CS3120 Database Management Systems Laboratory

Assignment 5

Mayank Singla
111901030

Using VSCode to write the queries in a file and then execute that file from the terminal using "`\i file.sql`"

Data Types

Q1. Update the address of the customer to now hold their exact coordinates (latitude and longitude) with precision up to 18 digits and the decimal part scaling over 15 digits. Insert at least one record with address ID = '123456' into the modified address and display (only) this record.

```
3   ALTER TABLE address
4   ADD COLUMN latitude DECIMAL(18, 15) CHECK (
5           latitude >= -90
6           AND latitude <= 90
7       );
    ▶ Run SQL
8   ALTER TABLE address
9   ADD COLUMN longitude DECIMAL(18, 15) CHECK (
10          longitude >= -180
11          AND longitude <= 180
12      );
    ▶ Run SQL
13  UPDATE address
14  SET longitude = 27.2046,
15      latitude = 77.4977;
```

```
dvdrental=# \i solution.sql
ALTER TABLE
ALTER TABLE
UPDATE 603
dvdrental=#
```

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

▶ Run SQL

```sql
INSERT INTO address(
        address_id,
        address,
        district,
        city_id,
        postal_code,
        phone,
        latitude,
        longitude
    )
VALUES (
        123456,
        '102 Ahalia Hostel',
        'Palakkad',
        495,
        '678557',
        '919888888888',
        27.2046,
        77.4977
    );
```

```
dvdrental=# \i solution.sql
INSERT 0 1
```

▶ Run SQL

```sql
SELECT *
FROM address
WHERE address_id = 123456;
```

```
 address_id |      address      | address2 | district | city_id | postal_code |
   phone     |        last_update         |      latitude      |     longitude

-----------+-------------------+----------+----------+---------+-------------+-
-----------+----------------------------+--------------------+---------------
----
     123456 | 102 Ahalia Hostel |          | Palakkad |     495 | 678557      |
919888888888 | 2022-03-04 07:59:23.631218 | 27.204600000000000 | 77.497700000000
000
(1 row)

~
```

Q2. Update the payment details of the customer to now hold the transaction id corresponding to each payment that occurred. The transaction id is a 32-digit unique identifier that is automatically generated by the computer. Display at least 5 payment records after adding the transaction ID.

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

```
39  /
    ************************************************************
    ***************/
40  /* Q2: Update the payment details of the customer to now
    hold the transaction id corresponding to each payment that
    occurred.The transaction id is a 32 - digit unique
    identifier that is automatically generated by the computer.
    Display at least 5 payment records after adding the
    transaction ID. */
    ▶ Run SQL
41  ALTER TABLE payment
42  ADD COLUMN transaction_id UUID DEFAULT UUID_GENERATE_V4();
    ▶ Run SQL
43  SELECT *
44  FROM payment
45  LIMIT 6;
46
```

