Title Page

- **Title:** DB Assignment 4
- **Name**: Nicolas Zywalewski
- **Due Date:** 4 November 2025

SQL Section

- Creating constraints along with primary/foreign keys, consistent with the assignment directions:

```
-- ADD ALL PRIMARY KEYS
alter table actor add constraint actor_pk primary key (actor_id);
alter table address add constraint address_pk primary key (address_id);
alter table category add constraint category_pk primary key (category_id);
alter table city add constraint city_pk primary key (city_id);
alter table country add constraint country_pk primary key (country_id);
alter table customer add constraint customer_pk primary key (customer_id);
alter table film add constraint film_pk primary key (film_id);
alter table film_actor add constraint film_actor_pk primary key (actor_id, film_id); -- composite pk
alter table film_category add constraint film_category_pk primary key (film_id, category_id); -- composite pk
alter table inventory add constraint inventory_pk primary key (inventory_id);
alter table language add constraint language_pk primary key (language_id);
alter table payment add constraint payment_pk primary key (payment_id) ;
alter table rental add constraint rental_pk primary key (rental_id);
alter table staff add constraint staff_pk primary key (staff_id);
alter table store add constraint store_pk primary key (store_id);
```

```
-- FOREIGN KEYS AND CONSTRAINTS
-- ----------------------------
-- ADDRESS
alter table address
    add constraint address_fk_city_id foreign key (city_id) references city(city_id)
        on update cascade on delete cascade;   -- fk

-- CATEGORY
alter table category
    add constraint category_name_const check (name in
        ('Animation','Comedy','Family','Foreign','Sci-Fi','Travel',
        'Children','Drama','Horror','Action','Classics','Games',
        'New','Documentary','Sports','Music'));   -- valid categories

-- CITY
alter table city
    add constraint city_fk_country_id foreign key (country_id) references country(country_id)
        on update cascade on delete cascade;   -- fk

-- CUSTOMER
alter table customer
    add constraint customer_fk_address_id foreign key (address_id) references address(address_id)
        on update cascade on delete cascade,   -- fk
    add constraint customer_fk_store_id foreign key (store_id) references store(store_id)
        on update cascade on delete cascade,   -- fk
    add constraint customer_active_const check (active in (0,1));   -- 0: inactive and 1: active
```

```sql
-- FILM
alter table film
    add constraint film_fk_language_id foreign key (language_id) references language(language_id)
        on update cascade on delete cascade,   -- fk
    add constraint film_rental_duration_const check (rental_duration between 2 and 8),   -- valid duration
    add constraint film_rental_rate_const check (rental_rate between 0.99 and 6.99),   -- valid rate
    add constraint film_length_const check (length between 30 and 200),   -- valid length
    add constraint film_rating_const check (rating in ('PG','G','NC-17','PG-13','R')),   -- valid rating
    add constraint film_replacement_cost_const check (replacement_cost between 5.00 and 100.00),   -- valid cost
    add constraint film_special_features_const check (special_features in ('Behind the Scenes', 'Commentaries', 'Deleted Scenes', 'Trailers')), -- valid attributes
    add constraint film_feature2_const check (feature2 in ('Behind the Scenes', 'Commentaries', 'Deleted Scenes', 'Trailers')
        or feature2 = ''), -- valid attributes
    add constraint film_feature3_const check (feature3 in ('Behind the Scenes', 'Commentaries', 'Deleted Scenes', 'Trailers')
        or feature3 = ''), -- valid attributes
    add constraint film_feature4_const check (feature4 in ('Behind the Scenes', 'Commentaries', 'Deleted Scenes', 'Trailers')
        or feature4 = ''); -- valid attributes

-- FILM_ACTOR
alter table film_actor
    add constraint film_actor_fk_actor_id foreign key (actor_id) references actor(actor_id)
        on update cascade on delete cascade,   -- fk
    add constraint film_actor_fk_film_id foreign key (film_id) references film(film_id)
        on update cascade on delete cascade;   -- fk

-- FILM_CATEGORY
alter table film_category
    add constraint film_category_fk_film_id foreign key (film_id) references film(film_id)
        on update cascade on delete cascade,   -- fk
    add constraint film_category_fk_category_id foreign key (category_id) references category(category_id)
        on update cascade on delete cascade;   -- fk

-- INVENTORY
alter table inventory
    add constraint inventory_fk_film_id foreign key (film_id) references film(film_id)
        on update cascade on delete cascade,   -- fk
    add constraint inventory_fk_store_id foreign key (store_id) references store(store_id)
        on update cascade on delete cascade;   -- fk

-- PAYMENT
alter table payment
    modify column payment_date datetime, -- valid date
    add constraint payment_fk_customer_id foreign key (customer_id) references customer(customer_id)
        on update cascade on delete cascade,   -- fk
    add constraint payment_fk_staff_id foreign key (staff_id) references staff(staff_id)
        on update cascade on delete cascade,   -- fk
    add constraint payment_fk_rental_id foreign key (rental_id) references rental(rental_id)
        on update cascade on delete cascade,   -- fk
    add constraint payment_amount_const check (amount >= 0);   -- non-negative amount
```

```sql
-- RENTAL
alter table rental
    modify column rental_date datetime, -- valid date
    modify column return_date datetime, -- valid date
    add constraint rental_fk_inventory_id foreign key (inventory_id) references inventory(inventory_id)
        on update cascade on delete cascade,   -- fk
    add constraint rental_fk_customer_id foreign key (customer_id) references customer(customer_id)
        on update cascade on delete cascade,   -- fk
    add constraint rental_fk_staff_id foreign key (staff_id) references staff(staff_id)
        on update cascade on delete cascade,   -- fk
    add constraint rental_unique unique (rental_date,inventory_id,customer_id); -- unique contraint according to legend

-- STAFF
alter table staff
    add constraint staff_fk_address_id foreign key (address_id) references address(address_id)
        on update cascade on delete cascade,   -- fk
    add constraint staff_fk_store_id foreign key (store_id) references store(store_id)
        on update cascade on delete cascade,   -- fk
    add constraint staff_active_const check (active in (0,1));   -- active flag

-- STORE
alter table store
    add constraint store_fk_address_id foreign key (address_id) references address(address_id)
        on update cascade on delete cascade;   -- fk
```

