCost	Number



Property	Value
Display Control	Combo Box
Row Source Type	Table/Query
Row Source	SELECT ViewerID, ViewerFirstName FROM Viewer
Bound Column	1
Column Count	2
Column Heads	No
Column Widths	
List Rows	16

Note again that the database has been submitted along with this document. In addition, here is the file:



cinemate_redone.accdb

V. Summary: Results and Recommendations

A. The DB Prototype

The DB Prototype worked very well. By normalizing the data into four different related tables, the design minimizes redundancy and the insert, update, and delete anomalies that result when data is repeated. Rather than having to update data in several places when a single value changes, the database designer needs to change data in one place, and all parts of the database that refer to that one data entry can continue to refer to it, but now to the updated value. It was difficult to enter the data into foreign key fields of the Rental table, but this was made easier by defining some fields as Lookup fields in Access. The queries used to test the prototype – all of which were fairly complicated in that they joined multiple tables together – suggest that the prototype works well.

B. Issues with the model

The first version of this project employed a much more extensive data model. It contained thirteen tables, all of which were fully normalized, and it stored particulars about the directors, actors, composers, and production companies associated with each movie. That was the original vision for this project, and it would be good to return to it so that the database can serve as a source of interesting movie trivia in addition to keeping track of rental purchase.

The model currently doesn't allow two or more people to indicate that they, together, rented a movie. The relationship between Viewer and Rental is one-to-many. To enable multiple users to make the same rental would require converting this relationship to a many-to-many, which would require introducing an association.

The fully normalized nature of the model seems to have mitigated all update, delete, and insert anomalies, and that is good news.

C. Next Steps

The next step is to restore the original thirteen-table version of this project. The initial vision for this project was for the database to serve as a treasure trove of movie information. This far more limited version lacks that level of detail, so restoring the previous version is first on the list of future steps.

Enabling the model to accommodate multiple users participating on a single rental is another feature that would be good to add. This was cited as a shortcoming in the preceding section.

It may be helpful to expand the database to store movie fan *clubs* rather than just individual movie fans. Each fan could belong to one or more clubs, and each club would have one or more members. This many-to-many relationship would require an association to implement, but it would allow the database to store the viewing habits of multiple groups of people so that they may be compared and – perhaps – appealed to through marketing efforts.

Although using lookup fields made data entry easier, it could be made a lot easier by using Access Forms instead. A form would provide a visual way to add and edit data. If the database is

used to hold data about many, many movies and rentals, creating and using forms would provide great benefit.

Moving the database to another platform that is commonly used for web applications would make it possible to use the data with a website. Such a website could be used to add new movies to the collection and edit existing ones. It could also be used as a place where movie fans could share information. Access isn't usually used as a backend for websites, but other databases that also speak SQL are. To make this tool accessible beyond Access and on the web, shifting to MySQL or some other database platform would be a wise step.