Assignment #1
DVD Rental Database Modelling using Postgres SQL

The given assignment is on DVD Rental database and the Entity-Relationship (ER) model diagram represents the schema of a DVD rental database system.
Comprehensive evaluation of schema are:-

Tables: The diagram shows 15 tables, each representing a different entities of the DVD rental business. Here are the tables

category: Contains information about the different categories of films available for rent.
Attributes -- category_id, name

film_category : A junction table that associates films with their respective categories
Attributes ---- film_id ,category_id

film: Holds details about the films,
Attributes --- film_id, title, description, release_year , language_id,rental_duration,, rental_rate, lengt, replacement_cost,rating, lastupdate,special_features,fulltext

language: Stores languages that films are available in,
Attributes --language_id and name.

film_actor: Another junction table linking actors to the films they've appeared in
Attributes --actor_id and film_id.

actor: Contains data on actors
Attributes -- actor_id, first_name, and last_name.

inventory: Tracks the physical copies of films available at different store locations
Attributes --inventory_id, film_id, and store_id.

rental: Records the details of rentals
Attributes --rental_id, rental_date, inventory_id, customer_id, and return_date,staff_id.

customer: Stores customer information
Attributes --customer_id, store_id, first_name, last_name, email, address_id, activebool(status indicator), create-date,active

address: Holds the address information for both customers and staff
Attributes -- address_id, address, district, city_id, and postal_code, phone.

city: Contains city details
Attributes -- city_id, city, and country_id.

country: Lists countries
Attributes --country_id and country.

store: Represents the physical locations of the DVD rental stores
Attributes -- store_id, manager_staff_id, and address_id.

Staff: Contains information about the staff members
Attributes -- staff_id, first_name, last_name, email, store_id, and username ,password, picture

payment: Records payment transactions
Attributes -- payment_id, customer_id, staff_id, rental_id, and amount.

2. Relationships:
Relationships in an ER diagram show how different entities interact with each other

- category - This table has a one-to-many relationship with film_category as each category can be associated with many films but each film_category association involves one category.
- film_category - Serves as a junction table between film and category, implementing a many-to-many relationship where a film can belong to many categories and a category can include many films. It has foreign keys from both film and category.
- film - It has a one-to-many relationship with inventory, a many-to-many relationship with actor through film_actor, and a one-to-many relationship with film_category.
- language - It has a one-to-many relationship with film, meaning each language can be associated with multiple films, but each film is associated with only one language.
- film_actor - This is a junction table for the many-to-many relationship between film and actor, meaning an actor can star in many films, and a film can have many actors.
- actor - It is related to film_actor as one actor can be related to many film_actor entries (an actor can be in multiple films).
- inventory - It has a one-to-many relationship with rental, as each inventory item can be rented multiple times, and a many-to-one relationship with store, meaning each inventory item belongs to one store.
- rental - It relates to inventory, customer, payment, and staff. Each rental involves one inventory item, one customer, may have multiple payments, and is processed by one staff member.
- customer - It has a one-to-many relationship with rental (a customer can have multiple rentals) and payment (a customer can make many payments). It is also related to address through address_id.
- address - It is related to customer and staff in a one-to-many relationship (one address can be associated with multiple customers or staff members), and to city as each address is in one city.
- city - It has a one-to-many relationship with address (each city can have multiple addresses) and is related to country as each city is located in one country.
- country - It has a one-to-many relationship with city, as a country can have multiple cities.
- store - It is related to inventory as a one-to-many relationship (a store can have many inventory items). It also has a relationship with staff where each store is managed by one

staff member (manager_staff_id) and can have multiple staff members working in it. It's also connected to address.

- staff - It has a one-to-many relationship with rental (staff can process many rentals) and payment (staff can receive many payments). Each staff member also has one address.
- payment - It has a many-to-one relationship with customer, staff, and rental, as each payment is made by one customer, processed by one staff member, and associated with one rental.

2. List primary keys and foreign keys

In an Entity-Relationship (ER) model, primary keys are unique identifiers for the records in a table. Foreign keys are identifiers that link a record to a primary key in another table, thus establishing a relationship between the two tables.

Based on the ER model of the DVD rental database here are the primary keys (PK) and foreign keys (FK) for each table:

1. Actor. - PK: actor_id
2. Address - PK: address_id , FK: city_id (links to city table)
3. Category - PK: category_id
4. City - PK: city_id , FK: country_id (links to country table)
5. Country - PK: country_id
6. Customer - PK: customer_id , FK: store_id (links to store table) , FK: address_id (links to address table)
7. Film - PK: film_id
8. film_actor- PK: Composite key (actor_id, film_id) , FK: actor_id (links to actor table) , FK: film_id (links to film table)
9. film_category - PK: Composite key (film_id, category_id) , FK: film_id (links to film table) , FK: category_id (links to category table)
10. inventory - PK: inventory_id , FK: film_id (links to film table) , FK: store_id (links to store table)
11. language - PK: language_id
12. payment - PK: payment_id , FK: customer_id (links to customer table) , FK: staff_id (links to staff table) , FK: rental_id (links to rental table)
13. rental - PK: rental_id , FK: inventory_id (links to inventory table), FK: customer_id (links to customer table) , FK: staff_id (links to staff table)
14. staff - PK: staff_id , FK: store_id (links to store table) , FK: address_id (links to address table)
15. store - PK: store_id , FK: manager_staff_id (links to staff table) , FK: address_id (links to address table)

The composite keys in film_actor and film_category are primary keys that consist of a combination of two foreign keys. They serve as both primary keys for their respective tables and as foreign keys linking back to the actor, film, and category tables.

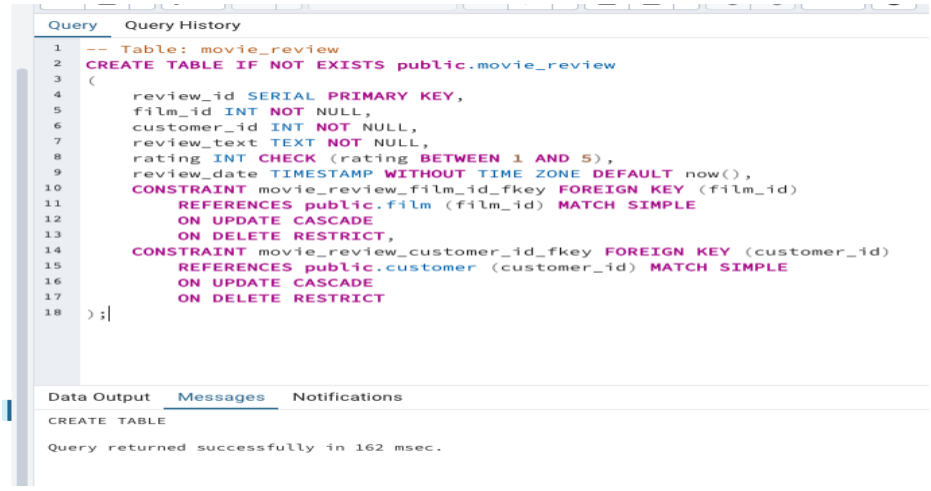
3. Create at least 3 sample Table with constraints and dummy data -Create at least three sample tables with appropriate constraints (e.g., NOT NULL, UNIQUE, CHECK) and populate them with dummy data.

I have created

1. movie_review – This table allows customers to leave reviews and ratings for films they have rented.
2. staff_schedule - To help manage the work schedules of the staff members.
3. rental_history -- To keep track of the full rental history of items, even after they are returned.
4. film_recommendations - To suggest films to customers based on their rental history or preferences.
5. late_fee - To manage late fees for overdue rentals.
6. promotions - For managing promotional offers on rentals.

