
HANNAHFLIX

Table Of Contents:

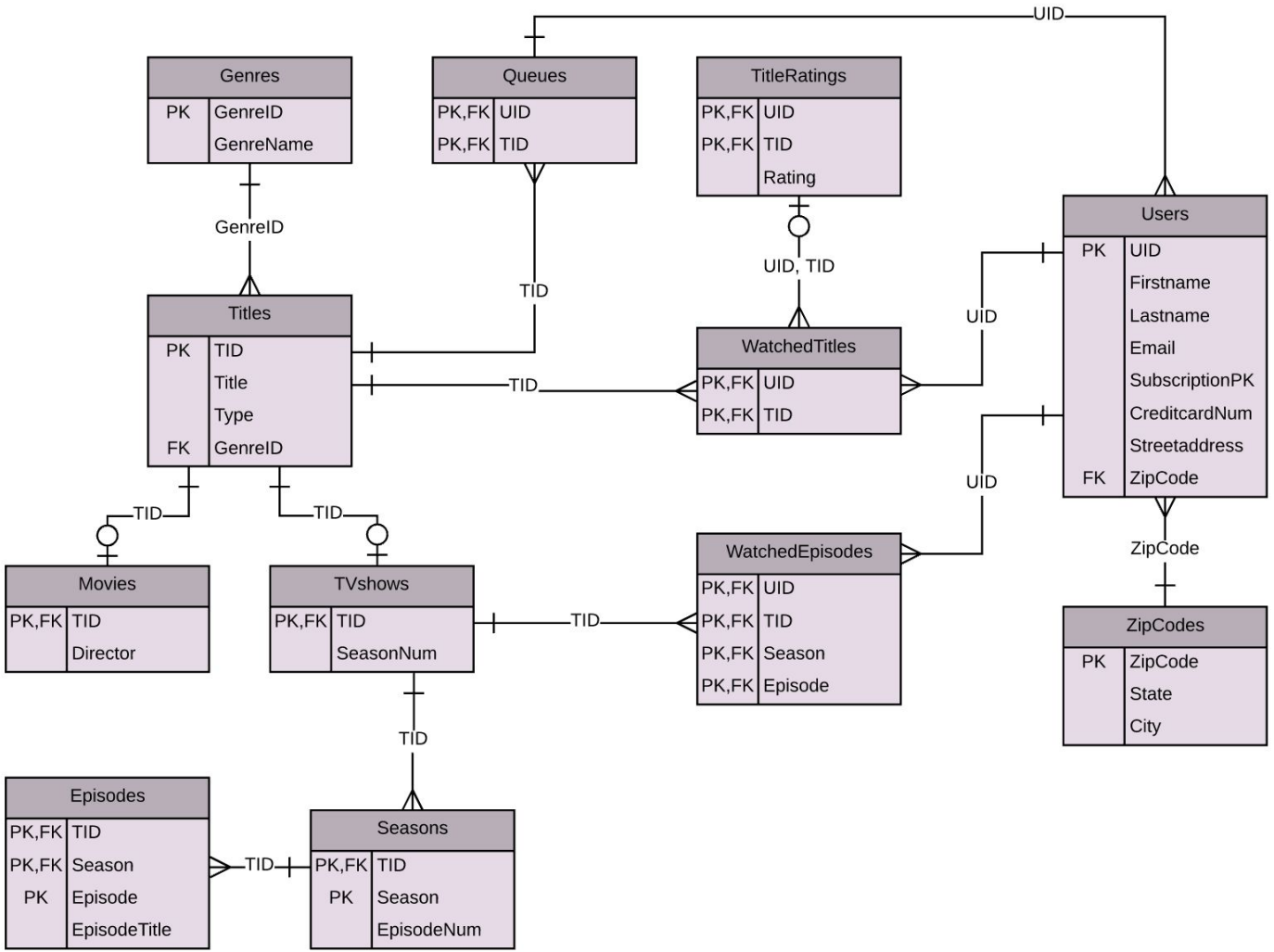
Executive Summary.....3
Entity Relationship Diagram.....4
Table Statements.....5
View Statements.....18
Reports.....22
Stored Procedures.....26
Triggers.....31
Security.....34
Implementation Notes and Known Problems.....37
Future Enhancements.....38

Executive Summary

HannahFlix is a new high quality streaming service with more titles than ever before. This service prides itself with high quality movies that are hand picked by film critics from around the world, while also providing the latest and greatest TV shows for their customers to enjoy.

