UNIT 2: INTRODUCTION TO DATA AND DATABASES

Queries & Subqueries

LAGUARDIA COMMUNITY COLLEGE

The WHY!

Week 3

Upcoming Assignments (Required):

- Assignment 4A - Holiday (Optional/Extra Credit Review | Complete What You Want!)
- Assignment 4B - Due Sunday, April 14th, 11:59PM (Finding a Data Set & Practicing Queries)
- **Challenge Question:** Using AI with Databases I

Note: If you would like to complete any of our challenge questions throughout the semester for practice, please let me know on Slack and I will open them for you.

FORMING QUERIES

- A query is a request for data or information from a database table or combination of tables.
- Queries can both retrieve and manipulate data, depending on the needs of the user.

FORMING QUERIES - A Standard Process

Step 1 - Define Objective
Clearly identify what you want to achieve. Example: "List all customers who rented movies last month.”

Step 2 - Identify Relevant Tables
Determine which tables contain the data you need. Example: `customer`, `rental`.

Step 3 - Specify Required Data
Decide which pieces of information (columns) you need in your result. Example: Customer names, rental dates.

Step 4 - Consider the Relationship
Understand how data across different tables is related. Do you need to use JOINs?

Step 5 - Filtering Data
Decide if you need to filter data using conditions (WHERE clause). (i.e. Examples from Last Month)

Step 6 - Aggregate the Data
Determine if you need to summarize data (using GROUP BY, COUNT, SUM, etc.). Example: Total rentals per customer.

Step 7 - Ordering Results
Consider if the order of the results matters (ORDER BY clause). Example: Order by rental date.

SUBQUERY

- Subqueries allow for multiple SELECT statement, allowing for more detailed queries!
- A subquery is a SQL query nested inside a larger query.
- A subquery may occur in :
  - A SELECT clause
  - A FROM clause
  - A WHERE clause

Column Subqueries

SELECT

first_name, last_name

FROM

actor

WHERE

actor_id IN (SELECT actor_id FROM film_actor WHERE film_id = 1);

Row Subqueries

SELECT * FROM customer

WHERE (customer_id, store_id) = (SELECT customer_id, store_id FROM rental WHERE rental_id = 1);

Table Subqueries

SELECT

dt.customer_id, COUNT(*) AS total_rentals

FROM

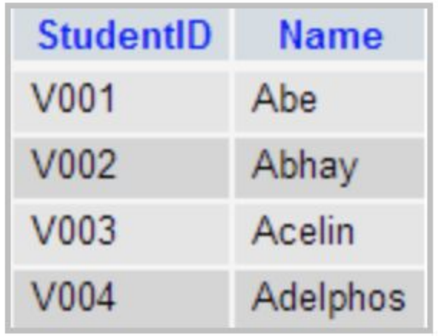
(SELECT customer_id FROM rental WHERE return_date IS NULL) AS dt

GROUP BY

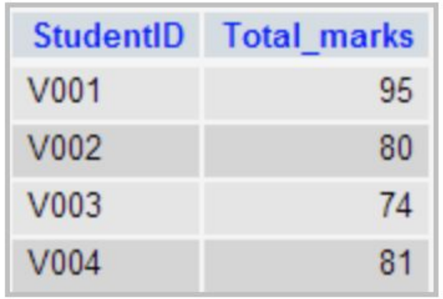
dt.customer_id;

SUBQUERY … WHY??

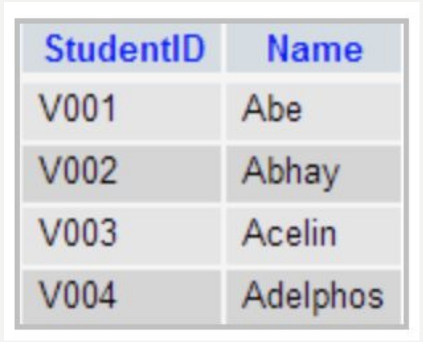
EXAMPLE TABLES:



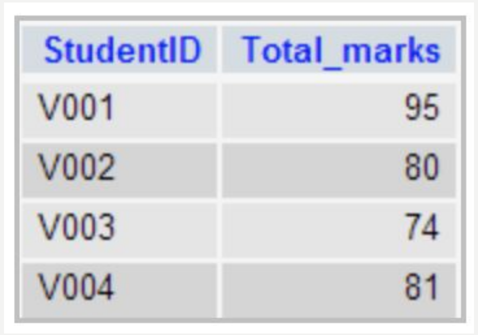
StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos



StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

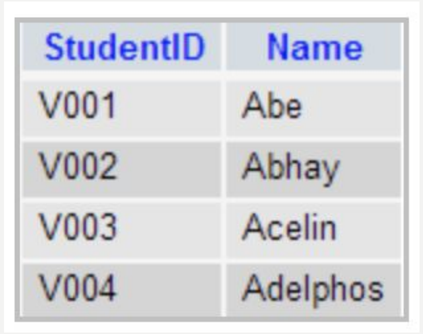


StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos

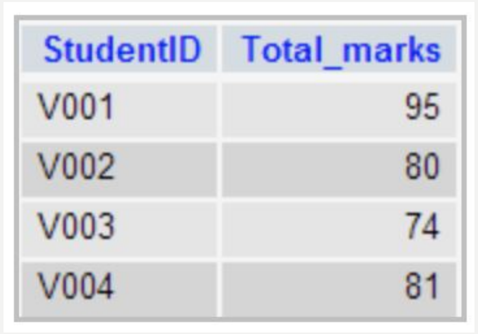


StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

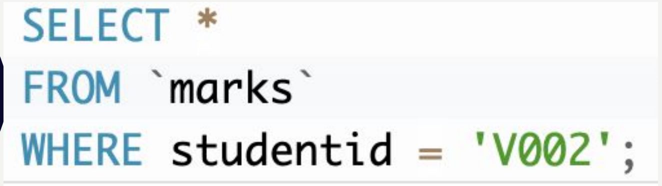
- We want to write a query to identify all students who get better marks than that of the student whose StudentID is 'V002', but we do not know the mark of 'V002'.
- To solve the problem, we require two queries. One query returns the marks (stored in Total_marks field) of 'V002' and a second query identifies the students who get better marks than the result of the first query.



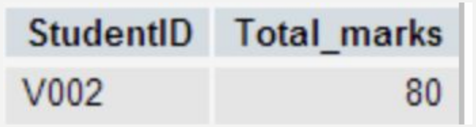
StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos



StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

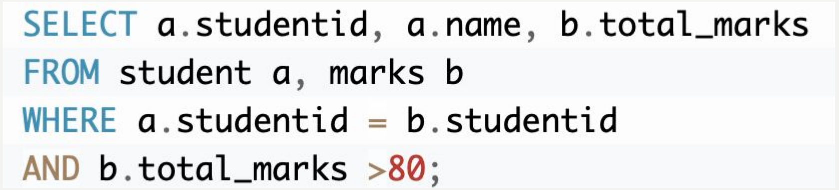


```sql
SELECT *
FROM `marks`
WHERE studentid = 'V002';
```

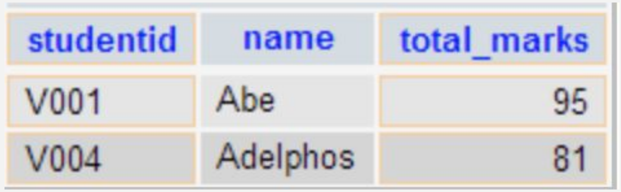


StudentID	Total_marks
V002	80

Using the results of the first query, we can write the second



```sql
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >80;
```

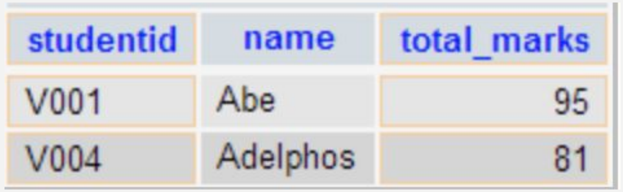


studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

WITH THE SUBQUERY, YOU CAN COMBINE BOTH QUERIES INTO ONE

- With the subquery, you place one query inside of the other.

```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.student id = b.studentid
AND b.total_marks > ( SELECT total_marks FROM marks WHERE studentid = ‘V002’);
```



studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

SUBQUERIES … FOLLOW ALONG IN YOUR DB

We want to find the films whose rental rate is higher than the average rental rate.

Similar to the previous example, we can do this in two steps.

