descipr

The Definitive *guide*

to practice SQL case studies to ace technical data science rounds!

"Upskill to unstop, not just fit in!"

@descipr

www.descipr.com

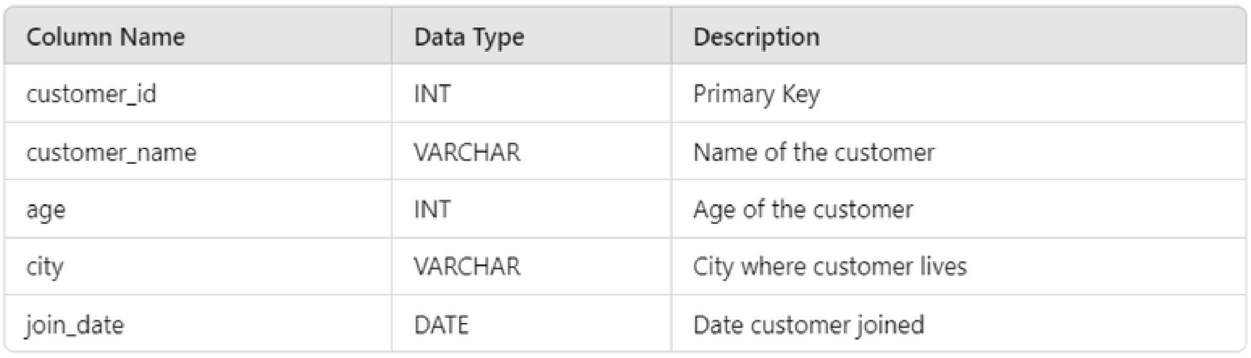
hello@descipr.com

+919148398744

Credit card transactions insights

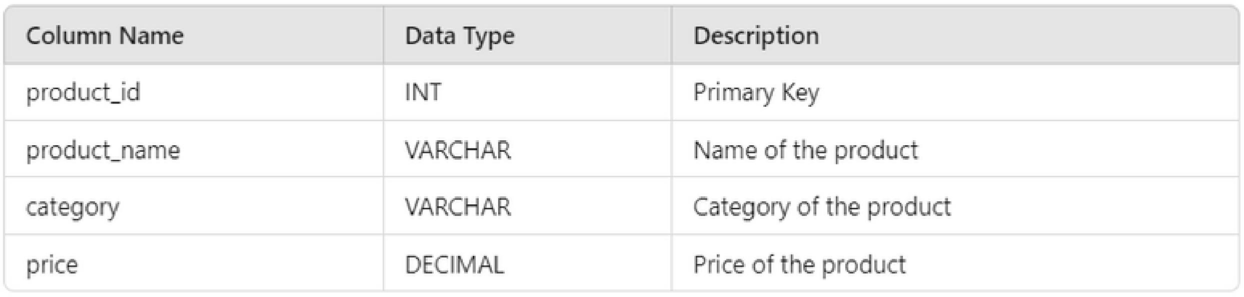
Table Structures:

`Customers`



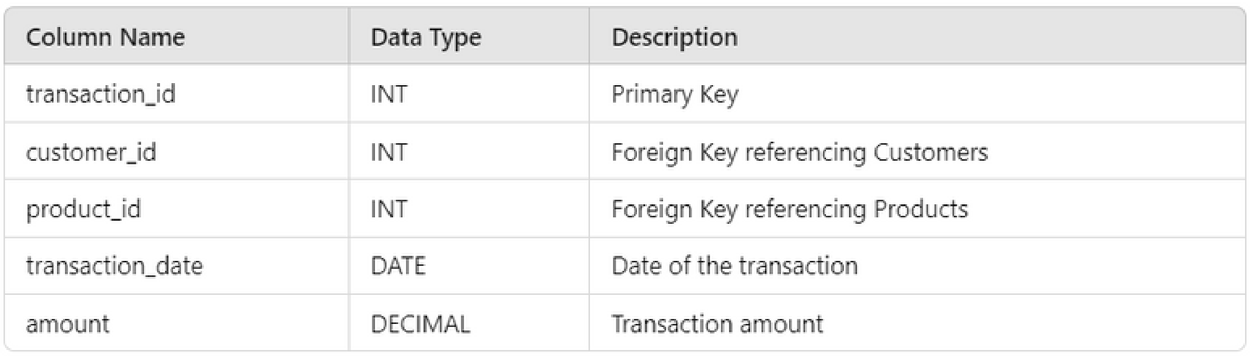
Column Name	Data Type	Description
customer_id	INT	Primary Key
customer_name	VARCHAR	Name of the customer
age	INT	Age of the customer
city	VARCHAR	City where customer lives
join_date	DATE	Date customer joined