These tables expand the functionality of the DVD Rental system to cover aspects like customer engagement through reviews, internal management through staff scheduling, financials through late fee tracking, marketing through promotions, and enhancing customer experience with film recommendations.

movie_review



staff_schedule

```
);

-- Table: staff_schedule
CREATE TABLE IF NOT EXISTS public.staff_schedule
(
    schedule_id SERIAL PRIMARY KEY,
    staff_id INT NOT NULL,
    work_date DATE NOT NULL,
    start_time TIME WITHOUT TIME ZONE NOT NULL,
    end_time TIME WITHOUT TIME ZONE NOT NULL,
    CONSTRAINT staff_schedule_staff_id_fkey FOREIGN KEY (staff_id)
        REFERENCES public.staff (staff_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 88 msec.
```

Rental_ history

```

CREATE TABLE IF NOT EXISTS public.rental_history
(
    history_id SERIAL PRIMARY KEY,
    rental_id INT NOT NULL,
    rental_date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    return_date TIMESTAMP WITHOUT TIME ZONE,
    customer_id INT NOT NULL REFERENCES public.customer(customer_id),
    film_id INT NOT NULL REFERENCES public.film(film_id),
    store_id INT NOT NULL,
    staff_id INT NOT NULL
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 33 msec.
```

Film Recommendations

```
);

CREATE TABLE IF NOT EXISTS public.film_recommendations
(
    recommendation_id SERIAL PRIMARY KEY,
    customer_id INT NOT NULL REFERENCES public.customer(customer_id),
    film_id INT NOT NULL REFERENCES public.film(film_id),
    recommendation_score DECIMAL(4, 2),
    last_update TIMESTAMP WITHOUT TIME ZONE DEFAULT now()
);

```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 31 msec.
```

Late_fee

```
55
56  CREATE TABLE IF NOT EXISTS public.late_fee
57  (
58      fee_id SERIAL PRIMARY KEY,
59      rental_id INT NOT NULL REFERENCES public.rental(rental_id),
60      days_overdue INT NOT NULL,
61      fee_amount DECIMAL(5, 2) NOT NULL,
62      status VARCHAR(50) NOT NULL DEFAULT 'Unpaid' CHECK (status IN ('Unpaid', 'Paid', 'Waived')),
63      last_update TIMESTAMP WITHOUT TIME ZONE DEFAULT now()
64  );
65
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 150 msec.
```

Promotions

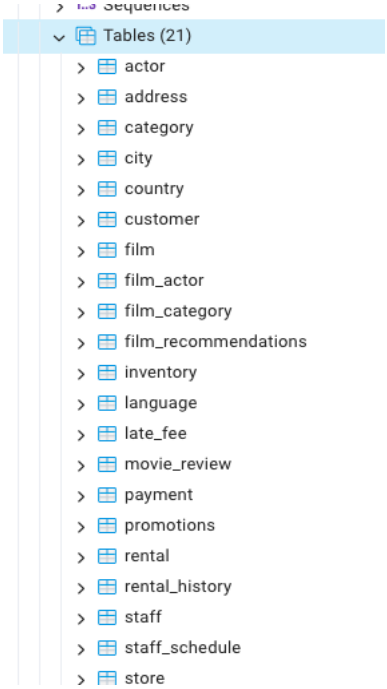
```
66
67  CREATE TABLE IF NOT EXISTS public.promotions
68  (
69      promotion_id SERIAL PRIMARY KEY,
70      name VARCHAR(255) NOT NULL,
71      description TEXT,
72      start_date DATE NOT NULL,
73      end_date DATE NOT NULL,
74      discount_rate DECIMAL(3, 2) NOT NULL CHECK (discount_rate >= 0 AND discount_rate <= 1),
75      last_update TIMESTAMP WITHOUT TIME ZONE DEFAULT now()
76  );
77
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 35 msec.
```

These are the total tables that are added in the schema of DVD_rental.



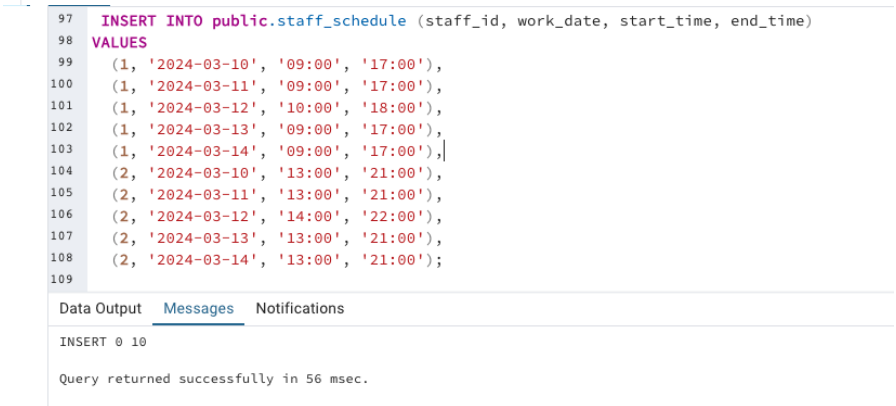
I have populated those tables with 10 rows of data.

```
INSERT INTO public.movie_review (film_id, customer_id, review_text, rating, review_date)
VALUES
  (1, 1, 'Incredible storytelling and memorable characters.', 5, '2024-01-10 10:00:00'),
  (1, 2, 'The special effects were top-notch!', 4, '2024-01-10 11:00:00'),
  (2, 1, 'A bit too slow to get going, but a great finish.', 3, '2024-01-10 12:00:00'),
  (2, 3, 'I loved the soundtrack more than the movie itself.', 4, '2024-01-10 13:00:00'),
  (1, 3, 'Would watch it again! Two thumbs up!', 5, '2024-01-10 14:00:00'),
  (3, 1, 'The acting was superb, brought the characters to life.', 4, '2024-01-10 15:00:00'),
  (4, 2, 'My kids loved it, a new family favorite.', 5, '2024-01-10 16:00:00'),
  (3, 2, 'Not as good as the original, but still worth a watch.', 3, '2024-01-10 17:00:00'),
  (2, 4, 'A solid entry in the sci-fi genre. Really made me think.', 4, '2024-01-10 18:00:00'),
  (2, 4, 'Decent movie, but I probably wouldnt watch it a second time.', 3, '2024-01-10 19:00:00');
```

Data Output | Messages | Notifications

```
INSERT 0 10

Query returned successfully in 112 msec.
```



```
INSERT INTO public.staff_schedule (staff_id, work_date, start_time, end_time)
VALUES
  (1, '2024-03-10', '09:00', '17:00'),
  (1, '2024-03-11', '09:00', '17:00'),
  (1, '2024-03-12', '10:00', '18:00'),
  (1, '2024-03-13', '09:00', '17:00'),
  (1, '2024-03-14', '09:00', '17:00'),
  (2, '2024-03-10', '13:00', '21:00'),
  (2, '2024-03-11', '13:00', '21:00'),
  (2, '2024-03-12', '14:00', '22:00'),
  (2, '2024-03-13', '13:00', '21:00'),
  (2, '2024-03-14', '13:00', '21:00');
```

Data Output | Messages | Notifications

```
INSERT 0 10

Query returned successfully in 56 msec.
```

```sql
INSERT INTO public.rental_history (rental_id, rental_date, return_date, customer_id, film_id, store_id, staff_id)
VALUES
  (101, '2024-03-01 14:00:00', '2024-03-05 10:00:00', 1, 1, 1, 1),
  (102, '2024-03-02 16:00:00', '2024-03-06 12:00:00', 2, 2, 1, 1),
  (103, '2024-03-03 18:00:00', '2024-03-07 14:00:00', 3, 3, 2, 2),
  (104, '2024-03-04 20:00:00', '2024-03-08 16:00:00', 4, 4, 2, 2),
  (105, '2024-03-05 10:00:00', '2024-03-09 10:00:00', 5, 5, 1, 1),
  (106, '2024-03-06 12:00:00', '2024-03-10 12:00:00', 6, 1, 2, 2),
  (107, '2024-03-07 14:00:00', '2024-03-11 14:00:00', 7, 2, 1, 1),
  (108, '2024-03-08 16:00:00', '2024-03-12 16:00:00', 8, 3, 2, 2),
  (109, '2024-03-09 18:00:00', '2024-03-13 18:00:00', 9, 4, 1, 1),
  (110, '2024-03-10 20:00:00', '2024-03-14 20:00:00', 10, 5, 2, 2);
```

Data Output | Messages | Notifications

```
INSERT 0 10

Query returned successfully in 65 msec.
```

```sql
INSERT INTO public.film_recommendations (customer_id, film_id, recommendation_score, last_update)
VALUES
  (1, 10, 8.5, '2024-03-10 09:00:00'),
  (2, 15, 9.0, '2024-03-10 09:15:00'),
  (3, 20, 7.5, '2024-03-10 09:30:00'),
  (4, 25, 6.0, '2024-03-10 09:45:00'),
  (5, 30, 9.5, '2024-03-10 10:00:00'),
  (1, 35, 7.0, '2024-03-10 10:15:00'),
  (2, 40, 8.0, '2024-03-10 10:30:00'),
  (3, 45, 9.2, '2024-03-10 10:45:00'),
  (4, 50, 6.5, '2024-03-10 11:00:00'),
  (5, 55, 8.8, '2024-03-10 11:15:00');
```

Data Output | Messages | Notifications

