COMP 421 GROUP 46

Deneille Guiseppi	260722102
Luka Ma	260745824
Petar Basta	260735072

COMP 421- PROJECT DELIVERABLE 2

PART 1:

Entities:

Payment(pid, date, amount, accid) (accid ref Account)

Account(accid)

Region(regName)

Tag(tid, tagName)

Media(mid, title, releaseYear, isComplete)

Weak Entities:

AccountUser(userName, accid, regName) (accid ref Account, regName ref Region)

Episode(epNum, seasonNum, mid, epTitle) (seasonNum, mid ref Season)

Season(seasonNum, mid) (mid ref Media)

Rating(mid, name, accid, value) (mid ref Media, userName ref AccountUser, accid ref userName)

Relationships:

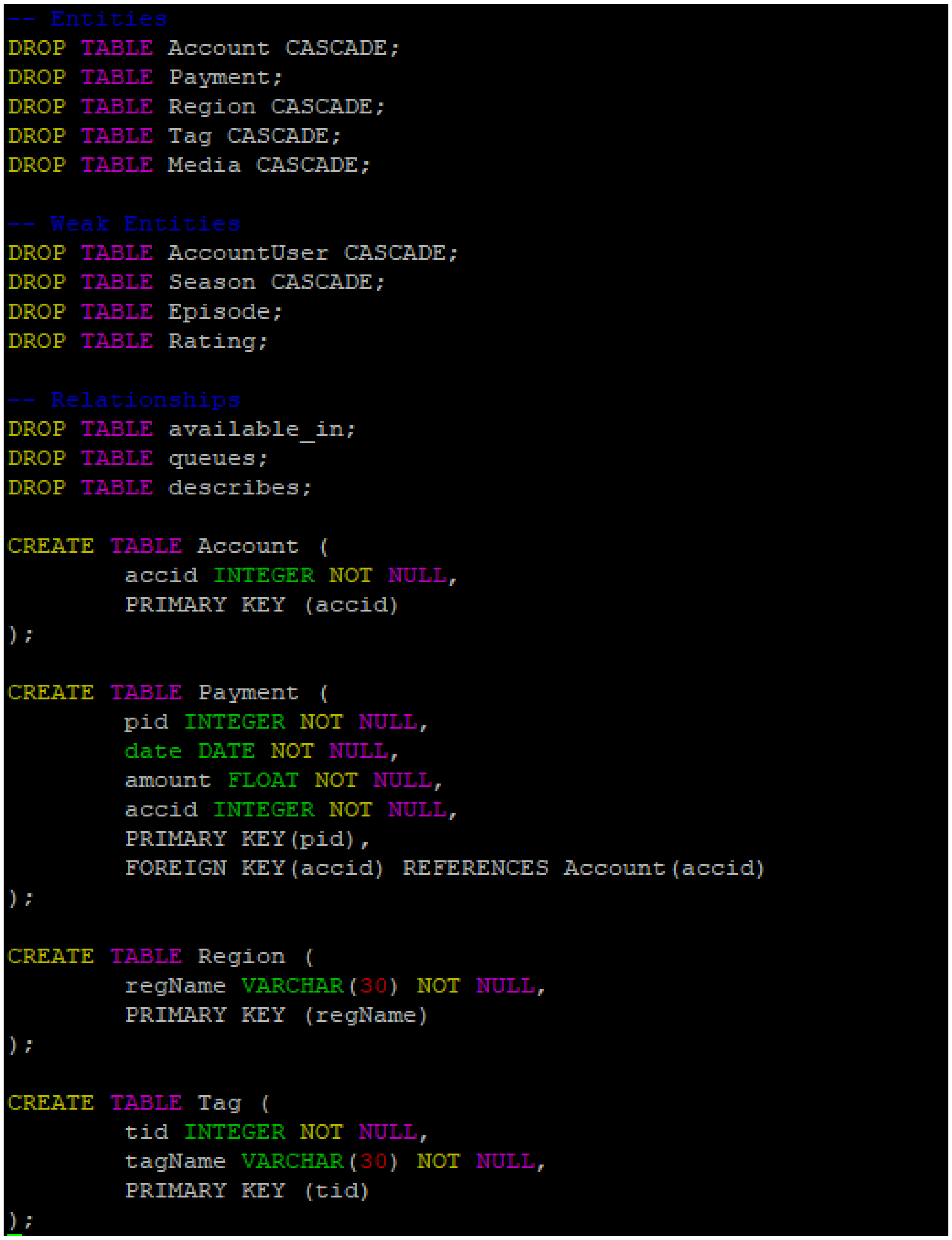
available_in(mid, regName) (mid ref Media, regName ref Region)

queues(userName, accid, mid) (userName ref AccountUser, accid ref AccountUser, mid ref Media)

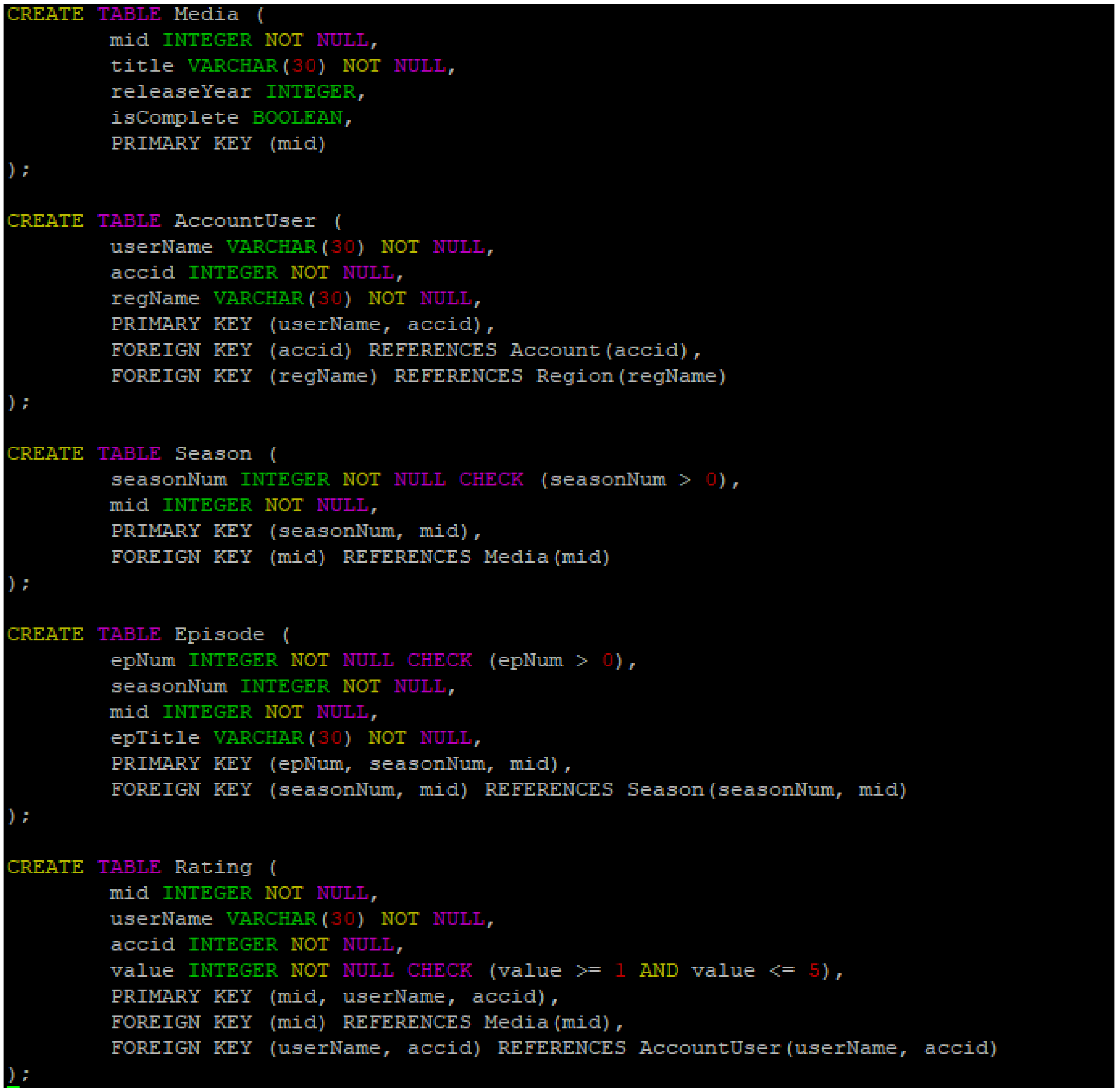
describes(tid, mid) (tid ref tag, mid ref Media)

PART 2:

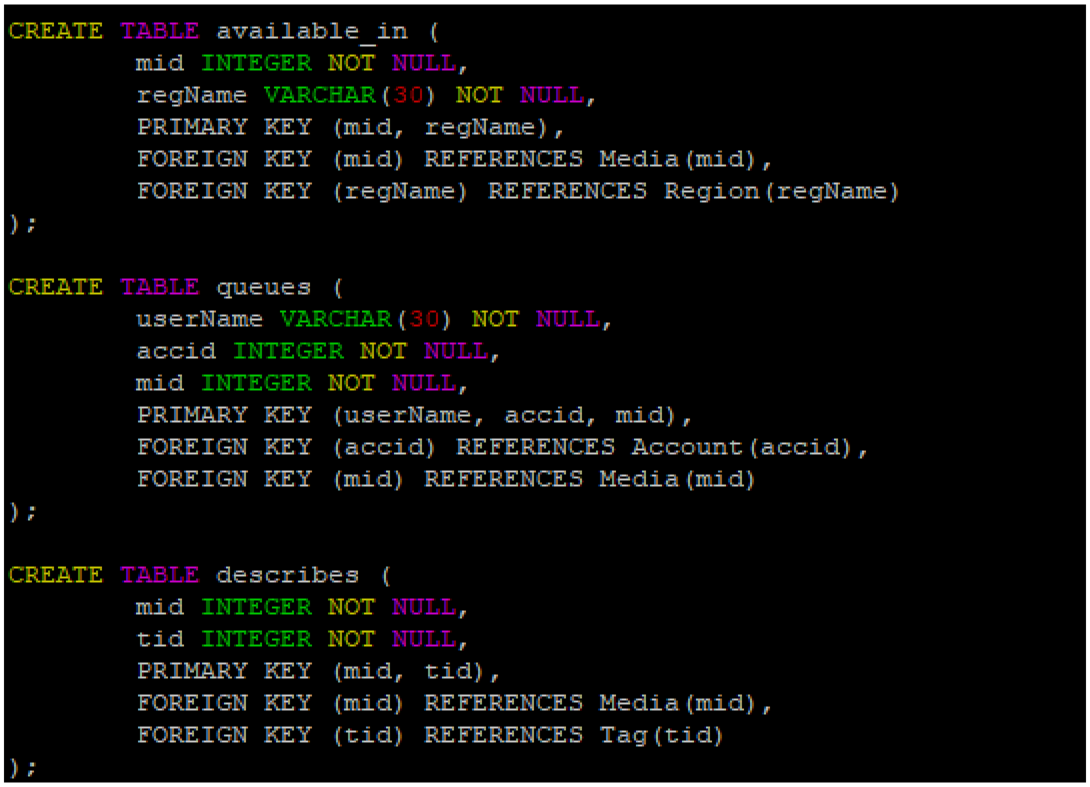
(i) createTables.sql:



```
-- Entities
DROP TABLE Account CASCADE;
DROP TABLE Payment;
DROP TABLE Region CASCADE;
DROP TABLE Tag CASCADE;
DROP TABLE Media CASCADE;

-- Weak Entities
DROP TABLE AccountUser CASCADE;
DROP TABLE Season CASCADE;
DROP TABLE Episode;
DROP TABLE Rating;

-- Relationships
DROP TABLE available_in;
DROP TABLE queues;
DROP TABLE describes;

CREATE TABLE Account (
        accid INTEGER NOT NULL,
        PRIMARY KEY (accid)
);

CREATE TABLE Payment (
        pid INTEGER NOT NULL,
        date DATE NOT NULL,
        amount FLOAT NOT NULL,
        accid INTEGER NOT NULL,
        PRIMARY KEY(pid),
        FOREIGN KEY(accid) REFERENCES Account(accid)
);

CREATE TABLE Region (
        regName VARCHAR(30) NOT NULL,
        PRIMARY KEY (regName)
);

CREATE TABLE Tag (
        tid INTEGER NOT NULL,
        tagName VARCHAR(30) NOT NULL,
        PRIMARY KEY (tid)
);
```



