Home Theatre Video

Project definition, scope

Home Theater Video, a small video rental store, is faced with the challenge of tracking the rental/sales of DVDs and movie information requested by their customers. Management has realized that the current method of using spreadsheets will no longer be feasible to address their needs of maintaining data and searching for information. Home Theater Video has decided to enter the age of maintaining information in a database to provide adequate support to their customers when searching for a movie/actor, managing customer rental transactions, and support of their back-end processes.

Functional Requirement

HTV customers frequently ask for movies starring specific actors. So they would like to keep track of the leading actors appearing in each movie. Not all of our movies have leading actors, documentaries for instance. Customers like to know each actor's "real" birth name and date of birth. We track only actors who appear in the movies in our inventory. The Screen Actors Guild does not permit two actors to have the same studio name and the studio name never contains a middle name or initial.

We would like to search for an actor by their full stage name. There is also no need to have separate fields for their first, middle, and last real names. We are currently using a spreadsheet to keep track of our videos.

There are several thousand customers. We only rent videos to people who have joined our "video club." To belong to our club, they must have good credit. For each club member, we'd like to keep their first and last name, current phone number, and current complete address. Each club member has a membership number; the customer list is currently managed with a spreadsheet.

HTV needs to track the videos each customer currently has checked out, as well as a complete history of the videos each customer has rented. They also would like to track video rental revenue, and late fees collected, - this is to be maintained in a separate details table.

A customer may check out multiple videos at any given time. In addition, Home Theater Video would like to track customer payments to maintain the video rental sales. Also, the customers have different payment options. Each transaction may be paid by Credit Card, Cash, or Check.

For each employee, HTV would like to maintain their first and last name, email address, and home phone, we not only track the employee information but also their video sales.

Users

The HTV Owner, Employees would be the users, every transaction is feed into the system by the employee currently on duty and the Owner or the Employees can see the reports from the system for sales, dues and other collection related queries of the customer.

Database Requirements

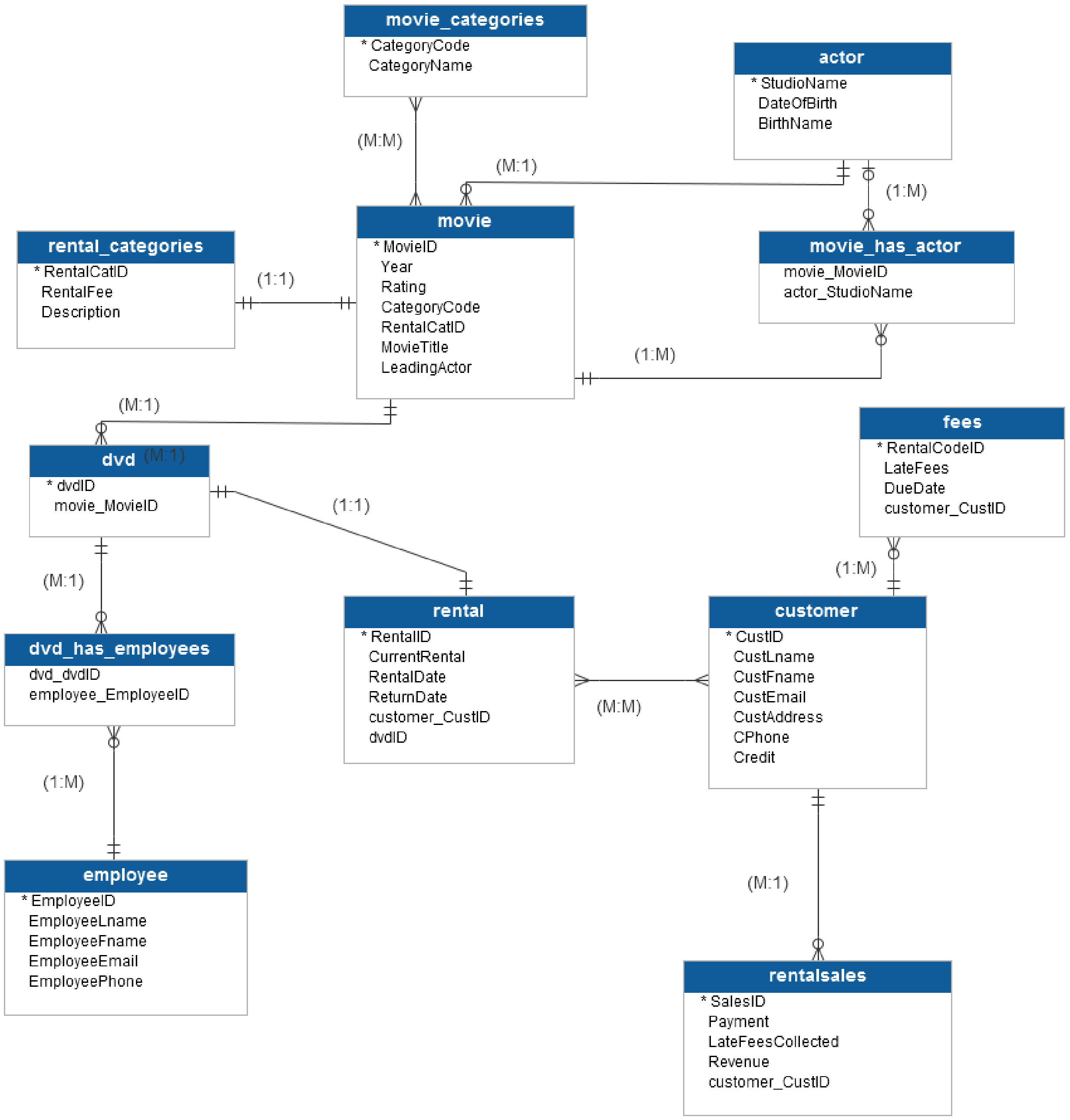
HTV has over 3,000 DVDs that they need to track. Each of the DVDs is assigned a unique number. For each movie, we need to know its title and category (e.g. comedy, suspense, drama, sci-fi). Below is the list of movie categories we use (Table 1). HTV would like to assign each movie category a unique code, which is included in the list.

