**SQL Lesson 1: SELECT queries 101**

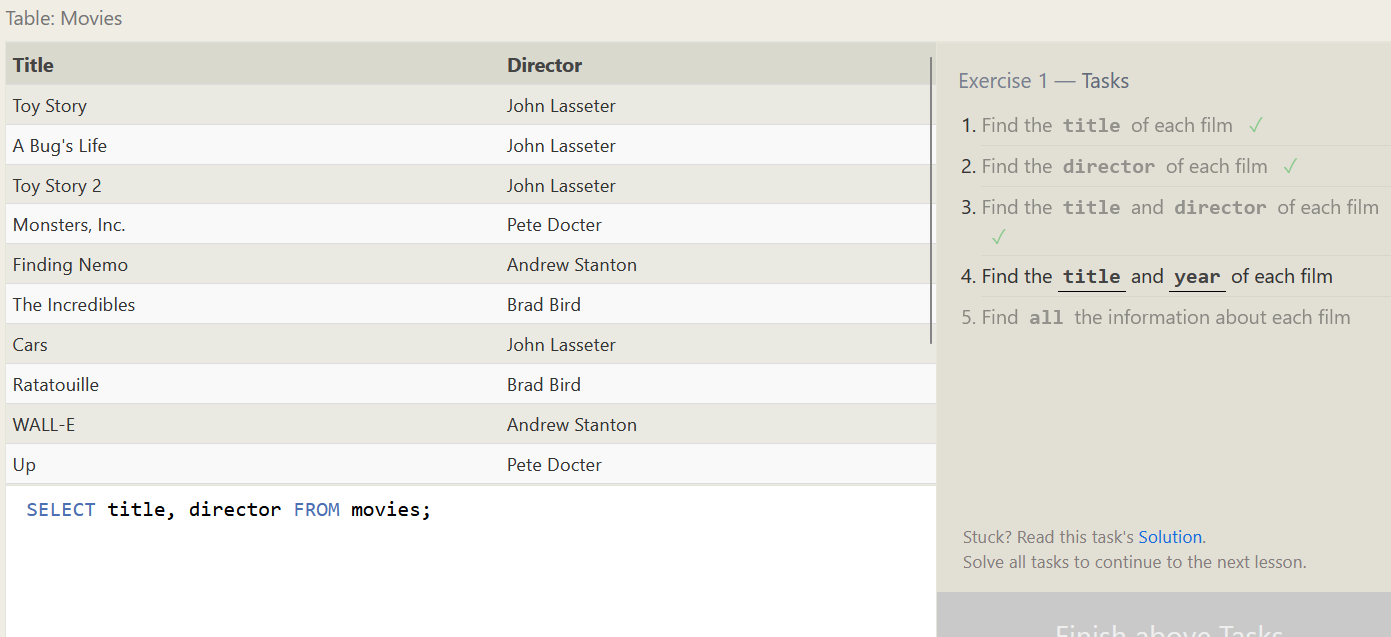
1. Find the title of each film.

>>> SELECT title FROM movies; 

2. Find the director of each film

>>> SELECT director FROM movies; 

3. Find the title and director of each film

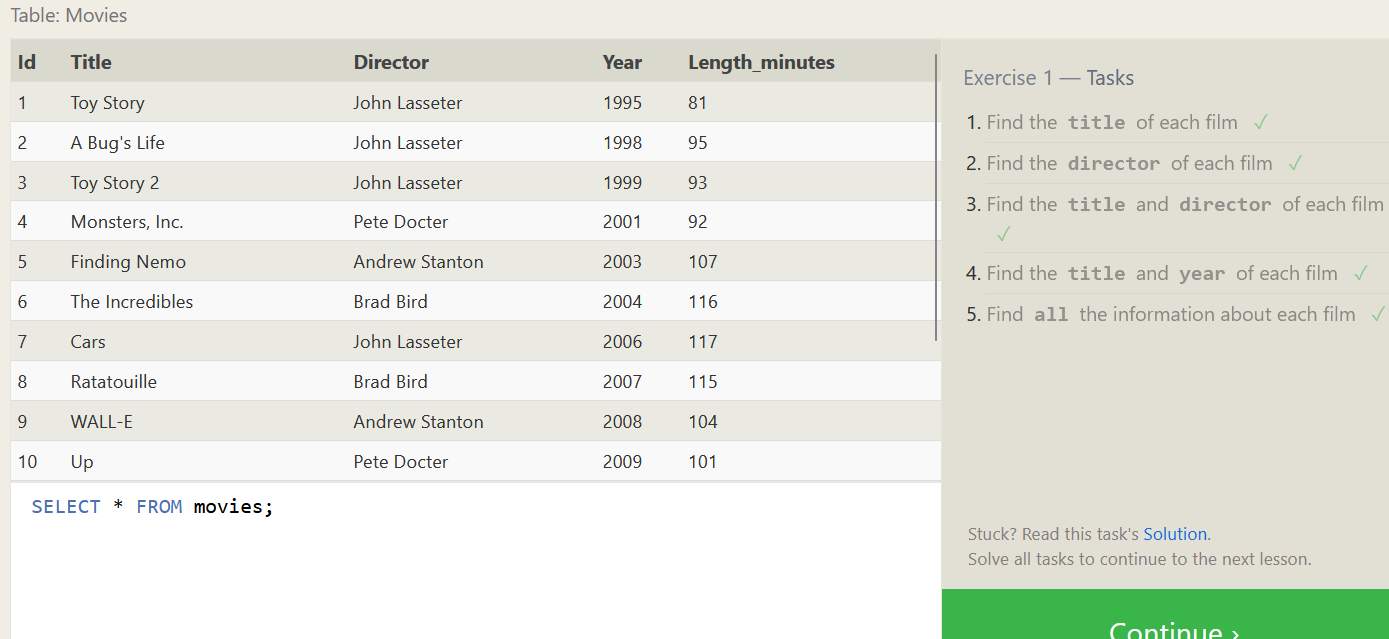
>>> SELECT title, director FROM movies; 

4. Find the title and year of each film

>>> SELECT title, year FROM movies; 

5. Find all the information about each film

>>> SELECT \* FROM movies;



**SQL Lesson 2: Queries with constraints (Pt. 1)**

1. Find the movie with a row id of 6

>>> SELECT \* FROM movies where id = 6; 

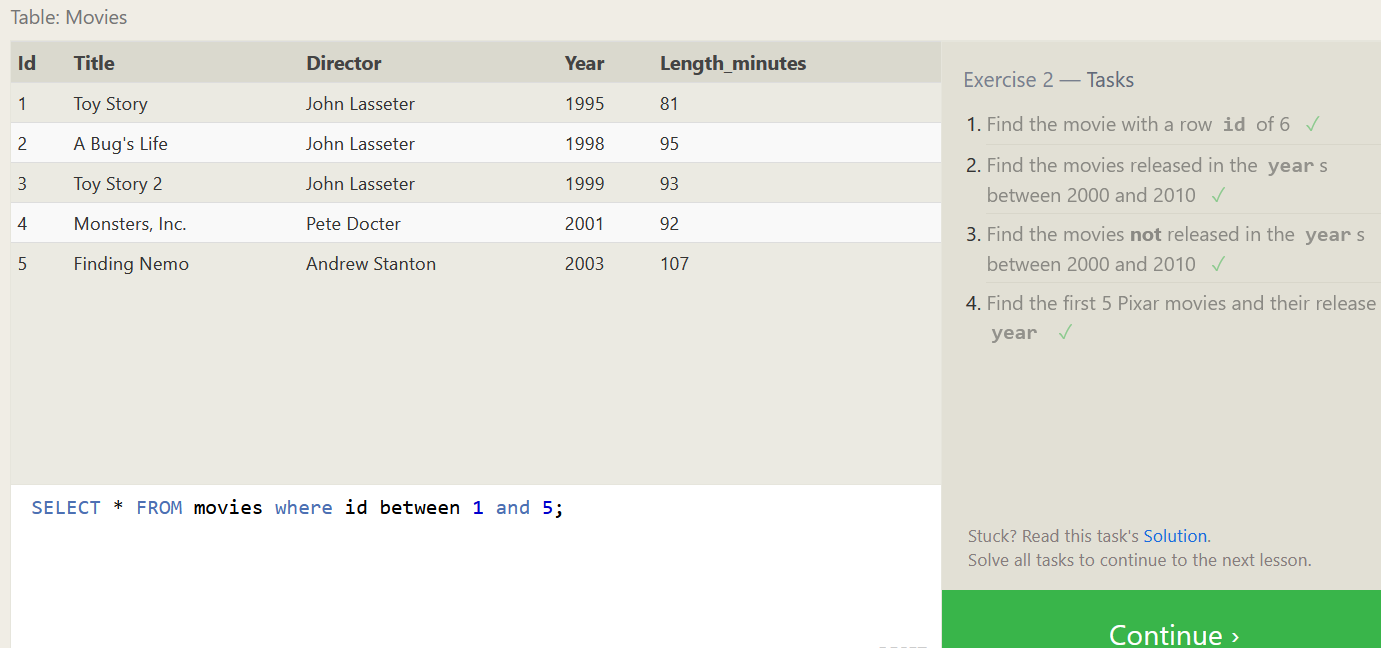
2. Find the movies released in the years between 2000 and 2010.