Find the AVG(rental_rate)

Use the result from our first query in the second statement

SUBQUERIES … FOLLOW ALONG IN YOUR DB

Like we spoke about earlier, requiring two steps isn't very optimized. We need to quickly solve business cases.

We can pass one query into a second query.

GUIDELINES

- A subquery must be enclosed in parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Subqueries cannot manipulate their results internally, therefore ORDER BY clause **cannot** be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.

RECAP

- Subqueries allow for multiple SELECT Statement, allowing for more detailed queries!

- A subquery is a SQL query nested inside a larger query. A way to think of it is that it works like a placeholder.

Column Subquery

```
SELECT
first_name , last_name
FROM
Actor
WHERE actor_id IN (SELECT actor_id FROM film_actor WHERE film_id = 1);
```

Row Subquery

```
SELECT * FROM customer WHERE (customer_id, store_id) = (SELECT customer_id, store_id FROM rental WHERE rental_id = 1);
```

Table Subquery

```
SELECT
dt.customer_id , COUNT(*) AS total_rentals
FROM
(SELECT customer_id FROM rental WHERE return_date IS NULL) AS dt
GROUP BY
dt.customer_id;
```

Scalar Subquery

```
SELECT
    (SELECT AVG(rental_rate) FROM film) AS average_rental_rate,
    (SELECT AVG(length) FROM film) AS average_film_length;
```

____

```
SELECT
    (SELECT AVG(rental_rate) FROM film) AS average_rental_rate,
    (SELECT name FROM category
     JOIN film_category ON category.category_id = film_category.category_id
     GROUP BY name
     ORDER BY COUNT(film_category.film_id) DESC
     LIMIT 1) AS most_popular_genre;
```

SELF-JOINS

SELF JOINS

A self JOIN is a regular join, but the table is joined with itself.

Self joins are used when you want to combine rows with other rows from the same table.

To perform the self join, you must use a table alias so SQL can determine which is the LEFT and RIGHT table from the same table.

SELF JOINS

Previously, inner and outer joins were used to help deal with combining data from multiple tables.

Self join allows you to refer to the same table twice – as if it were two separate tables.

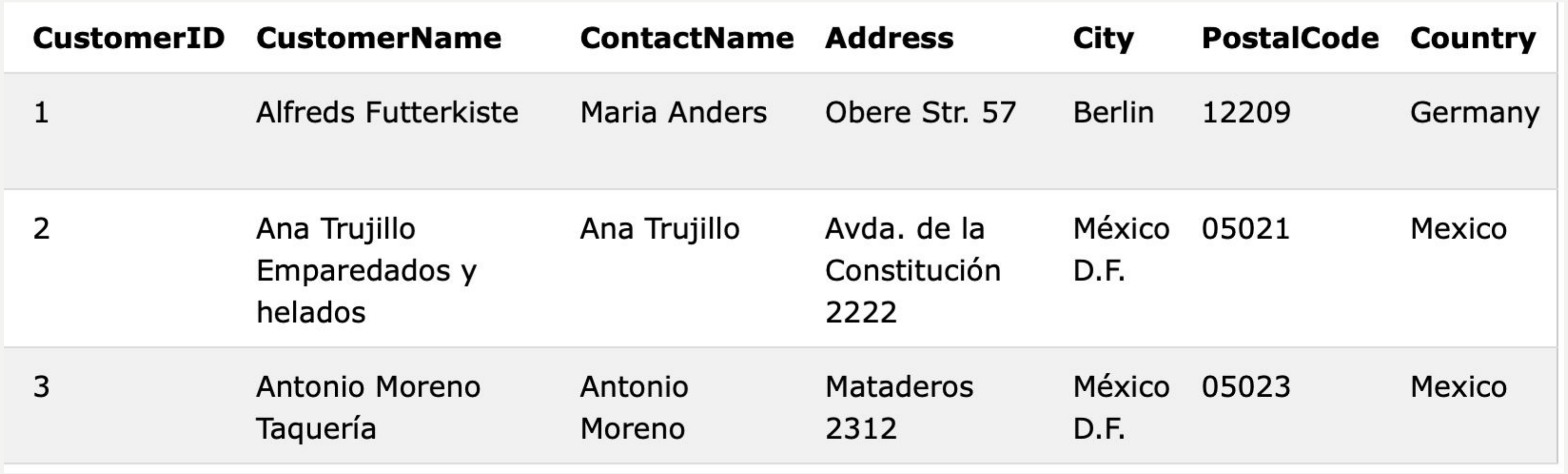
The self join essential creates a virtual view of a table, allowing it to be used more than once.