HTV gives each movie title a specific id and tracks which movie is recorded on each DVD. HTV also collects the year and rating of each movie. Videos are only stocked in DVD format. We always stock at least one DVD for each movie we carry, but for many of our more popular movies we stock multiple DVDs of the same movie. Each DVD contains a single specific movie.

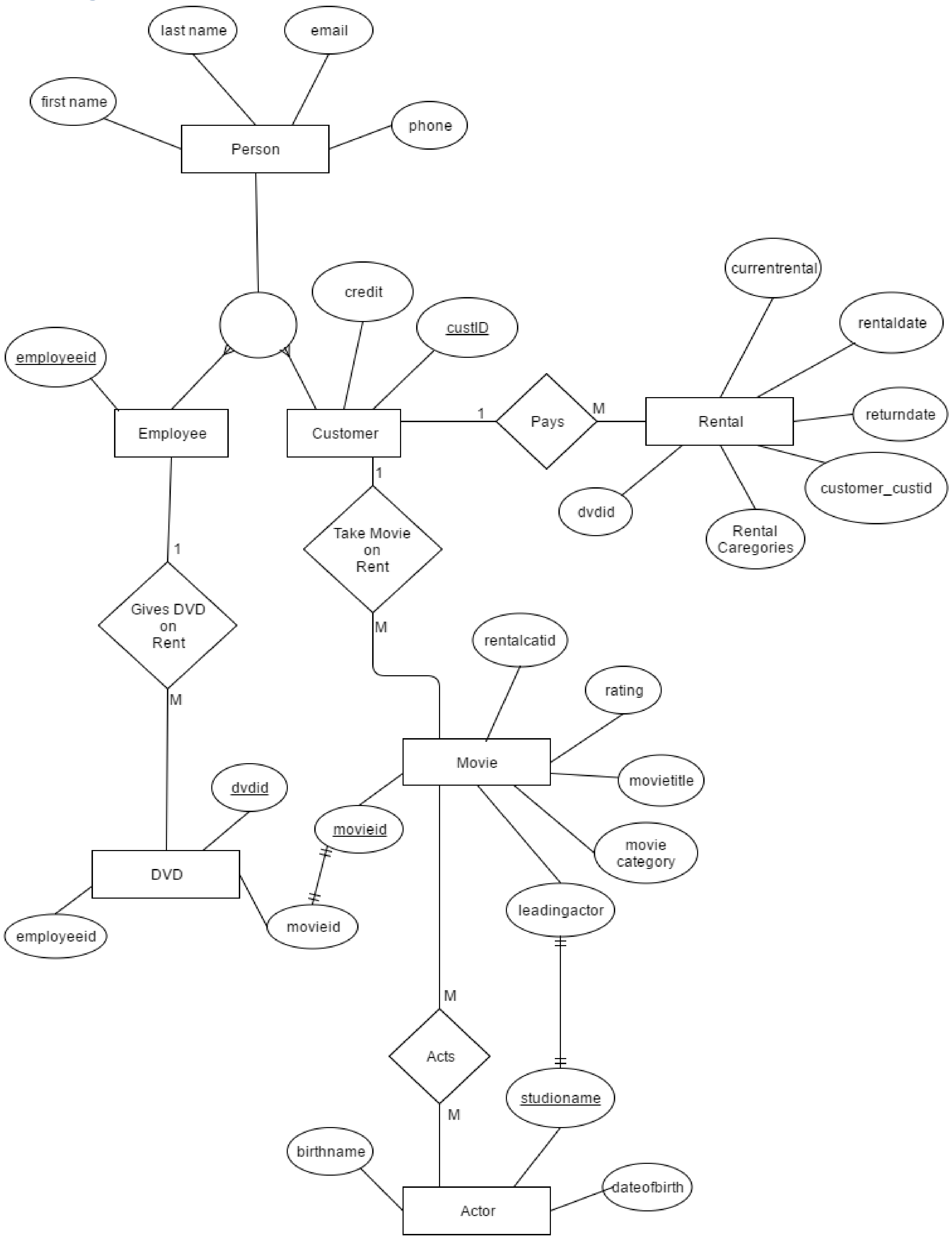
We don't have any movies that require multiple DVDs. The rental fees for our videos vary by the rental code we assign them. The movies are rented for a 5-day period. A late fee of $1.00 accrues for each day a video is returned after the due date.