>>> SELECT \* FROM movies where year between 2000 and 2010; 

3. Find the movies not released in the years between 2000 and 2010.

>>> SELECT \* FROM movies where year not between 2000 and 2010;

4. Find the first 5 Pixar movies and their release year.

>>> SELECT \* FROM movies where id between 1 and 5; 

**SQL Lesson 3: Queries with constraints (Pt. 2)**

1. Find all the Toy Story movies.

>>> SELECT \* FROM movies where Title like "%Toy Story%"; 

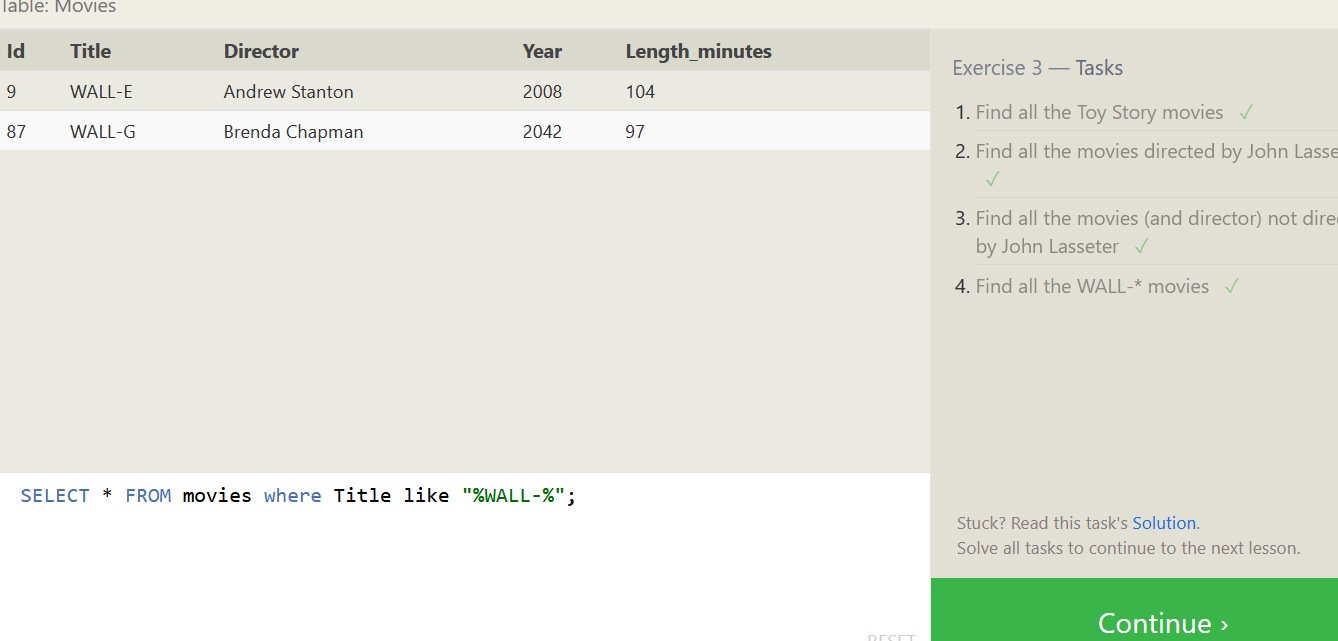
2. Find all the movies directed by John Lasseter.

>>> SELECT \* FROM movies where director ="John Lasseter"; 

3. Find all the movies (and director) not directed by John Lasseter.

>>> SELECT \* FROM movies where director !="John Lasseter"; 

4. Find all the WALL-\* movies.

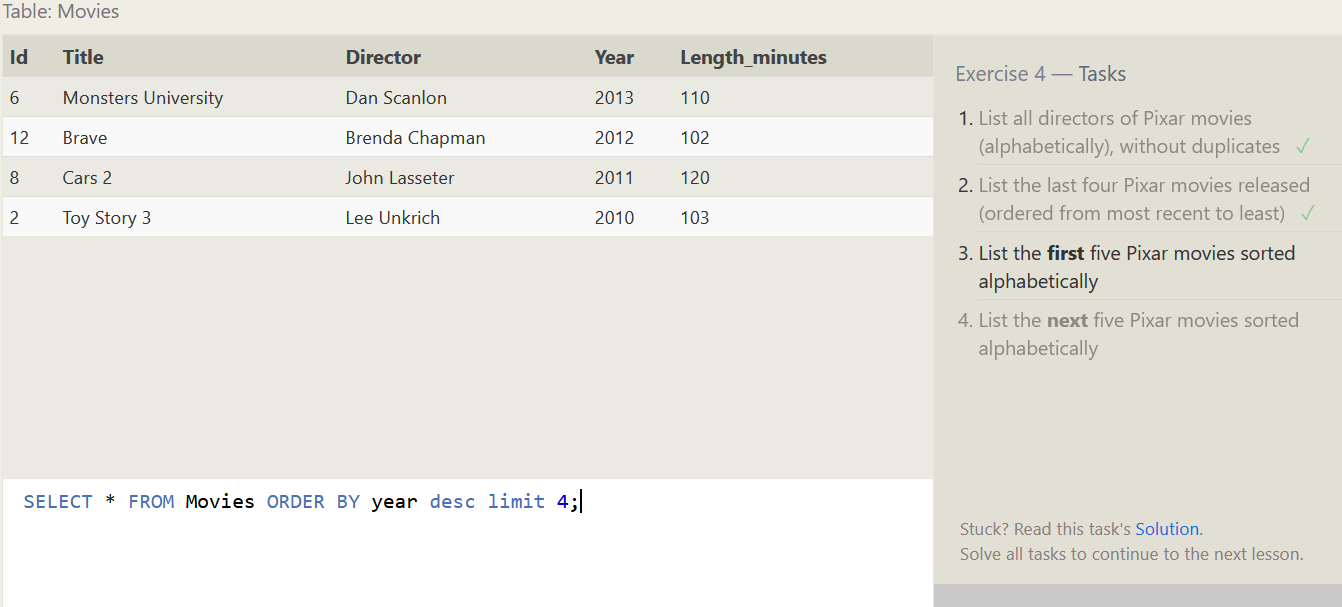
>>> SELECT \* FROM movies where Title like "%WALL-%"; 

**SQL Lesson 4: Filtering and sorting Query results**

1. List all directors of Pixar movies (alphabetically), without duplicates.

>>> SELECT DISTINCT Director FROM Movies ORDER BY Director; 

2. List the last four Pixar movies released (ordered from most recent to least).

>>> SELECT \* FROM Movies ORDER BY year desc limit 4; 

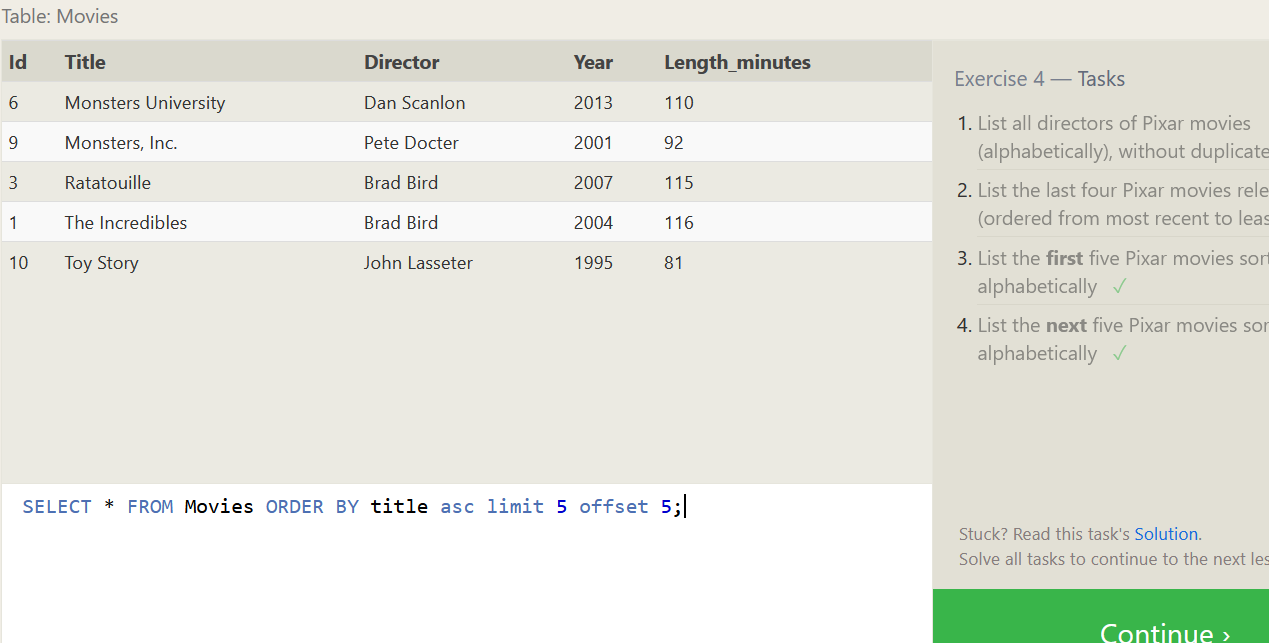
3. List the first five Pixar movies sorted alphabetically.