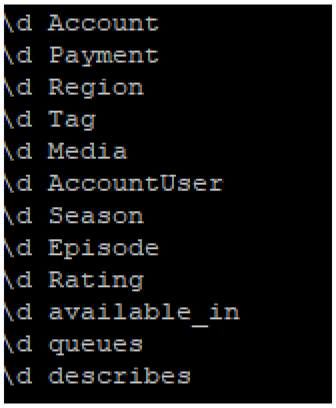
```
CREATE TABLE Media (
        mid INTEGER NOT NULL,
        title VARCHAR(30) NOT NULL,
        releaseYear INTEGER,
        isComplete BOOLEAN,
        PRIMARY KEY (mid)
);

CREATE TABLE AccountUser (
        userName VARCHAR(30) NOT NULL,
        accid INTEGER NOT NULL,
        regName VARCHAR(30) NOT NULL,
        PRIMARY KEY (userName, accid),
        FOREIGN KEY (accid) REFERENCES Account(accid),
        FOREIGN KEY (regName) REFERENCES Region(regName)
);

CREATE TABLE Season (
        seasonNum INTEGER NOT NULL CHECK (seasonNum > 0),
        mid INTEGER NOT NULL,
        PRIMARY KEY (seasonNum, mid),
        FOREIGN KEY (mid) REFERENCES Media(mid)
);

CREATE TABLE Episode (
        epNum INTEGER NOT NULL CHECK (epNum > 0),
        seasonNum INTEGER NOT NULL,
        mid INTEGER NOT NULL,
        epTitle VARCHAR(30) NOT NULL,
        PRIMARY KEY (epNum, seasonNum, mid),
        FOREIGN KEY (seasonNum, mid) REFERENCES Season(seasonNum, mid)
);

CREATE TABLE Rating (
        mid INTEGER NOT NULL,
        userName VARCHAR(30) NOT NULL,
        accid INTEGER NOT NULL,
        value INTEGER NOT NULL CHECK (value >= 1 AND value <= 5),
        PRIMARY KEY (mid, userName, accid),
        FOREIGN KEY (mid) REFERENCES Media(mid),
        FOREIGN KEY (userName, accid) REFERENCES AccountUser(userName, accid)
);
```



```
CREATE TABLE available_in (
        mid INTEGER NOT NULL,
        regName VARCHAR(30) NOT NULL,
        PRIMARY KEY (mid, regName),
        FOREIGN KEY (mid) REFERENCES Media(mid),
        FOREIGN KEY (regName) REFERENCES Region(regName)
);

CREATE TABLE queues (
        userName VARCHAR(30) NOT NULL,
        accid INTEGER NOT NULL,
        mid INTEGER NOT NULL,
        PRIMARY KEY (userName, accid, mid),
        FOREIGN KEY (accid) REFERENCES Account(accid),
        FOREIGN KEY (mid) REFERENCES Media(mid)
);

CREATE TABLE describes (
        mid INTEGER NOT NULL,
        tid INTEGER NOT NULL,
        PRIMARY KEY (mid, tid),
        FOREIGN KEY (mid) REFERENCES Media(mid),
        FOREIGN KEY (tid) REFERENCES Tag(tid)
);
```

The script above creates the tables for Account, Payment, Region, Tag, Media, AccountUser, Season, Episode, Rating, available_in, queues and describes respectively.

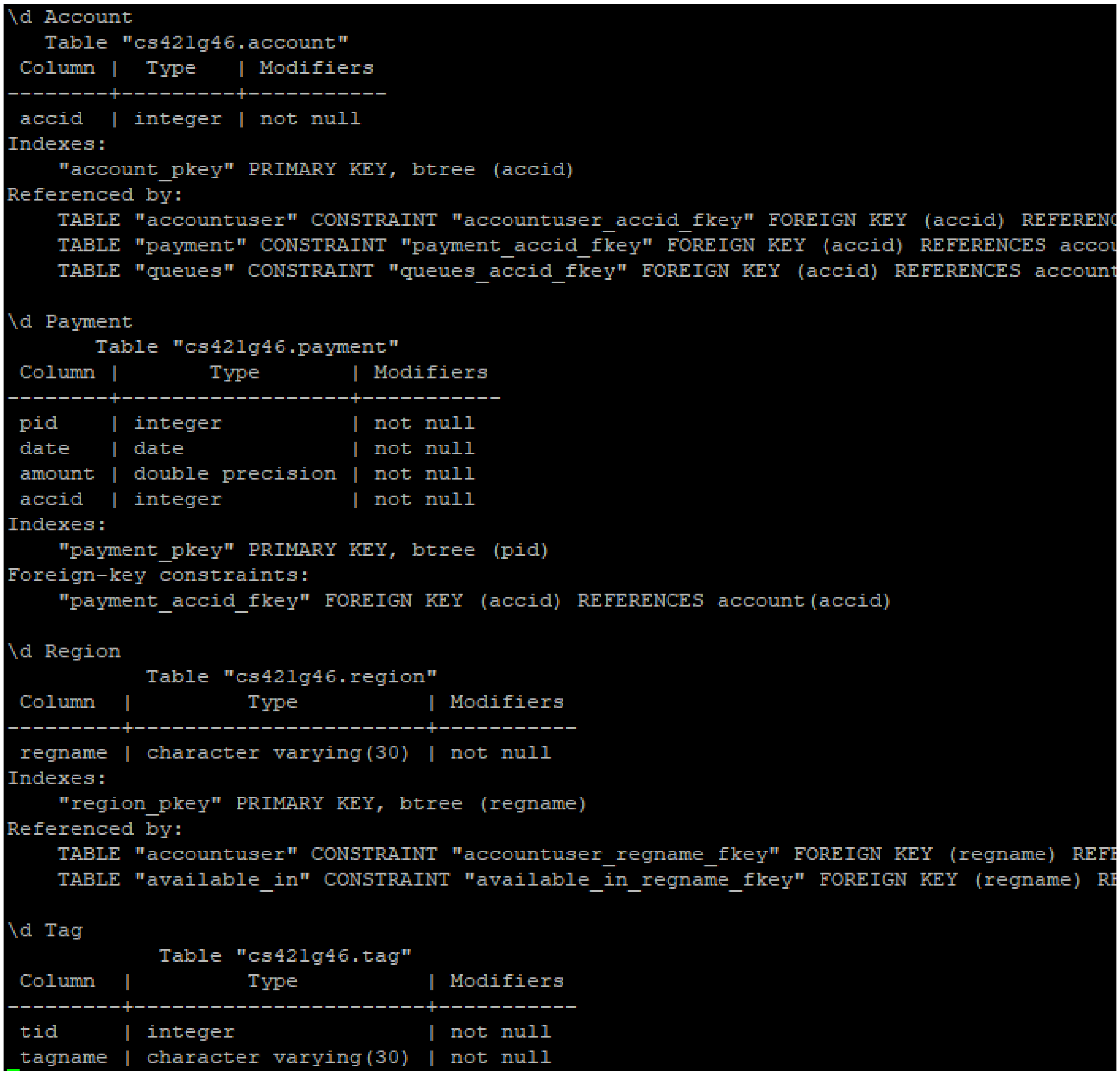
(ii) displayTableSchemas.sql:



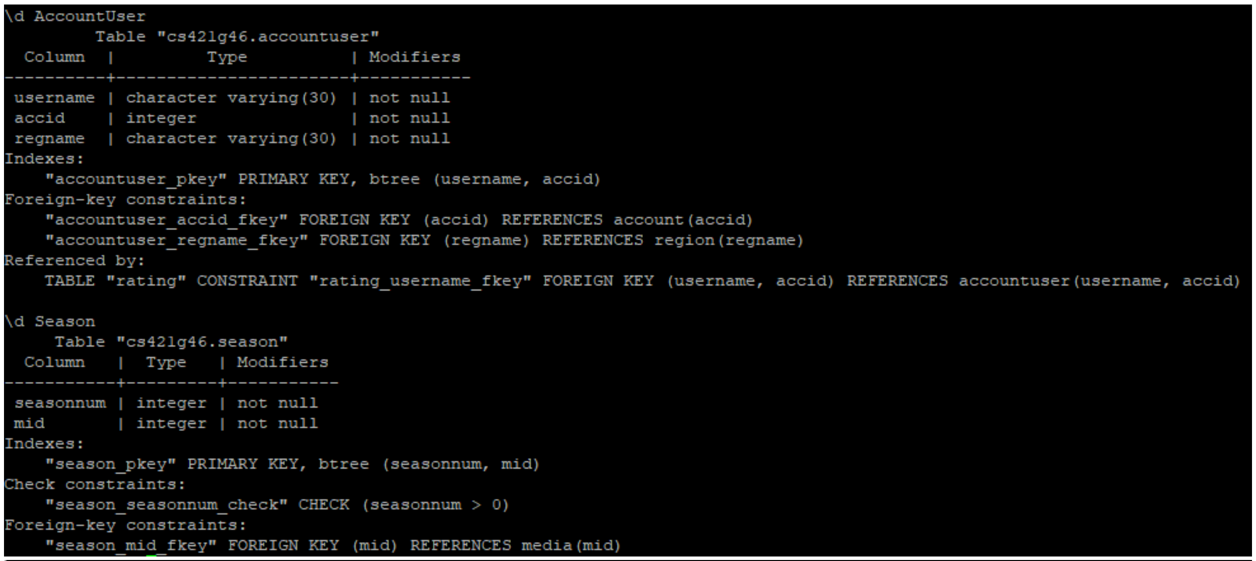
```
\d Account
\d Payment
\d Region
\d Tag
\d Media
\d AccountUser
\d Season
\d Episode
\d Rating
\d available_in
\d queues
\d describes
```

As in its name, this script displays the tables of the schema.

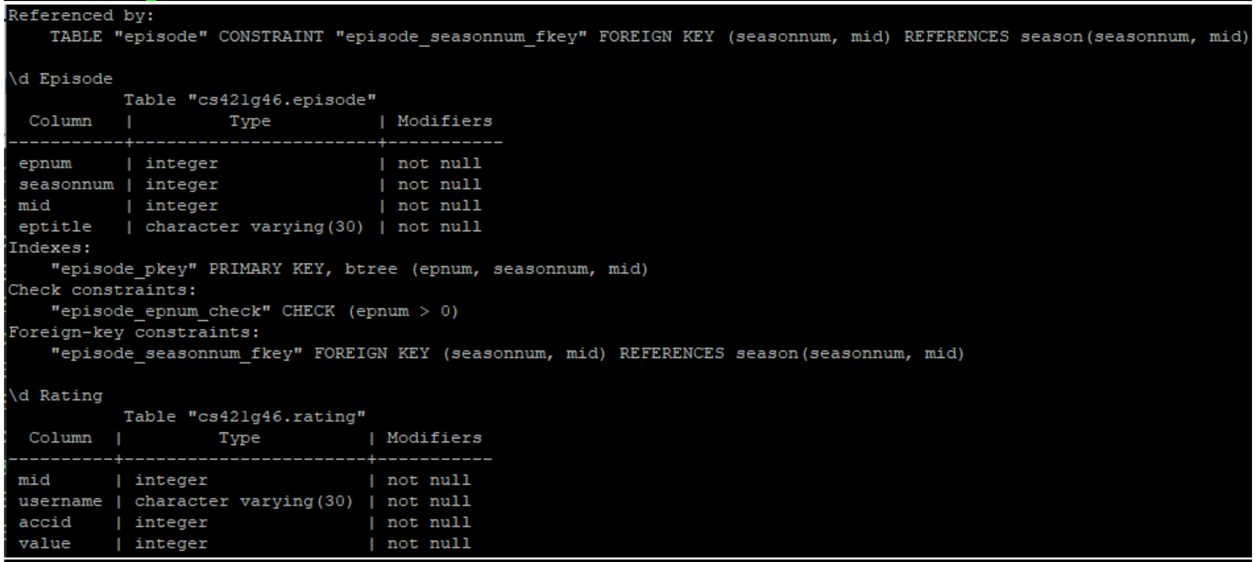
displayTableSchemas.log:



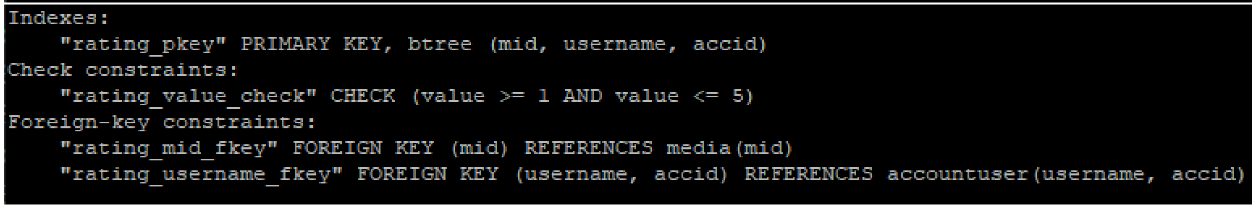
```
\d Account
   Table "cs421g46.account"
 Column |  Type   | Modifiers
--------+---------+-----------
 accid  | integer | not null
Indexes:
    "account_pkey" PRIMARY KEY, btree (accid)
Referenced by:
    TABLE "accountuser" CONSTRAINT "accountuser_accid_fkey" FOREIGN KEY (accid) REFERENC
    TABLE "payment" CONSTRAINT "payment_accid_fkey" FOREIGN KEY (accid) REFERENCES accou
    TABLE "queues" CONSTRAINT "queues_accid_fkey" FOREIGN KEY (accid) REFERENCES account

\d Payment
       Table "cs421g46.payment"
 Column |       Type       | Modifiers
--------+------------------+-----------
 pid    | integer          | not null
 date   | date             | not null
 amount | double precision | not null
 accid  | integer          | not null
Indexes:
    "payment_pkey" PRIMARY KEY, btree (pid)
Foreign-key constraints:
    "payment_accid_fkey" FOREIGN KEY (accid) REFERENCES account(accid)

\d Region
           Table "cs421g46.region"
 Column  |         Type          | Modifiers
---------+-----------------------+-----------
 regname | character varying(30) | not null
Indexes:
    "region_pkey" PRIMARY KEY, btree (regname)
Referenced by:
    TABLE "accountuser" CONSTRAINT "accountuser_regname_fkey" FOREIGN KEY (regname) REFE
    TABLE "available_in" CONSTRAINT "available_in_regname_fkey" FOREIGN KEY (regname) RE

\d Tag
            Table "cs421g46.tag"
 Column  |         Type          | Modifiers
---------+-----------------------+-----------
 tid     | integer               | not null
 tagname | character varying(30) | not null
```



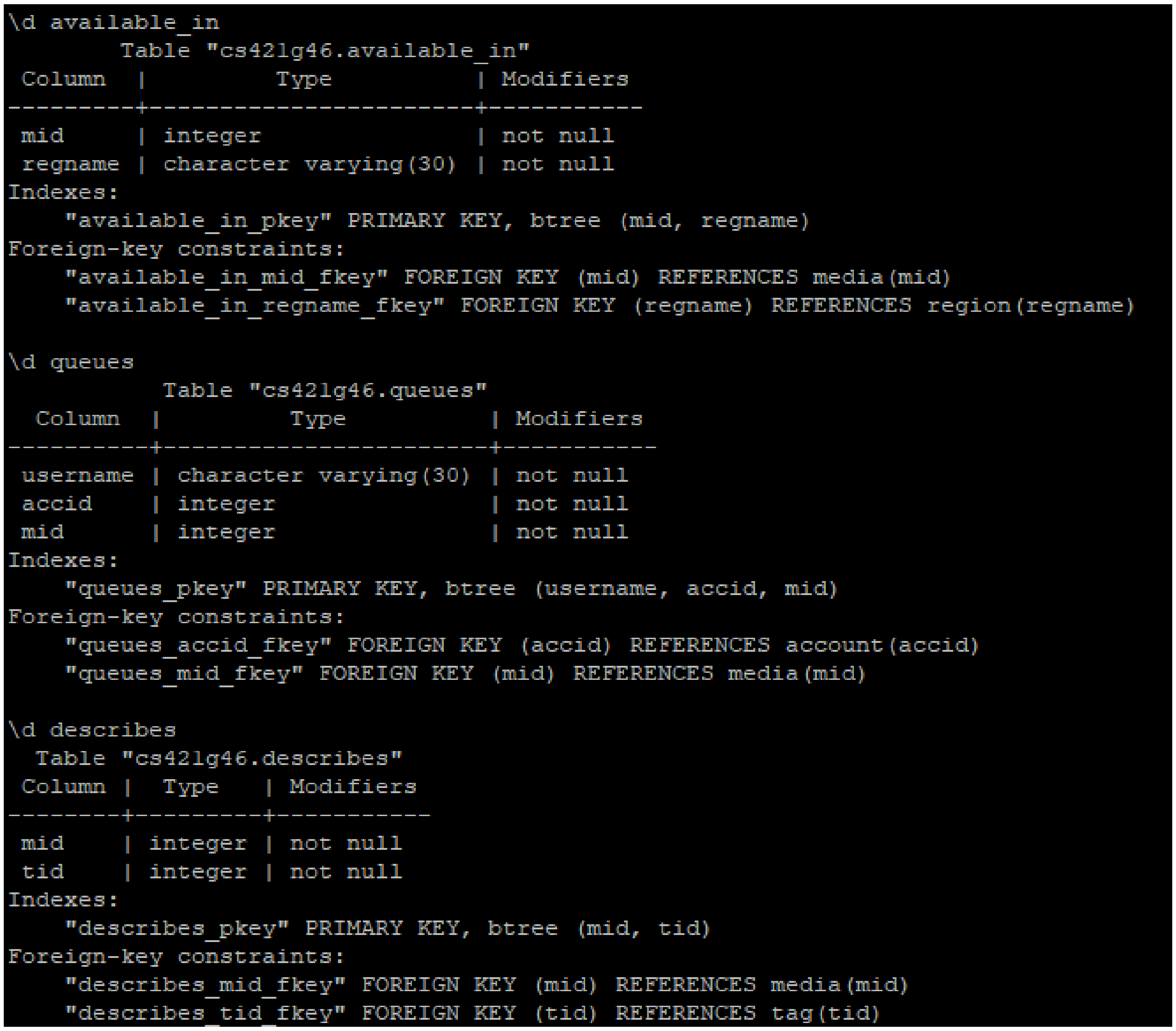
```
\d AccountUser
         Table "cs421g46.accountuser"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 username | character varying(30) | not null
 accid    | integer               | not null
 regname  | character varying(30) | not null
Indexes:
    "accountuser_pkey" PRIMARY KEY, btree (username, accid)
Foreign-key constraints:
    "accountuser_accid_fkey" FOREIGN KEY (accid) REFERENCES account(accid)
    "accountuser_regname_fkey" FOREIGN KEY (regname) REFERENCES region(regname)
Referenced by:
    TABLE "rating" CONSTRAINT "rating_username_fkey" FOREIGN KEY (username, accid) REFERENCES accountuser(username, accid)

\d Season
     Table "cs421g46.season"
  Column   |  Type   | Modifiers
-----------+---------+-----------
 seasonnum | integer | not null
 mid       | integer | not null
Indexes:
    "season_pkey" PRIMARY KEY, btree (seasonnum, mid)
Check constraints:
    "season_seasonnum_check" CHECK (seasonnum > 0)
Foreign-key constraints:
    "season_mid_fkey" FOREIGN KEY (mid) REFERENCES media(mid)
```



```
Referenced by:
    TABLE "episode" CONSTRAINT "episode_seasonnum_fkey" FOREIGN KEY (seasonnum, mid) REFERENCES season(seasonnum, mid)

\d Episode
           Table "cs421g46.episode"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 epnum     | integer               | not null
 seasonnum | integer               | not null
 mid       | integer               | not null
 eptitle   | character varying(30) | not null
Indexes:
    "episode_pkey" PRIMARY KEY, btree (epnum, seasonnum, mid)
Check constraints:
    "episode_epnum_check" CHECK (epnum > 0)
Foreign-key constraints:
    "episode_seasonnum_fkey" FOREIGN KEY (seasonnum, mid) REFERENCES season(seasonnum, mid)

\d Rating
           Table "cs421g46.rating"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 mid      | integer               | not null
 username | character varying(30) | not null
 accid    | integer               | not null
 value    | integer               | not null
```



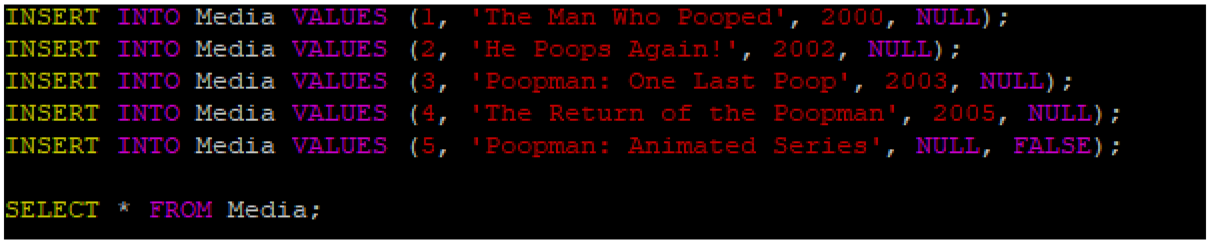
```
Indexes:
    "rating_pkey" PRIMARY KEY, btree (mid, username, accid)
Check constraints:
    "rating_value_check" CHECK (value >= 1 AND value <= 5)
Foreign-key constraints:
    "rating_mid_fkey" FOREIGN KEY (mid) REFERENCES media(mid)
    "rating_username_fkey" FOREIGN KEY (username, accid) REFERENCES accountuser(username, accid)
```



```
\d available_in
        Table "cs421g46.available_in"
 Column  |         Type          | Modifiers
---------+-----------------------+-----------
 mid     | integer               | not null
 regname | character varying(30) | not null
Indexes:
    "available_in_pkey" PRIMARY KEY, btree (mid, regname)
Foreign-key constraints:
    "available_in_mid_fkey" FOREIGN KEY (mid) REFERENCES media(mid)
    "available_in_regname_fkey" FOREIGN KEY (regname) REFERENCES region(regname)

\d queues
           Table "cs421g46.queues"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 username | character varying(30) | not null
 accid    | integer               | not null
 mid      | integer               | not null
Indexes:
    "queues_pkey" PRIMARY KEY, btree (username, accid, mid)
Foreign-key constraints:
    "queues_accid_fkey" FOREIGN KEY (accid) REFERENCES account(accid)
    "queues_mid_fkey" FOREIGN KEY (mid) REFERENCES media(mid)

\d describes
  Table "cs421g46.describes"
 Column |  Type   | Modifiers
--------+---------+-----------
 mid    | integer | not null
 tid    | integer | not null
Indexes:
    "describes_pkey" PRIMARY KEY, btree (mid, tid)
Foreign-key constraints:
    "describes_mid_fkey" FOREIGN KEY (mid) REFERENCES media(mid)
    "describes_tid_fkey" FOREIGN KEY (tid) REFERENCES tag(tid)
```

PART 3:

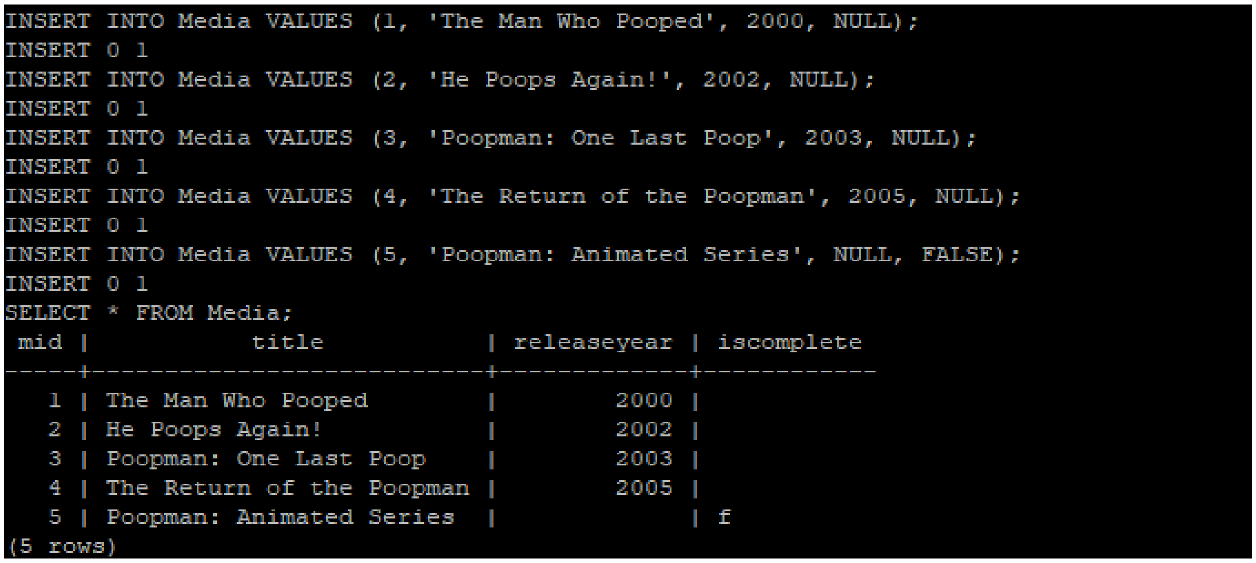
insertRecords5.sql



```
INSERT INTO Media VALUES (1, 'The Man Who Pooped', 2000, NULL);
INSERT INTO Media VALUES (2, 'He Poops Again!', 2002, NULL);
INSERT INTO Media VALUES (3, 'Poopman: One Last Poop', 2003, NULL);
INSERT INTO Media VALUES (4, 'The Return of the Poopman', 2005, NULL);
INSERT INTO Media VALUES (5, 'Poopman: Animated Series', NULL, FALSE);

SELECT * FROM Media;
```

This file deals with table insertion. The above is making insertions to the Media table.