ER, EER and 3NF Diagram

ER Diagram

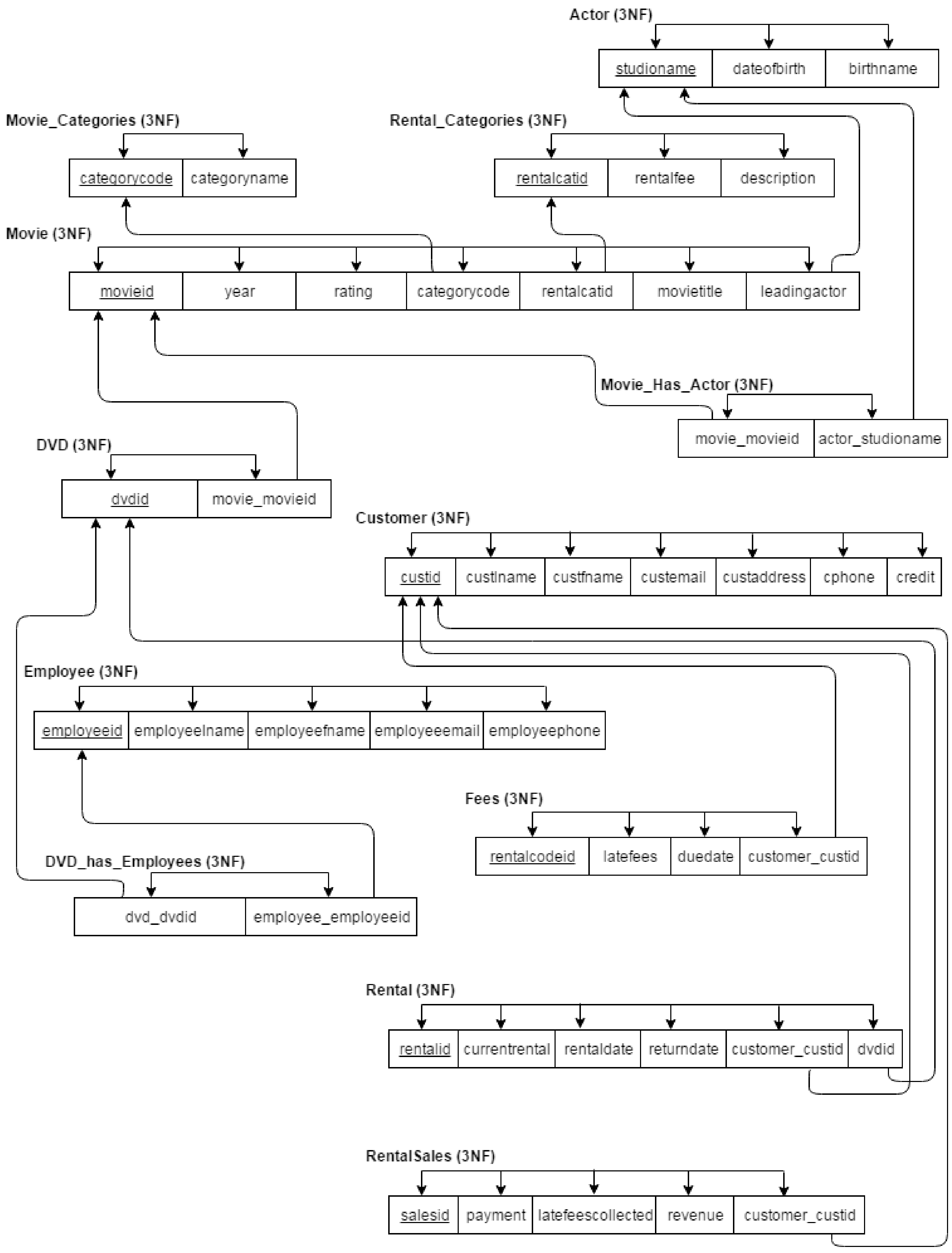


EER Diagram



last name
email
first name
phone
Person
currentrental
credit
custID
rentaldate
employeeid
Employee
Customer
1
Pays
M
Rental
returndate
1
customer_custid
dvdid
Rental Caregories
1
Take Movie on Rent
Gives DVD on Rent
M
rentalcatid
M
rating
Movie
movietitle
dvdid
movieid
movie category
DVD
leadingactor
employeeid
movieid
M
Acts
M
studioname
birthname
dateofbirth
Actor

3NF Diagram


Actor (3NF)
studioname
dateofbirth
birthname
Movie_Categories (3NF)
categorycode
categoryname
Rental_Categories (3NF)
rentalcatid
rentalfee
description
Movie (3NF)
movieid
year
rating
categorycode
rentalcatid
movietitle
leadingactor
Movie_Has_Actor (3NF)
movie_movieid
actor_studioname
DVD (3NF)
dvdid
movie_movieid
Customer (3NF)
custid
custlname
custfname
custemail
custaddress
cphone
credit
Employee (3NF)
employeeid
employeelname
employeefname
employeeemail
employeephone
Fees (3NF)
rentalcodeid
latefees
duedate
customer_custid
DVD_has_Employees (3NF)
dvd_dvdid
employee_employeeid
Rental (3NF)
rentalid
currentrental
rentaldate
returndate
customer_custid
dvdid
RentalSales (3NF)
salesid
payment
latefeescollected
revenue
customer_custid

Database Structure

Tables and their Constraints

1. Actor:

Table contains Actor's info, about their real name, Date of Birth and their studio name, this table is required to fetch the records as per the actor's name, this table act as the master table.