>>> SELECT \* FROM Movies ORDER BY title asc limit 5;



4. List the next five Pixar movies sorted alphabetically.

>>> SELECT \* FROM Movies ORDER BY title asc limit 5 offset 5;

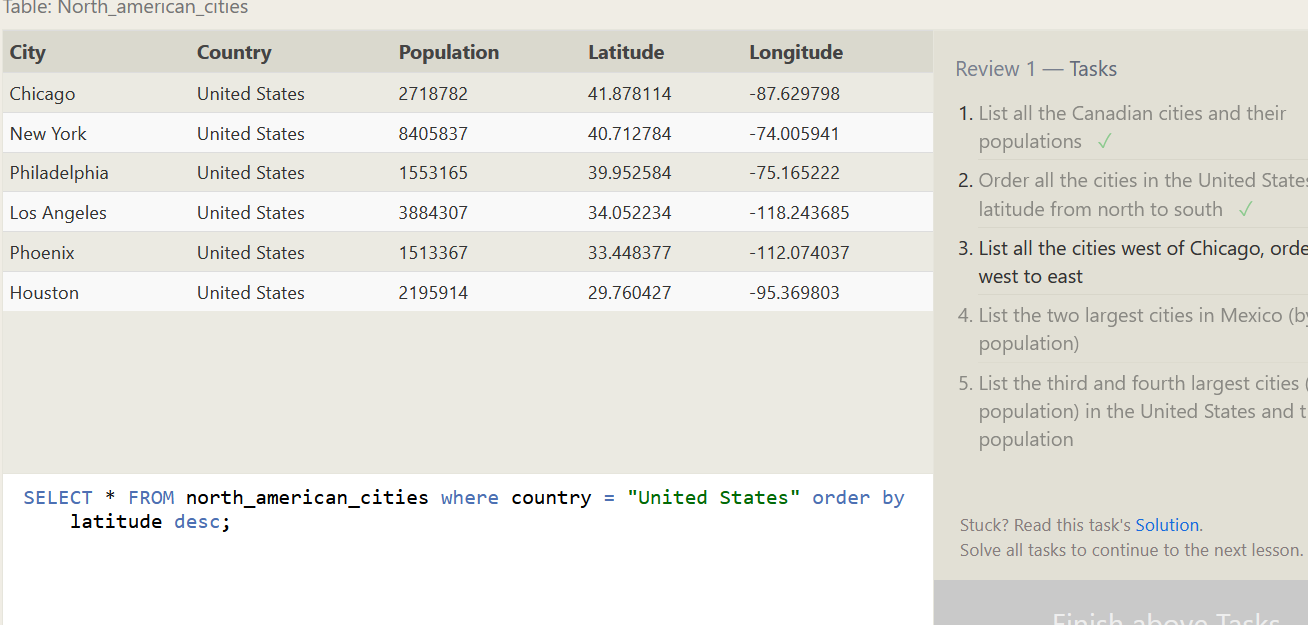


**SQL Review: Simple SELECT Queries**

1. List all the Canadian cities and their populations

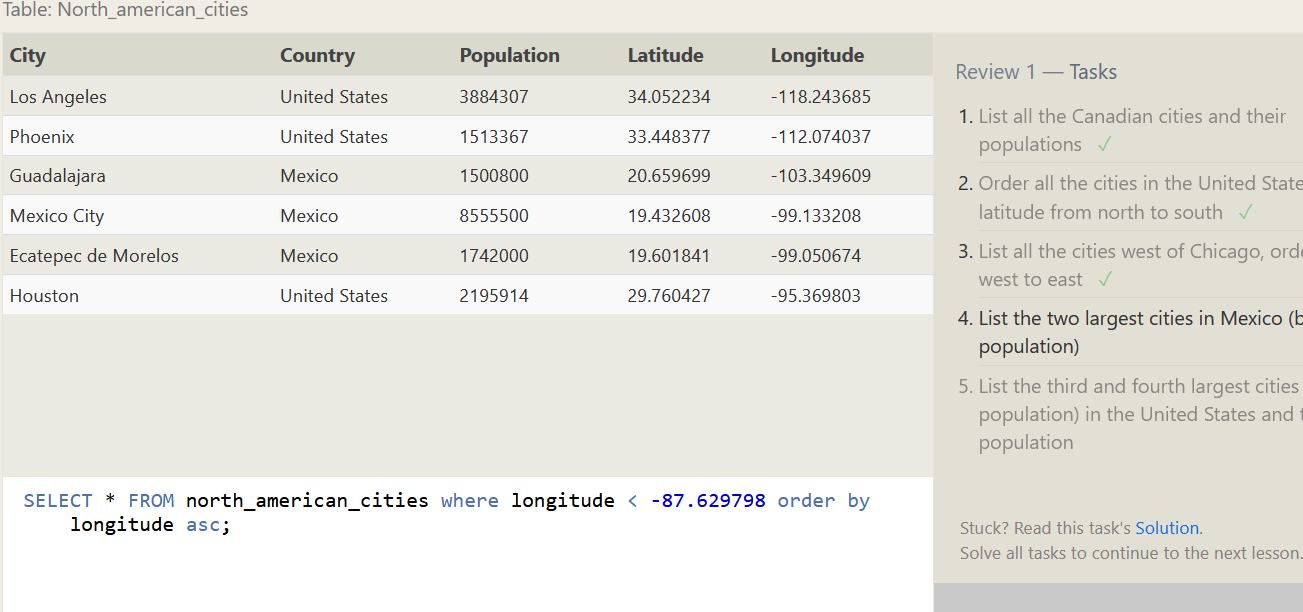
>>> SELECT \* FROM north\_american\_cities where country = "Canada"; 

2. Order all the cities in the United States by their latitude from north to south

>>> SELECT \* FROM north\_american\_cities where country = "United States" order by latitude desc; 

3. List all the cities west of Chicago, ordered from west to east.

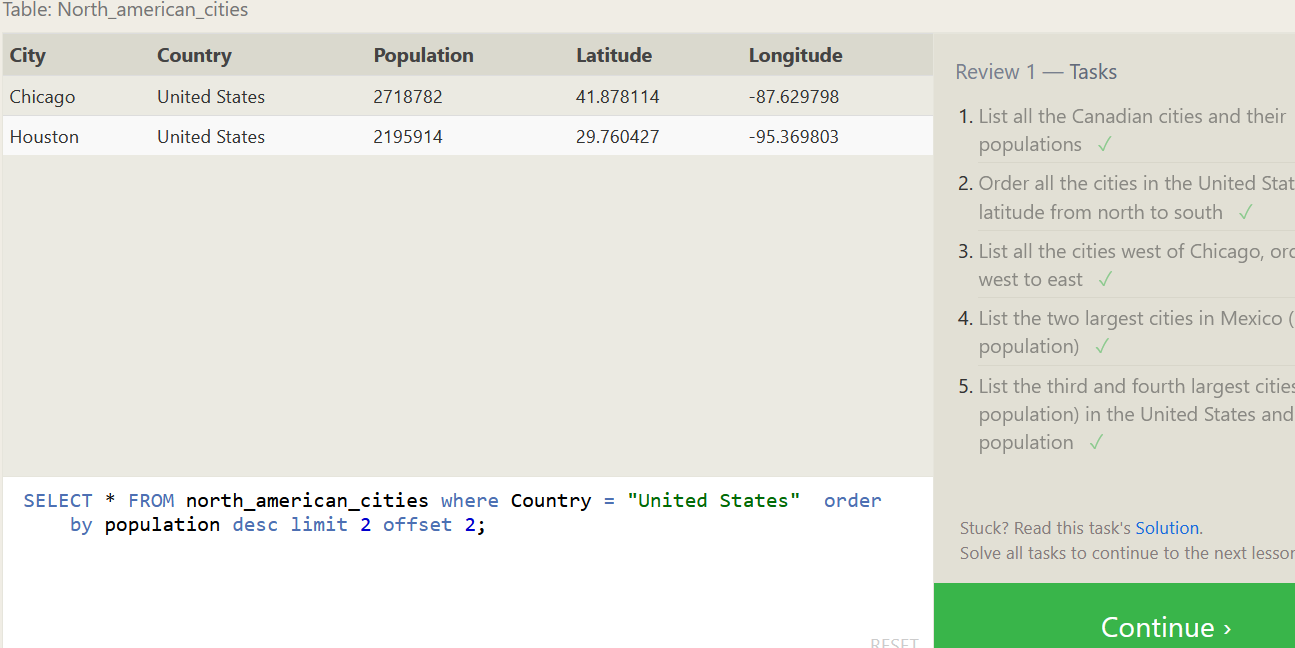
>>> SELECT \* FROM north\_american\_cities where longitude < -87.629798 order by longitude asc;



4. List the two largest cities in Mexico (by population)

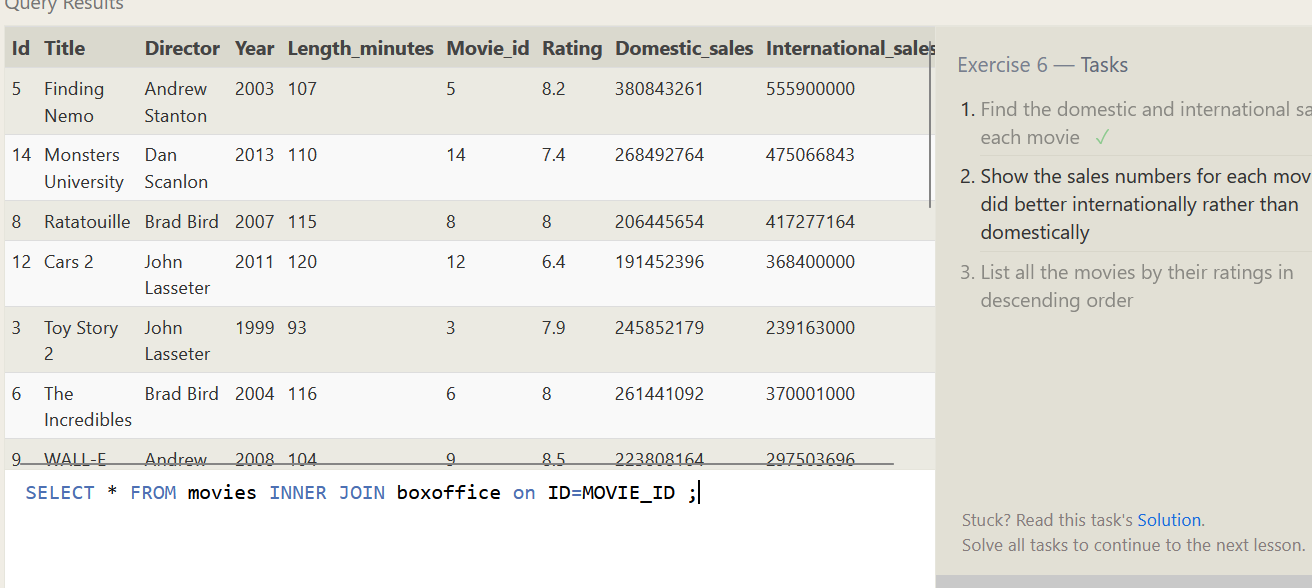
>>> SELECT \* FROM north\_american\_cities where Country = "Mexico" order by population desc limit 2; 