This database will be used to keep track of films, tv shows and users. HannahFlix administrators will be able to see who is watching what and better tailor every user's experience based on what films and shows they watch the most.

Entity Relationship Diagram



Tables

Tables: Users

This table is used to keep track of Users of the HannahFlix system and what their information is for billing purposes.

```
CREATE TABLE Users (
  uid int not null,
  firstname text not null,
  lastname text not null,
  email text not null,
  subscriptionPK text not null,
  creditcardNum bigint not null,
  streetaddress text not null,
  zipcode int not null references ZipCodes(zipcode),
  unique(uid),
  PRIMARY KEY (uid)
);
```

Functional Dependencies:
uid→ firstname,lastname, email,subscriptionPK, creditcardNum,streetaddress, zipcode.

uid integer	firstname text	lastname text	email text	subscriptionpk text	creditcardnum bigint	streetaddress text	zipcode integer
1	Hannah	Riedman	Hannah.riedman@marist.edu	Premium	5105105105105100	2630 East St	14424
2	Alan	Labouseur	alan@labouseur.com	Basic	4012888888881881	3399 North Rd	12601
3	Phobe	Robinson	PRobinson@gmail.com	Basic	5105675895105100	600 45th St	11201
4	Charlie	Donner	Charliedon@yahoo.com	Basic	4012889988384881	17 5th St	10065

Tables: Titles

tid integer	title text	type text	genreid integer
1	The Walking Dead	TVshow	9
2	Cosmos	TVShow	12
3	Arrested Development	TVshow	7
4	The Office	TVShow	7
5	Parks and Rec	TVShow	7
6	Black Mirror	TVShow	1
7	Rick and Morty	TVShow	7
8	Mr.Robot	TVShow	8
9	Breaking Bad	TVShow	9
10	Do The Right Thing	Movie	9
11	City of God	Movie	8
12	Back to the Future	Movie	5
13	Monty Python and the Holy Grail	Movie	7
14	North by Northwest	Movie	1
15	Strangers on a Train	Movie	1
16	Kill Bill Vol. 1	Movie	6
17	Kill Bill Vol. 2	Movie	6
18	Forrest Gump	Movie	9
19	Edward Scissorhands	Movie	2
20	American Psycho	Movie	1
21	Goldfinger	Movie	6

This table is used to keep track of the titles currently available. Each Title has a name and a type (TV show or Movie) and each title is assigned a genreID from the genre table.

```
CREATE TABLE Titles (
tid int not null,
title text not null,
type text not null,
genreID int not null references Genres(genreID),
unique(tid),
PRIMARY KEY (tid)
);
```

Functional Dependencies:
tid→title ,type, genreID

Tables: Genres

genreid integer	genrename text
1	Thriller
2	Romance
3	Epic
4	Mystery
5	Science Fiction
6	Action
7	Comedy
8	Crime
9	Drama
10	Romantic Comedy
11	Fantasy
12	Documentary

This table is used to keep track of what genres there are. More genres can easily be added if a title is added with no appropriate genre.

```
CREATE TABLE Genres (
  genreID int not null,
  genreName text not null,
  PRIMARY KEY (genreID)
);
```

Functional Dependencies:
genreID → genreName

Tables: Movies

This table is used to keep track of all the Movies available. It also holds information for the Directors of Movies

```
CREATE TABLE Movies (
  tid int not null references titles(tid),
  director text not null,
  unique(tid),
  PRIMARY KEY (tid)
);
```

Functional Dependencies:
tid→ director

tid integer	director text
10	Spike Lee
11	Fernando Meirelles and Kátia Lund
12	Robert Zemeckis
13	Terry Gilliam and Terry Jones
14	Alfred Hitchcock
15	Alfred Hitchcock
16	Quentin Tarantino
17	Quentin Tarantino
18	Robert Zemeckis
19	Tim Burton
20	Mary Harron
21	Guy Hamilton

Tables: TV shows

This table is used to keep track of all the TV shows available. It also holds information for the number of seasons.

```
CREATE TABLE TVshows (
  tid int not null references titles(tid),
  seasonNum int not null,
  unique(tid),
  PRIMARY KEY (tid)
);
```