Syntax:

```
CREATE TABLE IF NOT EXISTS actor (
 StudioName VARCHAR (50) NOT NULL,
 DateOfBirth date NULL,
 BirthName VARCHAR (50) NULL,
 PRIMARY KEY (StudioName));
```

Constraints:

PRIMARY KEY (StudioName):

Actor's studioName are unique, as per the guidelines, The Screen Actors Guild does not permit two actors to have the same studio name and the studio name never contains a middle name or initial.

2. Movie Categories:

This table contains the categories of movie that are present in the Home Theatre Video Store. If a customer wants to have the list of all 'War' movies present in Store, then we will use this table for fetching those records.

Syntax:

```
Create Table IF NOT EXISTS movie_categories (
CategoryCode CHAR (3),
CategoryName VARCHAR (100),
Primary Key (CategoryCode));
```

Constraints:

Primary Key (CategoryCode):

Category Code is unique for a category like, War, Fantasy etc. have their unique category code.

3. Rental Categories:

Each movie falls under a category for fee based on the popularity, rating and other factor, this information is stored here.

Syntax:

```
Create Table IF NOT EXISTS rental_categories (
RentalCatID INT,
RentalFee NUMERIC,
Description VARCHAR (100),
Primary Key (RentalCatID));
```

Constraints:

Primary Key (RentalCatID):
Each movie falls in any one of the rental categories that may change as per movie, but still they are unique.

4. Movie:

List of all the movies and their related info available in the Store are present in this table.

Syntax:

```
CREATE TABLE IF NOT EXISTS movie (
  MovieID INT NOT NULL,
  Year INT NULL,
  Rating CHAR (5) NULL,
  CategoryCode CHAR (3),
  RentalCatID INT,
  MovieTitle VARCHAR (80) NULL,
  LeadingActor VARCHAR (50) NOT NULL,
 Primary Key (MovieID),
 Constraint fk_movie_StudioName
   Foreign Key (LeadingActor)
   References actor (StudioName),
 Constraint fk_CategoryCode
   Foreign Key (CategoryCode)
   References movie_categories (CategoryCode),
 Constraint fk_rental_categories
   Foreign Key (RentalCatID)
   References rental_categories (RentalCatID));
```

Constraints:

Primary Key (MovieID):
Each Movie has been assigned a unique ID to identify it, thus it is a primary key.

Constraint fk_movie_StudioName Foreign Key (LeadingActor):
Leading Actor in the Movie should be a part of the Actor Table, hence it has an integrity constraint.

Constraint fk_CategoryCode Foreign Key (CategoryCode):
Category Code is matched from the movie category table and should be available.

Constraint fk_rental_categories Foreign Key (RentalCatID):
RentalCatID is a child of its parent table Rantal Categories, so every RentalID in the child table movie should be there in rental_categories.

5. Customer:

All the register customers in the Store have their information stored in this table, including their email, address, phone number etc.

Syntax:

```
CREATE TABLE IF NOT EXISTS customer (
 CustID INT NOT NULL,
 CustLname VARCHAR (45) NULL,
 CustFname VARCHAR (45) NULL,
 CustEmail VARCHAR (60) NULL,
 CustAddress VARCHAR (75) NULL,
 CPhone VARCHAR (10) NULL,
 Credit Char (4) NULL,
 PRIMARY KEY (CustID));
```

Constraints:

PRIMARY KEY (CustID)):
Customer ID is a unique key to identify each customer in the table.

6. Rental:

The movies that are currently on rent or were rented in past have a record in this table with the Customer id who took it and date and also the return date with particular DVD id as the store have multiple copies of same DVD.

Syntax:

```
CREATE TABLE IF NOT EXISTS rental (
 RentalID INT NOT NULL,
 CurrentRental VARCHAR (45) NULL,
 RentalDate VARCHAR (45) NULL,
 ReturnDate VARCHAR (60) NULL,
 customer_CustID INT NOT NULL,
 dvdID INT,
 Primary Key (RentalID),
Constraint fk_customer_CustID
   Foreign Key (customer_CustID)
   References customer (CustID));
```

Constraints:

Primary Key (RentalID):
RentalCatID is a child of its parent table Rantal Categories, so every RentalID in the child table rental, should be there in rental_categories.

Constraint fk_customer_CustID Foreign Key (customer_CustID):
customer_CustID is a child element of parent table Customer, so every customer who is being rented a movie, should be registered in the customer table.

7. Employee:

The employees and their related information is stored in this table their personal info like phone number and email address are also there.