```
INSERT 0 10

Query returned successfully in 115 msec.
```

Query | Query History

```sql
INSERT INTO public.late_fee (rental_id, days_overdue, fee_amount, status, last_update)
VALUES
  (101, 2, 1.99, 'Unpaid', '2024-03-15 09:00:00'),
  (102, 1, 0.99, 'Unpaid', '2024-03-15 09:15:00'),
  (103, 3, 2.99, 'Unpaid', '2024-03-15 09:30:00'),
  (104, 5, 4.99, 'Unpaid', '2024-03-15 09:45:00'),
  (105, 2, 1.99, 'Unpaid', '2024-03-15 10:00:00'),
  (106, 4, 3.99, 'Unpaid', '2024-03-15 10:15:00'),
  (107, 1, 0.99, 'Unpaid', '2024-03-15 10:30:00'),
  (108, 6, 5.99, 'Unpaid', '2024-03-15 10:45:00'),
  (109, 2, 1.99, 'Unpaid', '2024-03-15 11:00:00'),
  (110, 7, 6.99, 'Unpaid', '2024-03-15 11:15:00');
```

Data Output | Messages | Notifications

```
INSERT 0 10

Query returned successfully in 164 msec.
```

```sql
INSERT INTO public.promotions (name, description, start_date, end_date, discount_rate, last_update)
VALUES
  ('Summer Bonanza', 'Discount on all summer blockbusters.', '2024-06-01', '2024-08-31', 0.20, NOW()),
  ('Weekend Special', 'Special rates for all movies during weekends.', '2024-05-01', '2024-05-03', 0.15, NOW()),
  ('Holiday Cheer', 'Enjoy the holiday season with classic movies.', '2024-12-20', '2025-01-05', 0.25, NOW()),
  ('RomCom Festival', 'Romantic Comedies at half price.', '2024-02-14', '2024-02-20', 0.50, NOW()),
  ('Sci-Fi September', 'Explore the universe with a 30% discount on all sci-fi movies.', '2024-09-01', '2024-09-30', 0.30, NOW()),
  ('Halloween Horrors', 'Spooky savings on horror films.', '2024-10-25', '2024-10-31', 0.35, NOW()),
  ('Thanksgiving Offer', 'Grateful for our customers with these great offers.', '2024-11-23', '2024-11-30', 0.20, NOW()),
  ('Back to School', 'Family-friendly films to enjoy together.', '2024-08-15', '2024-09-15', 0.10, NOW()),
  ('New Year New Movies', 'Start the year right with our selection of new releases.', '2025-01-01', '2025-01-31', 0.20, NOW()),
  ('Classic Film Series', 'Journey back in time with these classic films.', '2024-03-01', '2024-04-01', 0.40, NOW());
```

Data Output | Messages | Notifications

```
INSERT 0 10

Query returned successfully in 309 msec.
```

Implementation of queries -Implement a set of queries that demonstrate various database operations such as SELECT, INSERT, UPDATE, JOIN, etc.

Here are the queries that focus on INSERT, UPDATE, SELECT, JOIN etc..

Each query serves a distinct purpose, from managing and viewing movie-related data to handling staff schedules and customer records. They demonstrate a range of SQL functionalities including data insertion, updates, deletions, and retrieval with conditions and aggregations.

1. Get the list of all movies:This query retrieves every record from the public.film table, effectively listing all movies in the database without any filtering or sorting.

```sql
-- Get the list of all movies.
SELECT * FROM public.film;
```

Data Output | Messages | Notifications

	film_id [PK] integer	title character varying (255)	description text	release_year integer	languag smallint
1	133	Chamber Italian	A Fateful Reflection of a Moose And a Husband who must Overcome a Monkey in Nigeria	2006	
2	384	Grosse Wonderful	A Epic Drama of a Cat And a Explorer who must Redeem a Moose in Australia	2006	
3	8	Airport Pollock	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006	
4	98	Bright Encounters	A Fateful Yarn of a Lumberjack And a Feminist who must Conquer a Student in A Jet Boat	2006	
5	1	Academy Dinosaur	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006	
6	2	Ace Goldfinger	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	2006	
7	3	Adaptation Holes	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory	2006	
8	4	Affair Prejudice	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank	2006	
9	5	African Egg	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico	2006	
10	6	Agent Truman	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006	
11	7	Airplane Sierra	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat	2006	
12	9	Alabama Devil	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat	2006	
13	10	Aladdin Calendar	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China	2006	

Total rows: 1000 of 1000 | Query complete 00:00:00.110 | Ln 167, Col

2. Retrieve all film titles and their associated reviews and ratings: Joins the public.movie_review table with the public.film table on the film_id to fetch the title of each film alongside the text and rating of each review. It provides a comprehensive view of films and their customer feedback.

```
168
169 -- Retrieve all film titles and their associated reviews and ratings.
170 SELECT f.title, mr.review_text, mr.rating
171 FROM public.movie_review AS mr
172 JOIN public.film AS f ON mr.film_id = f.film_id;
173
```

Data Output | Messages | Notifications

	title character varying (255)	review_text text	rating integer
1	Academy Dinosaur	Incredible storytelling and memorable characters.	5
2	Academy Dinosaur	The special effects were top-notch!	4
3	Ace Goldfinger	A bit too slow to get going, but a great finish.	3
4	Ace Goldfinger	I loved the soundtrack more than the movie itself.	4
5	Academy Dinosaur	Would watch it again! Two thumbs up!	5
6	Adaptation Holes	The acting was superb, brought the characters to life.	4
7	Affair Prejudice	My kids loved it, a new family favorite.	5
8	Adaptation Holes	Not as good as the original, but still worth a watch.	3
9	Ace Goldfinger	A solid entry in the sci-fi genre. Really made me think.	4
10	Ace Goldfinger	Decent movie, but I probably wouldnt watch it a second ti...	3

3. Change the status of all late fees that are over 5 days overdue to 'Waived': Updates the public.late_fee table, setting the status to 'Waived' for all entries where the days_overdue is greater than 5. This could be part of a forgiveness policy for overdue fees.
Calculate the total amount of late fees per customer

```
173
174 -- UPDATE: Change the status of all late fees that are over 5 days overdue to 'Waived'.
175 UPDATE public.late_fee
176 SET status = 'Waived'
177 WHERE days_overdue > 5;
178
```

Data Output | Messages | Notifications

```
UPDATE 2

Query returned successfully in 75 msec.
```

4. Combines data from the public.late_fee, public.rental, and public.customer tables to sum up the total late fees each customer owes. It groups the results by customer, allowing for easy tracking of outstanding fees.

```
-- Calculate the total amount of late fees per customer.
SELECT c.first_name, c.last_name, SUM(lf.fee_amount) AS total_late_fees
FROM public.late_fee AS lf
JOIN public.rental AS r ON lf.rental_id = r.rental_id
JOIN public.customer AS c ON r.customer_id = c.customer_id
GROUP BY c.customer_id;
```

Data Output | Messages | Notifications

	first_name character varying (45)	last_name character varying (45)	total_late_fees numeric
1	Alma	Austin	3.99
2	Ruth	Martinez	6.99
3	Tim	Cary	1.99
4	Glenda	Frazier	5.99
5	Douglas	Graf	0.99
6	Angel	Barclay	1.99
7	Tracy	Cole	1.99
8	Daniel	Cabral	4.99
9	Edward	Baugh	0.99
10	Ernest	Stepp	2.99

5. Insert a new promotion that gives a 10% discount for all films that have an average rating of 5. Inserts a new record into the public.promotions table for a discount promotion, but only if there exists at least one film with an average rating of 5. This conditional insert is based on the aggregate average rating from the public.movie_review table.