5. List the third and fourth largest cities (by population) in the United States and their population

>>> SELECT \* FROM north\_american\_cities where Country = "United States" order by population desc limit 2 offset 2; 

**SQL Lesson 6: Multi-table queries with JOINs**

1. Find the domestic and international sales for each movie

>>> SELECT \* FROM movies INNER JOIN boxoffice on ID=MOVIE\_ID ; 

2. Show the sales numbers for each movie that did better internationally rather than domestically

>>> SELECT \* FROM movies INNER JOIN boxoffice on ID=MOVIE\_ID WHERE international\_sales > domestic\_sales; 

3. List all the movies by their ratings in descending order

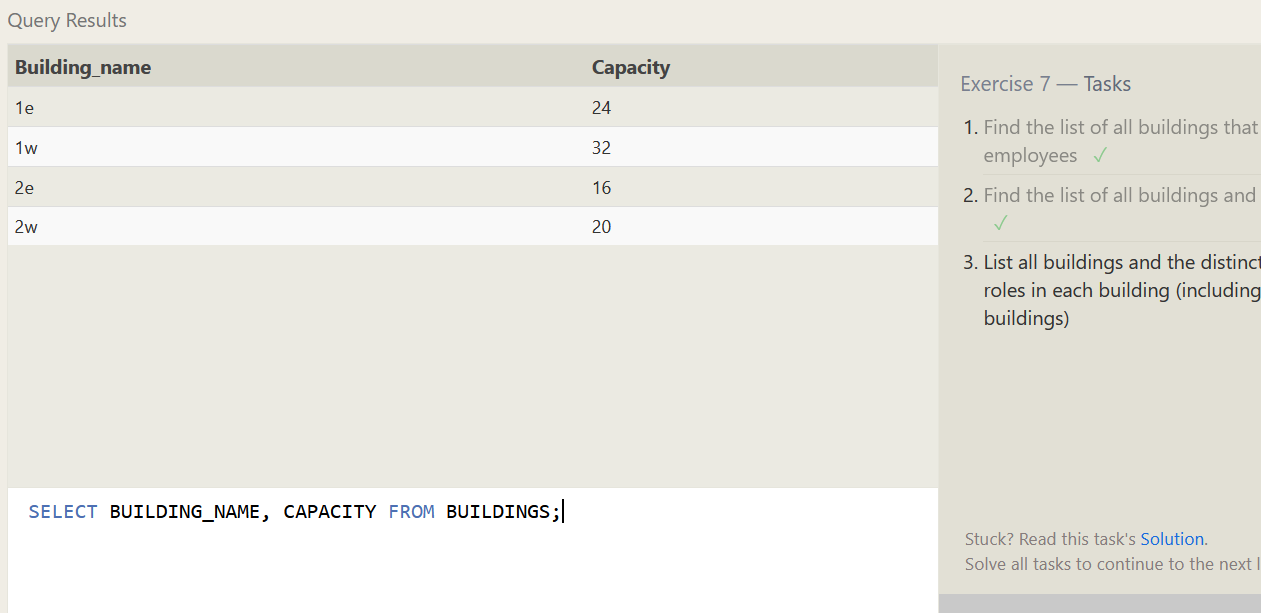
>>> SELECT \* FROM movies INNER JOIN boxoffice on ID=MOVIE\_ID ORDER BY RATING DESC; 

**SQL Lesson 7: OUTER JOINs**

1. Find the list of all buildings that have employees

>>> SELECT DISTINCT BUILDING FROM employees LEFT JOIN buildings on building=building\_name; 

2. Find the list of all buildings and their capacity

>>> SELECT BUILDING\_NAME, CAPACITY FROM BUILDINGS; 

3. List all buildings and the distinct employee roles in each building (including empty buildings)

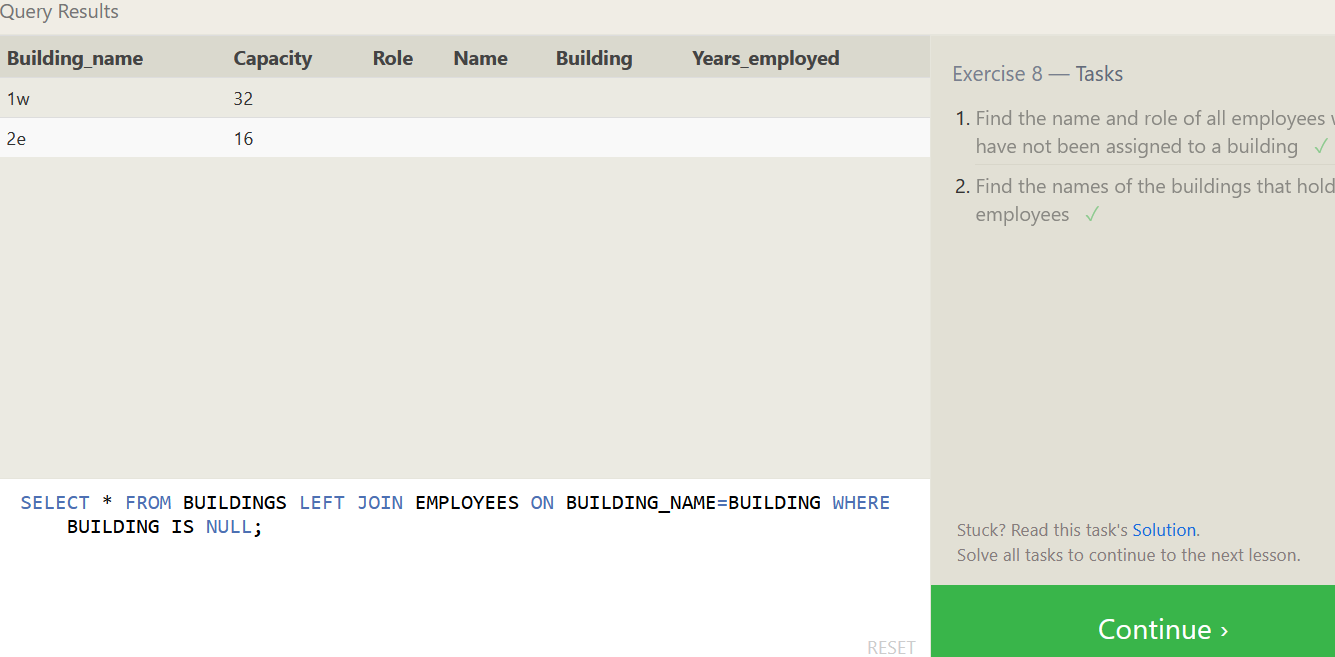
>>> SELECT DISTINCT Building\_name, Role FROM Buildings LEFT JOIN employees ON building\_name = building; 

**SQL Lesson 8: A short note on NULLs**

1. Find the name and role of all employees who have not been assigned to a building

>>> SELECT NAME, ROLE FROM employees WHERE BUILDING IS NULL; 

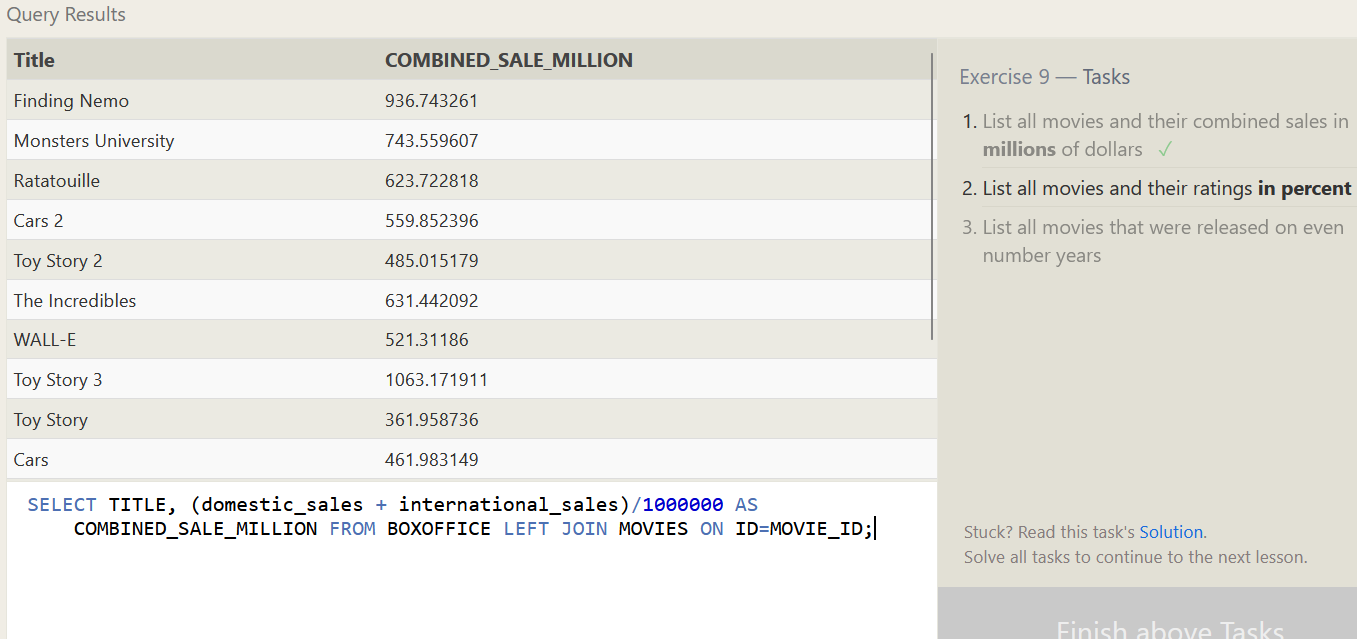
2. Find the names of the buildings that hold no employees