Functional Dependencies:
tid→seasonNum

tid integer	seasonnum integer
1	7
2	1
3	4
4	9
5	7
6	3
7	2
8	2
9	5

Tables: Seasons

This table is used to keep track of all the Seasons available. It also holds information for how many episodes are in each season.

```
CREATE TABLE Seasons (
  tid int not null references TVshows(tid),
  season int not null,
  episodeNum int not null,
  unique(tid,season),
  PRIMARY KEY (tid,season)
);
```

Functional Dependencies:
tid, season→ episodeNum

	tid integer	season integer	episodenum integer
19	4	7	26
20	4	8	24
21	4	9	25
22	5	1	6
23	5	2	24
24	5	3	16
25	5	4	22
26	5	5	22
27	5	6	22
28	5	7	13
29	6	1	3
30	6	2	3
31	6	3	6
32	7	1	11
33	7	2	10
34	8	1	10
35	8	2	12
36	9	1	7
37	9	2	13
38	9	3	13
39	9	4	13
40	9	5	16

	tid integer	season integer	episodenum integer
1	1	1	6
2	1	2	13
3	1	3	16
4	1	4	16
5	1	5	16
6	1	6	16
7	1	7	16
8	2	1	13
9	3	1	22
10	3	2	18
11	3	3	13
12	3	4	15
13	4	1	6
14	4	2	22
15	4	3	25
16	4	4	19
17	4	5	28
18	4	6	26

Tables: Episodes

This table is used to keep track of all the Episodes available and their respective season number, TV show and episode title.

	tid integer	season integer	episode integer	episodetitle text
1	1	1	1	Days Gone Bye
2	1	2	1	What Lies Ahead
3	1	3	1	Seed
4	1	4	1	30 days without an accident
5	1	5	1	No Sanctuary
6	1	6	1	First Time Again
7	1	7	1	The Day will come when you wont be
8	2	1	1	The Shores of the Cosmic Ocean
9	2	1	13	Who Speaks for Earth?
10	3	1	1	Pilot
11	3	2	1	The One Where Michael Leaves
12	3	3	1	The Cabin Show
13	3	4	1	Flight of the Phoenix
14	4	1	1	Pilot
15	4	2	1	The Dundies
16	4	3	1	Gay Witch Hunt
17	4	4	1	Fun Run
18	4	5	1	Weight Loss
19	4	6	1	Gossip

```
CREATE TABLE Episodes (
  tid int not null,
  season int not null,
  episode int not null,
  episodeTitle text not null,
  FOREIGN KEY(tid, season) REFERENCES Seasons(tid, season),
  PRIMARY KEY (tid, season, episode)
);
```

Functional Dependencies:

tid, season, episode → episodeTitle

Tables: Episodes-- more sample data

	tid integer	season integer	episode integer	episodetitle text
20	4	7	1	Nepotism
21	4	8	1	The List
22	4	9	1	New Guys
23	5	1	1	Pilot
24	5	2	1	Pawnee Zoo
25	5	3	1	Go Big or Go Home
26	5	4	1	Im Leslie Knope
27	5	5	1	Ms.Knope Goes to Washington
28	5	6	1	London
29	5	7	1	2017
30	6	1	1	The National Anthem
31	6	2	1	Be Right Back
32	6	3	1	NoseDive
33	6	3	6	Hated in a Nation
34	7	1	1	Pilot
35	7	2	1	A Rickle in Time
36	7	2	5	Get Schwifty
37	8	1	1	eps1.0_hellofriend.mov
38	8	2	1	eps2.0_unm4sk-pt1.tc
39	9	1	1	Pilot
40	9	2	1	Seven Thirty-Seven
41	9	3	1	No Más
42	9	4	1	Box Cutter
43	9	5	1	Live Free or Die

Tables: WatchedTitles

This table is used to keep track of all the Titles watched by each certain user. This includes movies and TV show titles.

```
CREATE TABLE WatchedTitles (
  uid int not null references Users(uid),
  tid int not null references Titles(tid),
  unique(uid, tid),
  PRIMARY KEY(uid, tid)
);
```

Functional Dependencies:

uid, tid →

	uid integer	tid integer
1	1	1
2	1	3
3	1	4
4	1	5
5	1	10
6	1	11
7	1	14
8	2	2
9	2	6
10	2	12
11	2	21
12	3	5
13	3	7
14	3	18
15	4	6
16	4	20
17	4	18