`Products`



Column Name	Data Type	Description
product_id	INT	Primary Key
product_name	VARCHAR	Name of the product
category	VARCHAR	Category of the product
price	DECIMAL	Price of the product

`Transactions`



Column Name	Data Type	Description
transaction_id	INT	Primary Key
customer_id	INT	Foreign Key referencing Customers
product_id	INT	Foreign Key referencing Products
transaction_date	DATE	Date of the transaction
amount	DECIMAL	Transaction amount

Credit card transactions insights

Retrieve all customer names and the products they have purchased.

```
SELECT c.customer_name, p.product_name
FROM Customers c
INNER JOIN Transactions t ON c.customer_id = t.customer_id
INNER JOIN Products p ON t.product_id = p.product_id;
```

Find the total amount spent by each customer, and display their city.

```
SELECT c.customer_name, c.city, SUM(t.amount) AS total_spent
FROM Customers c
INNER JOIN Transactions t ON c.customer_id = t.customer_id
GROUP BY c.customer_name, c.city;
```

List all transactions along with the corresponding customer names and product names.

```
SELECT t.transaction_id, c.customer_name, p.product_name,
t.amount
FROM Transactions t
INNER JOIN Customers c ON t.customer_id = c.customer_id
INNER JOIN Products p ON t.product_id = p.product_id;
```

Credit card transactions insights

Find customers who bought products from a specific category, e.g., 'Electronics'.

```sql
SELECT DISTINCT c.customer_name
FROM Customers c
INNER JOIN Transactions t ON c.customer_id = t.customer_id
INNER JOIN Products p ON t.product_id = p.product_id
WHERE p.category = 'Electronics';
```

Find the total revenue generated by each product category.

```sql
SELECT p.category, SUM(t.amount) AS total_revenue
FROM Products p
INNER JOIN Transactions t ON p.product_id = t.product_id
GROUP BY p.category;
```

List customers who have spent more than $500 overall.

```sql
SELECT c.customer_name, SUM(t.amount) AS total_spent
FROM Customers c
INNER JOIN Transactions t ON c.customer_id = t.customer_id
GROUP BY c.customer_name
HAVING SUM(t.amount) > 500;
```

Credit card transactions insights

Find the number of customers from each city who made a purchase after January 1, 2024.

```
SELECT c.city, COUNT(DISTINCT c.customer_id) AS total_customers
FROM Customers c
INNER JOIN Transactions t ON c.customer_id = t.customer_id
WHERE t.transaction_date > '2024-01-01'
GROUP BY c.city;
```

Rank customers based on their total spending, and show the top 5 spenders.

```
SELECT customer_name, total_spent, RANK() OVER (ORDER BY total_spent DESC) AS rank
FROM (
    SELECT c.customer_name, SUM(t.amount) AS total_spent
    FROM Customers c
    INNER JOIN Transactions t ON c.customer_id = t.customer_id
    GROUP BY c.customer_name
) AS customer_spending
WHERE rank <= 5;
```