Syntax:

```
CREATE TABLE IF NOT EXISTS employee (
 EmployeeID INT NOT NULL,
 EmployeeLname VARCHAR (45) NULL,
 EmployeeFname VARCHAR (45) NULL,
 EmployeeEmail VARCHAR (60) NULL,
 EmployeePhone VARCHAR (13) NULL,
 Primary Key (EmployeeID));
```

Constraints:

Primary Key (EmployeeID));
Employee ID ia a unique attribute here to identify every employee in the Store.

8. Movie_has_Actor:

All the lead actors of each movie available in Store are present in this table to have the knowledge of the actors in a specific movie.

Syntax:

```
CREATE TABLE IF NOT EXISTS movie_has_actor (
 movie_MovieID INT NOT NULL,
 actor_StudioName VARCHAR (50),
   Constraint fk_movie_has_actor_MovieID
   foreign key (movie_MovieID)
   References movie (MovieID),
 Constraint fk_movie_has_actor_StudioName
   foreign key (actor_StudioName)
   References actor (StudioName));
```

Constraints:

Constraint fk_movie_has_actor_MovieID Foreign key (movie_MovieID):
movie_MovieID column is a child element of its parent table Movie, it constraints the system from entering anonymous movie names.

Constraint fk_movie_has_actor_StudioName Foreign key (actor_StudioName):
actor_StudioName column is a child element of its parent table Actor, it constraints the system from entering anonymous Actor names.

9. DVD:

Store has multiple copies of same movie having different DVD ids, these details are stored in this table.

Syntax:

```
CREATE TABLE IF NOT EXISTS dvd (
 dvdID INT NOT NULL,
 movie_MovieID INT NOT NULL,
 PRIMARY KEY (dvdID),
 CONSTRAINT fk_dvd_MovieID
   FOREIGN KEY (movie_MovieID)
   REFERENCES movie (MovieID));
```

Constraints:

PRIMARY KEY (dvdID):
DVD is a unique identity in itself as an object which rented to customer, so it is uniquely identify using dvdID.

CONSTRAINT fk_dvd_MovieID FOREIGN KEY (movie_MovieID):
DVD is related to any one of the movie in the movie table, hence this referential integrity checks for any anonymous entry in the table.

10. DVD_has_employees:

DVD are rented to customers by employees and this information is saved to get the sales records by each employee in the store.

Syntax:

```
CREATE TABLE IF NOT EXISTS dvd_has_employees (
 dvd_dvdID INT NOT NULL,
 employee_EmployeeID INT NOT NULL,
   Constraint fk_dvd_has_employees_dvdID
   foreign key (dvd_dvdID)
   References dvd (dvdID),
 Constraint fk_dvd_has_employees_EmployeeID
   foreign key (employee_EmployeeID)
   References employee (EmployeeID));
```

Constraints:

Constraint fk_dvd_has_employees_dvdID FOREIGN KEY (dvd_dvdID):

Constraint fk_dvd_has_employees_EmployeeID FOREIGN KEY (employee_EmployeeID):

11. Fees:

Calculation of fees collected from the customer against every sales is stored in this table for customer wise audit, means to have the records for revenue generated from customer.

Syntax:

```
CREATE TABLE IF NOT EXISTS fees (
 RentalCodeID INT NOT NULL,
 LateFees DECIMAL (4,2) NULL,
 DueDate DATE NULL,
 customer_CustID INT NOT NULL,
 Primary Key (RentalCodeID),
 CONSTRAINT fk_fees_CustID
   FOREIGN KEY (customer_CustID)
   REFERENCES customer (CustID));
```

Constraints:

Primary Key (RentalCodeID),

Constraint fk_fees_CustID FOREIGN KEY (customer_CustID):

13. Rentalsales:

This table contains the payment collected from the customer against the sales(rental) including late fees and payment details.

Syntax:

```
CREATE TABLE IF NOT EXISTS rentalsales (
 SalesID INT NOT NULL,
 Payment DECIMAL (5,2) NULL,
 LateFeesCollected DECIMAL (4,2) NULL,
 Revenue DECIMAL (5,2) NULL,
 customer_CustID INT NOT NULL,
 primary key (SalesID),
 Constraint fk_rentalsales_CustID
   foreign key (customer_CustID)
   references customer (CustID));
```

Constraints:

Primary Key (SalesID):

Constraint fk_rentalsales_CustID Foreign Key (customer_CustID):

Queries Description

As HTV’s data collection grows in size, it is important for management and staff to extract or filter information to answer questions. HTV will require a variety of queries to extract information from the database. We have determined that the initial set of queries is to include the following:

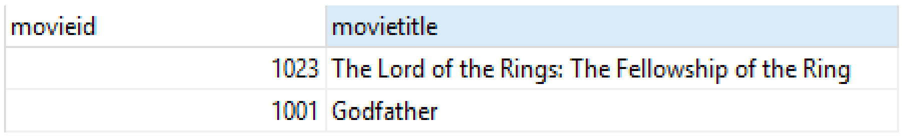
1. Movies Before 2004:

Movies released before 2004 does Home Theater Video have with Movie Id and Title.