SELF JOIN - BASIC SYNTAX

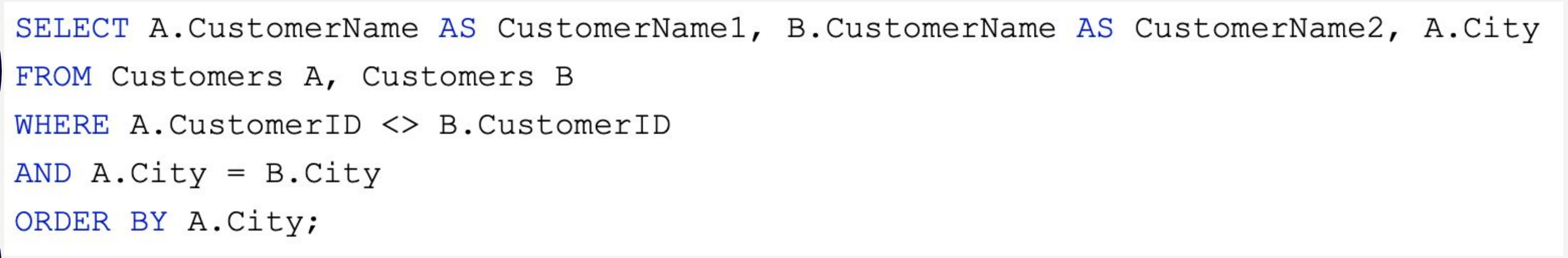
```
SELECT column_name(s)
FROM tableA t1, tableA t2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

EXAMPLE

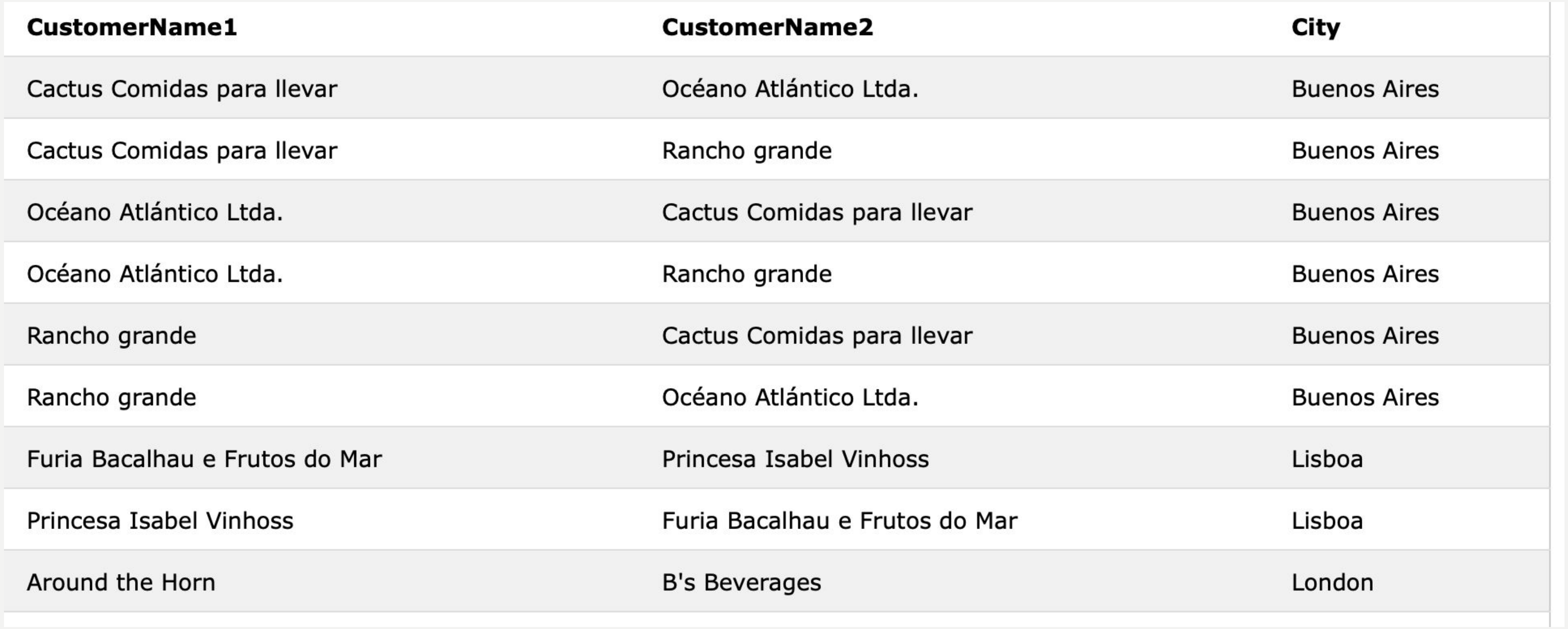


CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico



```sql
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

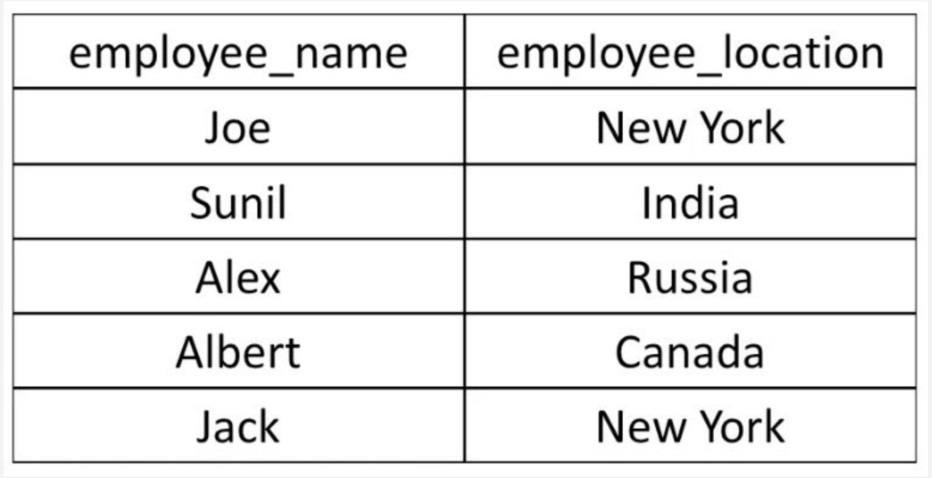
EXAMPLE RESULTS



CustomerName1	CustomerName2	City
Cactus Comidas para llevar	Océano Atlántico Ltda.	Buenos Aires
Cactus Comidas para llevar	Rancho grande	Buenos Aires
Océano Atlántico Ltda.	Cactus Comidas para llevar	Buenos Aires
Océano Atlántico Ltda.	Rancho grande	Buenos Aires
Rancho grande	Cactus Comidas para llevar	Buenos Aires
Rancho grande	Océano Atlántico Ltda.	Buenos Aires
Furia Bacalhau e Frutos do Mar	Princesa Isabel Vinhoss	Lisboa
Princesa Isabel Vinhoss	Furia Bacalhau e Frutos do Mar	Lisboa
Around the Horn	B's Beverages	London

SELF-JOIN: SAMPLE TABLE

Let’s try to find which employees are from the same location as the employee Joe - which is New York



employee_name	employee_location
Joe	New York
Sunil	India
Alex	Russia
Albert	Canada
Jack	New York

SELF-JOIN

- A subquery is the optimal query – especially for performance.
- Always use an aliases (AS)

```
SELECT
    f1.title AS "Film 1",
    f2.title AS "Film 2",
    f1.rating AS "Rating"
FROM
    film f1
INNER JOIN
    film f2 ON f1.rating = f2.rating AND f1.film_id < f2.film_id
ORDER BY
    f1.rating, f1.title;
```

JOINS REVIEW: ACTIVITY

1. Identify movies that are currently rented out. In the `dvdrental` database, the `rental` table has `return_date` to indicate when a movie was returned. If `return_date` is NULL, it means the movie hasn't been returned yet.

2. Imagine you have a DVD rental store. You want to list all customers, including those who have not rented any DVDs yet, along with any DVDs they have rented.