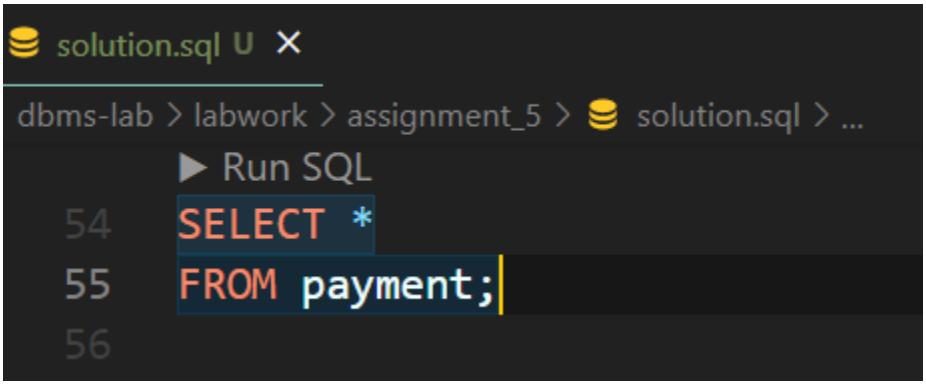
```
dvdrental=# \i solution.sql
ALTER TABLE
 payment_id | customer_id | staff_id | rental_id | amount |        payment_date        |            transaction_id
------------+-------------+----------+-----------+--------+----------------------------+--------------------------------------
      17503 |         341 |        2 |      1520 |   7.99 | 2007-02-15 22:25:46.996577 | 768f081f-b50f-49ec-9248-74f17b1d2e49
      17504 |         341 |        1 |      1778 |   1.99 | 2007-02-16 17:23:14.996577 | e934217d-7b57-43e4-a8c7-5873e9d04fba
      17505 |         341 |        1 |      1849 |   7.99 | 2007-02-16 22:41:45.996577 | 445a2041-b242-4a7b-9047-3a792171a9b2
      17506 |         341 |        2 |      2829 |   2.99 | 2007-02-19 19:39:56.996577 | 9f49ce5a-62c2-4888-97af-c6916055fd4a
      17507 |         341 |        2 |      3130 |   7.99 | 2007-02-20 17:31:48.996577 | bf47fe7d-a80c-4881-9850-2bce61d7cc02
      17508 |         341 |        1 |      3382 |   5.99 | 2007-02-21 12:33:49.996577 | 06827e93-ee65-4320-8a66-fcc807cbdf72
(6 rows)

dvdrental=#
```

Q3. Update the payment details of the customer to now hold the payment info corresponding to each payment that occurred. The payment info is a nested JSON string. Copy and insert the following JSON data after modifying the payment details and display the same.

```
▶ Run SQL
50  ALTER TABLE payment
51  ADD COLUMN payment_info JSON;
▶ Run SQL
52  UPDATE payment
53  SET payment_info = '{"intent": "sale", "payer": {
    "payment_method": "paypal" }, "redirect_urls": {
    "return_url": "http://return.url", "cancel_url": "http://
    cancel.url" }, "transactions": [{ "item_list": { "items": [
    { "name": "item", "sku": "item", "price": "1.00",
    "currency": "USD", "quantity": "1" }]}, "amount": {
    "currency": "USD", "total": "1.00" }, "description": "This
    is the payment information." }] }';
54
```

```
dvdrental=# \i solution.sql
ALTER TABLE
UPDATE 14596
dvdrental=#
```

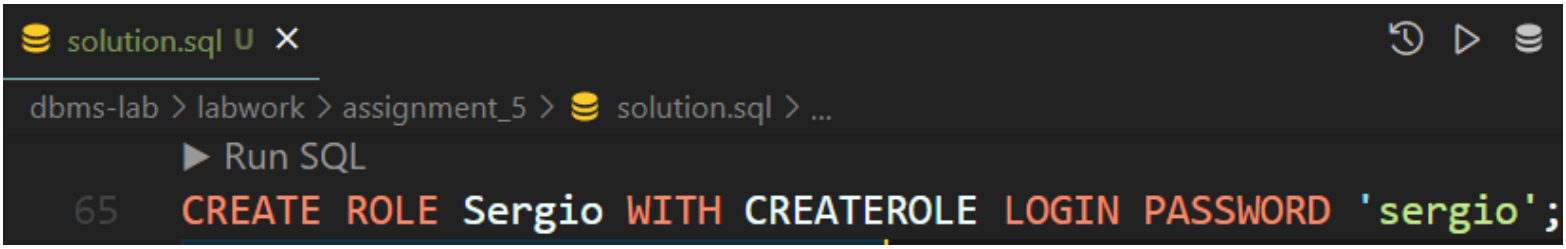



```
 payment_id | customer_id | staff_id | rental_id | amount |        payment_date        |
          transaction_id                          |
                                                  payment_info


------------+-------------+----------+-----------+--------+----------------------------+-
-------------------------------------+-----------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------
      17510 |         342 |        1 |      2914 |   5.99 | 2007-02-20 02:11:44.996577 |
acf9416b-e353-4435-9f45-9e61f02395e4 | {"intent": "sale", "payer": { "payment_method": "p
aypal" }, "redirect_urls": { "return_url": "http://return.url", "cancel_url": "http://can
cel.url" }, "transactions": [{ "item_list": { "items": [{ "name": "item", "sku": "item",
"price": "1.00", "currency": "USD", "quantity": "1" }]}, "amount": { "currency": "USD", "
total": "1.00" }, "description": "This is the payment information." }] }
      17511 |         342 |        1 |      3081 |   2.99 | 2007-02-20 13:57:39.996577 |
20d95f27-f636-4c3d-b230-4f8276e1e820 | {"intent": "sale", "payer": { "payment_method": "p
aypal" }, "redirect_urls": { "return_url": "http://return.url", "cancel_url": "http://can
cel.url" }, "transactions": [{ "item_list": { "items": [{ "name": "item", "sku": "item",
"price": "1.00", "currency": "USD", "quantity": "1" }]}, "amount": { "currency": "USD", "
total": "1.00" }, "description": "This is the payment information." }] }
      17512 |         343 |        2 |      1547 |   4.99 | 2007-02-16 00:10:50.996577 |
80222e33-8776-4ed3-9c72-c4a0948dcfa8 | {"intent": "sale", "payer": { "payment_method": "p
:
```