Tables: WatchedEpisodes

This table is used to keep track of all the episodes watched by each user. This is so HannahFlix can keep track if you have finished a show or what episode you are on in the series.

	uid integer	tid integer	season integer	episode integer
1	1	1	1	1
2	1	1	2	1
3	1	1	3	1
4	1	3	3	1
5	1	4	4	1
6	1	5	4	1
7	2	2	1	1
8	2	2	1	13
9	2	6	1	1
10	2	6	3	6

```
CREATE TABLE WatchedEpisodes (
  uid int not null references Users(uid),
  tid int not null,
  season int not null,
  episode int not null,
  FOREIGN KEY(tid, season, episode) REFERENCES Episodes(tid, season, episode),
  PRIMARY KEY (uid, tid, season, episode)
);
```

Functional Dependencies:

uid, tid, season, episode →

Tables: TitleRatings

This table is used to keep track of all the titles rated by the user on a 1 to 5 star rating scale. The user can rate movies or a TV show as a whole but only if the user has watched the title (and is therefore in the WatchedTitles table).

```
CREATE TABLE TitleRatings (
  uid int not null,
  tid int not null,
  rating int not null check (rating > 0 and rating < 6),
  FOREIGN KEY(uid, tid) REFERENCES WatchedTitles(uid, tid),
  PRIMARY KEY(uid, tid)
);
```

Functional Dependencies:
uid, tid → rating

	uid integer	tid integer	rating integer
1	1	3	5
2	1	10	5
3	1	11	5
4	2	21	4
5	3	5	4
6	3	7	5
7	4	6	5
8	4	20	5
9	4	18	2

Tables: Queues

This table is used to keep track of what titles are on a user's list. A user can add any movie or tv show to their Queue to watch later.

```
CREATE TABLE Queues (
  uid int not null references Users(uid),
  tid int not null references Titles(tid),
  unique(uid, tid),
  PRIMARY KEY(uid, tid)
);
```

Functional Dependencies:

uid, tid →

	uid integer	tid integer
1	1	9
2	1	19
3	1	14
4	1	15
5	2	10
6	2	11


Views

Views: view_all_movies

This view is used to see all the available movies with the movie titles and directors.

```
CREATE VIEW view_all_movies
AS
SELECT t.tid, title as "Movie title", director
FROM movies m INNER JOIN titles t ON m.tid = t.tid;
```

	tid integer	Movie title text	director text
1	10	Do The Right Thing	Spike Lee
2	11	City of God	Fernando Meirelles and Kátia Lund
3	12	Back to the Future	Robert Zemeckis
4	13	Monty Python and the Holy Grail	Terry Gilliam and Terry Jones
5	14	North by Northwest	Alfred Hitchcock
6	15	Strangers on a Train	Alfred Hitchcock
7	16	Kill Bill Vol. 1	Quentin Tarantino
8	17	Kill Bill Vol. 2	Quentin Tarantino
9	18	Forrest Gump	Robert Zemeckis
10	19	Edward Scissorhands	Tim Burton
11	20	American Psycho	Mary Harron
12	21	Goldfinger	Guy Hamilton

Views: view_all_tvshow_episodes

This view is used to see all the available tv show episodes with the tvshow title, season number, episode number, and episode titles.

```
CREATE VIEW view_all_tvshow_episodes
AS
SELECT t.tid, title as "TV show title", season, episode, episodetitle
FROM episodes e INNER JOIN titles t ON e.tid = t.tid;
```

	tid integer	TV show title text	season integer	episode integer	episodetitle text
1	1	The Walking Dead	1	1	Days Gone Bye
2	1	The Walking Dead	2	1	What Lies Ahead
3	1	The Walking Dead	3	1	Seed
4	1	The Walking Dead	4	1	30 days without an accident
5	1	The Walking Dead	5	1	No Sanctuary
6	1	The Walking Dead	6	1	First Time Again
7	1	The Walking Dead	7	1	The Day will come when you wont be
8	2	Cosmos	1	1	The Shores of the Cosmic Ocean
9	2	Cosmos	1	13	Who Speaks for Earth?
10	3	Arrested Development	1	1	Pilot
11	3	Arrested Development	2	1	The One Where Michael Leaves
12	3	Arrested Development	3	1	The Cabin Show
13	3	Arrested Development	4	1	Flight of the Phoenix
14	4	The Office	1	1	Pilot
15	4	The Office	2	1	The Dundies
16	4	The Office	3	1	Gay Witch Hunt
17	4	The Office	4	1	Fun Run
18	4	The Office	5	1	Weight Loss
19	4	The Office	6	1	Gossip
20	4	The Office	7	1	Nepotism
21	4	The Office	8	1	The List