Credit card transactions insights

Find the 3rd highest transaction amount for each customer.

```
SELECT customer_name, amount, transaction_date
FROM (
    SELECT c.customer_name, t.amount, t.transaction_date,
           ROW_NUMBER() OVER (PARTITION BY c.customer_id ORDER
BY t.amount DESC) AS rn
    FROM Customers c
    INNER JOIN Transactions t ON c.customer_id = t.customer_id
) AS ranked_transactions
WHERE rn = 3;
```

For each customer, find their most recent transaction date and the product they purchased.

```
SELECT customer_name, product_name, transaction_date
FROM (
                    SELECT c.customer_name, p.product_name,
t.transaction_date,
           ROW_NUMBER() OVER (PARTITION BY c.customer_id ORDER
BY t.transaction_date DESC) AS rn
    FROM Customers c
    INNER JOIN Transactions t ON c.customer_id = t.customer_id
    INNER JOIN Products p ON t.product_id = p.product_id
) AS recent_transactions
WHERE rn = 1;
```

Ride sharing customer insights

Table Structures:

`Customers`

Column Name	Data Type	Description
customer_id	INT	Primary Key
customer_name	VARCHAR	Name of the customer
city	VARCHAR	City where the customer lives
join_date	DATE	Date customer joined

`Rides`

Column Name	Data Type	Description
ride_id	INT	Primary Key
customer_id	INT	Foreign Key referencing Customers
driver_id	INT	Driver who completed the ride
ride_date	DATE	Date of the ride
distance_km	DECIMAL	Distance of the ride in kilometers

`Payments`

Column Name	Data Type	Description
payment_id	INT	Primary Key
ride_id	INT	Foreign Key referencing Rides
amount	DECIMAL	Amount paid for the ride
payment_method	VARCHAR	Method of payment (e.g., card, cash)

`DriverRatings`

Column Name	Data Type	Description
rating_id	INT	Primary Key
ride_id	INT	Foreign Key referencing Rides
customer_id	INT	Foreign Key referencing Customers
rating	DECIMAL	Rating given by customer (1-5)

Ride sharing customer insights

Retrieve all ride details along with the customer names.

```
SELECT r.ride_id, c.customer_name, r.ride_date, r.distance_km
FROM Rides r
INNER JOIN Customers c ON r.customer_id = c.customer_id;
```

List all rides where the distance is greater than 10 kilometers, along with customer names and payment amounts.

```
SELECT c.customer_name, r.ride_id, r.distance_km, p.amount
FROM Rides r
INNER JOIN Customers c ON r.customer_id = c.customer_id
INNER JOIN Payments p ON r.ride_id = p.ride_id
WHERE r.distance_km > 10;
```

Retrieve all payments made by customers who joined after January 1, 2024.

```
SELECT c.customer_name, p.amount, p.payment_method
FROM Payments p
INNER JOIN Rides r ON p.ride_id = r.ride_id
INNER JOIN Customers c ON r.customer_id = c.customer_id
WHERE c.join_date > '2024-01-01';
```

Ride sharing customer insights

Find all customers who paid by card.

```
SELECT DISTINCT c.customer_name
FROM Customers c
INNER JOIN Rides r ON c.customer_id = r.customer_id
INNER JOIN Payments p ON r.ride_id = p.ride_id
WHERE p.payment_method = 'card';
```

Find the total amount paid by each customer.

```
SELECT c.customer_name, SUM(p.amount) AS total_amount
FROM Customers c
INNER JOIN Rides r ON c.customer_id = r.customer_id
INNER JOIN Payments p ON r.ride_id = p.ride_id
GROUP BY c.customer_name;
```

List the number of rides taken by each customer in a specific city, e.g., 'New York'.

```
SELECT c.customer_name, COUNT(r.ride_id) AS total_rides
FROM Customers c
INNER JOIN Rides r ON c.customer_id = r.customer_id
WHERE c.city = 'New York'
GROUP BY c.customer_name;
```