Users, Roles, and Authorization

Q1. Add a new user ‘Sergio’ who has the role of an owner and can create new roles for new/existing users in the database.
a. Display all active users and their roles.
b. Restart the psql session as user ‘Sergio’ and try creating a database. Share the screenshot of the output.




```
mayank=# \i solution.sql
CREATE ROLE
mayank=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 mayank    | Superuser                                                  | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 sergio    | Create role                                                | {}

mayank=#
```

```
└─ psql -U sergio -d dvdrental
Password for user sergio:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=> CREATE DATABASE sergio;
ERROR:  permission denied to create database
dvdrental=>
```

Q2. (Logged-in as ‘Sergio’ only) Create the role of a ‘Receptionist’ who manages only the rental data, the customer data, and the payment details. A receptionist can perform any data modification over the aforementioned data (fetch / add / delete / update records).

a. Display all privileges for the ‘Receptionist’ over the specified data scope.

b. Restart the psql session as user ‘Receptionist’ and try displaying all staff records using its appropriate relation. Share the screenshot of the output.

```
└─ psql -U sergio -d dvdrental
Password for user sergio:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=> CREATE ROLE Receptionist WITH LOGIN PASSWORD 'receptionist';
CREATE ROLE
dvdrental=> \du
                                   List of roles
  Role name   |                         Attributes                         | Member of
--------------+------------------------------------------------------------+-----------
 mayank       | Superuser                                                  | {}
 postgres     | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 receptionist |                                                            | {}
 sergio       | Create role                                                | {}

dvdrental=>
```

```
mayank=# ALTER ROLE sergio WITH SUPERUSER;
ALTER ROLE
mayank=# \du
                                   List of roles
  Role name   |                         Attributes                         | Member of
--------------+------------------------------------------------------------+-----------
 mayank       | Superuser                                                  | {}
 postgres     | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 receptionist |                                                            | {}
 sergio       | Superuser, Create role                                     | {}

mayank=#
```

```
└─ psql -U sergio -d dvdrental
Password for user sergio:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=# GRANT ALL ON rental, payment, customer TO receptionist;
GRANT
dvdrental=#
```