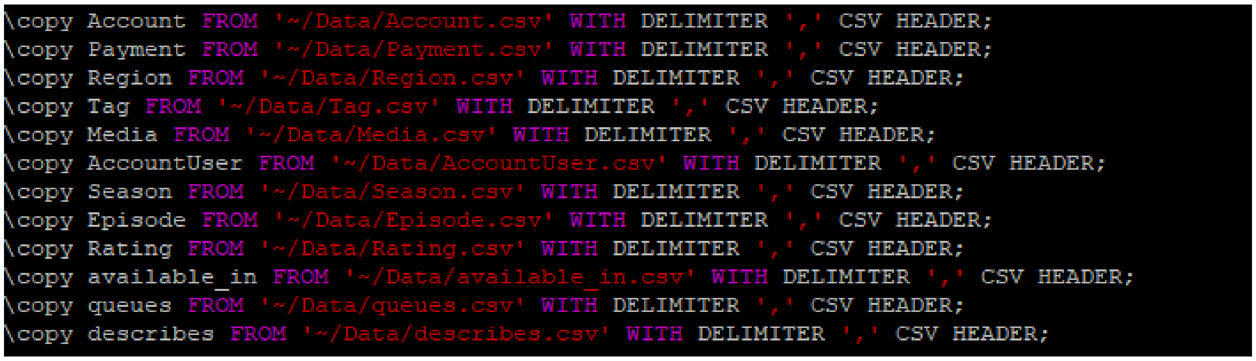
insertRecords5.sql:



```
INSERT INTO Media VALUES (1, 'The Man Who Pooped', 2000, NULL);
INSERT 0 1
INSERT INTO Media VALUES (2, 'He Poops Again!', 2002, NULL);
INSERT 0 1
INSERT INTO Media VALUES (3, 'Poopman: One Last Poop', 2003, NULL);
INSERT 0 1
INSERT INTO Media VALUES (4, 'The Return of the Poopman', 2005, NULL);
INSERT 0 1
INSERT INTO Media VALUES (5, 'Poopman: Animated Series', NULL, FALSE);
INSERT 0 1
SELECT * FROM Media;
 mid |           title           | releaseyear | iscomplete
-----+---------------------------+-------------+------------
   1 | The Man Who Pooped        |        2000 |
   2 | He Poops Again!           |        2002 |
   3 | Poopman: One Last Poop    |        2003 |
   4 | The Return of the Poopman |        2005 |
   5 | Poopman: Animated Series  |             | f
(5 rows)
```

PART 4:

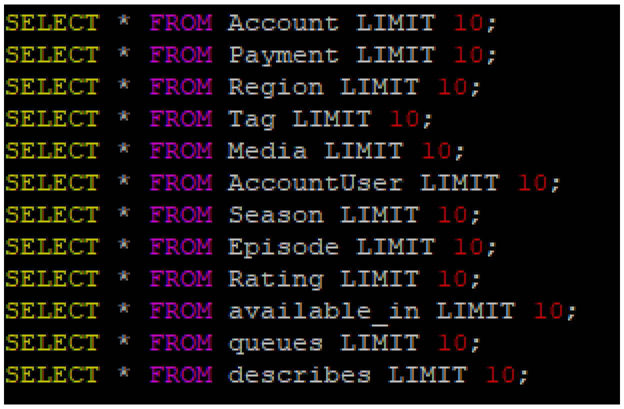
populateRecords.sql:



```
\copy Account FROM '~/Data/Account.csv' WITH DELIMITER ',' CSV HEADER;
\copy Payment FROM '~/Data/Payment.csv' WITH DELIMITER ',' CSV HEADER;
\copy Region FROM '~/Data/Region.csv' WITH DELIMITER ',' CSV HEADER;
\copy Tag FROM '~/Data/Tag.csv' WITH DELIMITER ',' CSV HEADER;
\copy Media FROM '~/Data/Media.csv' WITH DELIMITER ',' CSV HEADER;
\copy AccountUser FROM '~/Data/AccountUser.csv' WITH DELIMITER ',' CSV HEADER;
\copy Season FROM '~/Data/Season.csv' WITH DELIMITER ',' CSV HEADER;
\copy Episode FROM '~/Data/Episode.csv' WITH DELIMITER ',' CSV HEADER;
\copy Rating FROM '~/Data/Rating.csv' WITH DELIMITER ',' CSV HEADER;
\copy available_in FROM '~/Data/available_in.csv' WITH DELIMITER ',' CSV HEADER;
\copy queues FROM '~/Data/queues.csv' WITH DELIMITER ',' CSV HEADER;
\copy describes FROM '~/Data/describes.csv' WITH DELIMITER ',' CSV HEADER;
```

This file was used to load in a large amount of data (in the form of csv files placed in ~/Data directory).

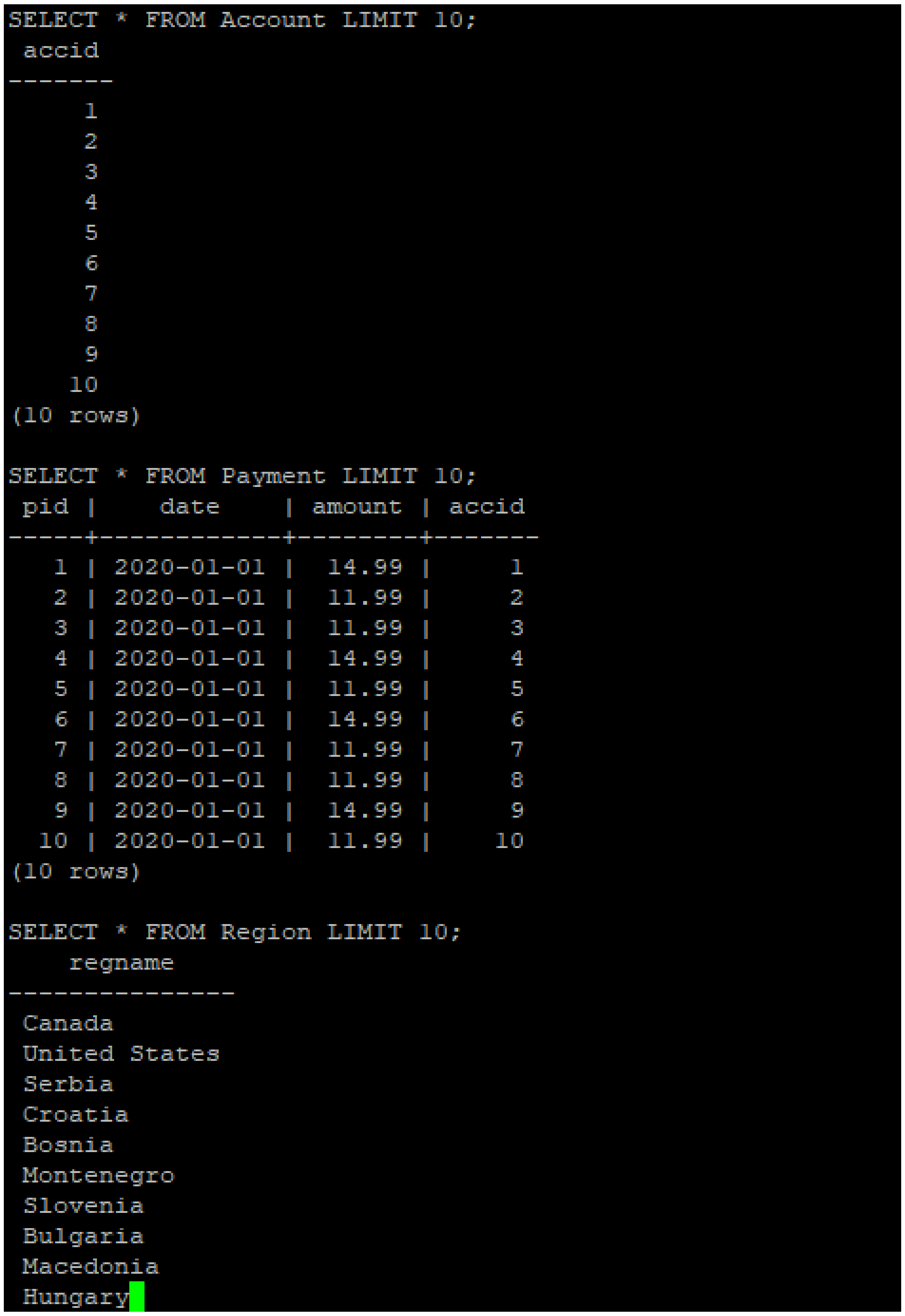
displayData.sql:



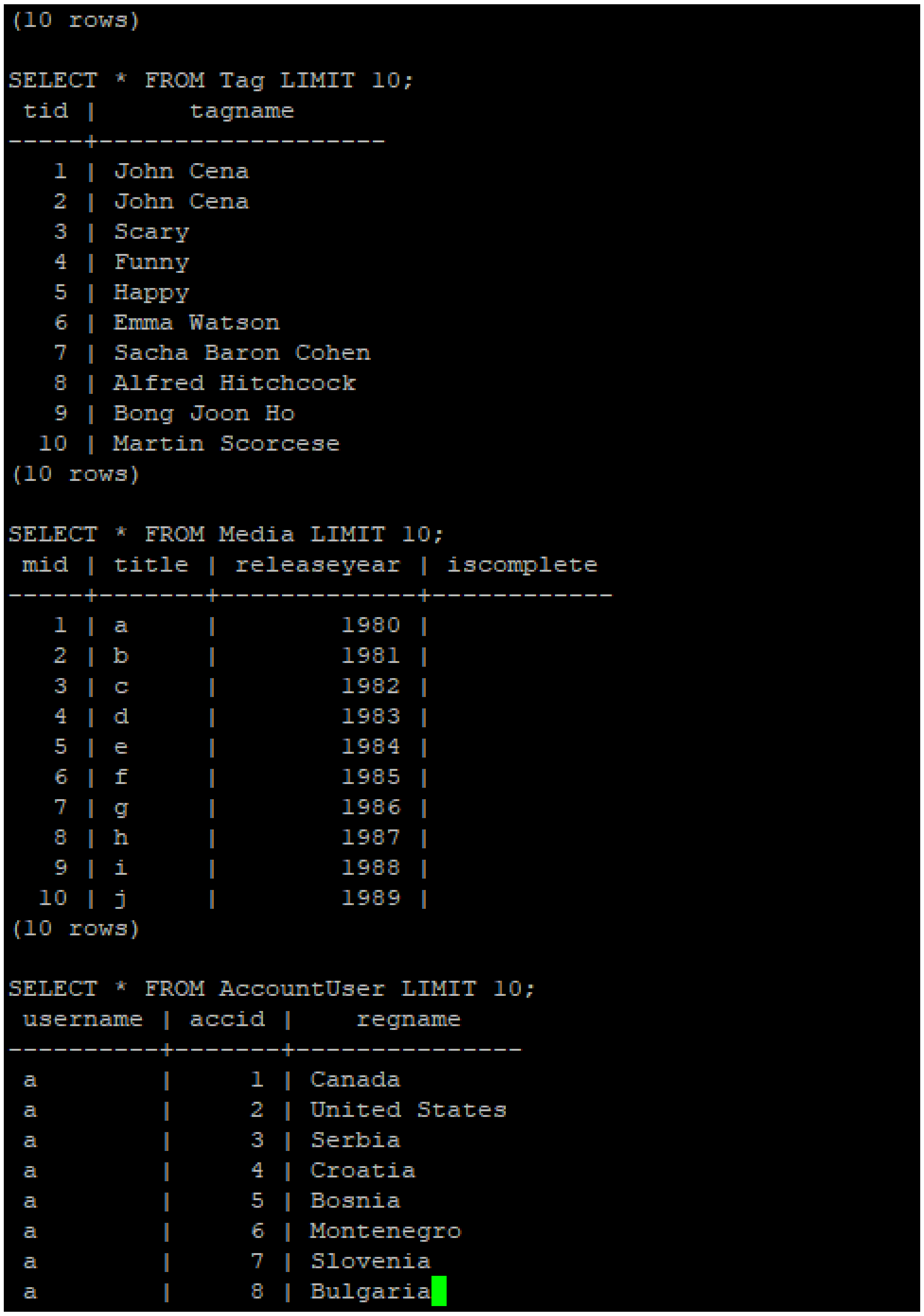
```
SELECT * FROM Account LIMIT 10;
SELECT * FROM Payment LIMIT 10;
SELECT * FROM Region LIMIT 10;
SELECT * FROM Tag LIMIT 10;
SELECT * FROM Media LIMIT 10;
SELECT * FROM AccountUser LIMIT 10;
SELECT * FROM Season LIMIT 10;
SELECT * FROM Episode LIMIT 10;
SELECT * FROM Rating LIMIT 10;
SELECT * FROM available_in LIMIT 10;
SELECT * FROM queues LIMIT 10;
SELECT * FROM describes LIMIT 10;
```

This file was created to display 10 rows of each table. You can see the result in the file below.

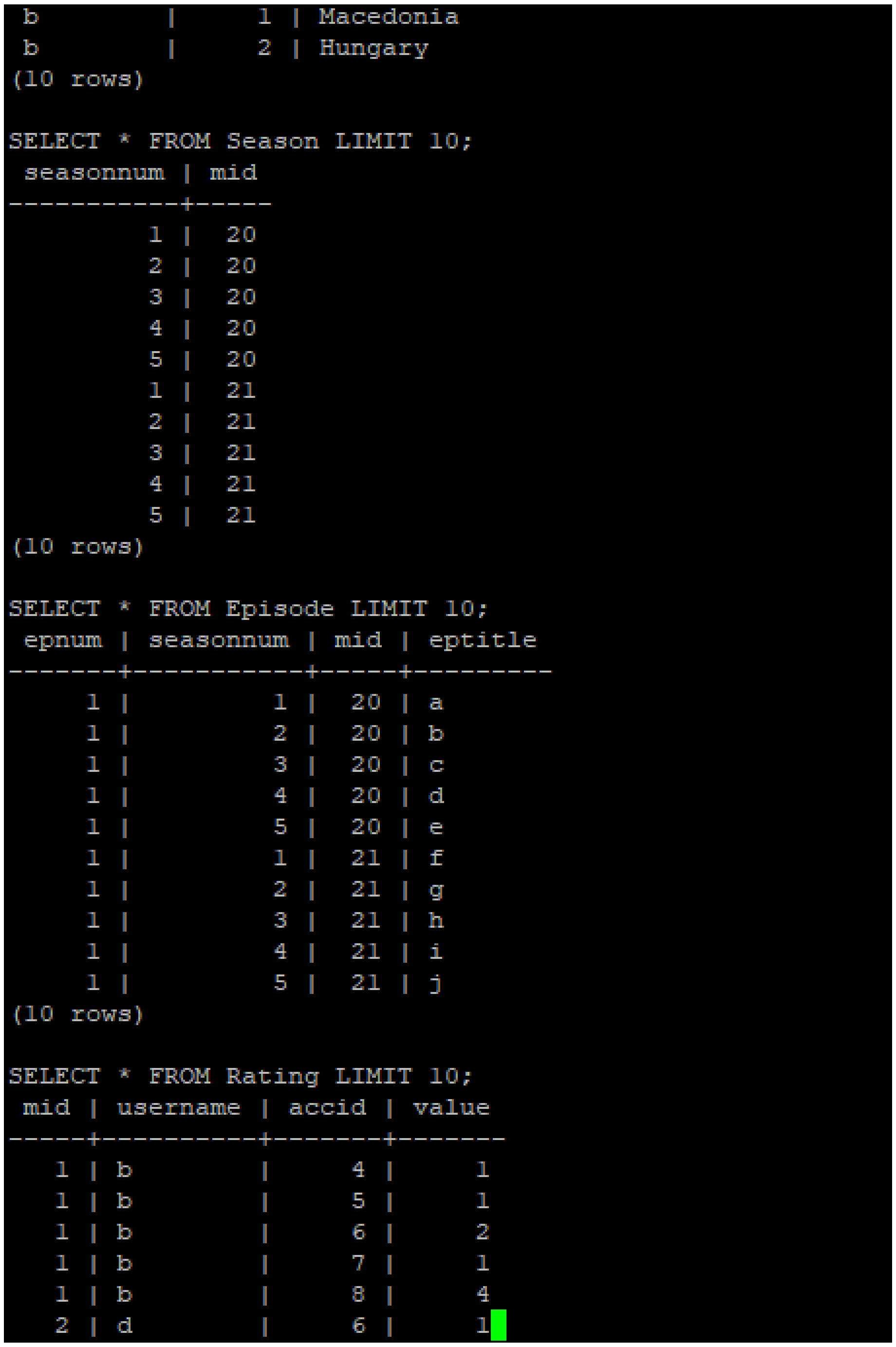
displayData.log:

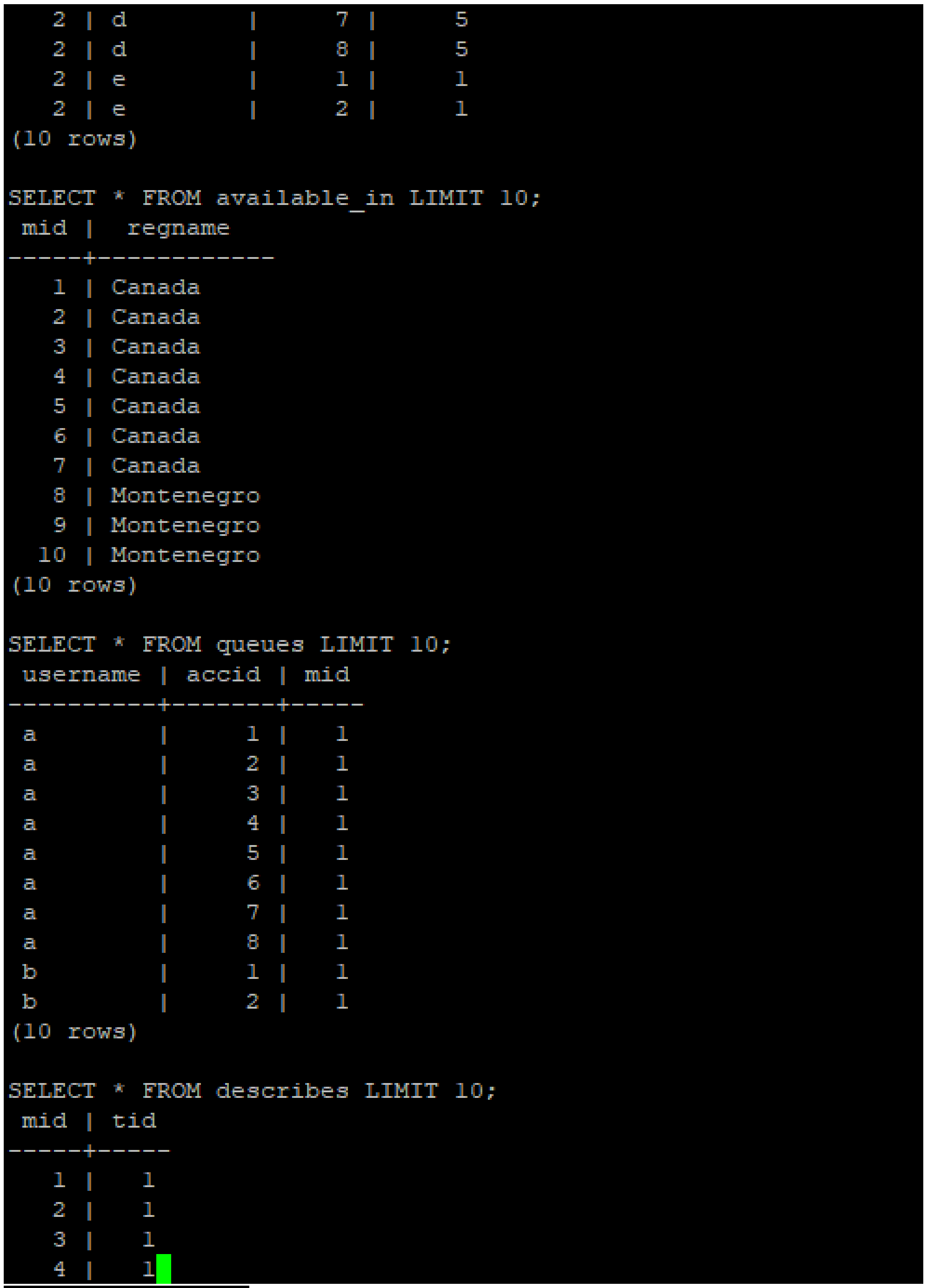


```
SELECT * FROM Account LIMIT 10;
 accid
-------
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
(10 rows)

SELECT * FROM Payment LIMIT 10;
 pid |    date    | amount | accid
-----+------------+--------+-------
   1 | 2020-01-01 |  14.99 |     1
   2 | 2020-01-01 |  11.99 |     2
   3 | 2020-01-01 |  11.99 |     3
   4 | 2020-01-01 |  14.99 |     4
   5 | 2020-01-01 |  11.99 |     5
   6 | 2020-01-01 |  14.99 |     6
   7 | 2020-01-01 |  11.99 |     7
   8 | 2020-01-01 |  11.99 |     8
   9 | 2020-01-01 |  14.99 |     9
  10 | 2020-01-01 |  11.99 |    10
(10 rows)

SELECT * FROM Region LIMIT 10;
    regname
---------------
 Canada
 United States
 Serbia
 Croatia
 Bosnia
 Montenegro
 Slovenia
 Bulgaria
 Macedonia
 Hungary
```

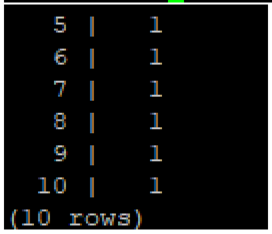


```
(10 rows)

SELECT * FROM Tag LIMIT 10;
 tid |      tagname
-----+-------------------
   1 | John Cena
   2 | John Cena
   3 | Scary
   4 | Funny
   5 | Happy
   6 | Emma Watson
   7 | Sacha Baron Cohen
   8 | Alfred Hitchcock
   9 | Bong Joon Ho
  10 | Martin Scorcese
(10 rows)

SELECT * FROM Media LIMIT 10;
 mid | title | releaseyear | iscomplete
-----+-------+-------------+------------
   1 | a     |        1980 |
   2 | b     |        1981 |
   3 | c     |        1982 |
   4 | d     |        1983 |
   5 | e     |        1984 |
   6 | f     |        1985 |
   7 | g     |        1986 |
   8 | h     |        1987 |
   9 | i     |        1988 |
  10 | j     |        1989 |
(10 rows)

SELECT * FROM AccountUser LIMIT 10;
 username | accid |    regname
----------+-------+---------------
 a        |     1 | Canada
 a        |     2 | United States
 a        |     3 | Serbia
 a        |     4 | Croatia
 a        |     5 | Bosnia
 a        |     6 | Montenegro
 a        |     7 | Slovenia
 a        |     8 | Bulgaria
```


```
 b          |      1 | Macedonia
 b          |      2 | Hungary
(10 rows)

SELECT * FROM Season LIMIT 10;
 seasonnum | mid
-----------+-----
         1 |  20
         2 |  20
         3 |  20
         4 |  20
         5 |  20
         1 |  21
         2 |  21
         3 |  21
         4 |  21
         5 |  21
(10 rows)

SELECT * FROM Episode LIMIT 10;
 epnum | seasonnum | mid | eptitle
-------+-----------+-----+---------
     1 |         1 |  20 | a
     1 |         2 |  20 | b
     1 |         3 |  20 | c
     1 |         4 |  20 | d
     1 |         5 |  20 | e
     1 |         1 |  21 | f
     1 |         2 |  21 | g
     1 |         3 |  21 | h
     1 |         4 |  21 | i
     1 |         5 |  21 | j
(10 rows)

SELECT * FROM Rating LIMIT 10;
 mid | username | accid | value
-----+----------+-------+-------
   1 | b        |     4 |     1
   1 | b        |     5 |     1
   1 | b        |     6 |     2
   1 | b        |     7 |     1
   1 | b        |     8 |     4
   2 | d        |     6 |     1
```