Syntax:

```
select MovieID, MovieTitle from movie where YEAR < ‘2004’;
```

Output:



movieid	movietitle
1023	The Lord of the Rings: The Fellowship of the Ring
1001	Godfather

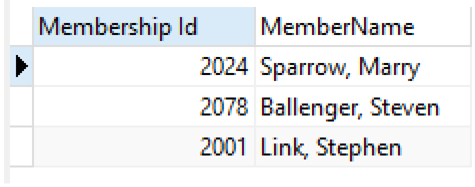
2. Membership List:

A membership list that displays the member's membership number, last name and first name separated by a comma and a space.

Syntax:

```
select CustID as "Membership Id", CustLname||', '||CustFname as "MemberName"
FROM Customer;
```

Output:



Membership Id	MemberName
2024	Sparrow, Marry
2078	Ballenger, Steven
2001	Link, Stephen

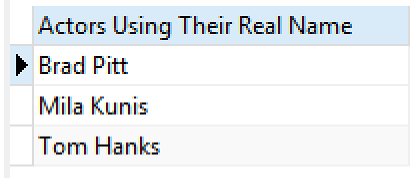
3. Actors Using Real Name:

List of the actors whose studio name is the same as their real name.

Syntax:

```
select StudioName as "Actors Using Their Real Name"
from actor
where StudioName = BirthName
order by StudioName;
```

Output:



Actors Using Their Real Name
Brad Pitt
Mila Kunis
Tom Hanks

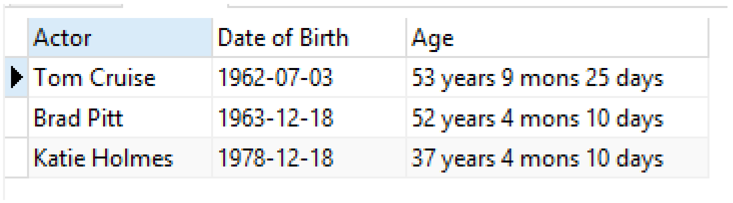
4. Actors Age:

A list of actors that displays their studio name, date of birth and age. Sort by age with the oldest appearing first on the list.

Syntax:

select StudioName as "Actor", DateOfBirth as "Date of Birth",
Age(DateOfBirth) as "Age"
from actor
order by "Age" DESC;

Output:



Actor	Date of Birth	Age
Tom Cruise	1962-07-03	53 years 9 mons 25 days
Brad Pitt	1963-12-18	52 years 4 mons 10 days
Katie Holmes	1978-12-18	37 years 4 mons 10 days

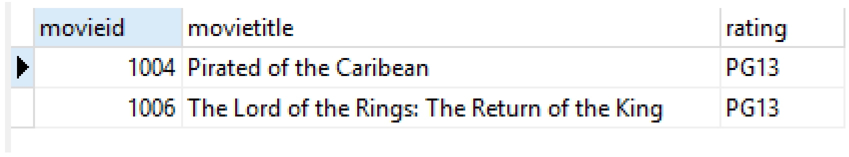
5. Fantasy DVDs:

List of Fantasy DVDs Home Theater Video have.

Syntax:

select MovieID,MovieTitle,Rating from movie
INNER JOIN movie_categories ON movie.CategoryCode = movie_categories.CategoryCode
where movie_categories.CategoryName = 'Fantasy';

Output:



movieid	movietitle	rating
1004	Pirated of the Caribean	PG13
1006	The Lord of the Rings: The Return of the King	PG13

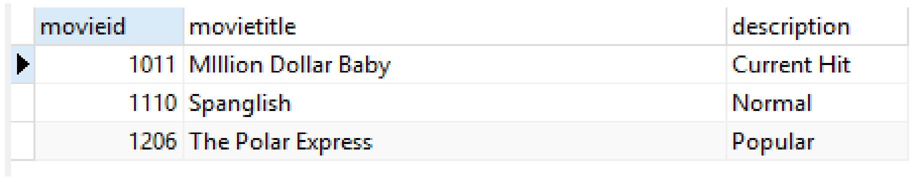
6. Movie Rental Categories:

A listing of Home Theater Video's movies and their respective rental category sorted by Rental Code.

Syntax:

select MovieID,MovieTitle,Description from movie
INNER JOIN rental_categories ON movie.RentalCatID = rental_categories.RentalCatID
ORDER BY rental_categories.RentalCatID;

Output:



movieid	movietitle	description
1011	MIllion Dollar Baby	Current Hit
1110	Spanglish	Normal
1206	The Polar Express	Popular

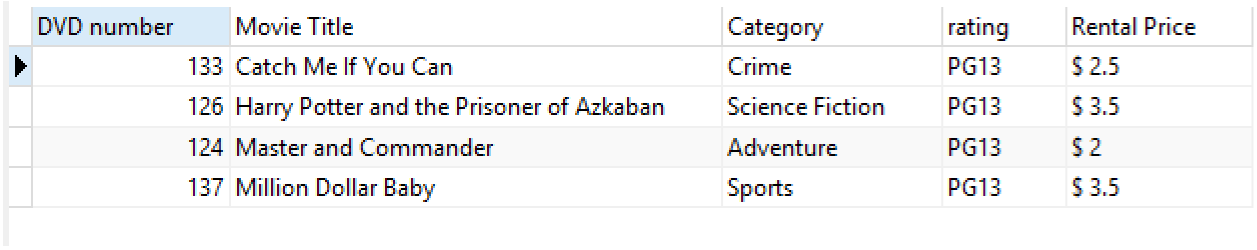
7. Price List:

A listing of individual DVD prices for Home Theater Video. Display DVD Number, Movie Title, Movie Category, Rating, and Rental Price.