Ride sharing customer insights

Find customers who have spent more than $500 in total across all rides.

```
SELECT c.customer_name, SUM(p.amount) AS total_spent
FROM Customers c
INNER JOIN Rides r ON c.customer_id = r.customer_id
INNER JOIN Payments p ON r.ride_id = p.ride_id
GROUP BY c.customer_name
HAVING SUM(p.amount) > 500;
```

Rank customers based on the total distance traveled and show the top 3.

```
SELECT customer_name, total_distance, RANK() OVER (ORDER BY
total_distance DESC) AS rank
FROM (
        SELECT   c.customer_name,   SUM(r.distance_km)   AS
total_distance
    FROM Customers c
    INNER JOIN Rides r ON c.customer_id = r.customer_id
    GROUP BY c.customer_name
) AS customer_distances
WHERE rank <= 3;
```

Ride sharing customer insights

Find the 3rd longest ride for each customer.

```
SELECT customer_name, distance_km, ride_date
FROM (
    SELECT c.customer_name, r.distance_km, r.ride_date,
        ROW_NUMBER() OVER (PARTITION BY c.customer_id ORDER
BY r.distance_km DESC) AS rn
    FROM Customers c
    INNER JOIN Rides r ON c.customer_id = r.customer_id
) AS ranked_rides
WHERE rn = 3;
```

For each customer, find the average rating of the drivers they have rated more than once.

```
SELECT customer_name, AVG(rating) AS average_rating
FROM (
    SELECT c.customer_name, dr.rating, COUNT(dr.rating_id) AS
rating_count
    FROM Customers c
        INNER JOIN DriverRatings dr ON c.customer_id =
dr.customer_id
    GROUP BY c.customer_name, dr.rating
    HAVING COUNT(dr.rating_id) > 1
) AS customer_ratings
GROUP BY customer_name;
```

Website traffic Google analytics

Table Structures:

`Users`

Column Name	Data Type	Description
user_id	INT	Primary Key
user_name	VARCHAR	Name of the user
registration_date	DATE	Date the user registered
country	VARCHAR	Country of the user

`PageViews`

Column Name	Data Type	Description
pageview_id	INT	Primary Key
user_id	INT	Foreign Key referencing Users
page_url	VARCHAR	URL of the page viewed
view_timestamp	TIMESTAMP	Time of the pageview
session_duration	INT	Duration of the session in seconds

`Purchases`

Column Name	Data Type	Description
purchase_id	INT	Primary Key
user_id	INT	Foreign Key referencing Users
purchase_date	DATE	Date of the purchase
total_amount	DECIMAL	Total amount spent on the purchase

`TrafficSources`

Column Name	Data Type	Description
traffic_id	INT	Primary Key
user_id	INT	Foreign Key referencing Users
source	VARCHAR	Traffic source (e.g., 'organic', 'paid', 'referral')
medium	VARCHAR	Marketing medium (e.g., 'email', 'CPC', 'social')
campaign	VARCHAR	Campaign identifier

Website traffic
Google analytics

Find the last page each user viewed before making a purchase.

```
SELECT u.user_name, p.page_url, pv.view_timestamp
FROM Purchases p
INNER JOIN PageViews pv ON p.user_id = pv.user_id
INNER JOIN Users u ON p.user_id = u.user_id
WHERE pv.view_timestamp < p.purchase_date
ORDER BY pv.view_timestamp DESC;
```

List all users who visited more than 3 distinct pages during any single session.

```
SELECT u.user_name, COUNT(DISTINCT pv.page_url) AS
pages_viewed
FROM Users u
INNER JOIN PageViews pv ON u.user_id = pv.user_id
GROUP BY u.user_name, pv.session_duration
HAVING COUNT(DISTINCT pv.page_url) > 3;
```