Problem 1:

```
124     -- Query 1: What is the average length of films in each category? List the results in alphabetic order of categories.
125 •   select c.name, avg(f.length) as AvgLength
126     from film f
127     inner join film_category fc
128         on f.film_id = fc.film_id
129     inner join category c
130         on fc.category_id = c.category_id
131     group by c.name
132     order by c.name;
133
```

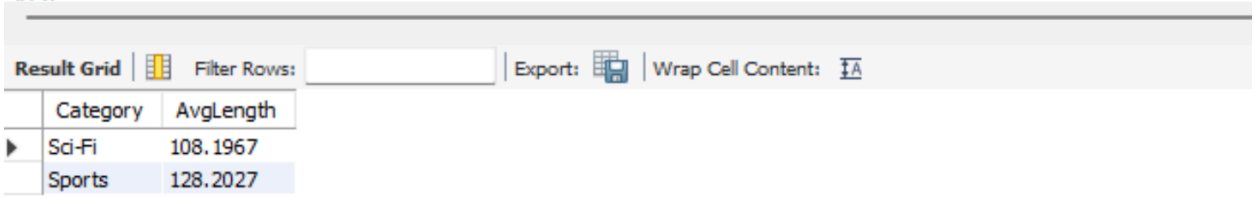
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	AvgLength
Action	111.6094
Animation	111.0152
Children	109.8000
Classics	111.6667
Comedy	115.8276
Documentary	108.7500
Drama	120.8387
Family	114.7826
Foreign	121.6986
Games	127.8361
Horror	112.4821
Music	113.6471
New	111.1270
Sci-Fi	108.1967
Sports	128.2027
Travel	113.3158

Explanation: The film, film_category, and category tables are joined to connect each film with its category. This query calculates the average film length for each category using “avg(length)” and groups by category name. Results are ordered alphabetically so each category’s average length is easy to compare.