>>> SELECT \* FROM BUILDINGS LEFT JOIN EMPLOYEES ON BUILDING\_NAME=BUILDING WHERE BUILDING IS NULL; 

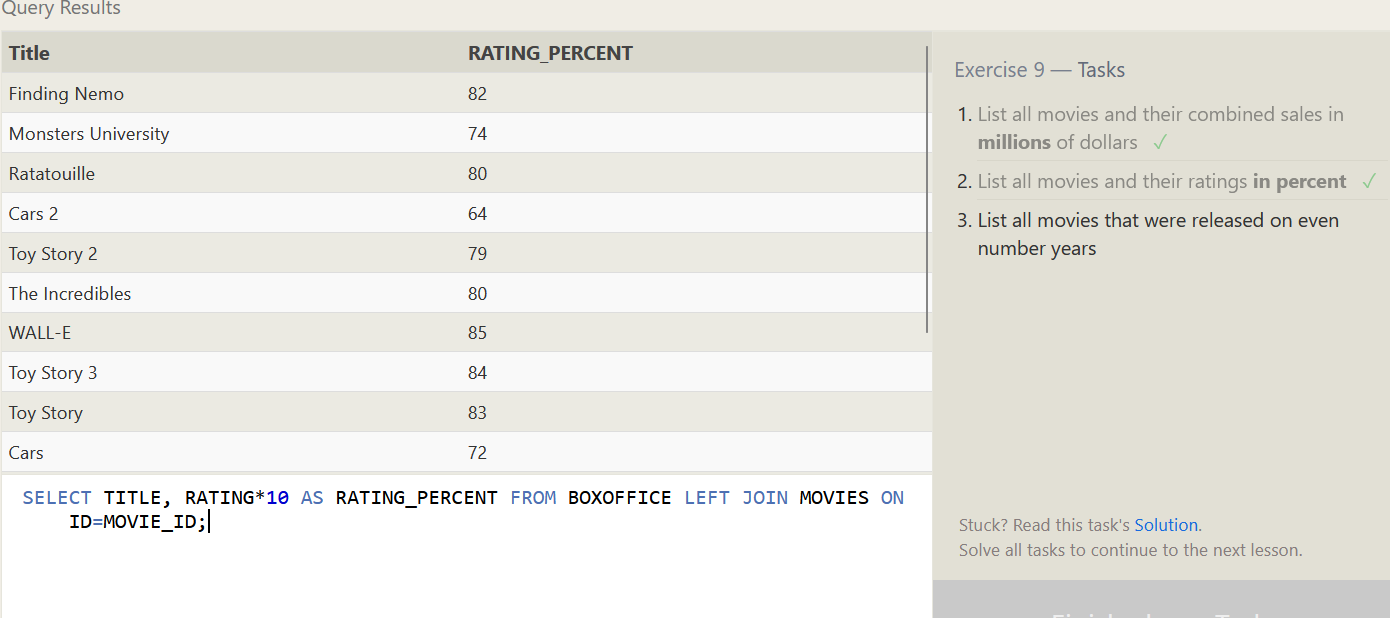
**SQL Lesson 9: Queries with expressions**

1. List all movies and their combined sales in millions of dollars

>>> SELECT TITLE, (domestic\_sales + international\_sales)/1000000 AS COMBINED\_SALE\_MILLION FROM BOXOFFICE LEFT JOIN MOVIES ON ID=MOVIE\_ID;



2. List all movies and their ratings in percent

>>> SELECT TITLE, RATING\*10 AS RATING\_PERCENT FROM BOXOFFICE LEFT JOIN MOVIES ON ID=MOVIE\_ID; 

3. List all movies that were released on even number years

>>> SELECT TITLE,YEAR FROM MOVIES WHERE YEAR%2 != 1;



**SQL Lesson 10: Queries with aggregates (Pt. 1)**

1. Find the longest time that an employee has been at the studio

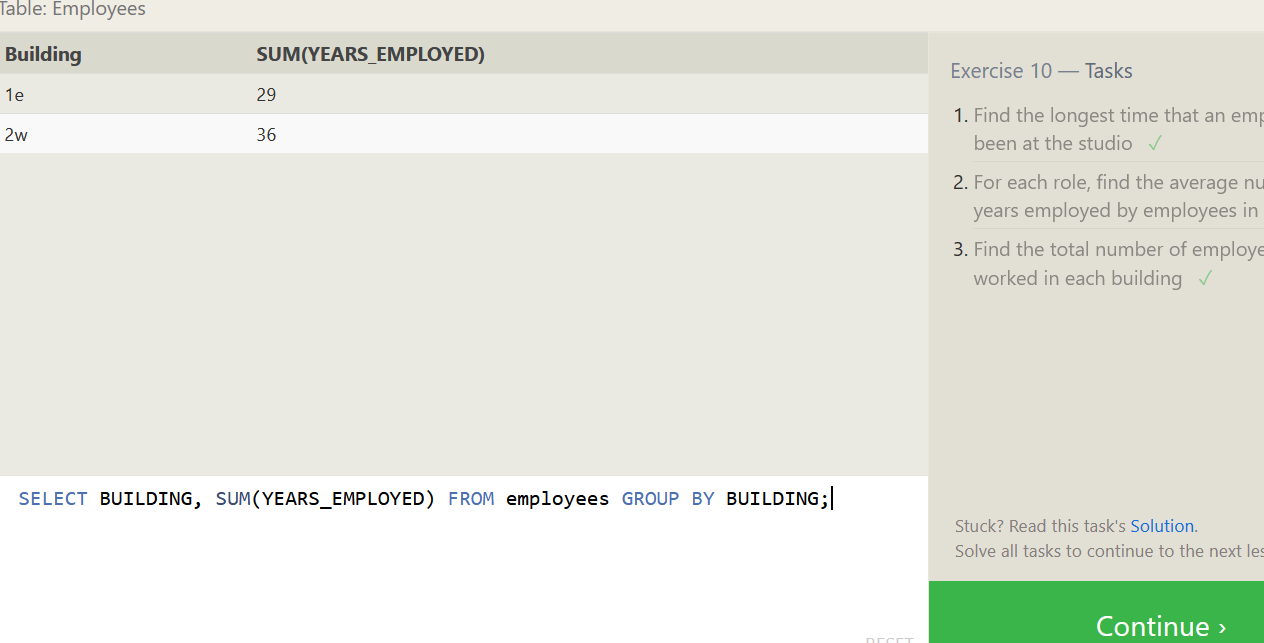
>>> SELECT \*, MAX(YEARS\_EMPLOYED) FROM employees; 

2. For each role, find the average number of years employed by employees in that role

>>> SELECT ROLE, AVG(YEARS\_EMPLOYED) FROM employees GROUP BY ROLE;



3. Find the total number of employee years worked in each building

>>> SELECT BUILDING, SUM(YEARS\_EMPLOYED) FROM employees GROUP BY BUILDING; 

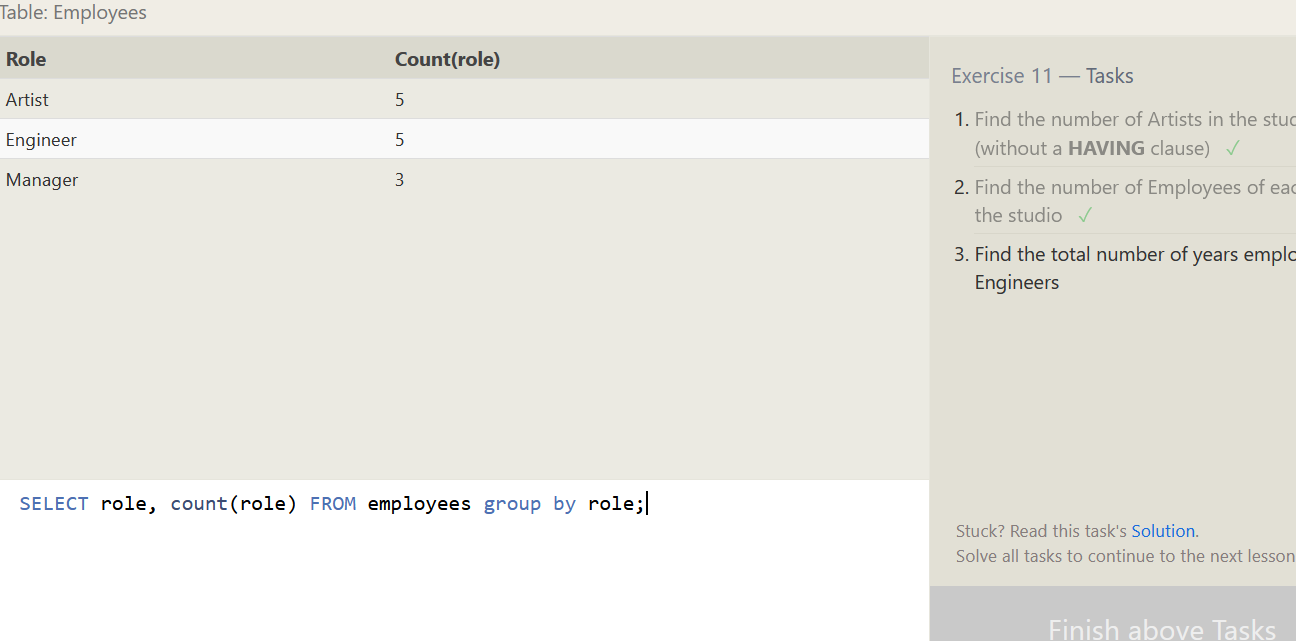
**SQL Lesson 11: Queries with aggregates (Pt. 2)**