```
-- Insert a new promotion that gives a 10% discount for all films that have an average rating of 5.
INSERT INTO public.promotions (name, description, start_date, end_date, discount_rate, last_update)
SELECT 'Perfect Score Discount', 'A 10% discount for our highest-rated films.', '2024-07-01', '2024-07-31', 0.10, NOW()
FROM public.film
WHERE film_id IN (
  SELECT film_id
  FROM public.movie_review
  GROUP BY film_id
  HAVING AVG(rating) = 5
);
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 69 msec.
```

6. Get a list of all staff members and their corresponding work hours for a particular date. Retrieves the work schedule (work date, start time, end time) for all staff members from the public.staff and public.staff_schedule tables for a specific date ('2024-03-10'). It's useful for managing staff assignments and planning.

```
-- Get a list of all staff members and their corresponding work hours for a particular date.
SELECT s.first_name, s.last_name, ss.work_date, ss.start_time, ss.end_time
FROM public.staff AS s
JOIN public.staff_schedule AS ss ON s.staff_id = ss.staff_id
WHERE ss.work_date = '2024-03-10';
```

Data Output | Messages | Notifications

	first_name character varying (45)	last_name character varying (45)	work_date date	start_time time without time zone	end_time time without time zone
1	Mike	Hillyer	2024-03-10	09:00:00	17:00:00
2	Jon	Stephens	2024-03-10	13:00:00	21:00:00

7. Retrieve film titles along with the names of the customers who rented them.
By joining the public.rental, public.inventory, public.film, and public.customer tables, this query shows which customers rented which films, including the customer's name and the film's title.

```
--Retrieve film titles along with the names of the customers who rented them.
SELECT f.title, c.first_name, c.last_name
FROM public.rental AS r
JOIN public.inventory AS i ON r.inventory_id = i.inventory_id
JOIN public.film AS f ON i.film_id = f.film_id
JOIN public.customer AS c ON r.customer_id = c.customer_id;
```

Data Output | Messages | Notifications

	title character varying (255)	first_name character varying (45)	last_name character varying (45)
1	Freaky Pocus	Tommy	Collazo
2	Graduate Lord	Manuel	Murrell
3	Love Suicides	Andrew	Purdy
4	Idols Snatchers	Delores	Hansen
5	Mystic Truman	Nelson	Christenson
6	Swarm Gold	Cassandra	Walters
7	Lawless Vision	Minnie	Romero
8	Matrix Snowman	Ellen	Simpson
9	Hanging Deep	Danny	Isom
10	Whale Bikini	April	Burns
11	Games Bowfinger	Deanna	Byrd
12	King Evolution	Raymond	Mcwhorter
13	Monterey Labyrinth	Theodore	Culp

Total rows: 1000 of 16044 | Query complete 00:00:00.097

8. Remove a staff schedule entry: Deletes a specific record from the public.staff_schedule table based on the schedule_id. This operation would be used to cancel or remove a previously scheduled work shift.

```
211
212 -- Remove a staff schedule entry.
213 DELETE FROM public.staff_schedule WHERE schedule_id = 1;
214
215
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 226 msec.
```

9. Add a new customer:Inserts a new customer record into the public.customer table with specified values for the customer's name, email, address ID, store ID, and active status. This query expands the customer base with a new entry.

```
215 -- Add a new customer.
216 INSERT INTO public.customer (first_name, last_name, email, address_id, store_id, activebool)
217 VALUES ('Sam', 'Wilson', 'sam.wilson@example.com', 1, 1, true);
218
219
220
221
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 197 msec.
```

10. Update a customer's last name.
Updates the last name of a specific customer in the public.customer table. This might be used to correct a spelling error or update information following a name change.

```
219
220 -- Update a customer's last name.
221 UPDATE public.customer SET last_name = 'Smith' WHERE customer_id = 1;
222
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 205 msec.
```

Create Views

A view in SQL is a virtual table based on the result-set of an SQL statement. Creating views is a convenient way to save query results as a virtual table. However, unlike a table, a view does not store data physically in the database; it dynamically generates data when you query it.

I have created the following views

1. view_customer_rentals: This view shows all movies rented by each customer, including the customer's ID, first and last name, the title of the rented film, and the rental date. It provides a comprehensive look at customer rental activity.

```
224 -- Shows all movies rented by each customer.
225 CREATE VIEW public.view_customer_rentals AS
226 SELECT c.customer_id, c.first_name, c.last_name, f.title, r.rental_date
227 FROM public.customer AS c
228 JOIN public.rental AS r ON c.customer_id = r.customer_id
229 JOIN public.inventory AS i ON r.inventory_id = i.inventory_id
230 JOIN public.film AS f ON i.film_id = f.film_id;
231
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 244 msec.
```

```
233 SELECT * FROM public.view_customer_rentals;
234
235
```

Data Output | Messages | Notifications

	customer_id integer	first_name character varying (45)	last_name character varying (45)	title character varying (255)	rental_date timestamp without time zone
1	459	Tommy	Collazo	Freaky Pocus	2005-05-24 22:54:33
2	408	Manuel	Murrell	Graduate Lord	2005-05-24 23:03:39
3	333	Andrew	Purdy	Love Suicides	2005-05-24 23:04:41
4	222	Delores	Hansen	Idols Snatchers	2005-05-24 23:05:21
5	549	Nelson	Christenson	Mystic Truman	2005-05-24 23:08:07
6	269	Cassandra	Walters	Swarm Gold	2005-05-24 23:11:53
7	239	Minnie	Romero	Lawless Vision	2005-05-24 23:31:46
8	126	Ellen	Simpson	Matrix Snowman	2005-05-25 00:00:40
9	399	Danny	Isom	Hanging Deep	2005-05-25 00:02:21
10	142	April	Burns	Whale Bikini	2005-05-25 00:09:02
11	261	Deanna	Byrd	Games Bowfinger	2005-05-25 00:19:27
12	334	Raymond	Mcwhorter	King Evolution	2005-05-25 00:22:55
13	446	Theodore	Culp	Monterey Labyrinth	2005-05-25 00:31:15

Total rows: 1000 of 16044 | Query complete 00:00:00.393 | Ln 234, Col 1

2. view_film_ratings: This view displays the average ratings for each film, combining film information (film ID and title) with the average rating it has received from customer reviews. It's useful for evaluating overall film popularity and quality as perceived by customers.

```
233  -- Displays average ratings for each film.
234  CREATE VIEW public.view_film_ratings AS
235  SELECT f.film_id, f.title, AVG(mr.rating) AS average_rating
236  FROM public.film AS f
237  JOIN public.movie_review AS mr ON f.film_id = mr.film_id
238  GROUP BY f.film_id;
239
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 58 msec.
```

```
242
243  SELECT * FROM public.view_film_ratings;
244
245  -- Lists complete work schedules for all staff.
```

Data Output | Messages | Notifications

	film_id integer	title character varying (255)	average_rating numeric
1	2	Ace Goldfinger	3.5000000000000000
2	3	Adaptation Holes	3.5000000000000000
3	4	Affair Prejudice	5.0000000000000000
4	1	Academy Dinosaur	4.6666666666666667

3. view_staff_schedules: Lists complete work schedules for all staff members, detailing each staff's ID, first and last name, along with their work dates, start times, and end times. It offers a snapshot of the staffing schedule, helping in workforce management

```
239
240  -- Lists complete work schedules for all staff.
241  CREATE VIEW public.view_staff_schedules AS
242  SELECT s.staff_id, s.first_name, s.last_name, ss.work_date, ss.start_time, ss.end_time
243  FROM public.staff AS s
244  JOIN public.staff_schedule AS ss ON s.staff_id = ss.staff_id;
245
246  -- Summarizes the late fees owed by each customer.
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 66 msec.
```

```
SELECT * FROM public.view_staff_schedules;
```

Data Output | Messages | Notifications

	staff_id integer	first_name character varying (45)	last_name character varying (45)	work_date date	start_time time without time zone	end_time time without time zone
1	1	Mike	Hillyer	2024-03-11	09:00:00	17:00:00
2	1	Mike	Hillyer	2024-03-12	10:00:00	18:00:00
3	1	Mike	Hillyer	2024-03-13	09:00:00	17:00:00
4	1	Mike	Hillyer	2024-03-14	09:00:00	17:00:00
5	2	Jon	Stephens	2024-03-10	13:00:00	21:00:00
6	2	Jon	Stephens	2024-03-11	13:00:00	21:00:00
7	2	Jon	Stephens	2024-03-12	14:00:00	22:00:00
8	2	Jon	Stephens	2024-03-13	13:00:00	21:00:00
9	2	Jon	Stephens	2024-03-14	13:00:00	21:00:00

4. view_late_fees_owed: Summarizes the total late fees owed by each customer, showing the customer's ID, first name, last name, and the aggregate late fees. This view helps in financial tracking and customer account management.