	tid integer	TV show title text	season integer	episode integer	episodetitle text
20	4	The Office	7	1	Nepotism
21	4	The Office	8	1	The List
22	4	The Office	9	1	New Guys
23	5	Parks and Rec	1	1	Pilot
24	5	Parks and Rec	2	1	Pawnee Zoo
25	5	Parks and Rec	3	1	Go Big or Go Home
26	5	Parks and Rec	4	1	Im Leslie Knope
27	5	Parks and Rec	5	1	Ms.Knope Goes to Washington
28	5	Parks and Rec	6	1	London
29	5	Parks and Rec	7	1	2017
30	6	Black Mirror	1	1	The National Anthem
31	6	Black Mirror	2	1	Be Right Back
32	6	Black Mirror	3	1	NoseDive
33	6	Black Mirror	3	6	Hated in a Nation
34	7	Rick and Morty	1	1	Pilot
35	7	Rick and Morty	2	1	A Rickle in Time
36	7	Rick and Morty	2	5	Get Schwifty
37	8	Mr.Robot	1	1	eps1.0_hellofriend.mov
38	8	Mr.Robot	2	1	eps2.0_unm4sk-pt1.tc
39	9	Breaking Bad	1	1	Pilot
40	9	Breaking Bad	2	1	Seven Thirty-Seven
41	9	Breaking Bad	3	1	No Más
42	9	Breaking Bad	4	1	Box Cutter

Views: most_popular_titles

This view is used to show administrators how many users are watching each title. This will help our film and tv show curators decide what kind of titles are most desired.

```
CREATE VIEW most_popular_titles
AS
SELECT t.tid, title, type, count(*) as "Users Watched"
FROM titles t INNER JOIN watchedtitles wt ON t.tid = wt.tid
GROUP BY  t.tid
ORDER BY count(*) DESC, t.tid;
```

	tid integer	title text	type text	Users Watched bigint
1	5	Parks and Rec	TVShow	2
2	6	Black Mirror	TVShow	2
3	18	Forrest Gump	Movie	2
4	1	The Walking Dead	TVshow	1
5	2	Cosmos	TVShow	1
6	3	Arrested Development	TVshow	1
7	4	The Office	TVShow	1
8	7	Rick and Morty	TVShow	1
9	10	Do The Right Thing	Movie	1
10	11	City of God	Movie	1
11	12	Back to the Future	Movie	1
12	14	North by Northwest	Movie	1
13	20	American Psycho	Movie	1
14	21	Goldfinger	Movie	1


Reports

Reports: Percent of Users that are Basic

This Report will be used to find the percentage of how many users have a basic subscription package. This is useful to help HannahFlix see what packages are the most popular.

```
SELECT  round(
                ( (count(*) filter (WHERE subscriptionPK = 'Basic')::numeric
                / count(*)::numeric) * 100)::numeric, 2
                ) as  "Percent Users Basic"
FROM users
```

Percent Users Basic numeric
75.00

Reports: Most Popular Genre

This report will be used to find the most popular genre of titles users watch. This will help HannahFlix in finding more titles in that category to add to their library.

```
SELECT genreName as “Most Popular Genre”
FROM genres
WHERE genreID = (SELECT genreID
                 FROM (SELECT genreID,count(*)
                     FROM watchedtitles wt INNER JOIN titles t ON wt.tid = t.tid
                     GROUP BY genreID
                     ORDER BY count(*) DESC
                     LIMIT 1) as "GenreID");
```

Most Popular Genre text
Comedy

Reports: Most Popular Director

This report will be used to find the most popular Director of titles users watch. This will help HannahFlix in finding more titles from that Director to add to their library.

```
SELECT director as "Most Popular Director"
FROM (SELECT director, count(*)
    FROM watchedtitles wt INNER JOIN movies m ON wt.tid = m.tid
    GROUP BY director
    ORDER BY count(*) DESC
    limit 1) as "director";
```

Most Popular Director text
Robert Zemeckis


Stored
Procedures

Stored Procedures: Recommendations

This procedure will look at a user's watched titles and take the most watched genre of titles they have watched and recommended titles from that genre for the user to watch while also ensuring that no titles that the user has watched will be recommended.