1. Find the number of Artists in the studio (without a HAVING clause)

>>> SELECT count(\*) FROM employees WHERE ROLE = "Artist"; 

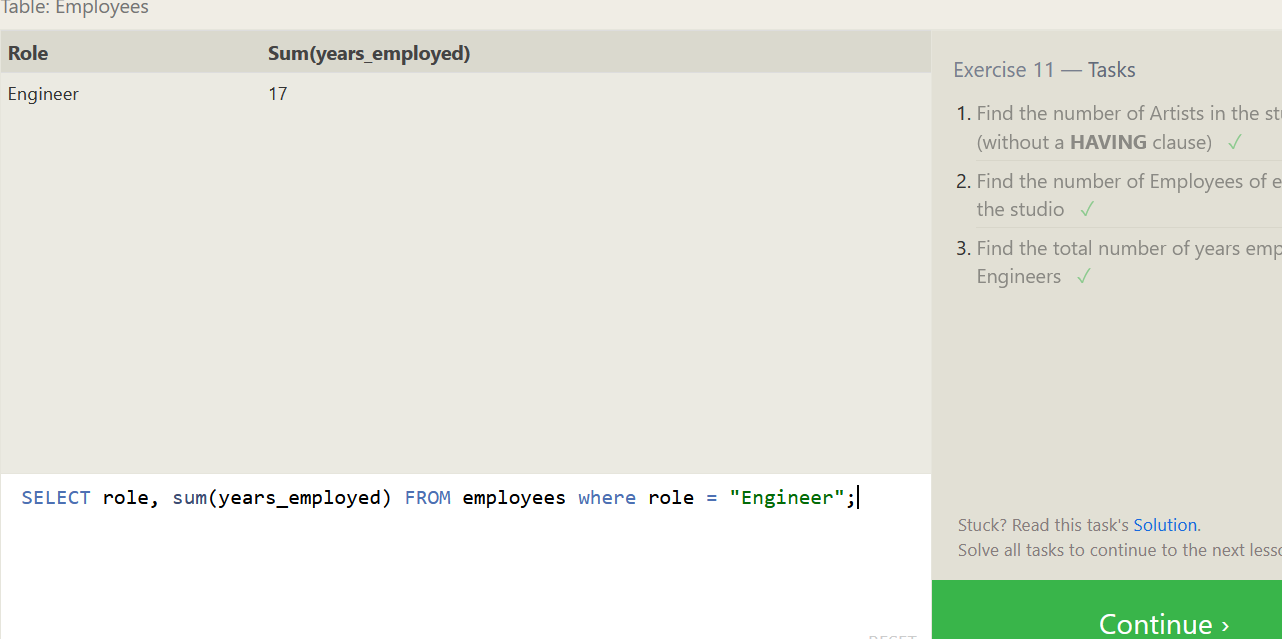
2. Find the number of Employees of each role in the studio

>>> SELECT role, count(role) FROM employees group by role;



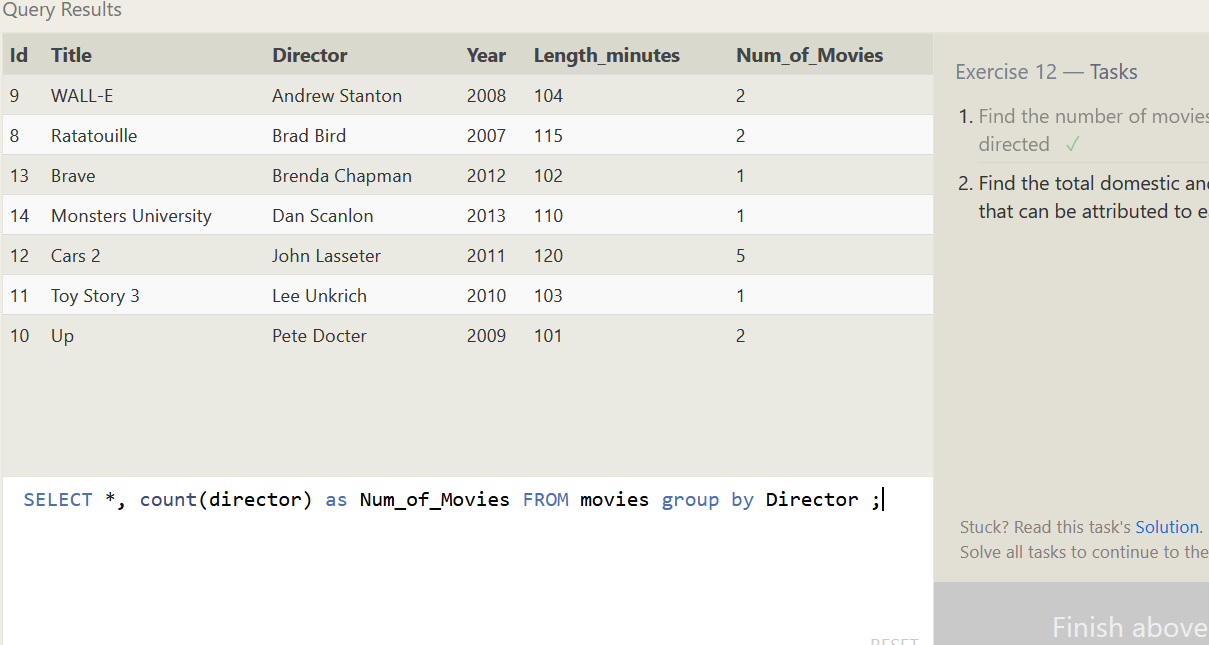
3. Find the total number of years employed by all Engineers

>>> SELECT role, sum(years\_employed) FROM employees where role = "Engineer";

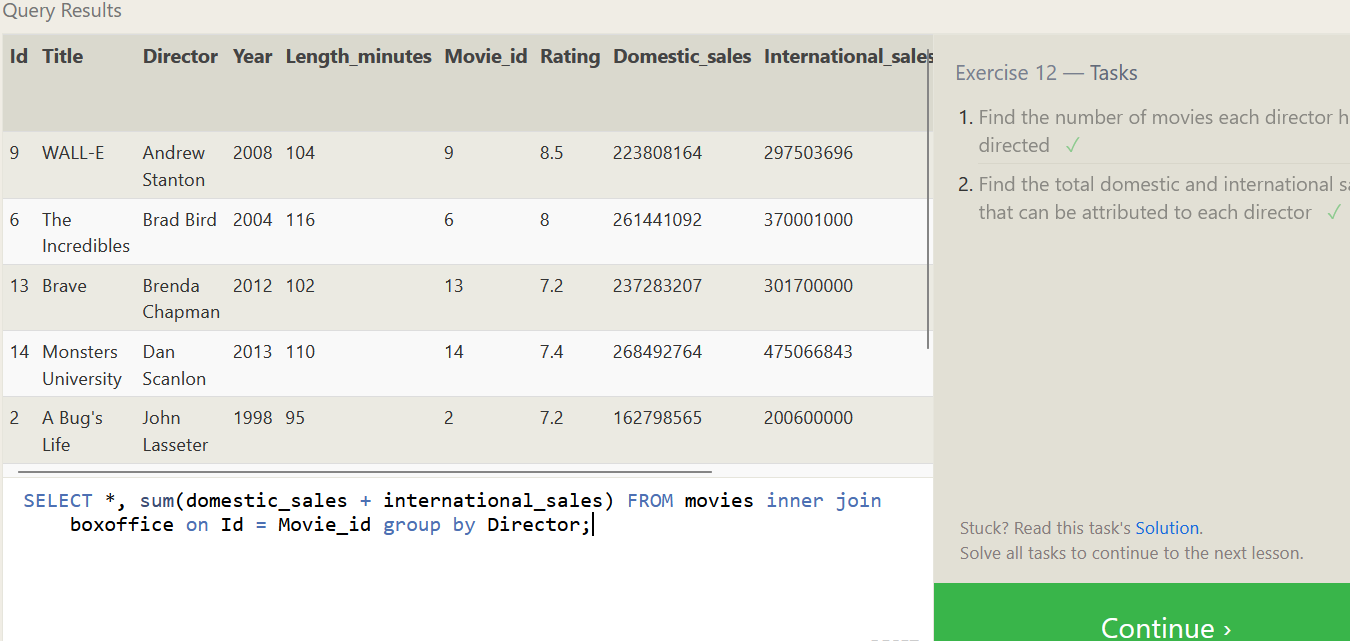


**SQL Lesson 12: Order of execution of a Query**

1. Find the number of movies each director has directed

>>> SELECT \*, count(director) as Num\_of\_Movies FROM movies group by Director ; 