```
-- Summarizes the late fees owed by each customer.
CREATE VIEW public.view_late_fees_owed AS
SELECT c.customer_id, c.first_name, c.last_name, SUM(lf.fee_amount) AS total_late_fees
FROM public.late_fee AS lf
JOIN public.rental AS r ON lf.rental_id = r.rental_id
JOIN public.customer AS c ON r.customer_id = c.customer_id
GROUP BY c.customer_id;
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 160 msec.
```

```
SELECT * FROM public.view_late_fees_owed;
```

Data Output | Messages | Notifications

	customer_id integer	first_name character varying (45)	last_name character varying (45)	total_late_fees numeric
1	196	Alma	Austin	3.99
2	19	Ruth	Martinez	6.99
3	468	Tim	Cary	1.99
4	242	Glenda	Frazier	5.99
5	343	Douglas	Graf	0.99
6	503	Angel	Barclay	1.99
7	108	Tracy	Cole	1.99
8	310	Daniel	Cabral	4.99
9	317	Edward	Baugh	0.99
10	384	Ernest	Stepp	2.99

5. view_current_promotions: Shows active promotions by listing their ID, name, description, start and end dates, and the discount rate. It's a tool for marketing analysis and customer engagement, highlighting currently available offers.

```
254
255 -- Shows active promotions.
256 CREATE VIEW public.view_current_promotions AS
257 SELECT p.promotion_id, p.name, p.description, p.start_date, p.end_date, p.discount_rate
258 FROM public.promotions AS p
259 WHERE p.start_date <= CURRENT_DATE AND p.end_date >= CURRENT_DATE;
260
261
```

Data Output | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 171 msec.
```

```
271
272 SELECT * FROM public.view_current_promotions;
273
274
```

Data Output | Messages | Notifications

	promotion_id integer	name character varying (255)	description text	start_date date	end_date date	discount_rate numeric (3,2)
1	10	Classic Film Series	Journey back in time with these classic film...	2024-03-01	2024-04-01	0.40

Clauses—
Use of the following Clauses: Example: order by, between, group by, having, order by, AND, OR, with

1. Retrieve Staff Schedules Between Two Dates: Useful for managing staff workload and planning, this query fetches staff schedules within a specified date range.

```
-- Retrieve staff schedules between two dates.
SELECT *
FROM public.view_staff_schedules
WHERE work_date BETWEEN '2024-03-10' AND '2024-03-14';
```

Data Output | Messages | Notifications

	staff_id integer	first_name character varying (45)	last_name character varying (45)	work_date date	start_time time without time zone	end_time time without time zone
1	1	Mike	Hillyer	2024-03-11	09:00:00	17:00:00
2	1	Mike	Hillyer	2024-03-12	10:00:00	18:00:00
3	1	Mike	Hillyer	2024-03-13	09:00:00	17:00:00
4	1	Mike	Hillyer	2024-03-14	09:00:00	17:00:00
5	2	Jon	Stephens	2024-03-10	13:00:00	21:00:00
6	2	Jon	Stephens	2024-03-11	13:00:00	21:00:00
7	2	Jon	Stephens	2024-03-12	14:00:00	22:00:00
8	2	Jon	Stephens	2024-03-13	13:00:00	21:00:00
9	2	Jon	Stephens	2024-03-14	13:00:00	21:00:00

2. Find the Total Number of Rentals for Each Film: This aggregation helps in understanding film popularity by counting how many times each film has been rented.

```
-- Find the total number of rentals for each film.
SELECT f.title, COUNT(r.rental_id) AS rental_count
FROM public.rental AS r
JOIN public.inventory AS i ON r.inventory_id = i.inventory_id
JOIN public.film AS f ON i.film_id = f.film_id
GROUP BY f.title;
```

Data Output | Messages | Notifications

	title character varying (255)	rental_count bigint
1	Graceland Dynamite	6
2	Opus Ice	11
3	Braveheart Human	5
4	Wonderful Drop	9
5	Rush Goodfellas	31
6	Purple Movie	13
7	Minority Kiss	12
8	Luke Mummy	7
9	Fantasy Troopers	26
10	Grinch Massage	14
11	Gaslight Crusade	16
12	Microcosmos Paradise	13
13	Saturn Name	13

Total rows: 958 of 958 | Query complete 00:00:00.354

3 List Film Name and Its Release Year Based on Ascending Order: By sorting films first by their release year and then by title, this provides a chronological catalogue of available films.

```
289
290 -- List film name and its release yeat based on the ascending order of release year and title
291 SELECT title, release_year
292 FROM public.film
293 ORDER BY release_year ASC, title ASC;
294
295
```

Data Output | Messages | Notifications

	title character varying (255)	release_year integer
1	Academy Dinosaur	2006
2	Ace Goldfinger	2006
3	Adaptation Holes	2006
4	Affair Prejudice	2006
5	African Egg	2006
6	Agent Truman	2006
7	Airplane Sierra	2006
8	Airport Pollock	2006
9	Alabama Devil	2006
10	Aladdin Calendar	2006
11	Alamo Videotape	2006
12	Alaska Phantom	2006
13	Ali Forever	2006

4. List Films with Average Rating Above 4: Identifies high-quality films as perceived by customers, aiding in recommendations and marketing strategies.

```
295 --  List films that have received an average rating above 4.
296 SELECT f.title, AVG(mr.rating) AS average_rating
297 FROM public.movie_review AS mr
298 JOIN public.film AS f ON mr.film_id = f.film_id
299 GROUP BY f.title
300 HAVING AVG(mr.rating) > 4;
301
302
```

Data Output | Messages | Notifications

	title character varying (255)	average_rating numeric
1	Affair Prejudice	5.0000000000000000
2	Academy Dinosaur	4.6666666666666667

5. Shows Active Promotions: This view aggregates active promotions based on the current date, helping in identifying which offers are available to customers right now.

```
-- Show promotions that are active in July 2024 and have a discount rate of at least 20%.
SELECT *
FROM public.promotions
WHERE start_date <= '2024-07-31'
AND end_date >= '2024-07-01'
AND discount_rate >= 0.20;
```

Data Output | Messages | Notifications

	promotion_id [PK] integer	name character varying (255)	description text	start_date date	end_date date	discount_rate numeric (3,2)	last_update timestamp without time zone
1	1	Summer Bonanza	Discount on all summer blockbusters.	2024-06-01	2024-08-31	0.20	2024-03-07 00:59:09.1314

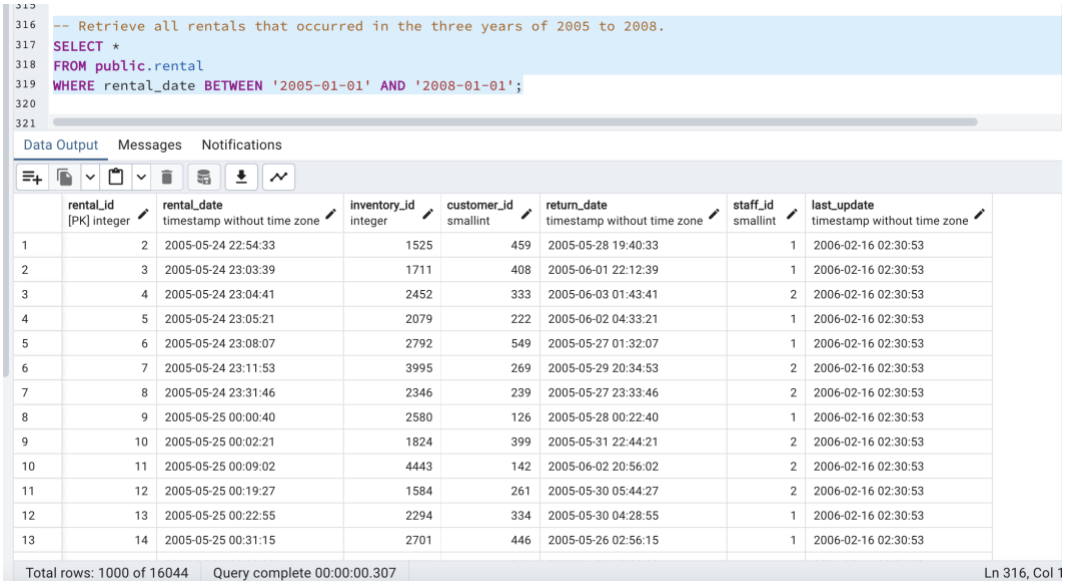
6. Find Movie Reviews with a 5-Star Rating or Written for Film ID 1: Helps in quickly identifying exceptional reviews or reviews for a specific film, useful for quality or content analysis.

```
-- Find movie reviews that either have a 5-star rating or were written for film ID 1.
SELECT *
FROM public.movie_review
WHERE rating = 5
OR film_id = 1;
```