Website traffic
Google analytics

Find all traffic sources for users who registered in the last 30 days.

```
SELECT u.user_name, t.source, t.medium
FROM Users u
INNER JOIN TrafficSources t ON u.user_id = t.user_id
WHERE u.registration_date > CURRENT_DATE - INTERVAL '30 days';
```

Retrieve the first page visited by every user.

```
SELECT u.user_name, pv.page_url, MIN(pv.view_timestamp) AS first_view
FROM Users u
INNER JOIN PageViews pv ON u.user_id = pv.user_id
GROUP BY u.user_name, pv.page_url
ORDER BY first_view;
```

Website traffic
Google analytics

Find the average session duration for users who have made at least one purchase.

```
SELECT u.user_name, AVG(pv.session_duration) AS avg_session_duration
FROM Users u
INNER JOIN PageViews pv ON u.user_id = pv.user_id
INNER JOIN Purchases p ON u.user_id = p.user_id
GROUP BY u.user_name;
```

Find users who generated more than 50 page views within a single session.

```
SELECT u.user_name, pv.session_duration, COUNT(pv.pageview_id) AS views_in_session
FROM Users u
INNER JOIN PageViews pv ON u.user_id = pv.user_id
GROUP BY u.user_name, pv.session_duration
HAVING COUNT(pv.pageview_id) > 50;
```

Website traffic
Google analytics

Calculate the total amount spent by users who came from 'organic' traffic sources.

```
SELECT u.user_name, SUM(p.total_amount) AS total_spent
FROM Users u
INNER JOIN Purchases p ON u.user_id = p.user_id
INNER JOIN TrafficSources t ON u.user_id = t.user_id
WHERE t.source = 'organic'
GROUP BY u.user_name;
```

Find the user with the longest cumulative session time.

```
SELECT user_name, total_session_time
FROM (
        SELECT u.user_name, SUM(pv.session_duration) AS total_session_time,
        RANK() OVER (ORDER BY SUM(pv.session_duration) DESC) AS rank
    FROM Users u
    INNER JOIN PageViews pv ON u.user_id = pv.user_id
    GROUP BY u.user_name
) AS session_ranks
WHERE rank = 1;
```

Website traffic
Google analytics

Find each user's 2nd most visited page URL.

```
SELECT user_name, page_url, view_count
FROM (
    SELECT u.user_name, pv.page_url, COUNT(pv.pageview_id) AS view_count,
            ROW_NUMBER() OVER (PARTITION BY u.user_id ORDER BY COUNT(pv.pageview_id) DESC) AS rn
    FROM Users u
    INNER JOIN PageViews pv ON u.user_id = pv.user_id
    GROUP BY u.user_name, pv.page_url
) AS ranked_views
WHERE rn = 2;
```

Find users who have used more than 3 different traffic sources.

```
SELECT user_name, traffic_sources_count
FROM (
            SELECT u.user_name, COUNT(DISTINCT t.source) AS traffic_sources_count
    FROM Users u
    INNER JOIN TrafficSources t ON u.user_id = t.user_id
    GROUP BY u.user_name
) AS user_sources
WHERE traffic_sources_count > 3;
```

descipr

Ecommerce purchase behaviour analysis

Table Structures:

`Customers`

Column Name	Data Type	Description
customer_id	INT	Primary Key
customer_name	VARCHAR	Name of the customer
registration_date	DATE	Date the customer registered
country	VARCHAR	Country of the customer

`Orders`

Column Name	Data Type	Description
order_id	INT	Primary Key
customer_id	INT	Foreign Key referencing `Customers`
order_date	DATE	Date when the order was placed
total_amount	DECIMAL	Total amount spent on the order

`OrderDetails`

Column Name	Data Type	Description
order_id	INT	Foreign Key referencing `Orders`
product_id	INT	Foreign Key referencing `Products`
quantity	INT	Quantity of the product in the order
unit_price	DECIMAL	Price of each unit