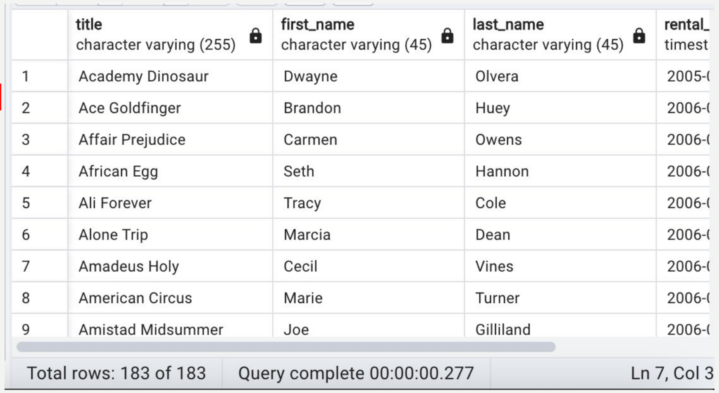
3. In the same DVD rental store, you're interested in seeing all DVDs, regardless of whether they've been rented out, and details about any rentals.

4. For a promotional event, you want to create a list of all possible pairings of customers with DVDs for a special offer mail-out.

5. Your DVD rental store is planning a multicultural film night and wants to highlight movies in various languages to cater to a diverse audience. The task is to compile a list of all unique film languages available in your inventory to help with the selection process.

JOINS REVIEW: ACTIVITY

Answer 1 to beat the 'game'; and 3 to win! (1



	title character varying (255)	first_name character varying (45)	last_name character varying (45)	rental_ timest
1	Academy Dinosaur	Dwayne	Olvera	2005-(
2	Ace Goldfinger	Brandon	Huey	2006-(
3	Affair Prejudice	Carmen	Owens	2006-(
4	African Egg	Seth	Hannon	2006-(
5	Ali Forever	Tracy	Cole	2006-(
6	Alone Trip	Marcia	Dean	2006-(
7	Amadeus Holy	Cecil	Vines	2006-(
8	American Circus	Marie	Turner	2006-(
9	Amistad Midsummer	Joe	Gilliland	2006-(

Total rows: 183 of 183 Query complete 00:00:00.277 Ln 7, Col 3

Format for Answering Questions:

1. What is being asked here? (i.e. objective of question)
2. What should my output look like?
3. Type of JOIN and/or other Clauses
4. Query & Output

Example: You're tasked with identifying customers who have not yet rented any movies.

1. Create a list of all customers who have not rented any film yet.
2. The output should be a list of customers, including their `customer_id`, `first_name`, and `last_name`, who have no entries in the `rental` table associated with their `customer_id`.
3. NO JOIN CLAUSE | WHERE, FROM, SELECT, NOT EXISTS
4. Query:

```
SELECT customer_id, first_name, last_name
FROM customer
WHERE NOT EXISTS (
   SELECT 1
   FROM rental
   WHERE rental.customer_id = customer.customer_id
);
```

JOINS REVIEW: Group 1

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: Group 2

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: Group 3

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: Group 4

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: *Answers*

1. Identify Movies that are currently rented out.

```
SELECT film.title
FROM rental
JOIN inventory ON
rental.inventory_id =
inventory.inventory_id
JOIN film ON inventory.film_id =
film.film_id
WHERE rental.return_date IS
NULL;
```

2. List all customers, including those who have not rented any DVDs yet, along with any DVDs they have rented

```
SELECT customer.customer_id,
customer.first_name,
customer.last_name, film.title
FROM customer
LEFT JOIN rental ON
customer.customer_id =
rental.customer_id
LEFT JOIN inventory ON
rental.inventory_id =
inventory.inventory_id
LEFT JOIN film ON
inventory.film_id = film.film_id;
```

3. See all DVDs, regardless of whether they've been rented out, and details about any rentals

```
SELECT film.title,
rental.rental_date,
customer.first_name,
customer.last_name
FROM film
LEFT JOIN inventory ON
film.film_id = inventory.film_id
LEFT JOIN rental ON
inventory.inventory_id =
rental.inventory_id
LEFT JOIN customer ON
rental.customer_id =
customer.customer_id;
```

JOINS REVIEW: *Answers*

4. Create a list of all possible pairings of customers with DVDs for a special offer mail-out	5. Compile a list of all unique film languages available in your inventory

SELECT customer.first_name, customer.last_name, film.title
FROM customer
CROSS JOIN film;
	SELECT DISTINCT language.name
	FROM film
	JOIN language ON film.language_id = language.language_id;
	
	OR
	
	SELECT DISTINCT name
	FROM language;