Syntax:

```
select dvdID as "DVD number”, MovieTitle as "Movie Title",
CategoryName as "Category",Rating,'$ ' ||RentalFee as "Rental Price"
from movie
INNER JOIN dvd ON movie.MovieID = dvd.movie_MovieID
INNER JOIN rental_categories ON movie.RentalCatID = rental_categories.RentalCatID
INNER JOIN movie_Categories ON movie.CategoryCode = movie_categories.CategoryCode
ORDER BY "Movie Title","Category";
```

Output:



DVD number	Movie Title	Category	rating	Rental Price
133	Catch Me If You Can	Crime	PG13	$ 2.5
126	Harry Potter and the Prisoner of Azkaban	Science Fiction	PG13	$ 3.5
124	Master and Commander	Adventure	PG13	$ 2
137	Million Dollar Baby	Sports	PG13	$ 3.5

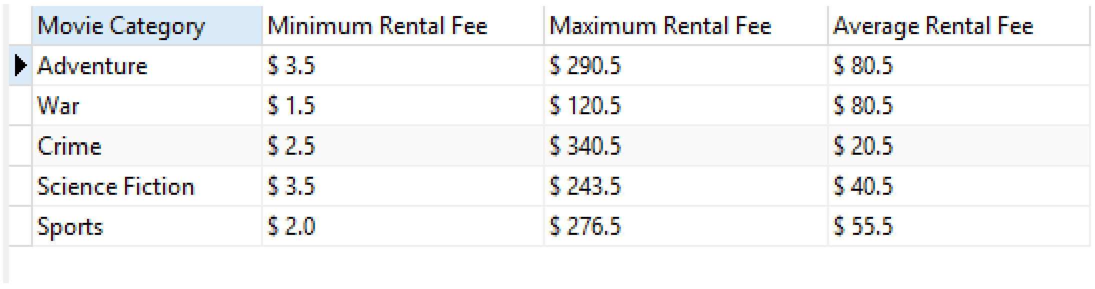
8. Rental Fee Stats:

A query to display the minimum, maximum, and average rental fee for each movie category.

Syntax:

```
select CategoryName as "Movie Category",
'$ ' || MIN(RentalFee) as "Minimum Rental Fee",
'$ ' || MAX(RentalFee) as "Maximum Rental Fee",
'$ ' || AVG(RentalFee) as "Average Rental Fee"
from movie
INNER JOIN rental_categories ON movie.RentalCatID = rental_categories.RentalCatID
INNER JOIN movie_categories ON movie.CategoryCode = movie_categories.CategoryCode
GROUP BY "Movie Category";
```

Output:



Movie Category	Minimum Rental Fee	Maximum Rental Fee	Average Rental Fee
Adventure	$ 3.5	$ 290.5	$ 80.5
War	$ 1.5	$ 120.5	$ 80.5
Crime	$ 2.5	$ 340.5	$ 20.5
Science Fiction	$ 3.5	$ 243.5	$ 40.5
Sports	$ 2.0	$ 276.5	$ 55.5

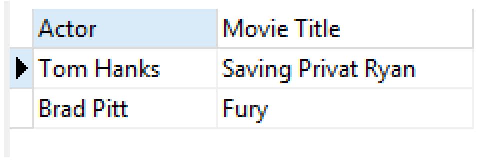
9. Actors in War Movies:

A listing to display the actors that star in war movies. Displays Actor's Stage Name and Movie Title. Sorted by Actor.

Syntax:

```
select actor_StudioName as "Actor",
MovieTitle as "Movie Title"
FROM movie
INNER JOIN movie_categories ON movie.CategoryCode = movie_categories.CategoryCode
INNER JOIN movie_has_actor ON movie.MovieID = movie_has_actor.movie_MovieID
Where movie_categories.CategoryName = 'War'
ORDER BY "Actor";
```

Output:



Actor	Movie Title
Tom Hanks	Saving Privat Ryan
Brad Pitt	Fury

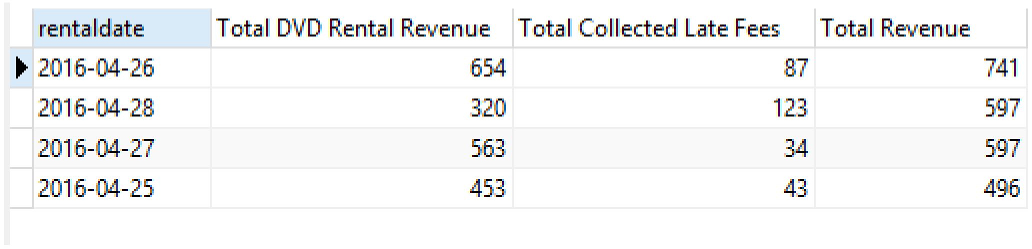
10. Daily Revenue:

A query to display the daily revenue by day. Display Rental Date, Total DVD Rental Revenue, Total Collected Late Fees, and Total Revenue. Total Revenue is the sum of DVD Rental Revenue and Collected Late Fees. Collected Late Fees need to be associated with the Returned Date. Sorted by rental date.