`Products`

Column Name	Data Type	Description
product_id	INT	Primary Key
product_name	VARCHAR	Name of the product
category	VARCHAR	Category of the product
price	DECIMAL	Price of the product

Ecommerce purchase behaviour analysis

Find all orders placed by each customer along with the total amount spent.

```
SELECT c.customer_name, o.order_id, o.total_amount
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id;
```

List all products purchased by a customer along with their price and order date.

```
SELECT c.customer_name, p.product_name, o.order_date,
od.unit_price
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id
INNER JOIN OrderDetails od ON o.order_id = od.order_id
INNER JOIN Products p ON od.product_id = p.product_id;
```

Ecommerce purchase behaviour analysis

Retrieve all products along with the total quantity sold for each.

```
SELECT p.product_name, SUM(od.quantity) AS
total_quantity_sold
FROM Products p
INNER JOIN OrderDetails od ON p.product_id = od.product_id
GROUP BY p.product_name;
```

Find all customers who made at least one purchase in the last 30 days.

```
SELECT DISTINCT c.customer_name
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.order_date > CURRENT_DATE - INTERVAL '30 days';
```

Ecommerce purchase behaviour analysis

Find the average amount spent per customer.

```
SELECT c.customer_name, AVG(o.total_amount) AS
average_spent
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name;
```

Find the total revenue generated by each product category.

```
SELECT p.category, SUM(od.quantity * od.unit_price) AS
total_revenue
FROM Products p
INNER JOIN OrderDetails od ON p.product_id = od.product_id
GROUP BY p.category;
```

Ecommerce purchase behaviour analysis

Find customers who have placed more than 5 orders.

```
SELECT c.customer_name, COUNT(o.order_id) AS order_count
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name
HAVING COUNT(o.order_id) > 5;
```

Find the top 3 products by total sales (quantity sold).

```
SELECT product_name, total_quantity
FROM (
   SELECT p.product_name, SUM(od.quantity) AS total_quantity,
         RANK() OVER (ORDER BY SUM(od.quantity) DESC) AS rank
   FROM Products p
   INNER JOIN OrderDetails od ON p.product_id = od.product_id
   GROUP BY p.product_name
) AS ranked_products
WHERE rank <= 3;
```

Ecommerce purchase behaviour analysis

Find each customer’s 2nd highest total order amount.

```
SELECT customer_name, total_amount
FROM (
    SELECT c.customer_name, o.total_amount,
           ROW_NUMBER() OVER (PARTITION BY c.customer_id ORDER
BY o.total_amount DESC) AS rn
    FROM Customers c
    INNER JOIN Orders o ON c.customer_id = o.customer_id
) AS ranked_orders
WHERE rn = 2;
```

Find the total revenue generated by customers who registered more than a year ago but made purchases in the last 3 months.

```
SELECT SUM(o.total_amount) AS total_revenue
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id
WHERE c.registration_date < CURRENT_DATE - INTERVAL '1 year'
  AND o.order_date > CURRENT_DATE - INTERVAL '3 months';
```

OTT streaming behaviour analysis

Table Structures:

`Users`

Column Name	Data Type	Description
user_id	INT	Primary Key
user_name	VARCHAR	Name of the user
subscription_date	DATE	Date the user subscribed
subscription_plan	VARCHAR	Subscription plan (e.g., 'Basic', 'Premium')
country	VARCHAR	Country of the user

`Streams`

Column Name	Data Type	Description
stream_id	INT	Primary Key
user_id	INT	Foreign Key referencing Users
content_id	INT	Foreign Key referencing Content
start_time	TIMESTAMP	When the streaming session started
end_time	TIMESTAMP	When the streaming session ended
duration_minutes	INT	Total duration of the session (in minutes)

OTT streaming behaviour analysis

Table Structures:

`Content`

Column Name	Data Type	Description
content_id	INT	Primary Key
content_title	VARCHAR	Title of the content
genre	VARCHAR	Genre of the content (e.g., 'Drama', 'Action')
release_year	INT	Year the content was released
duration_minutes	INT	Total duration of the content (in minutes)