Problem 2:

```sql
-- Query 2: Which categories have the longest and shortest average film lengths?
with category_averages as (
    select c.name as Category, avg(f.length) as AvgLength
    from film f
    inner join film_category fc
        on f.film_id = fc.film_id
    inner join category c
        on fc.category_id = c.category_id
    group by c.name
) -- this CTE gets the average length of each category of movie
-- using the CTE, we select the longest and shortest average lengths
select Category, AvgLength
from category_averages
where AvgLength = (select max(AvgLength) from category_averages)
or AvgLength = (select min(AvgLength) from category_averages);
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Category	AvgLength
Sci-Fi	108.1967
Sports	128.2027

Explanation: The same joins as number 1 are used inside a CTE to compute the average film length for every category. Using this CTE, the outer query selects the maximum and minimum averages to identify which categories have the longest and shortest films on average.

Problem 3:

```sql
-- Query 3: Which customers have rented action but not comedy or classic movies?
select cust.first_name, cust.last_name
    from customer cust
    inner join rental r on r.customer_id = cust.customer_id
    inner join inventory i on i.inventory_id = r.inventory_id
    inner join film_category fc on fc.film_id = i.film_id
    inner join category cat on cat.category_id = fc.category_id
    where cat.name = 'Action' -- customers that rented Action movies
except -- we remove the customers that rented Comedy or Classics movies
select cust.first_name, cust.last_name
    from customer cust
    inner join rental r on r.customer_id = cust.customer_id
    inner join inventory i on i.inventory_id = r.inventory_id
    inner join film_category fc on fc.film_id = i.film_id
    inner join category cat on cat.category_id = fc.category_id
    where cat.name = 'Comedy' or cat.name = 'Classics'; -- customers that rented Comedy or Classics
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name
LAWRENCE	LAWTON
MATTHEW	MAHAN
TOM	MILNER
JO	FOWLER
SCOTT	SHELLEY
EDWIN	BURK
JOANN	GARDNER
DONNA	THOMPSON
DON	BONE
JUAN	FRALEY
DOLORES	WAGNER
MICHEAL	FORMAN
AMBER	DIXON

Explanation: The customer, rental, inventory, film_category, and category tables are joined to find customers who rented Action movies. An “except” clause removes anyone who also rented Comedy or Classics. The resulting output lists only customers who rented Action films and avoided the other two categories.

Problem 4:

```
169     -- Query 4: Which actor has appeared in the most English-language movies?
170 ●   with act_lang_counts as (
171         select a.actor_id, a.first_name, a.last_name,
172         l.name as Language, count(distinct f.film_id) as MovieCount
173         from actor a
174         inner join film_actor fa on fa.actor_id = a.actor_id
175         inner join film f on f.film_id = fa.film_id
176         inner join language l on l.language_id = f.language_id
177         where l.name = 'English'
178         group by a.actor_id, a.first_name, a.last_name
179     ) -- this CTE gets each actor's count of English-language films
180     -- uisng the CTE, we select the actor with the max movie count
181     select first_name, last_name, MovieCount
182     from act_lang_counts
183     where MovieCount = (select max(MovieCount) from act_lang_counts);
```

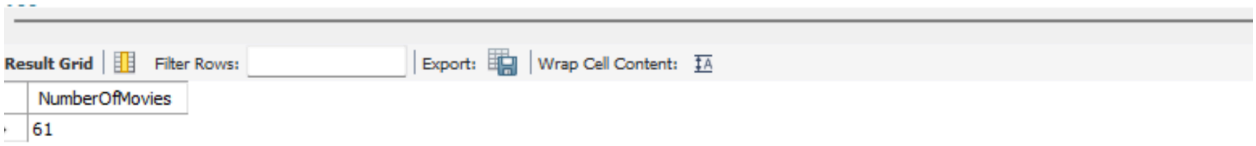
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	MovieCount
GINA	DEGENERES	42

Explanation: Actor, film_actor, film, and language tables are joined together to count how many English-language films each actor appeared in. The CTE summarizes those counts per actor, and the outer query selects the actor with the highest total. This identifies who has appeared in the most English-language movies.