Syntax:

```
select rental.RentalDate ,
SUM(rentalsales.Revenue) as "Total DVD Rental Revenue",
SUM(rentalsales.LateFeesCollected) as "Total Collected Late Fees",
SUM(rentalsales.Revenue)+SUM(rentalsales.LateFeesCollected) as "Total Revenue"
from rental
INNER JOIN rentalsales ON rentalsales.SalesID = rental.RentalID
GROUP BY rental.RentalDate
ORDER BY rental.RentalDate;
```

Output:



rentaldate	Total DVD Rental Revenue	Total Collected Late Fees	Total Revenue
2016-04-26	654	87	741
2016-04-28	320	123	597
2016-04-27	563	34	597
2016-04-25	453	43	496

Stored Procedure and Triggers

Stored Procedure

Print Receipt for Rented DVDs

```
CREATE OR REPLACE Procedure Print_recp_rented_dvd(a_rental_id IN int)
IS
v_RentalID rental.RentalID%type;
v_RentalDate rental.RentalDate%type;
v_customer_CustID rental.customer_CustID%type;
v_customer_name varchar (100);
v_dvdID rental.dvdID%type;
v_MovieTitle movie.MovieTitle%type;
v_CurrentRental rental.CurrentRental%type;
BEGIN

select
r.RentalID,
r.RentalDate,
r.customer_CustID,
(select c.CustLname||', '||c.CustFname from customer c where c.CustID = r.customer_CustID)
customer_name,
r.dvdID,
r.CurrentRental
into
v_RentalID,
v_RentalDate,
v_customer_CustID,
v_customer_name,
v_dvdID,
v_MovieTitle,
v_CurrentRental
from rental r
inner join dvd d on r.dvdID = d.dvdID
inner join movie m on m.MovieID = d.movie_MovieID
where rental_id = a_rental_id;

DBMS_OUTPUT.PUT_LINE('--------------Rental Receipt------------------');
DBMS_OUTPUT.PUT_LINE('Rental ID -'|| v_RentalID);
DBMS_OUTPUT.PUT_LINE('Date -'|| v_RentalDate);
DBMS_OUTPUT.PUT_LINE('Customer ID-'|| v_customer_CustID);
DBMS_OUTPUT.PUT_LINE('Customer Name-'|| v_customer_name);
DBMS_OUTPUT.PUT_LINE('DVD ID-'|| v_dvdID);
DBMS_OUTPUT.PUT_LINE('Movie Title -'|| v_MovieTitle);
DBMS_OUTPUT.PUT_LINE('Rental Charges -'|| v_ CurrentRental);
DBMS_OUTPUT.PUT_LINE('--------------------------------------------------');

END Print_recp_rented_dvd;
/
```

Print Report for Daily Revenue

```
CREATE OR REPLACE Procedure Print_daily_revenue()
IS
cnumber number;
cursor c1 is
select rental.RentalDate ,
SUM(rentalsales.Revenue) as tot_dev_rev ,
SUM(rentalsales.LateFeesCollected) as tot_coll_late_fee ,
SUM(rentalsales.Revenue)+SUM(rentalsales.LateFeesCollected) as tot_rev
from rental
INNER JOIN rentalsales ON rentalsales.SalesID = rental.RentalID
where rental.RentalDate = a_RentalDate
GROUP BY rental.RentalDate;
BEGIN
DBMS_OUTPUT.PUT_LINE('Date	|	Total DVD Rental Revenue	|	Total Collected Late
Fees	|	Total Revenue');
For i in c1
loop
DBMS_OUTPUT.PUT_LINE(i.RentalDate||'	|	'||i.tot_dev_rev||'	|	'||i.tot_coll_late_fee||'
	|	'||i.tot_rev);
end loop;
END Print_daily_revenue;
/
```

Triggers & Check

Pre-Rented Check

If a Customer has already rented the same movie with another dvdID , then he can't get the same movie again.

```
CREATE OR REPLACE TRIGGER pre_rented_check()
BEFORE INSERT
ON rental
DECLARE
v_cnt number(10);
BEGIN

select count('x') into v_cnt
from customer_has_movie where customer_CustID = :NEW.customer_CustID and
dvdID = :NEW.dvdID;

IF v_cnt > 0 THEN
RAISE_APPLICATION_ERROR("This Movie is Already Rented by the Customer");
END IF;

END pre_rented_check;
/
```

Update customer's records after receiving DVD

```
CREATE OR REPLACE TRIGGER upd_dvd_received()
AFTER UPDATE
ON rental
DECLARE
movie_MovieID dvd.movie_MovieID%type;
v_cnt number(10);
BEGIN
select movie_MovieID into v_movie_MovieID from dvd
where dvdID = :NEW.dvdID;

select count('x') into v_cnt
from customer_has_movie where movie_MovieID = v_movie_MovieID;

IF :NEW.ReturnDate = sysdate THEN
Delete from from customer_has_movie where movie_MovieID = v_movie_MovieID;
END IF;

END upd_dvd_received;
/
```