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

```sql
SELECT grantee,
    table_name,
    STRING_AGG(privilege_type, ', ') AS privileges
FROM information_schema.role_table_grants
WHERE (
        table_name = 'rental'
        OR table_name = 'customer'
        OR table_name = 'payment'
    )
    AND grantee = 'receptionist'
GROUP BY grantee,
    table_name;
```

```
dvdrental=# \i solution.sql
   grantee    | table_name |                          privileges
--------------+------------+---------------------------------------------------------------
 receptionist | customer   | TRIGGER, REFERENCES, TRUNCATE, INSERT, DELETE, UPDATE, SELECT
 receptionist | payment    | DELETE, INSERT, SELECT, UPDATE, TRUNCATE, REFERENCES, TRIGGER
 receptionist | rental     | INSERT, TRIGGER, REFERENCES, TRUNCATE, DELETE, UPDATE, SELECT
(3 rows)

dvdrental=#
```

```
dvdrental=# \dp rental
                                Access privileges
 Schema |  Name  | Type  |      Access privileges       | Column privileges | Policies
--------+--------+-------+------------------------------+-------------------+----------
 public | rental | table | postgres=arwdDxt/postgres    +|                   |
        |        |       | receptionist=arwdDxt/postgres |                   |
(1 row)

dvdrental=# \dp customer
                                Access privileges
 Schema |   Name   | Type  |      Access privileges       | Column privileges | Policies
--------+----------+-------+------------------------------+-------------------+----------
 public | customer | table | postgres=arwdDxt/postgres    +|                   |
        |          |       | receptionist=arwdDxt/postgres |                   |
(1 row)

dvdrental=# \dp payment
                                Access privileges
 Schema |  Name   | Type  |      Access privileges       | Column privileges | Policies
--------+---------+-------+------------------------------+-------------------+----------
 public | payment | table | postgres=arwdDxt/postgres    +|                   |
        |         |       | receptionist=arwdDxt/postgres |                   |
(1 row)

dvdrental=#
```

```
└─ psql -U receptionist -d dvdrental
Password for user receptionist:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=> SELECT * FROM staff;
ERROR:  permission denied for table staff
dvdrental=>
```

NOTE: Restart the session as a superuser (postgre) to attempt questions 3 & 4.

Q3. Create a new user ‘DB Administrator’ having complete access to the entire database. Log in as the administrator and modify the roles of the owner so that the owner now has all the privileges, not just creating roles. Display all the modified privileges for the owner.

```
└─ psql -U postgres -d dvdrental
Password for user postgres:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=# CREATE ROLE "DB Administrator" WITH SUPERUSER LOGIN PASSWORD 'dbadmin';
CREATE ROLE
dvdrental=#
```

```
└─ psql -U postgres -d dvdrental
Password for user postgres:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=# create role owner createrole;
CREATE ROLE
dvdrental=# \du
                                   List of roles
    Role name     |                         Attributes                         | Member of
------------------+----------------------------------------------------------------+-----------
 DB Administrator | Superuser                                                      | {}
 mayank           | Superuser                                                      | {}
 owner            | Create role, Cannot login                                      | {}
 postgres         | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 receptionist     |                                                                | {}
 sergio           | Superuser, Create role                                         | {}
```

```
└─ psql -U "DB Administrator" -d dvdrental
Password for user DB Administrator:
psql (14.2 (Ubuntu 14.2-1.pgdg20.04+1))
Type "help" for help.

dvdrental=# ALTER ROLE owner WITH SUPERUSER;
ALTER ROLE
dvdrental=# \du
                                   List of roles
    Role name     |                         Attributes                         | Member of
------------------+------------------------------------------------------------+-----------
 DB Administrator | Superuser                                                  | {}
 mayank           | Superuser                                                  | {}
 owner            | Superuser, Create role, Cannot login                       | {}
 postgres         | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 receptionist     |                                                            | {}
 sergio           | Superuser, Create role                                     | {}

dvdrental=#
```

