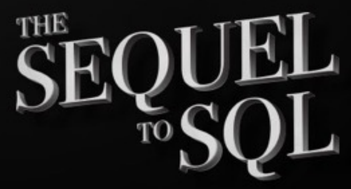
The Sequel to SQL:
Level 5
Subqueries



Performing Simple Subqueries

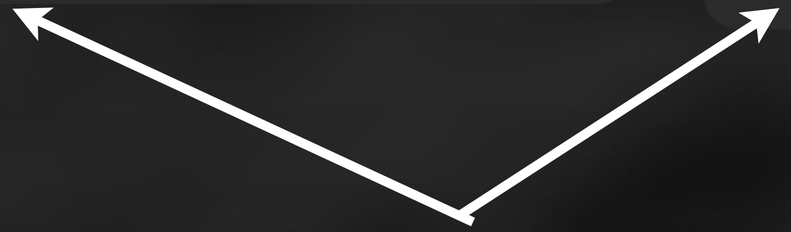
Let's find the **sum of all sales** for **Movies** that were showing as a **Non-cash** promotion.

Movies

id	title	sales	duration
1	Don Juan	45000	110
2	Peter Pan	30000	105
3	The Lost World	30000	106
4	Robin Hood	50000	143

Promotions

id	name	category	movie id
1	Half Off	Discount	1
2	Rewards	Cash Back	1
3	Matinee	Non-cash	1
4	Giveaways	Merchandise	3
5	Free Pass	Non-cash	4



*Let's use data from our **Promotions** table...*

*...and use it to complete a query on our **Movies** table.*

Creating Queries to Break Down Our Problem

We can create our inner query, which helps us find which **movie_id**s are Non-cash promotions.

Movies

id	title	sales	duration
1	Don Juan	45000	110
2	Peter Pan	30000	105
3	The Lost World	30000	106
4	Robin Hood	50000	143

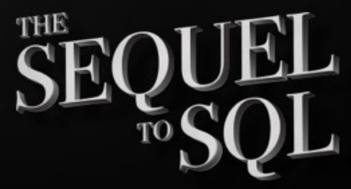
Promotions

id	name	category	movie_id
1	Half Off	Discount	1
2	Rewards	Cash Back	1
3	Matinee	Non-cash	1
4	Giveaways	Merchandise	3
5	Free Pass	Non-cash	4

We will call this our "inner query"

```
SELECT movie_id
FROM Promotions
WHERE category = 'Non-cash' ;
```

```
 movie_id
----------
        1
        4
(2 rows)
```



Using Our Inner Query on Our Outer Query

We can take the results from the inner query and add them to an outer query to find the SUM of sales.

Movies

id	title	sales	duration
1	Don Juan	45000	110
2	Peter Pan	30000	105
3	The Lost World	30000	106
4	Robin Hood	50000	143

Promotions

id	name	category	movie id
1	Half Off	Discount	1
2	Rewards	Cash Back	1
3	Matinee	Non-cash	1
4	Giveaways	Merchandise	3
5	Free Pass	Non-cash	4

```
SELECT SUM(sales)
FROM Movies
WHERE id IN
 (SELECT movie_id
 FROM Promotions
 WHERE category = 'Non-cash' );
```

This is a "query in a query."

*The inner query or **subquery** is surrounded by parentheses.*

Finding the Solution to Our Combined Queries

We can take the results from the inner query and add them to an outer query to find the SUM of sales.

Movies

id	title	sales	duration
1	Don Juan	45000	110
2	Peter Pan	30000	105
3	The Lost World	30000	106
4	Robin Hood	50000	143

Promotions

id	name	category	movie id
1	Half Off	Discount	1
2	Rewards	Cash Back	1
3	Matinee	Non-cash	1
4	Giveaways	Merchandise	3
5	Free Pass	Non-cash	4

```
SELECT SUM(sales)
FROM Movies
WHERE id IN
 (SELECT movie_id
 FROM Promotions
 WHERE category = 'Non-cash' );
```

```
SELECT SUM(sales)
FROM Movies
WHERE id IN
 (1,4);
```

This is the sum of 45000 and 50000.

```
  sum
-------
 95000
(1 row)
```

Comparing Subqueries vs. JOINs

We can also find the same results by using a JOIN.

Subquery — easier to read

```
SELECT SUM(sales)
FROM Movies
WHERE id IN
 (SELECT movie_id
 FROM Promotions
 WHERE category = 'Non-cash' );
```

JOIN query — better for performance

```
SELECT SUM(m.sales)
FROM Movies m
INNER JOIN Promotions p
ON m.id = p.movie_id
WHERE p.category = 'Non-cash';
```

Same result

```
  sum
-------
 95000
(1 row)
```

Subquery Syntax

```
WHERE <field> IN(<subquery>)
```

Filters rows that have a matching id

```
WHERE <field> NOT IN(<subquery>)
```

Filters rows that don't have a matching id

Using Correlated Subqueries

We want to find only films that have a duration greater than the average duration of all films.

Movies

id	title	genre	duration
1	Don Juan	Romance	110
2	Peter Pan	Adventure	105
3	The Lost World	Fantasy	106
4	Robin Hood	Adventure	143

```
SELECT * FROM Movies WHERE duration > AVG(duration);
```

```
ERROR:  aggregate functions are not allowed in WHERE
```

We need a correlated subquery!

Using Correlated Subqueries

Supplying data from the main query back into the subquery is called a correlated subquery.

Movies

id	title	genre	duration
1	Don Juan	Romance	110
2	Peter Pan	Adventure	105
3	The Lost World	Fantasy	106
4	Robin Hood	Adventure	143

```
SELECT * FROM Movies WHERE duration >
(SELECT AVG(duration) FROM Movies);
```

Both queries depend on each other's values to return a correct result.

```
 id |   title     |   genre   | duration
----+-------------+-----------+----------
  4 | Robin Hood | Adventure |      143
(1 row)
```