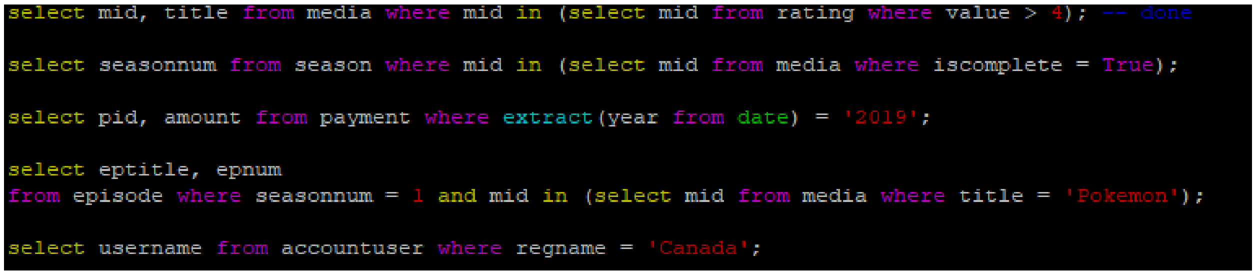
```
   2 | d          |      7 |      5
   2 | d          |      8 |      5
   2 | e          |      1 |      1
   2 | e          |      2 |      1
(10 rows)

SELECT * FROM available_in LIMIT 10;
 mid |  regname
-----+------------
   1 | Canada
   2 | Canada
   3 | Canada
   4 | Canada
   5 | Canada
   6 | Canada
   7 | Canada
   8 | Montenegro
   9 | Montenegro
  10 | Montenegro
(10 rows)

SELECT * FROM queues LIMIT 10;
 username | accid | mid
----------+-------+-----
 a        |     1 |   1
 a        |     2 |   1
 a        |     3 |   1
 a        |     4 |   1
 a        |     5 |   1
 a        |     6 |   1
 a        |     7 |   1
 a        |     8 |   1
 b        |     1 |   1
 b        |     2 |   1
(10 rows)

SELECT * FROM describes LIMIT 10;
 mid | tid
-----+-----
   1 |   1
   2 |   1
   3 |   1
   4 |   1
```



```
   5 |   1
   6 |   1
   7 |   1
   8 |   1
   9 |   1
  10 |   1
(10 rows)
```

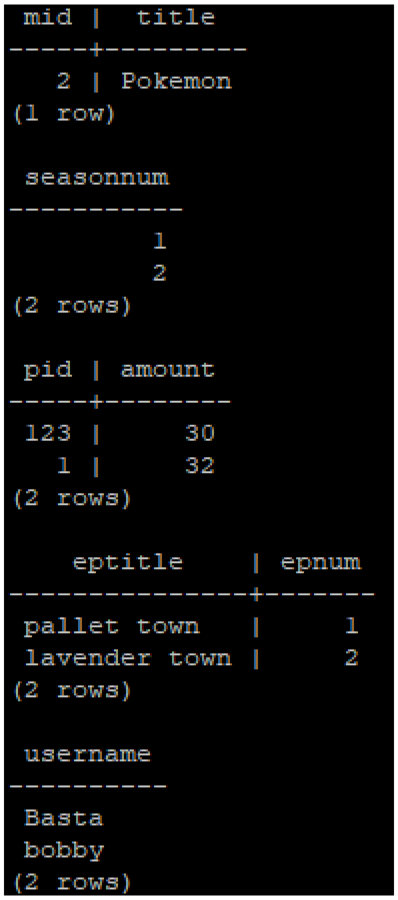
PART 5:

selectQueries5.sql:



```
select mid, title from media where mid in (select mid from rating where value > 4); -- done

select seasonnum from season where mid in (select mid from media where iscomplete = True);

select pid, amount from payment where extract(year from date) = '2019';

select eptitle, epnum
from episode where seasonnum = 1 and mid in (select mid from media where title = 'Pokemon');

select username from accountuser where regname = 'Canada';
```

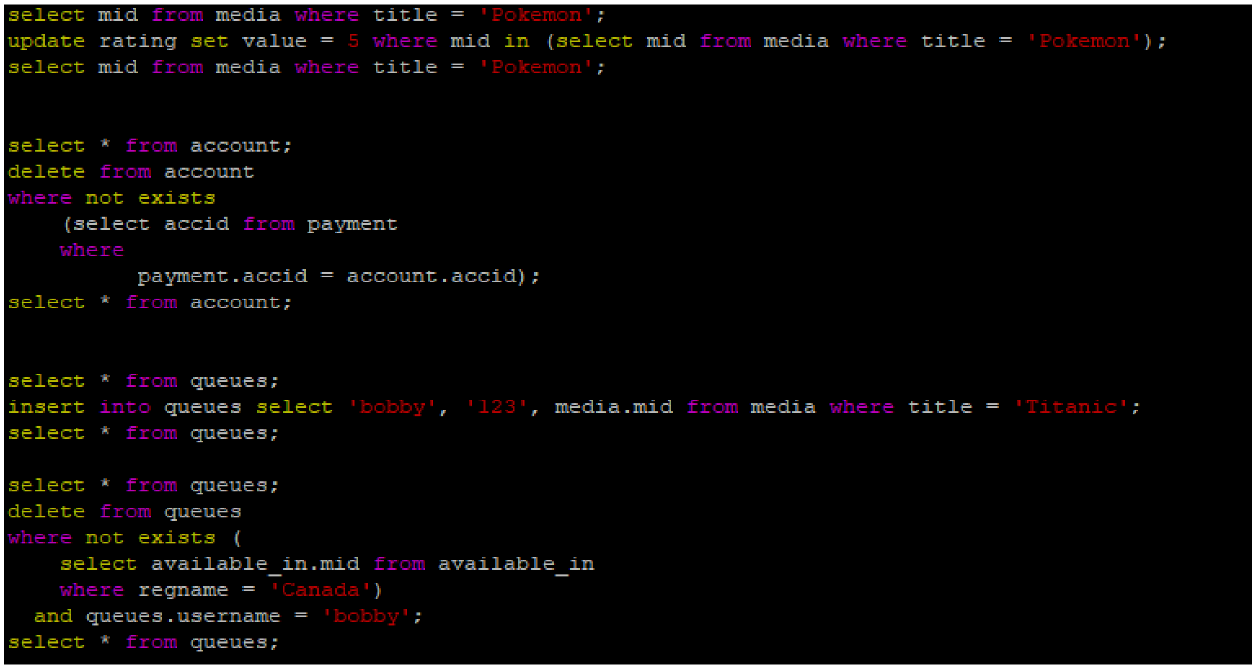
selectQueries5.log:



```
 mid |  title
-----+----------
   2 | Pokemon
(1 row)

 seasonnum
-----------
         1
         2
(2 rows)

 pid | amount
-----+--------
 123 |     30
   1 |     32
(2 rows)

    eptitle     | epnum
----------------+-------
 pallet town    |     1
 lavender town  |     2
(2 rows)

 username
----------
 Basta
 bobby
(2 rows)
```

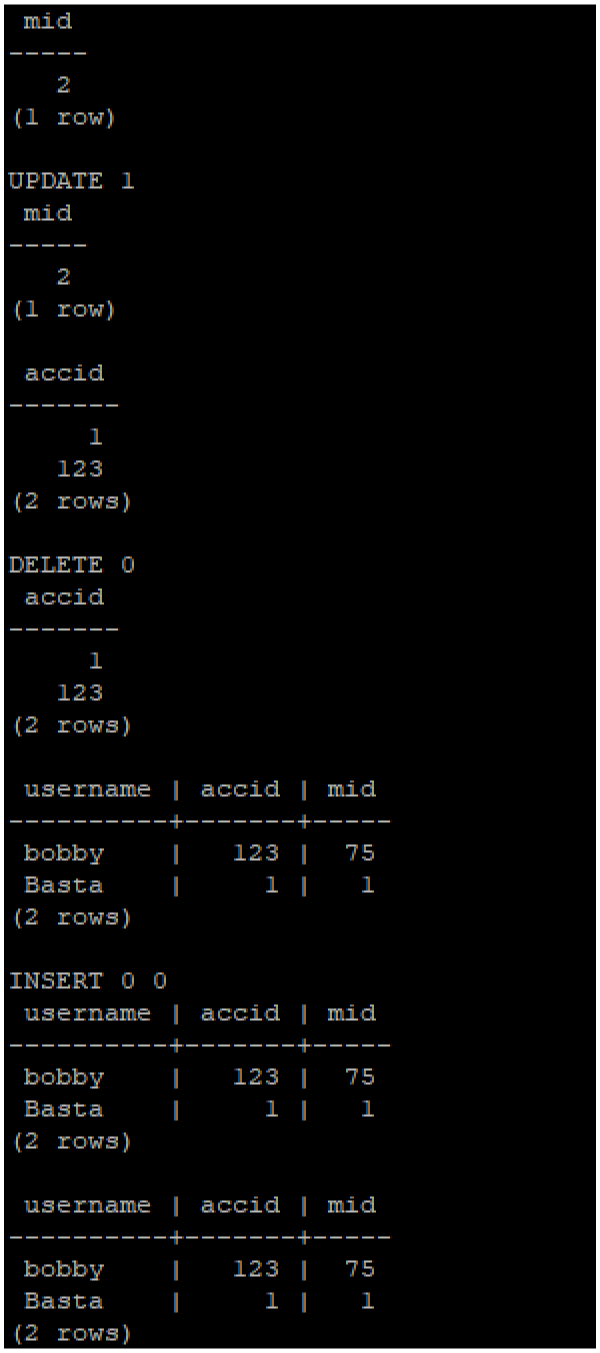
PART 6:

dataModify4.sql:

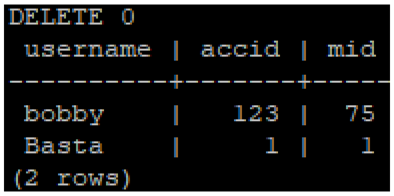


```
select mid from media where title = 'Pokemon';
update rating set value = 5 where mid in (select mid from media where title = 'Pokemon');
select mid from media where title = 'Pokemon';


select * from account;
delete from account
where not exists
    (select accid from payment
    where
          payment.accid = account.accid);
select * from account;


select * from queues;
insert into queues select 'bobby', '123', media.mid from media where title = 'Titanic';
select * from queues;

select * from queues;
delete from queues
where not exists (
    select available_in.mid from available_in
    where regname = 'Canada')
  and queues.username = 'bobby';
select * from queues;
```

dataModify4.log:



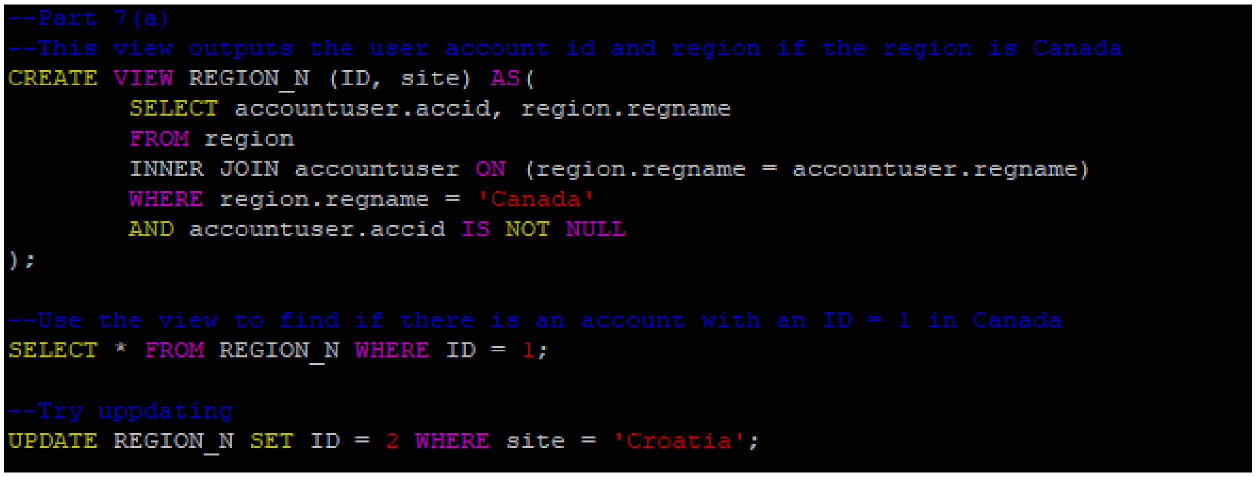
```
 mid
-----
   2
(1 row)

UPDATE 1
 mid
-----
   2
(1 row)

 accid
-------
     1
   123
(2 rows)

DELETE 0
 accid
-------
     1
   123
(2 rows)

 username | accid | mid
----------+-------+-----
 bobby    |   123 |  75
 Basta    |     1 |   1
(2 rows)

INSERT 0 0
 username | accid | mid
----------+-------+-----
 bobby    |   123 |  75
 Basta    |     1 |   1
(2 rows)

 username | accid | mid
----------+-------+-----
 bobby    |   123 |  75
 Basta    |     1 |   1
(2 rows)
```



```
DELETE 0
 username | accid | mid
----------+-------+-----
 bobby    |   123 |  75
 Basta    |     1 |   1
(2 rows)
```

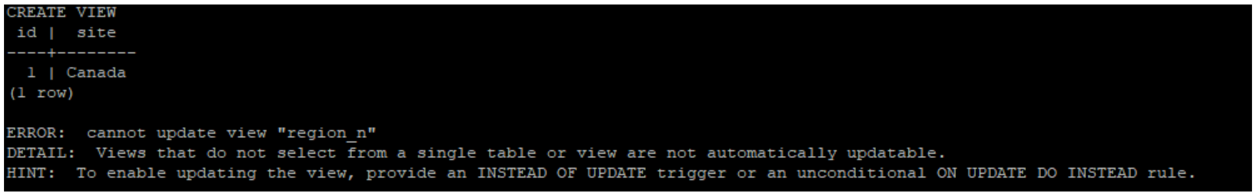
PART 7:

view1.sql:



```
--Part 7(a)
--This view outputs the user account id and region if the region is Canada
CREATE VIEW REGION_N (ID, site) AS(
        SELECT accountuser.accid, region.regname
        FROM region
        INNER JOIN accountuser ON (region.regname = accountuser.regname)
        WHERE region.regname = 'Canada'
        AND accountuser.accid IS NOT NULL
);

--Use the view to find if there is an account with an ID = 1 in Canada
SELECT * FROM REGION_N WHERE ID = 1;

--Try uppdating
UPDATE REGION_N SET ID = 2 WHERE site = 'Croatia';
```

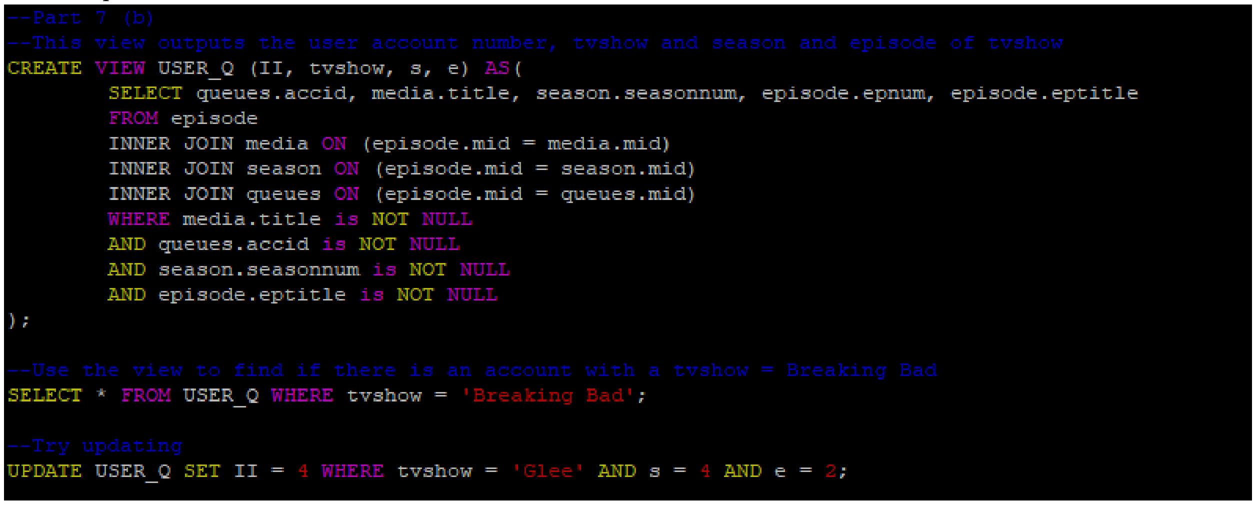
view1.log:



```
CREATE VIEW
 id |  site
----+--------
  1 | Canada
(1 row)

ERROR:  cannot update view "region_n"
DETAIL:  Views that do not select from a single table or view are not automatically updatable.
HINT:  To enable updating the view, provide an INSTEAD OF UPDATE trigger or an unconditional ON UPDATE DO INSTEAD rule.
```

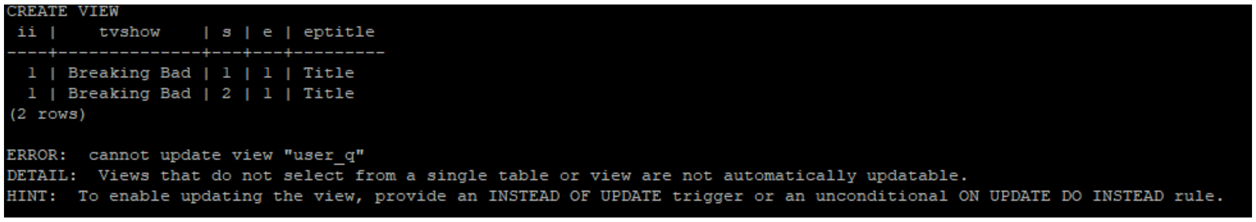
The view1.sql was not updateable because it used JOIN in its composition and incorporating multiple tables in your view will not allow it to be automatically updateable.

view2.sql:



```
--Part 7 (b)
--This view outputs the user account number, tvshow and season and episode of tvshow
CREATE VIEW USER_Q (II, tvshow, s, e) AS(
        SELECT queues.accid, media.title, season.seasonnum, episode.epnum, episode.eptitle
        FROM episode
        INNER JOIN media ON (episode.mid = media.mid)
        INNER JOIN season ON (episode.mid = season.mid)
        INNER JOIN queues ON (episode.mid = queues.mid)
        WHERE media.title is NOT NULL
        AND queues.accid is NOT NULL
        AND season.seasonnum is NOT NULL
        AND episode.eptitle is NOT NULL
);

--Use the view to find if there is an account with a tvshow = Breaking Bad
SELECT * FROM USER_Q WHERE tvshow = 'Breaking Bad';

--Try updating
UPDATE USER_Q SET II = 4 WHERE tvshow = 'Glee' AND s = 4 AND e = 2;
```

view2.log:

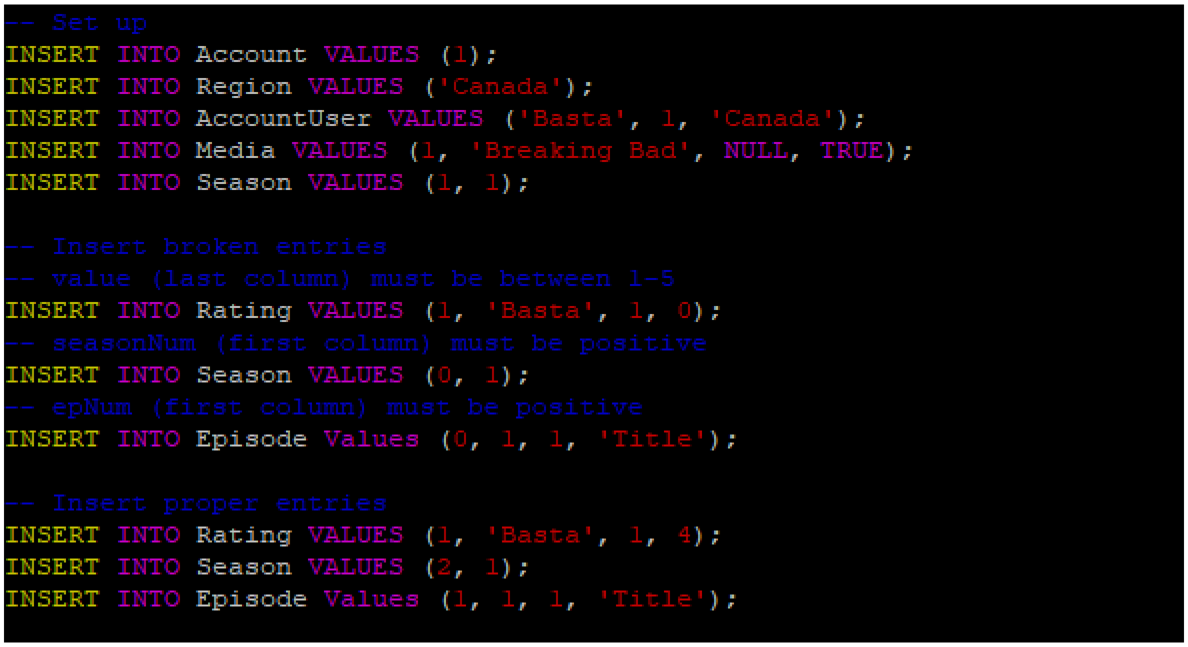


```
CREATE VIEW
 ii |    tvshow    | s | e | eptitle
----+--------------+---+---+---------
  1 | Breaking Bad | 1 | 1 | Title
  1 | Breaking Bad | 2 | 1 | Title
(2 rows)

ERROR:  cannot update view "user_q"
DETAIL:  Views that do not select from a single table or view are not automatically updatable.
HINT:  To enable updating the view, provide an INSTEAD OF UPDATE trigger or an unconditional ON UPDATE DO INSTEAD rule.
```

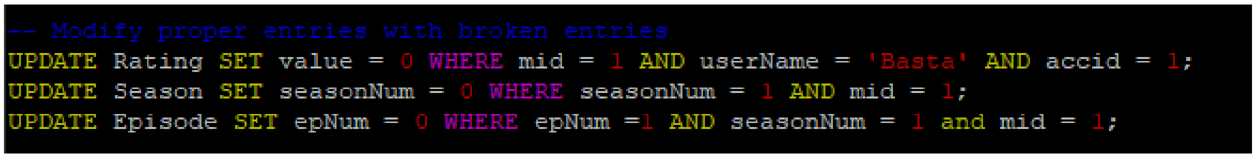
Like view1.sql, view2.sql was not updateable because it used JOIN in its composition and incorporating multiple tables in your view will not allow it to be automatically updateable.