Below is sample results for user, **Alan Labouseur** when the user's id is passed into the function. SQL for the procedure is shown on the next page.

Recommended Titles text
North by Northwest
Strangers on a Train
American Psycho

Stored Procedures: Recommendations cont.

```
DROP FUNCTION recommendations(theuser integer,resultSet refcursor);

CREATE or REPLACE function recommendations(theuser int,resultSet refcursor) returns refcursor as
$$
DECLARE
  theuser int := $1;
  resultSet  refcursor := $2;
BEGIN
  open resultset for
    SELECT title
    FROM  titles
    WHERE genreID = (SELECT genreID
                     FROM( SELECT genreID,count(*)
                           FROM watchedtitles wt INNER JOIN titles t ON wt.tid = t.tid
                           WHERE uid = theuser
                           GROUP BY genreID
                           ORDER BY count(*) DESC
                           LIMIT 1) as "genreID" )
    AND title not in (SELECT title
                      FROM watchedtitles wt INNER JOIN titles t ON wt.tid = t.tid
                      WHERE uid = theuser);
  return resultset;
END;
$$ language plpgsql
```

Stored Procedures: View by Genre

This procedure will accept a genre as text and then return all titles in that genre. This will be useful when users search titles by genre. Sample data is shown on the next page.

```
DROP FUNCTION viewbygenre(genre text,resultSet refcursor);

CREATE or REPLACE function viewbygenre(genre text,resultSet refcursor) returns refcursor as
$$
DECLARE
  genre text := $1;
  resultSet  refcursor := $2;
BEGIN
  open resultset for
    SELECT title, type
    FROM titles
    WHERE genreID = (SELECT genreID
                     FROM genres
                     WHERE genreName = genre);
  return resultset;
END;
$$ language plpgsql
```

Stored Procedures: View by Genre cont.

Here is some sample data for this procedure. Figure 1 shows data when the genre ‘Thriller’ is entered into the function. Figure 2 shows data when the genre ‘Drama’ is entered into the function.

Figure 1

	title text	type text
1	Black Mirror	TVShow
2	North by Northwest	Movie
3	Strangers on a Train	Movie
4	American Psycho	Movie

Figure 2

	title text	type text
1	The Walking Dead	TVshow
2	Breaking Bad	TVShow
3	Do The Right Thing	Movie
4	Forrest Gump	Movie


Triggers

Triggers: Credit Card Check

This Trigger will check the credit card entered for each user to ensure it is valid (has 16 digits) before allowing the user data into the Users table. The credicard_num() function will be executed whenever a new row is Inserted into the table.

```
CREATE OR REPLACE FUNCTION creditcard_num() RETURNS TRIGGER AS
$$
DECLARE
      creditcard text := cast(new.creditcardnum as text);
BEGIN
   IF creditcard is NULL THEN
         RAISE EXCEPTION 'creditcard cannot be null';
   END IF;
   IF (SELECT length(creditcard)
       FROM Users
       WHERE Uid = new.uid) <> 16 THEN
       RAISE EXCEPTION 'creditcard must be valid';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Triggers: Credit Card Check cont.

Here is the Trigger SQL and sample output when an incorrect credit card is entered.

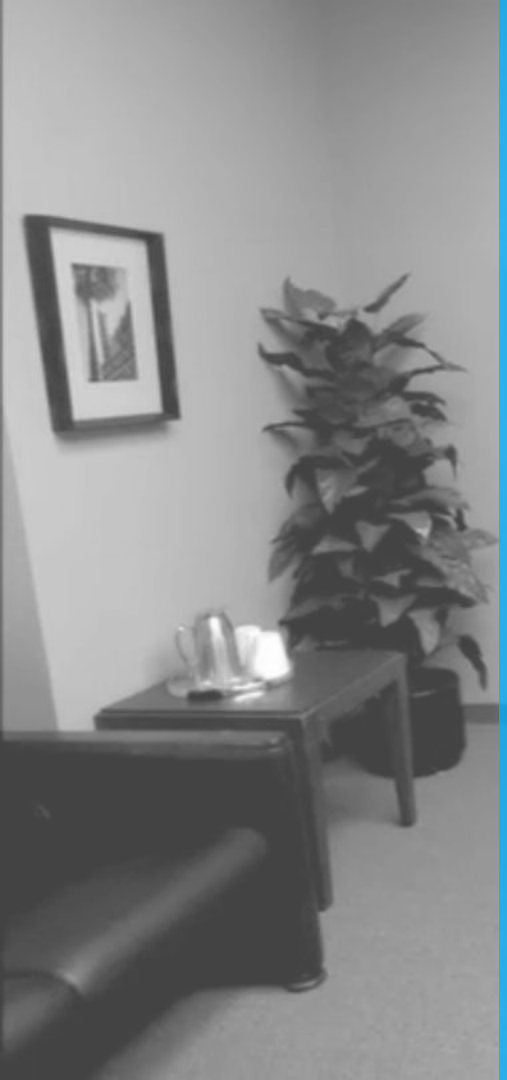
```
CREATE TRIGGER creditcard_check
 AFTER INSERT
 ON Users
 FOR EACH ROW
 EXECUTE PROCEDURE creditcard_num();
```