`ContentRatings`

Column Name	Data Type	Description
rating_id	INT	Primary Key
content_id	INT	Foreign Key referencing Content
user_id	INT	Foreign Key referencing Users
rating	DECIMAL	User rating for the content (1-5)

OTT streaming behaviour analysis

Find all the content titles streamed by each user.

```
SELECT u.user_name, c.content_title
FROM Users u
INNER JOIN Streams s ON u.user_id = s.user_id
INNER JOIN Content c ON s.content_id = c.content_id;
```

List all content in the 'Drama' genre along with the number of times it has been streamed.

```
SELECT c.content_title, COUNT(s.stream_id) AS total_streams
FROM Content c
INNER JOIN Streams s ON c.content_id = s.content_id
WHERE c.genre = 'Drama'
GROUP BY c.content_title;
```

OTT streaming behaviour analysis

Find the total time spent by each user watching content.

```
SELECT u.user_name, SUM(s.duration_minutes) AS
total_time_spent
FROM Users u
INNER JOIN Streams s ON u.user_id = s.user_id
GROUP BY u.user_name;
```

List all users who have streamed content for more than 1000 minutes in total.

```
SELECT u.user_name, SUM(s.duration_minutes) AS total_minutes
FROM Users u
INNER JOIN Streams s ON u.user_id = s.user_id
GROUP BY u.user_name
HAVING SUM(s.duration_minutes) > 1000;
```

OTT streaming behaviour analysis

Find the average duration of all streaming sessions for each genre.

```
SELECT c.genre, AVG(s.duration_minutes) AS
average_stream_duration
FROM Content c
INNER JOIN Streams s ON c.content_id = s.content_id
GROUP BY c.genre;
```

Find users who have streamed content from more than 3 distinct genres.

```
SELECT u.user_name, COUNT(DISTINCT c.genre) AS
distinct_genres
FROM Users u
INNER JOIN Streams s ON u.user_id = s.user_id
INNER JOIN Content c ON s.content_id = c.content_id
GROUP BY u.user_name
HAVING COUNT(DISTINCT c.genre) > 3;
```

OTT streaming behaviour analysis

Find the total number of streams for content released after 2020.

```
SELECT c.content_title, COUNT(s.stream_id) AS total_streams
FROM Content c
INNER JOIN Streams s ON c.content_id = s.content_id
WHERE c.release_year > 2020
GROUP BY c.content_title;
```

Find the top 3 most-watched content by total streaming time.

```
SELECT content_title, total_streaming_time
FROM (
        SELECT c.content_title, SUM(s.duration_minutes) AS total_streaming_time,
        RANK() OVER (ORDER BY SUM(s.duration_minutes) DESC) AS rank
    FROM Content c
    INNER JOIN Streams s ON c.content_id = s.content_id
    GROUP BY c.content_title
) AS ranked_content
WHERE rank <= 3;
```

OTT streaming behaviour analysis

Find each user's 2nd highest-rated content.

```
SELECT user_name, content_title, rating
FROM (
    SELECT u.user_name, c.content_title, cr.rating,
           ROW_NUMBER() OVER (PARTITION BY u.user_id ORDER BY
cr.rating DESC) AS rank
    FROM Users u
    INNER JOIN ContentRatings cr ON u.user_id = cr.user_id
    INNER JOIN Content c ON cr.content_id = c.content_id
) AS ranked_ratings
WHERE rank = 2;
```

Find the total time spent by users on the 'Premium' subscription plan for content released before 2019.

```
SELECT SUM(s.duration_minutes) AS total_time_spent
FROM Users u
INNER JOIN Streams s ON u.user_id = s.user_id
INNER JOIN Content c ON s.content_id = c.content_id
WHERE u.subscription_plan = 'Premium'
  AND c.release_year < 2019;
```