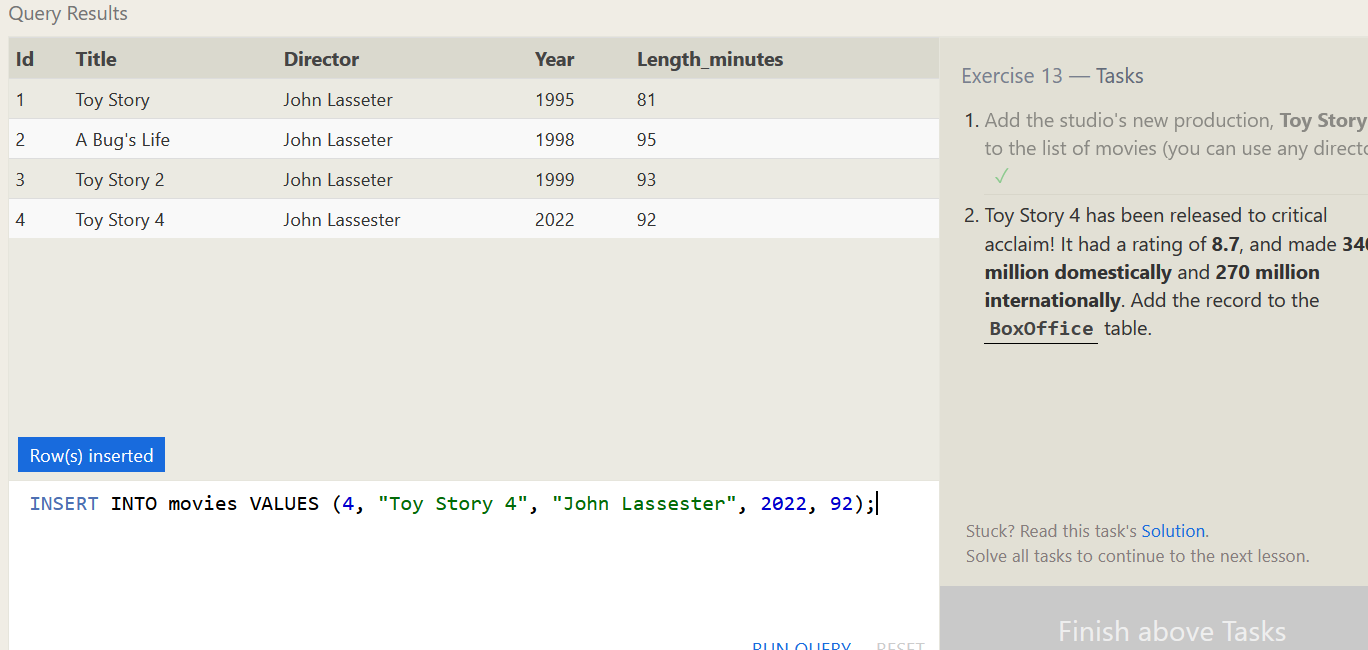
2. Find the total domestic and international sales that can be attributed to each director

>>> SELECT \*, sum(domestic\_sales + international\_sales) FROM movies inner join boxoffice on Id = Movie\_id group by Director; 

**SQL Lesson 13: Inserting rows**

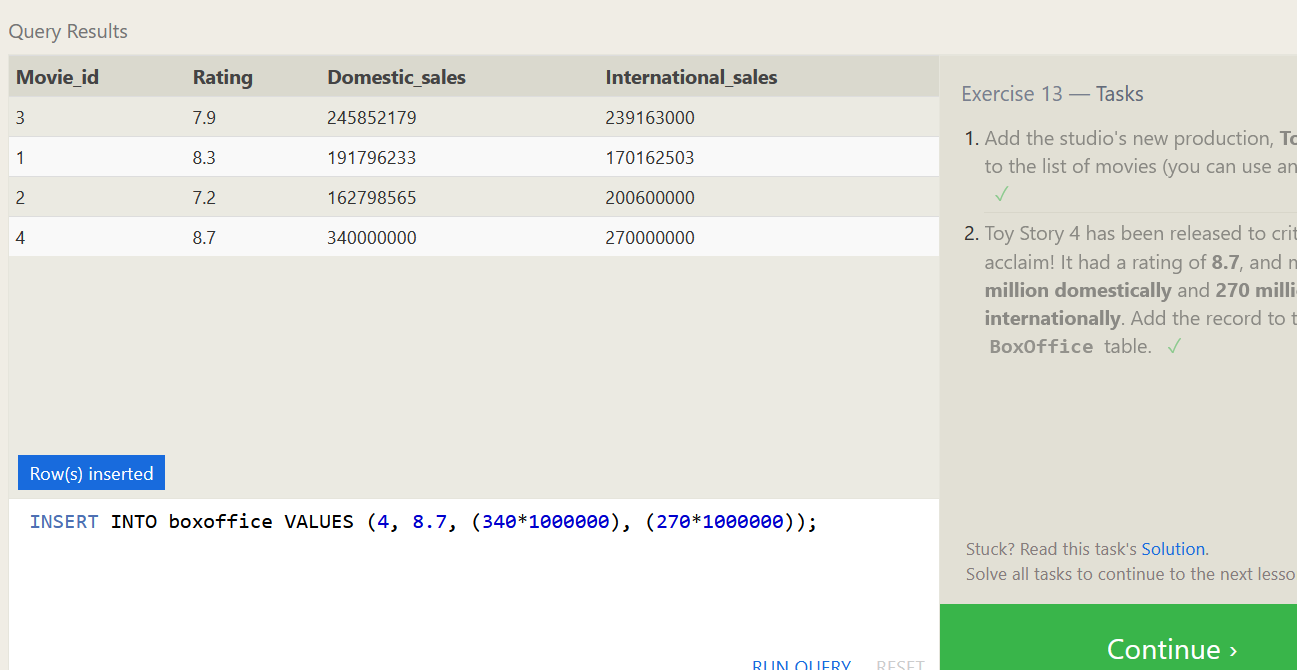
1. Add the studio's new production, Toy Story 4 to the list of movies (you can use any director)

>>> INSERT INTO movies VALUES (4, "Toy Story 4", "John Lassester", 2022, 92);



2. oy Story 4 has been released to critical acclaim! It had a rating of 8.7, and made 340 million domestically and 270 million internationally. Add the record to the BoxOffice table.

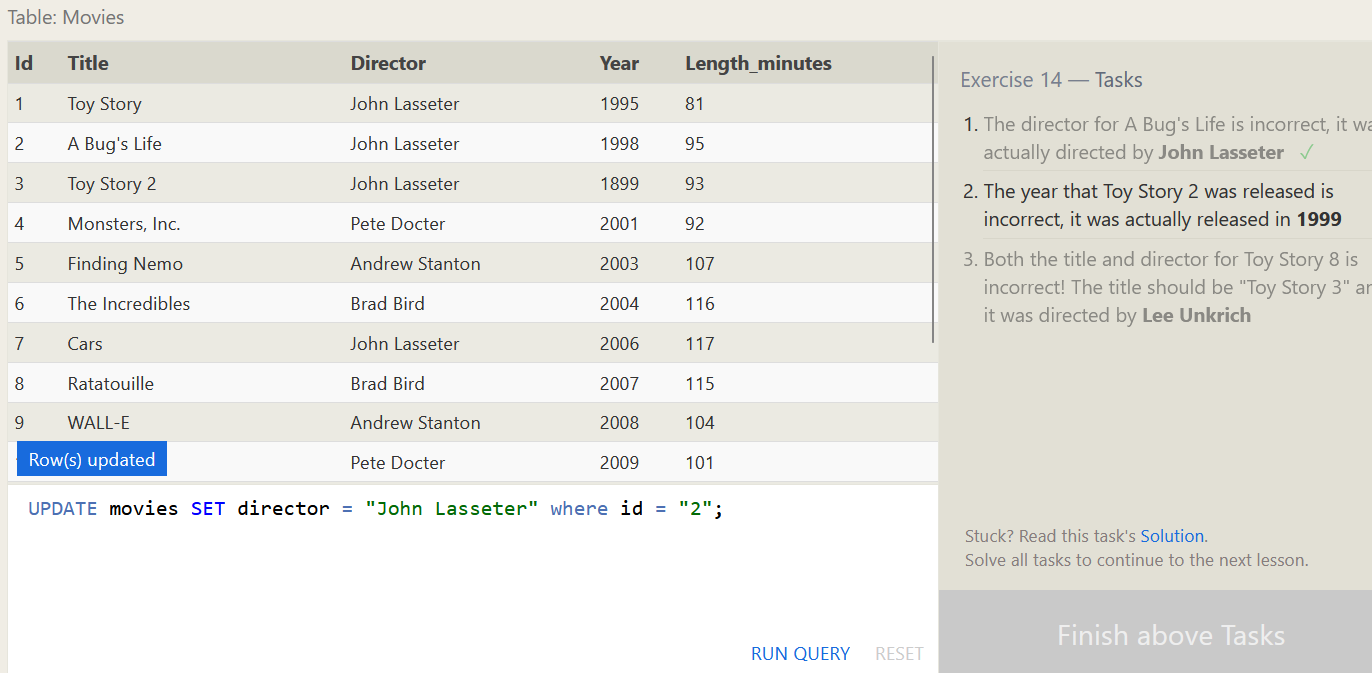
>>> INSERT INTO boxoffice VALUES (4, 8.7, (340\*1000000), (270\*1000000));