Q4. Create a new role of a ‘Manager’ and assign a new user ‘Max’ as the manager who requires a login to access the records. Create another role ‘Logged_in’ which keeps track of all the active users. Let ‘Max’ be logged in, thereby, is assigned the respective role.
a. Display all the users currently active in the database.
b. Remove the role of a manager completely but do not remove the employees who were managers previously. Share the screenshot of the status of the users in the system

```
dvdrental=# CREATE ROLE Manager;
CREATE ROLE
dvdrental=# CREATE ROLE Max;
CREATE ROLE
dvdrental=# CREATE ROLE Logged_in;
CREATE ROLE
dvdrental=# GRANT manager TO max;
GRANT ROLE
dvdrental=# GRANT logged_in TO max;
GRANT ROLE
dvdrental=# \du
                                   List of roles
    Role name     |                         Attributes                         |      Member of
------------------+----------------------------------------------------------------+---------------------
 DB Administrator | Superuser                                                      | {}
 logged_in        | Cannot login                                                   | {}
 manager          | Cannot login                                                   | {}
 max              | Cannot login                                                   | {manager,logged_in}
 mayank           | Superuser                                                      | {}
 owner            | Superuser, Create role, Cannot login                           | {}
 postgres         | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 receptionist     |                                                                | {}
 sergio           | Superuser, Create role                                         | {}

dvdrental=#
```

```
dvdrental=# DROP ROLE manager;
DROP ROLE
dvdrental=# \du
                                   List of roles
    Role name     |                         Attributes                         |  Member of
------------------+----------------------------------------------------------------+-------------
 DB Administrator | Superuser                                                      | {}
 logged_in        | Cannot login                                                   | {}
 max              | Cannot login                                                   | {logged_in}
 mayank           | Superuser                                                      | {}
 owner            | Superuser, Create role, Cannot login                           | {}
 postgres         | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 receptionist     |                                                                | {}
 sergio           | Superuser, Create role                                         | {}

dvdrental=#
```

Functions and Procedures

Q1. Create a function that consumes an ‘ID’ number as an argument and returns the number of films rented by the customer corresponding to that particular ID.

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

```
69  CREATE OR REPLACE FUNCTION get_num_films_by_cid(
70      cid INT
71  )
72  RETURNS INT AS $$
73  DECLARE
74      result INT;
    ▶ Run SQL
75  BEGIN
76      SELECT DISTINCT COUNT(film_id) INTO result
77      FROM rental
78          INNER JOIN customer ON rental.customer_id = customer.
            customer_id
79          INNER JOIN inventory ON rental.inventory_id = inventory.
            inventory_id
80      GROUP BY rental.customer_id
81      HAVING rental.customer_id = cid;
        ▶ Run SQL
82      RETURN result;
    ▶ Run SQL
83  END;
    ▶ Run SQL
84  $$ LANGUAGE plpgsql;
    ▶ Run SQL
85  SELECT *
86  FROM get_num_films_by_cid(459);
87
```

```
dvdrental=# \i solution.sql
CREATE FUNCTION
 get_num_films_by_cid
----------------------
                   38
(1 row)

dvdrental=#
```

Q2. Create a function that consumes an ‘ID’ number as an argument and returns the complete details of the customer corresponding to that particular ID ranging from the customer name (full), their address ID, and the amount paid by them including the payment date.

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

```
89  CREATE OR REPLACE FUNCTION get_customer_info(
90      cid INT
91  )
92  RETURNS TABLE(
93      first_name VARCHAR,
94      last_name VARCHAR,
95      address_id SMALLINT,
96      amount NUMERIC,
97      payment_date TIMESTAMP
98  )
99  AS $$
100 DECLARE
101     result INT;
    ▶ Run SQL
102 BEGIN
103     RETURN QUERY
104         SELECT customer.first_name,
105             customer.last_name,
106             customer.address_id,
107             payment.amount,
108             payment.payment_date
109         FROM customer
110             INNER JOIN payment ON customer.customer_id = payment.
                customer_id AND customer.customer_id = cid;
    ▶ Run SQL
111 END;
    ▶ Run SQL
112 $$ LANGUAGE plpgsql;
```

```
dvdrental=# \i solution.sql
CREATE FUNCTION
dvdrental=#
```

Q3. Create a function that consumes 'actor ID' as an argument and returns all the films the actor corresponding to the input actor ID has acted in. The function output should be a well-organized table enveloping the film ID, film title, and the complete name (first & last) of the actor.