Problem 5:

```
185     -- Query 5: How many distinct movies were rented for exactly 10 days from the store where Mike works?
186 •   with film_days as (
187         select f.film_id, datediff(r.return_date, r.rental_date) as DaysRented
188         from film f
189         inner join inventory i on i.film_id = f.film_id
190         inner join rental r on r.inventory_id = i.inventory_id
191         inner join store str on str.store_id = i.store_id
192         inner join staff stf on stf.store_id = str.store_id
193         where stf.first_name = 'Mike'
194     ) -- this CTE gets the number of days each film was rented from Mike's store
195     -- uisng this CTE, we count how many distinct films were rented for exactly 10 days
196     select count(distinct film_id) as NumberOfMovies
197     from film_days
198     where DaysRented = 10;
```



NumberOfMovies
61

Explanation: Film, inventory, rental, store, and staff tables are joined to get rentals from the store where Mike works. The CTE calculates how many days each film was rented by taking the date difference between return and rental dates. The outer query uses this CTE to count distinct films that were rented for exactly 10 days.

Problem 6:

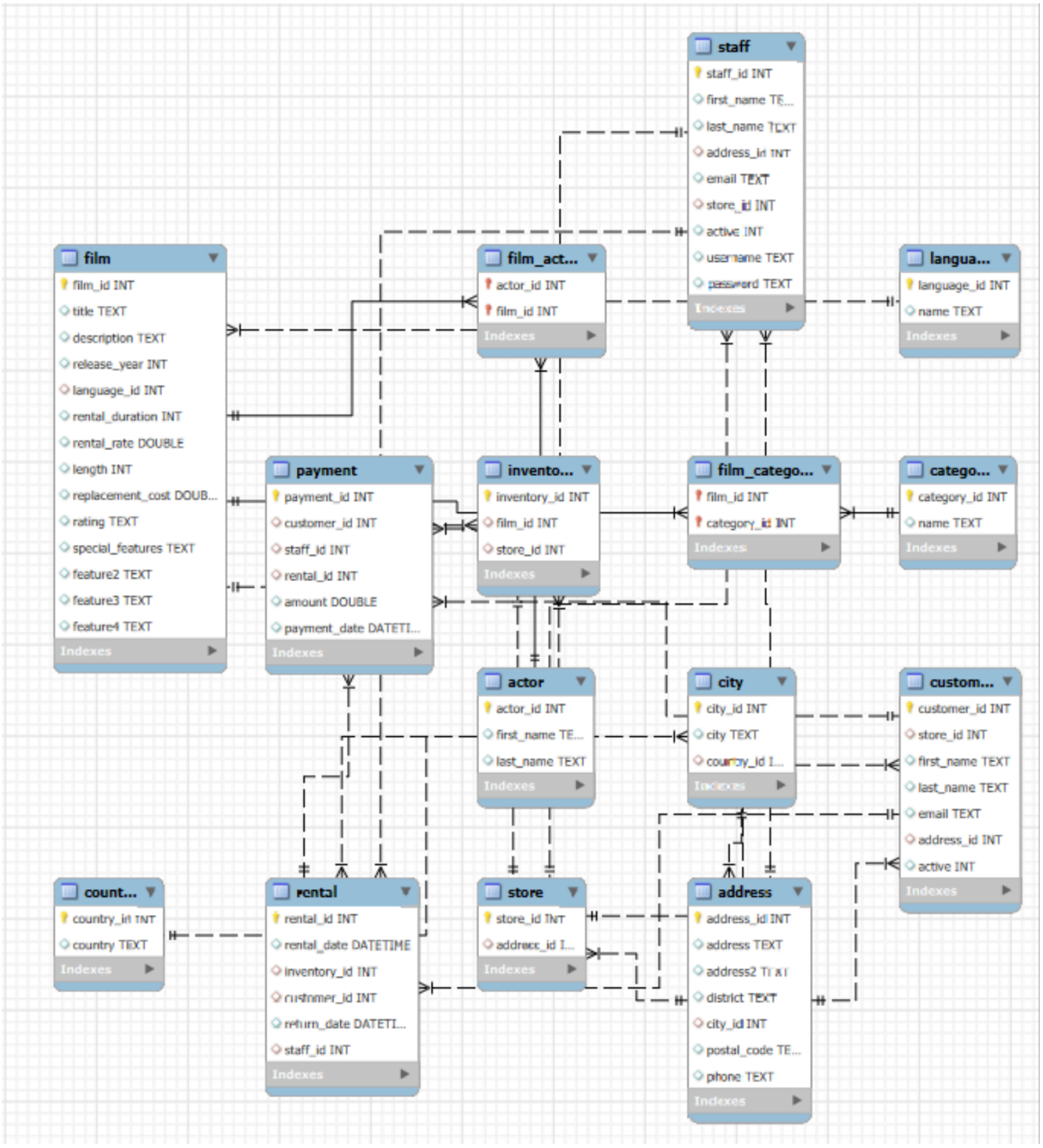
```
200     -- Query 6 Alphabetically list actors who appeared in the movie with the largest cast of actors.
201 •  with cast_sizes as (
202         select f.film_id as FilmID, f.title as MovieTitle,
203         count(distinct a.actor_id) as CastSize
204         from film f
205         inner join film_actor fa on fa.film_id = f.film_id
206         inner join actor a on a.actor_id = fa.actor_id
207         group by f.film_id
208     ) -- this CTE gets all movies with their casts sizes
209     -- using this CTE, we get the actors from the movie with the largest cast (using the cte)
210     select a.first_name as FirstName, a.last_name as LastName, MovieTitle
211     from cast_sizes cs
212     inner join film_actor fa on fa.film_id = cs.FilmID
213     inner join actor a on a.actor_id = fa.actor_id
214     where CastSize = (select max(CastSize) from cast_sizes)