Data Output | Messages | Notifications

	review_id [PK] integer	film_id integer	customer_id integer	review_text text	rating integer	review_date timestamp without time zone
1	1	1	1	Incredible storytelling and memorable characters.	5	2024-01-10 10:00:00
2	2	1	2	The special effects were top-notch!	4	2024-01-10 11:00:00
3	5	1	3	Would watch it again! Two thumbs up!	5	2024-01-10 14:00:00
4	7	4	2	My kids loved it, a new family favorite.	5	2024-01-10 16:00:00

7 Retrieve All Rentals That Occurred From 2005 to 2008: By fetching rental data within a specific timeframe, this query can aid in historical analysis or reporting.



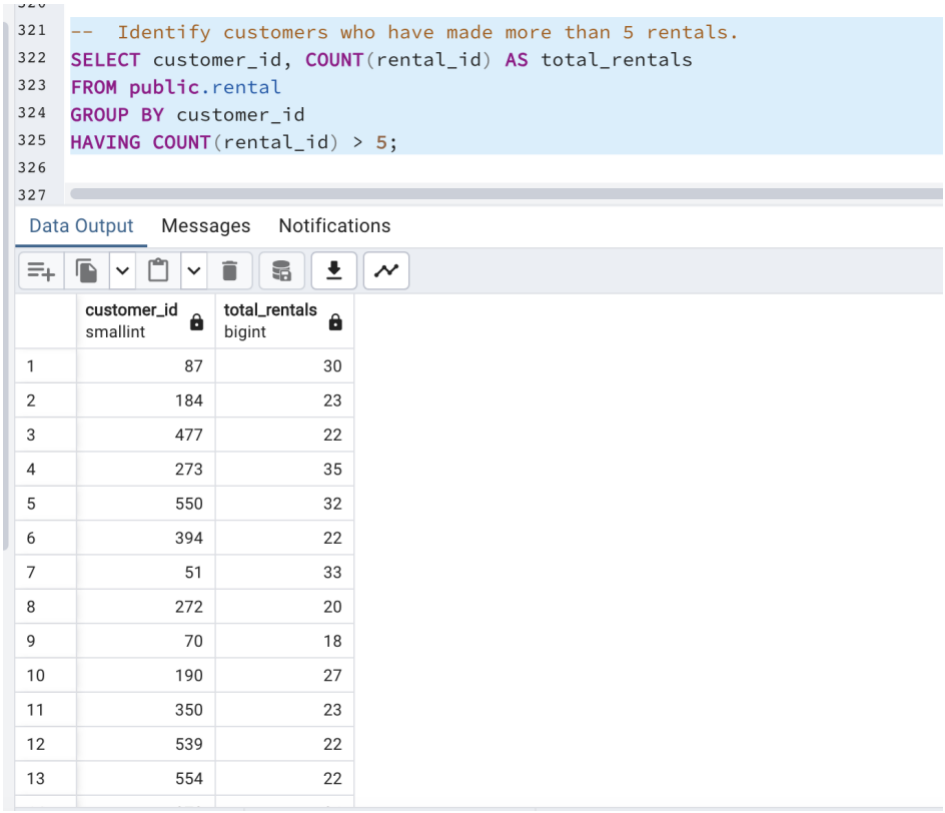
```
-- Retrieve all rentals that occurred in the three years of 2005 to 2008.
SELECT *
FROM public.rental
WHERE rental_date BETWEEN '2005-01-01' AND '2008-01-01';
```

Data Output | Messages | Notifications

	rental_id [PK] integer	rental_date timestamp without time zone	inventory_id integer	customer_id smallint	return_date timestamp without time zone	staff_id smallint	last_update timestamp without time zone
1	2	2005-05-24 22:54:33	1525	459	2005-05-28 19:40:33	1	2006-02-16 02:30:53
2	3	2005-05-24 23:03:39	1711	408	2005-06-01 22:12:39	1	2006-02-16 02:30:53
3	4	2005-05-24 23:04:41	2452	333	2005-06-03 01:43:41	2	2006-02-16 02:30:53
4	5	2005-05-24 23:05:21	2079	222	2005-06-02 04:33:21	1	2006-02-16 02:30:53
5	6	2005-05-24 23:08:07	2792	549	2005-05-27 01:32:07	1	2006-02-16 02:30:53
6	7	2005-05-24 23:11:53	3995	269	2005-05-29 20:34:53	2	2006-02-16 02:30:53
7	8	2005-05-24 23:31:46	2346	239	2005-05-27 23:33:46	2	2006-02-16 02:30:53
8	9	2005-05-25 00:00:40	2580	126	2005-05-28 00:22:40	1	2006-02-16 02:30:53
9	10	2005-05-25 00:02:21	1824	399	2005-05-31 22:44:21	2	2006-02-16 02:30:53
10	11	2005-05-25 00:09:02	4443	142	2005-06-02 20:56:02	2	2006-02-16 02:30:53
11	12	2005-05-25 00:19:27	1584	261	2005-05-30 05:44:27	2	2006-02-16 02:30:53
12	13	2005-05-25 00:22:55	2294	334	2005-05-30 04:28:55	1	2006-02-16 02:30:53
13	14	2005-05-25 00:31:15	2701	446	2005-05-26 02:56:15	1	2006-02-16 02:30:53

Total rows: 1000 of 16044 | Query complete 00:00:00.307 | Ln 316, Col 1

8. Identify Customers with More Than 5 Rentals: Useful for recognizing highly engaged customers, potentially for loyalty programs or targeted promotions.



```
--  Identify customers who have made more than 5 rentals.
SELECT customer_id, COUNT(rental_id) AS total_rentals
FROM public.rental
GROUP BY customer_id
HAVING COUNT(rental_id) > 5;
```

Data Output | Messages | Notifications

	customer_id smallint	total_rentals bigint
1	87	30
2	184	23
3	477	22
4	273	35
5	550	32
6	394	22
7	51	33
8	272	20
9	70	18
10	190	27
11	350	23
12	539	22
13	554	22

9. Find Movies Rated 'PG' or Longer Than 120 Minutes: This query can assist in content curation or meeting specific customer preferences.

```
-- Find movies that are either rated 'PG' or have a length greater than 120 minutes.
SELECT *
FROM public.film
WHERE rating = 'PG' OR length > 120;
```

Data Output | Messages | Notifications

	film_id [PK] integer	title character varying (255)	description text
1	1	Academy Dinosaur	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
2	5	African Egg	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico
3	6	Agent Truman	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China
4	11	Alamo Videotape	A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention
5	12	Alaska Phantom	A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia
6	13	Ali Forever	A Action-Packed Drama of a Dentist And a Crocodile who must Battle a Feminist in The Canadian Rockies
7	16	Alley Evolution	A Fast-Paced Drama of a Robot And a Composer who must Battle a Astronaut in New Orleans
8	19	Amadeus Holy	A Emotional Display of a Pioneer And a Technical Writer who must Battle a Man in A Baloon
9	21	American Circus	A Insightful Drama of a Girl And a Astronaut who must Face a Database Administrator in A Shark Tank
10	24	Analyze Hoosiers	A Thoughtful Display of a Explorer And a Pastry Chef who must Overcome a Feminist in The Sahara Desert
11	27	Anonymous Human	A Amazing Reflection of a Database Administrator And a Astronaut who must Outrace a Database Administrator in A Shark Tank
12	29	Antitrust Tomatoes	A Fateful Yarn of a Womanizer And a Feminist who must Succumb a Database Administrator in Ancient India
13	33	Apollo Teen	A Action-Packed Reflection of a Crocodile And a Explorer who must Find a Sumo Wrestler in An Abandoned Mine Shaft