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

```
115 CREATE OR REPLACE FUNCTION get_actor_films(
116     aid INT
117 )
118 RETURNS TABLE(
119     film_id INT,
120     title VARCHAr,
121     first_name VARCHAR,
122     last_name VARCHAR
123 )
124 AS $$
125 BEGIN
126     RETURN QUERY
127         SELECT DISTINCT film.film_id,
128             film.title,
129             actor.first_name,
130             actor.last_name
131         FROM actor
132             INNER JOIN film_actor ON actor.actor_id = film_actor.
            actor_id AND actor.actor_id = aid
133             INNER JOIN film ON film_actor.film_id = film.film_id;
    ▶ Run SQL
134 END;
    ▶ Run SQL
135 $$ LANGUAGE plpgsql;
136
```

```
dvdrental=# \i solution.sql
CREATE FUNCTION
dvdrental=# select * from get_actor_films(1);
 film_id |         title          | first_name | last_name
---------+------------------------+------------+-----------
       1 | Academy Dinosaur       | Penelope   | Guiness
      23 | Anaconda Confessions   | Penelope   | Guiness
      25 | Angels Life            | Penelope   | Guiness
     106 | Bulworth Commandments  | Penelope   | Guiness
     140 | Cheaper Clyde          | Penelope   | Guiness
     166 | Color Philadelphia     | Penelope   | Guiness
     277 | Elephant Trojan        | Penelope   | Guiness
     361 | Gleaming Jawbreaker    | Penelope   | Guiness
     438 | Human Graffiti         | Penelope   | Guiness
     499 | King Evolution         | Penelope   | Guiness
     506 | Lady Stage             | Penelope   | Guiness
     509 | Language Cowboy        | Penelope   | Guiness
     605 | Mulholland Beast       | Penelope   | Guiness
     635 | Oklahoma Jumanji       | Penelope   | Guiness
     749 | Rules Human            | Penelope   | Guiness
     832 | Splash Gump            | Penelope   | Guiness
     939 | Vertigo Northwest      | Penelope   | Guiness
     970 | Westward Seabiscuit    | Penelope   | Guiness
     980 | Wizard Coldblooded     | Penelope   | Guiness
(19 rows)

dvdrental=#
```

Q4. Create a procedure that displays all the film IDs in ascending order and the name (title) of each film corresponding to that particular ID.

solution.sql U

dbms-lab > labwork > assignment_5 > solution.sql > ...

```
▶ Run SQL
CREATE OR REPLACE PROCEDURE display_films()
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
▶ Run SQL
BEGIN
    FOR rec IN (SELECT film_id, title FROM film ORDER BY film_id ASC)
    LOOP
        RAISE NOTICE 'film_id: %, title: %', rec.film_id, rec.title;
    ▶ Run SQL
    END LOOP;
▶ Run SQL
END;
▶ Run SQL
$$;
```

```
dvdrental=# \i solution.sql
CREATE PROCEDURE
dvdrental=#
```

```
dvdrental=# CALL display_films();
NOTICE:  film_id: 1, title: Academy Dinosaur
NOTICE:  film_id: 2, title: Ace Goldfinger
NOTICE:  film_id: 3, title: Adaptation Holes
NOTICE:  film_id: 4, title: Affair Prejudice
NOTICE:  film_id: 5, title: African Egg
NOTICE:  film_id: 6, title: Agent Truman
NOTICE:  film_id: 7, title: Airplane Sierra
NOTICE:  film_id: 8, title: Airport Pollock
NOTICE:  film_id: 9, title: Alabama Devil
NOTICE:  film_id: 10, title: Aladdin Calendar
NOTICE:  film_id: 11, title: Alamo Videotape
NOTICE:  film_id: 12, title: Alaska Phantom
NOTICE:  film_id: 13, title: Ali Forever
NOTICE:  film_id: 14, title: Alice Fantasia
NOTICE:  film_id: 15, title: Alien Center
```

```
NOTICE:  film_id: 990, title: World Leathernecks
NOTICE:  film_id: 991, title: Worst Banger
NOTICE:  film_id: 992, title: Wrath Mile
NOTICE:  film_id: 993, title: Wrong Behavior
NOTICE:  film_id: 994, title: Wyoming Storm
NOTICE:  film_id: 995, title: Yentl Idaho
NOTICE:  film_id: 996, title: Young Language
NOTICE:  film_id: 997, title: Youth Kick
NOTICE:  film_id: 998, title: Zhivago Core
NOTICE:  film_id: 999, title: Zoolander Fiction
NOTICE:  film_id: 1000, title: Zorro Ark
CALL
dvdrental=#
```