```
-- Executing query:
INSERT INTO Users(uid, firstname, lastname, email, subscriptionPK, creditcardnum,streetaddress,zipcode)
VALUES(8,'New', 'User', 'alpacas@alpaca.gov','Premium',345628993928125690,'34 Alpaca Lane',12601);
ERROR:  creditcard must be valid
CONTEXT:  PL/pgSQL function creditcard_num() line 17 at RAISE

********** Error **********

ERROR: creditcard must be valid
SQL state: P0001
Context: PL/pgSQL function creditcard_num() line 17 at RAISE
```


Security



Security: Admin & Film Curator Roles

ADMIN

The admin of the database can see and edit all information for all the tables in the database.

```
CREATE ROLE Admin;
GRANT ALL
ON ALL TABLES IN SCHEMA PUBLIC
TO Admin;
```

FILMCURATOR

The Film Curator needs to have select access to view what films are in the database currently and also have insert access to add new films. In the case of outdated films or unwatched films, the Film curator can also delete movies.

```
CREATE ROLE FilmCurator;
GRANT SELECT, INSERT, DELETE
ON Genres, Titles, Movies,  most_popular_titles
TO FilmCurator;
```

Security: Show Specialist Role

SHOWSPECALIST

The Show Specialist, like the film curator, needs to have select access to view what tv shows are in the database currently and also have insert access to add new shows. In the case of outdated shows or unwatched shows, the Show Specialist can also delete tv shows. In addition, the Show Specialist can update TV shows since new seasons may be added periodically.

```
CREATE ROLE ShowSpecialist;
GRANT SELECT, INSERT, UPDATE, DELETE
ON Genres, Titles, TVshows, Seasons, Episodes, most_popular_titles
TO ShowSpecialist;
```

Implementation Notes & Known Problems

- The test data in the insert statements is not fully complete for the purpose of space and the size of this project.
  - The Episodes table only contains one episode per season due to the large amount of episodes per season.
  - Similarly the ZipCodes table only contains zip codes for the sample users since there is a large amount of zip codes.

- Currently the way the billing address is put into the users table does not account for users localized outside the US.

- The recommendation stored procedure only makes “recommendations” based on the most popular genre of watched titles by the user.
  - This can be an issue if the user has watched all the titles of that genre, then no more recommendations will be made.
  - Also this can be an issue if a user has watched a lot of one genre but has not liked some of them.

- Currently you can only add one genre per title and certain titles may have multiple genres.

Future Enhancements

- Once HannahFlix has been established in the US, we plan to include international customers and make the service available worldwide.
- Another Future Enhancement is multiple profiles per account since accounts are usually shared in families. This way, recommendations can be more accurate for each individual.
- We are working on the ability to add multiple genres for one title.
- We want to improve our recommendation function so it's more accurate. It might be interesting to include user ratings of watched titles for the recommendations in case a user has watched a lot of one genre but has not liked some of them.
- In the future, we might want to include a licences table to help admins know when licenses are expired, in order know when to renew licences for titles or delete titles.