Total rows: 569 of 569 | Query complete 00:00:00.097 | Ln 326, Col 1

10. Determine the Top 3 Most Reviewed Films: Highlights the most discussed or popular films within the review community, indicating viewer engagement.

```
-- Determine the top 3 most revie
SELECT f.title, COUNT(mr.review_id) AS review_count
FROM public.film AS f
JOIN public.movie_review AS mr ON f.film_id = mr.film_id
GROUP BY f.title
ORDER BY review_count DESC
LIMIT 3;
```

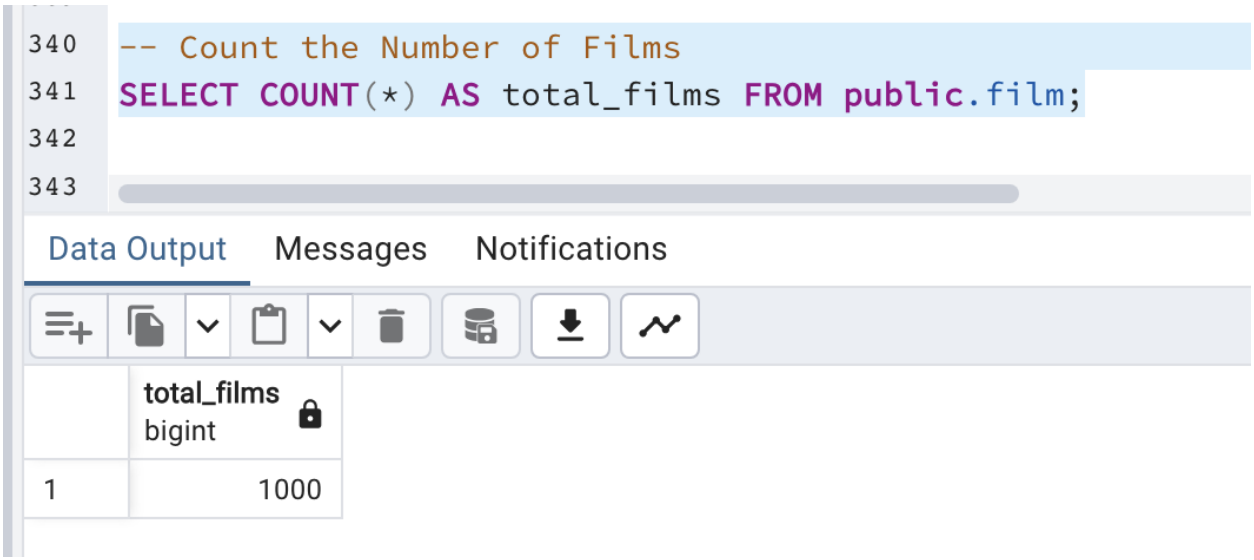
F5

Data Output | Messages | Notifications

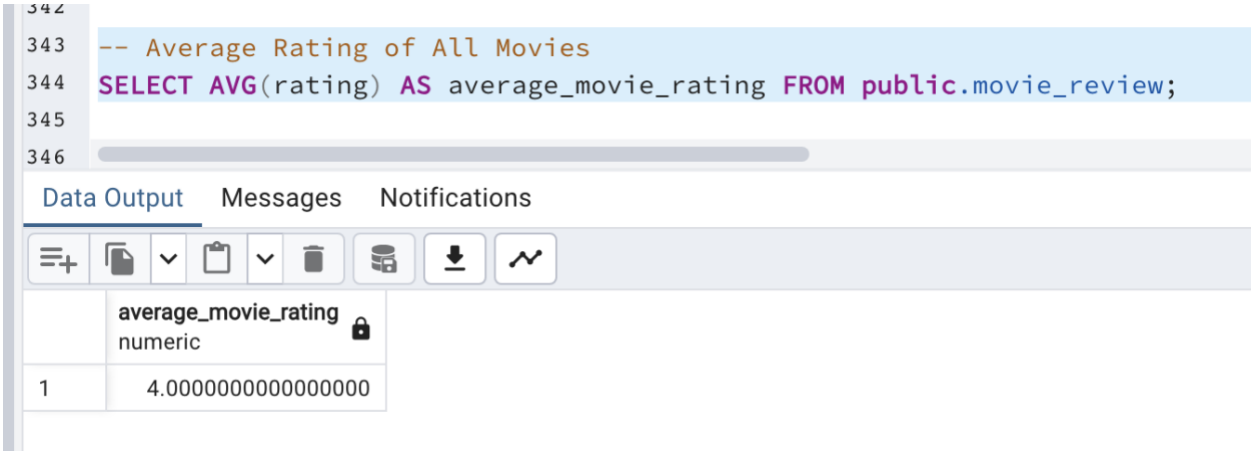
	title character varying (255)	review_count bigint
1	Ace Goldfinger	4
2	Academy Dinosaur	3
3	Adaptation Holes	2

Aggregate Functions
These were used to perform calculations across sets of rows, allowing for summarizations such as counting total entries, calculating averages, and determining maximum and minimum values.

1. Count the Number of Films: Provides a simple total count of the films available in the database, useful for inventory checks or catalog size assessments.



2. Average Rating of All Movies: Offers an overall sense of film quality across the entire movie database.



3. Total Number of Rentals for Each Customer: By identifying each customer's rental count, this helps in understanding customer activity levels.

```
345
346 -- Total Number of Rentals for Each Customer
347 SELECT customer_id, COUNT(rental_id) AS total_rentals
348 FROM public.rental
349 GROUP BY customer_id;
350
351
```

Data Output | Messages | Notifications

	customer_id smallint	total_rentals bigint
1	87	30
2	184	23
3	477	22
4	273	35
5	550	32
6	394	22
7	51	33
8	272	20
9	70	18
10	190	27
11	350	23
12	539	22
13	554	22

Total rows: 599 of 599 | Query complete 00:00:00.195

4. Maximum Late Fee Incurred by Any Rental: Identifies the highest late fee charged, useful for policy review or customer service interventions.

```
350
351 -- Maximum Late Fee by Any Rental
352 SELECT MAX(fee_amount) AS max_late_fee FROM public.late_fee;
353
354
```

Data Output | Messages | Notifications

	max_late_fee numeric
1	6.99

5. Average Length of All Films: Gives an insight into the typical film duration, aiding in content planning and customer expectation management.

```
353
354 -- Average Length of All Films
355 SELECT AVG(length) AS average_film_length FROM public.film;
356
```

Data Output | Messages | Notifications

	average_film_length numeric
1	115.2720000000000000

6. Count of Films by Rating: Provides a breakdown of films by their ratings, helping in understanding the content distribution and rating demographics.

```
356
357 -- Count of Films by Rating
358 SELECT rating, COUNT(film_id) AS number_of_films
359 FROM public.film
360 GROUP BY rating;
361
```

Data Output | Messages | Notifications

	rating mpaa_rating	number_of_films bigint
1	G	178
2	NC-17	210
3	PG	194
4	R	195
5	PG-13	223

7. Minimum and Maximum Replacement Cost of Films: Useful for financial planning and risk management, this query identifies the cost range for replacing films.

```
362 -- Minimum and Maximum Replacement Cost of Films
363 SELECT MIN(replacement_cost) AS min_replacement_cost, MAX(replacement_cost) AS max_replacement_cost FROM public.film;
364
365
```

Data Output | Messages | Notifications

	min_replacement_cost numeric	max_replacement_cost numeric
1	9.99	29.99

8. Average Recommendation Score for Each Film: Aids in film promotion and recommendation by highlighting films with higher scores.

```
364
365 -- Average Recommendation Score for Each Film
366 SELECT film_id, AVG(recommendation_score) AS avg_recommendation_score
367 FROM public.film_recommendations
368 GROUP BY film_id;
369
370
```

Data Output | Messages | Notifications

	film_id integer	avg_recommendation_score numeric
1	40	8.0000000000000000
2	15	9.0000000000000000
3	30	9.5000000000000000
4	10	8.5000000000000000
5	35	7.0000000000000000
6	45	9.2000000000000000
7	50	6.5000000000000000
8	25	6.0000000000000000
9	20	7.5000000000000000
10	55	8.8000000000000000