215     order by a.last_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FirstName	LastName	MovieTitle
JULIA	BARRYMORE	LAMBS CINCINATTI
VAL	BOLGER	LAMBS CINCINATTI
SCARLETT	DAMON	LAMBS CINCINATTI
LUCILLE	DEE	LAMBS CINCINATTI
WOODY	HOFFMAN	LAMBS CINCINATTI
MENA	HOPPER	LAMBS CINCINATTI
REESE	KILMER	LAMBS CINCINATTI
CHRISTIAN	NEESON	LAMBS CINCINATTI
JAYNE	NOLTE	LAMBS CINCINATTI
BURT	POSEY	LAMBS CINCINATTI
MENA	TEMPLE	LAMBS CINCINATTI
WALTER	TORN	LAMBS CINCINATTI
FAY	WINSLET	LAMBS CINCINATTI

Explanation: Film, film_actor, and actor tables are joined within a CTE to compute the number of actors per movie. The outer query uses this CTE to find the film with the largest cast size and lists all of the actors in this movie (in alphabetical order). This shows the complete list of actors for the movie with the largest cast.

ERD Diagram:


staff
staff_id INT
first_name TE...
last_name TEXT
address_id INT
email TEXT
store_id INT
active INT
username TEXT
password TEXT
Indexes
film
film_id INT
title TEXT
description TEXT
release_year INT
language_id INT
rental_duration INT
rental_rate DOUBLE
length INT
replacement_cost DOUB...
rating TEXT
special_features TEXT
feature2 TEXT
feature3 TEXT
feature4 TEXT
Indexes
film_act...
actor_id INT
film_id INT
Indexes
langua...
language_id INT
name TEXT
Indexes
payment
payment_id INT
customer_id INT
staff_id INT
rental_id INT
amount DOUBLE
payment_date DATETI...
Indexes
invento...
inventory_id INT
film_id INT
store_id INT
Indexes
film_catego...
film_id INT
category_id INT
Indexes
catego...
category_id INT
name TEXT
Indexes
actor
actor_id INT
first_name TE...
last_name TEXT
Indexes
city
city_id INT
city TEXT
country_id I...
Indexes
custom...
customer_id INT
store_id INT
first_name TEXT
last_name TEXT
email TEXT
address_id INT
active INT
Indexes
count...
country_id INT
country TEXT
Indexes
rental
rental_id INT
rental_date DATETIME
inventory_id INT
customer_id INT
return_date DATETI...
staff_id INT
Indexes
store
store_id INT
address_id I...
Indexes
address
address_id INT
address TEXT
address2 TEXT
district TEXT
city_id INT
postal_code TE...
phone TEXT
Indexes