PART 8:

checkConstraints.sql:

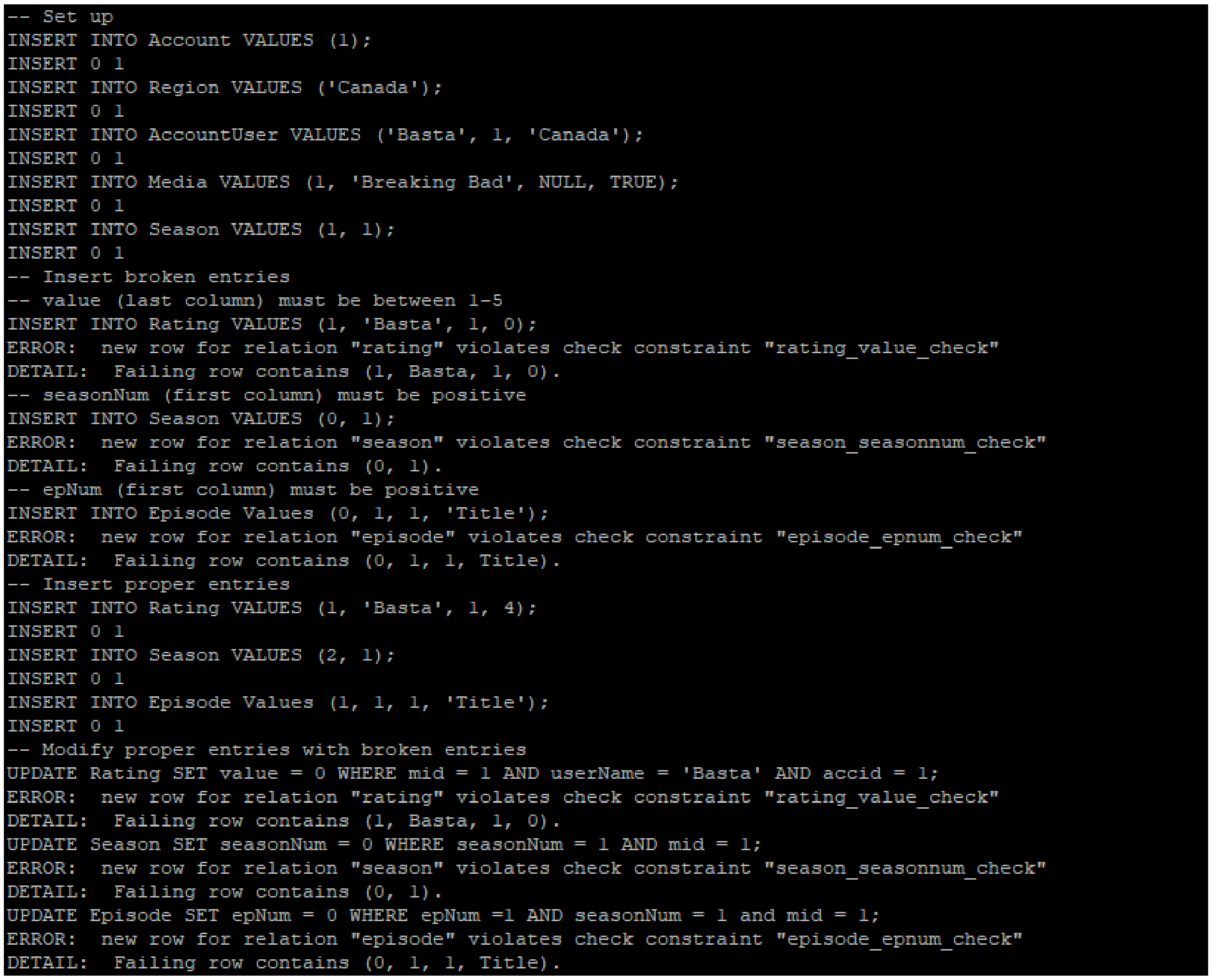


```
-- Set up
INSERT INTO Account VALUES (1);
INSERT INTO Region VALUES ('Canada');
INSERT INTO AccountUser VALUES ('Basta', 1, 'Canada');
INSERT INTO Media VALUES (1, 'Breaking Bad', NULL, TRUE);
INSERT INTO Season VALUES (1, 1);

-- Insert broken entries
-- value (last column) must be between 1-5
INSERT INTO Rating VALUES (1, 'Basta', 1, 0);
-- seasonNum (first column) must be positive
INSERT INTO Season VALUES (0, 1);
-- epNum (first column) must be positive
INSERT INTO Episode Values (0, 1, 1, 'Title');

-- Insert proper entries
INSERT INTO Rating VALUES (1, 'Basta', 1, 4);
INSERT INTO Season VALUES (2, 1);
INSERT INTO Episode Values (1, 1, 1, 'Title');
```



```
-- Modify proper entries with broken entries
UPDATE Rating SET value = 0 WHERE mid = 1 AND userName = 'Basta' AND accid = 1;
UPDATE Season SET seasonNum = 0 WHERE seasonNum = 1 AND mid = 1;
UPDATE Episode SET epNum = 0 WHERE epNum =1 AND seasonNum = 1 and mid = 1;
```

checkConstraints.log:

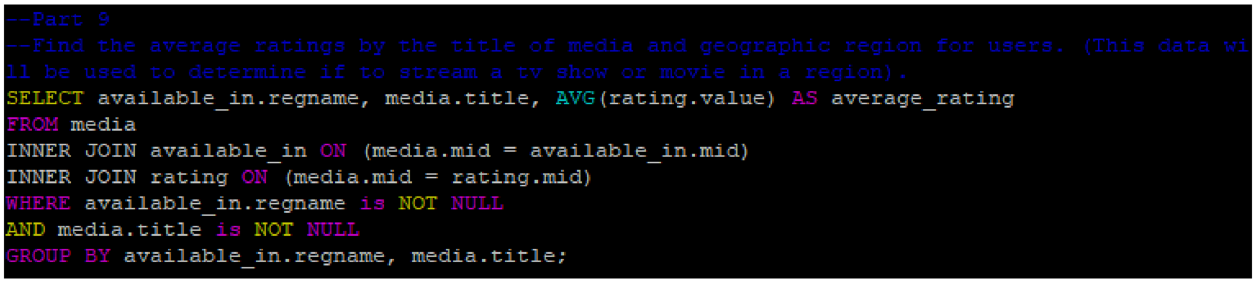


```
-- Set up
INSERT INTO Account VALUES (1);
INSERT 0 1
INSERT INTO Region VALUES ('Canada');
INSERT 0 1
INSERT INTO AccountUser VALUES ('Basta', 1, 'Canada');
INSERT 0 1
INSERT INTO Media VALUES (1, 'Breaking Bad', NULL, TRUE);
INSERT 0 1
INSERT INTO Season VALUES (1, 1);
INSERT 0 1
-- Insert broken entries
-- value (last column) must be between 1-5
INSERT INTO Rating VALUES (1, 'Basta', 1, 0);
ERROR:  new row for relation "rating" violates check constraint "rating_value_check"
DETAIL:  Failing row contains (1, Basta, 1, 0).
-- seasonNum (first column) must be positive
INSERT INTO Season VALUES (0, 1);
ERROR:  new row for relation "season" violates check constraint "season_seasonnum_check"
DETAIL:  Failing row contains (0, 1).
-- epNum (first column) must be positive
INSERT INTO Episode Values (0, 1, 1, 'Title');
ERROR:  new row for relation "episode" violates check constraint "episode_epnum_check"
DETAIL:  Failing row contains (0, 1, 1, Title).
-- Insert proper entries
INSERT INTO Rating VALUES (1, 'Basta', 1, 4);
INSERT 0 1
INSERT INTO Season VALUES (2, 1);
INSERT 0 1
INSERT INTO Episode Values (1, 1, 1, 'Title');
INSERT 0 1
-- Modify proper entries with broken entries
UPDATE Rating SET value = 0 WHERE mid = 1 AND userName = 'Basta' AND accid = 1;
ERROR:  new row for relation "rating" violates check constraint "rating_value_check"
DETAIL:  Failing row contains (1, Basta, 1, 0).
UPDATE Season SET seasonNum = 0 WHERE seasonNum = 1 AND mid = 1;
ERROR:  new row for relation "season" violates check constraint "season_seasonnum_check"
DETAIL:  Failing row contains (0, 1).
UPDATE Episode SET epNum = 0 WHERE epNum =1 AND seasonNum = 1 and mid = 1;
ERROR:  new row for relation "episode" violates check constraint "episode_epnum_check"
DETAIL:  Failing row contains (0, 1, 1, Title).
```

PART 9:

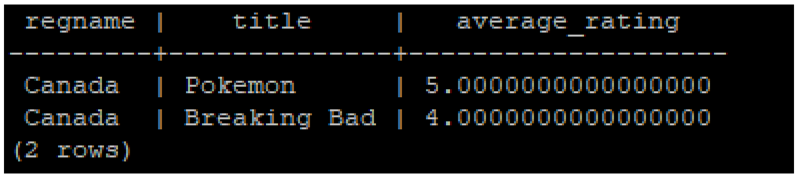
COMPLEX QUERY:

complexQuery.sql:



```
--Part 9
--Find the average ratings by the title of media and geographic region for users. (This data wi
ll be used to determine if to stream a tv show or movie in a region).
SELECT available_in.regname, media.title, AVG(rating.value) AS average_rating
FROM media
INNER JOIN available_in ON (media.mid = available_in.mid)
INNER JOIN rating ON (media.mid = rating.mid)
WHERE available_in.regname is NOT NULL
AND media.title is NOT NULL
GROUP BY available_in.regname, media.title;
```

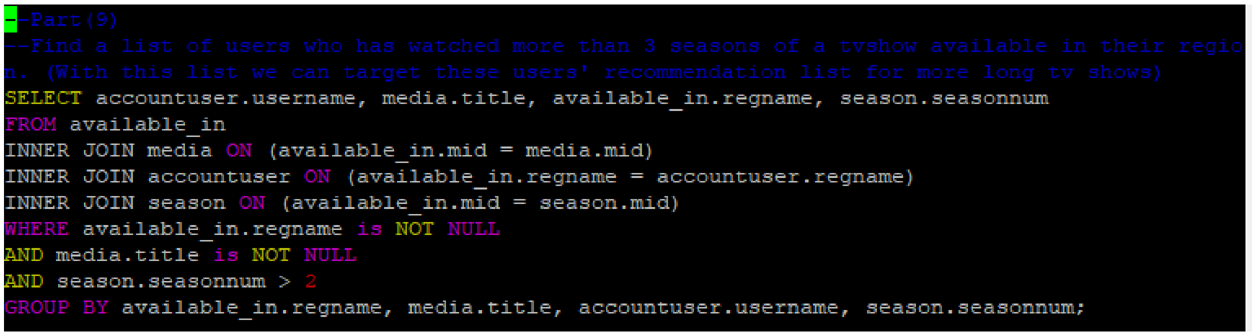
complexQuery.log:



```
 regname |     title      |    average_rating
---------+----------------+--------------------
 Canada  | Pokemon        | 5.0000000000000000
 Canada  | Breaking Bad   | 4.0000000000000000
(2 rows)
```

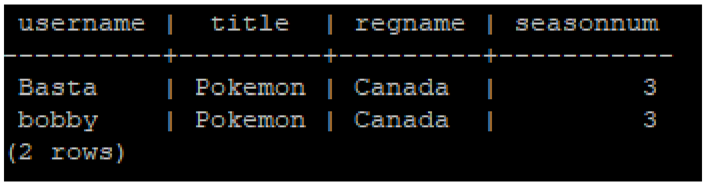
Business strategy for this complex query: Find the average ratings by the title of media and geographic region for users. (This data will be used to determine if to stream a tv show or movie in a region).

complexQuery1.sql:



```
--Part (9)
--Find a list of users who has watched more than 3 seasons of a tvshow available in their regio
n. (With this list we can target these users' recommendation list for more long tv shows)
SELECT accountuser.username, media.title, available_in.regname, season.seasonnum
FROM available_in
INNER JOIN media ON (available_in.mid = media.mid)
INNER JOIN accountuser ON (available_in.regname = accountuser.regname)
INNER JOIN season ON (available_in.mid = season.mid)
WHERE available_in.regname is NOT NULL
AND media.title is NOT NULL
AND season.seasonnum > 2
GROUP BY available_in.regname, media.title, accountuser.username, season.seasonnum;
```

complexQuery1.log:



```
 username |  title  | regname | seasonnum
----------+---------+---------+-----------
 Basta    | Pokemon | Canada  |         3
 bobby    | Pokemon | Canada  |         3
(2 rows)
```

Business strategy for this complex query: -Find a list of users who has watched more than 3 seasons of a tvshow available in their region. (With this list we can target these users' recommendation list for more long tv shows)