9. Number of Staff Scheduled Each Day: Essential for operational planning, this query shows daily staff allocation.

```
-- Number of Staff Scheduled Each Day
SELECT work_date, COUNT(distinct staff_id) AS number_of_staff
FROM public.staff_schedule
GROUP BY work_date;
```

Data Output | Messages | Notifications

	work_date date	number_of_staff bigint
1	2024-03-10	1
2	2024-03-11	2
3	2024-03-12	2
4	2024-03-13	2
5	2024-03-14	2

10. Monthly Review Activity Analysis: By examining the number of reviews each month, this helps in understanding customer engagement trends over time.

```
-- How active the review section is on a monthly basis:
SELECT
  EXTRACT(YEAR FROM review_date) AS review_year,
  EXTRACT(MONTH FROM review_date) AS review_month,
  COUNT(review_id) AS total_reviews
FROM public.movie_review
GROUP BY review_year, review_month
ORDER BY review_year, review_month;
```

Data Output | Messages | Notifications

	review_year numeric	review_month numeric	total_reviews bigint
1	2024	1	10

Nested queries

Films with Higher Than Average Replacement Costs: Helps in identifying premium or high-value films within the collection.

```
-- Find the titles of films that have a higher replacement cost than the average replacement cost of all films.
SELECT title, replacement_cost
FROM public.film
WHERE replacement_cost > (
  SELECT AVG(replacement_cost)
  FROM public.film
);
```

Data Output | Messages | Notifications

	title character varying (255)	replacement_cost numeric (5,2)
1	Grosse Wonderful	19.99
2	Academy Dinosaur	20.99
3	Affair Prejudice	26.99
4	African Egg	22.99
5	Airplane Sierra	28.99
6	Alabama Devil	21.99
7	Aladdin Calendar	24.99
8	Alaska Phantom	22.99

Films with At Least One Review and Their Average Rating: Useful for filtering out films with customer feedback and understanding their perceived quality.

```
-- List the average rating of films along with their titles, but only include films that have at least one review.
SELECT f.title, avg_reviews.avg_rating
FROM public.film AS f
JOIN (
  SELECT film_id, AVG(rating) AS avg_rating
  FROM public.movie_review
  GROUP BY film_id
) AS avg_reviews ON f.film_id = avg_reviews.film_id;
```

Data Output | Messages | Notifications

	title character varying (255)	avg_rating numeric
1	Academy Dinosaur	4.6666666666666667
2	Ace Goldfinger	3.5000000000000000
3	Adaptation Holes	3.5000000000000000
4	Affair Prejudice	5.0000000000000000

Film Rental Popularity and Demand: Indicates the demand for each film based on rental counts, useful for inventory and marketing decisions.

```
400
401 -- Retrieve the title of each film along with the total number of rentals for that film.
402 SELECT title,
403        (SELECT COUNT(*)
404         FROM public.rental AS r
405         JOIN public.inventory AS i ON r.inventory_id = i.inventory_id
406         WHERE i.film_id = f.film_id) AS total_rentals
407 FROM public.film AS f;
408
409
```

Data Output | Messages | Notifications

	title character varying (255)	total_rentals bigint
1	Chamber Italian	13
2	Grosse Wonderful	8
3	Airport Pollock	18
4	Bright Encounters	13
5	Academy Dinosaur	23
6	Ace Goldfinger	7
7	Adaptation Holes	12
8	Affair Prejudice	23

Total rows: 1000 of 1000 Query complete 00:00:00.720

Scalar Subquery

A scalar subquery is a subquery that returns exactly one column value from one row.

```
408
409 -- Get the title of the film that was rented most recently.
410 SELECT title
411 FROM public.film
412 WHERE film_id = (
413   SELECT i.film_id
414   FROM public.rental AS r
415   JOIN public.inventory AS i ON r.inventory_id = i.inventory_id
416   ORDER BY r.rental_date DESC
417   LIMIT 1
418 );
419
420
```

Data Output | Messages | Notifications

	title character varying (255)
1	Zhivago Core

Customer with Their Latest Rental Date: Provides personalized customer insights by tracking the most recent engagement.

```
-- Display Customer Names with Their Latest Rental Date:
SELECT first_name, last_name,
       (SELECT r.rental_date
        FROM public.rental AS r
        WHERE r.customer_id = c.customer_id
        ORDER BY r.rental_date DESC
        LIMIT 1) AS latest_rental_date
FROM public.customer AS c;
```

Data Output | Messages | Notifications

	first_name character varying (45)	last_name character varying (45)	latest_rental_date timestamp without time zone
1	Jared	Ely	2005-08-22 16:33:39
2	Patricia	Johnson	2005-08-23 17:39:35
3	Linda	Williams	2005-08-23 07:10:14
4	Barbara	Jones	2005-08-23 07:43:00
5	Elizabeth	Brown	2006-02-14 15:16:03
6	Jennifer	Davis	2005-08-23 06:41:32
7	Maria	Miller	2005-08-21 04:49:48
8	Susan	Wilson	2005-08-23 14:31:19

Total rows: 600 of 600 | Query complete 00:00:00.086

Customer with Maximum Rentals: Identifies the most active customer, potentially for loyalty rewards or targeted marketing.

```
-- Customer with Maximum Rentals
SELECT first_name, last_name
FROM public.customer
WHERE customer_id = (
    SELECT r.customer_id
    FROM public.rental AS r
    GROUP BY r.customer_id
    ORDER BY COUNT(r.rental_id) DESC
    LIMIT 1
);

-- Highest Average Rating by Film Category
```

Data Output | Messages | Notifications

	first_name character varying (45)	last_name character varying (45)
1	Eleanor	Hunt

Film with the Longest Duration Available: Helps in content management by identifying the longest films in the inventory.

```
-- Film with the Longest Duration in Inventory
SELECT title
FROM public.film
WHERE length = (
    SELECT MAX(length)
    FROM public.film AS f
    JOIN public.inventory AS i ON f.film_id = i.film_id
);
```

Data Output | Messages | Notifications

	title character varying (255)
1	Chicago North
2	Control Anthem
3	Darn Forrester
4	Gangs Pride
5	Home Pity
6	Muscle Bright
7	Pond Seattle
8	Soldiers Evolution

Total rows: 10 of 10 | Query complete 00:00:00.091

Category with Highest Average Film Rating: Aids in marketing and promotion by spotlighting the best-received film category.

```
-- Highest Average Rating by Film Category
SELECT c.name
FROM public.category AS c
WHERE c.category_id = (
    SELECT fc.category_id
    FROM public.film_category AS fc
    JOIN public.film AS f ON fc.film_id = f.film_id
    JOIN public.movie_review AS mr ON f.film_id = mr.film_id
    GROUP BY fc.category_id
    ORDER BY AVG(mr.rating) DESC
    LIMIT 1
);
```

Data Output | Messages | Notifications

	name character varying (25)
1	Documentary

Conclusion

Throughout this assignment, I have used a range of SQL queries from simple data retrieval to complex aggregations, manipulations, and the use of advanced SQL features such as nested queries, scalar subqueries, and the creation and querying of views. The process clarified several key topics of working with relational databases, particularly within a context similar to a DVD rental business.

Findings:

- I started with basic SELECT statements to fetch data directly from tables and then moved on to more complex queries involving JOIN clauses to retrieve related data from multiple tables.
- Insertion (INSERT), update (UPDATE), and deletion (DELETE) queries along with aggregate functions were used to modify the database contents, demonstrating how to manage data within a database actively.
- Through examples, I have learned how nested queries can be used to perform operations that require multiple steps of logic and how scalar subqueries can fetch specific data points for use in a larger query.

Challenges:

- Early on, there was a misunderstanding regarding the database schema, particularly the structure of relationships between films and categories. This led to an error in query design, showing the importance of thoroughly understanding the database schema before query construction.
- We encountered errors due to ambiguous column references in queries involving multiple tables with overlapping column names. This challenge covered the need for clear query syntax, especially when working with joins.
- Creating queries that involved multiple nested operations or conditional logic presented complexity in ensuring accuracy and efficiency.

Insights Gained:

- A deep understanding of the database schema is crucial for effective query writing. Knowing the relationships between tables enables more accurate and efficient queries.
- The assignment highlighted SQL's power in data manipulation and retrieval. Advanced features like nested queries and aggregate functions offered robust tools for data analysis.