**SQL Lesson 14: Updating rows**

1. The director for A Bug's Life is incorrect, it was actually directed by John Lasseter.

>>> UPDATE movies SET director = "John Lasseter" where id = "2";



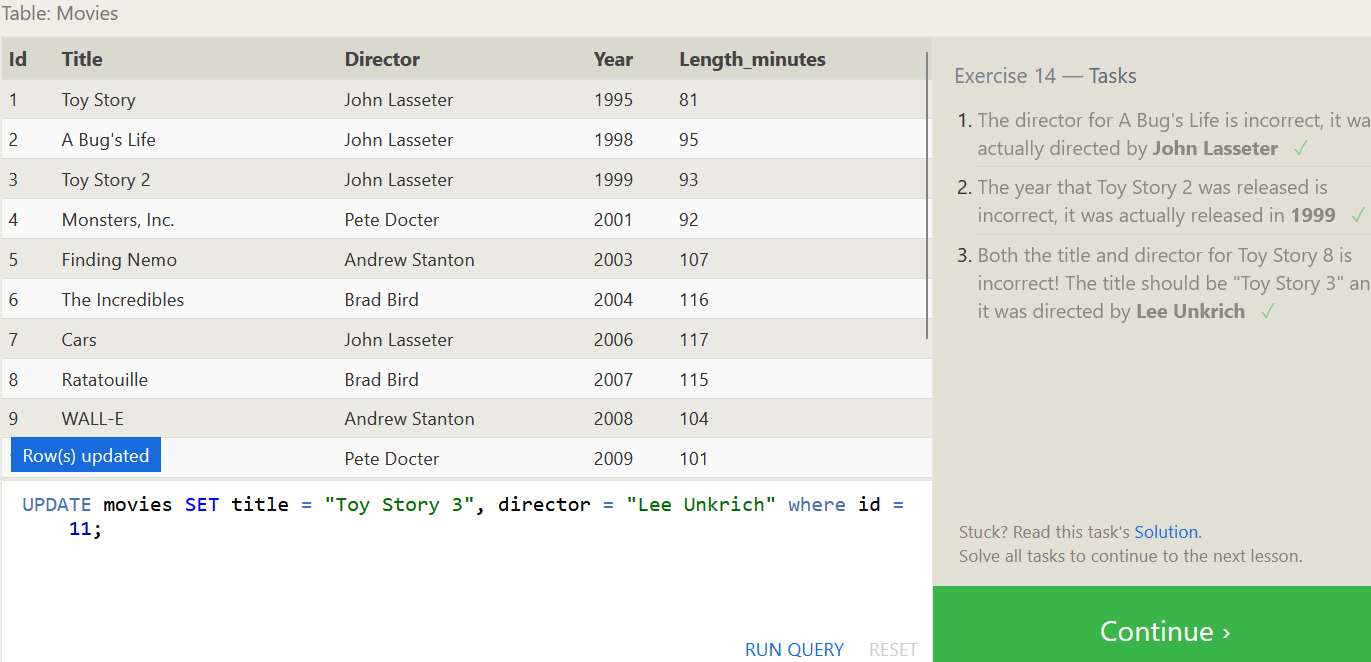
2. The year that Toy Story 2 was released is incorrect, it was actually released in 1999.

>>> UPDATE movies SET year = 1999 where title = "Toy Story 2";



3. Both the title and director for Toy Story 8 is incorrect! The title should be "Toy Story 3" and it was directed by Lee Unkrich.

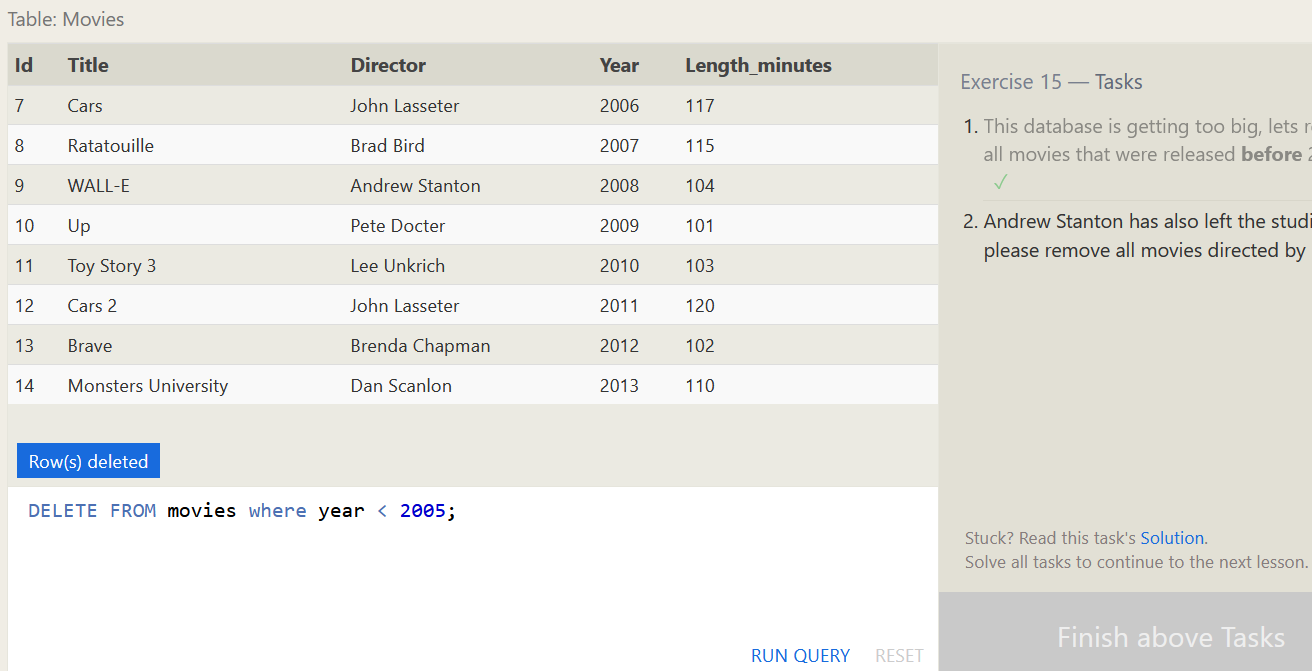
>>> UPDATE movies SET title = "Toy Story 3", director = "Lee Unkrich" where id = 11;



**SQL Lesson 15: Deleting rows**

1. This database is getting too big, lets remove all movies that were released before 2005.

>>> DELETE FROM movies where year < 2005;



2. Andrew Stanton has also left the studio, so please remove all movies directed by him.

>>> DELETE FROM movies where director = "Andrew Stanton";



**SQL Lesson 16: Creating tables**

1. Create a new table named Database with the following columns:

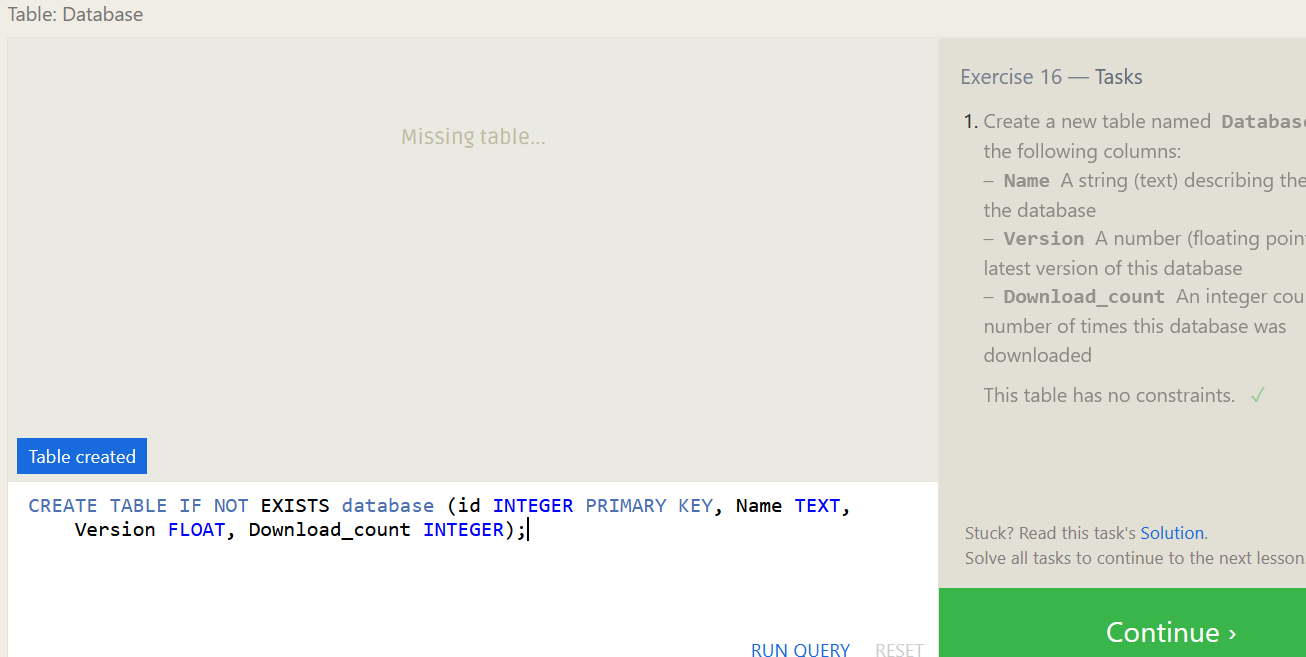
-- Name A string (text) describing the name of the database

-- Version A number (floating point) of the latest version of this database

-- Download\_count An integer count of the number of times this database was downloaded

This table has no constraints.

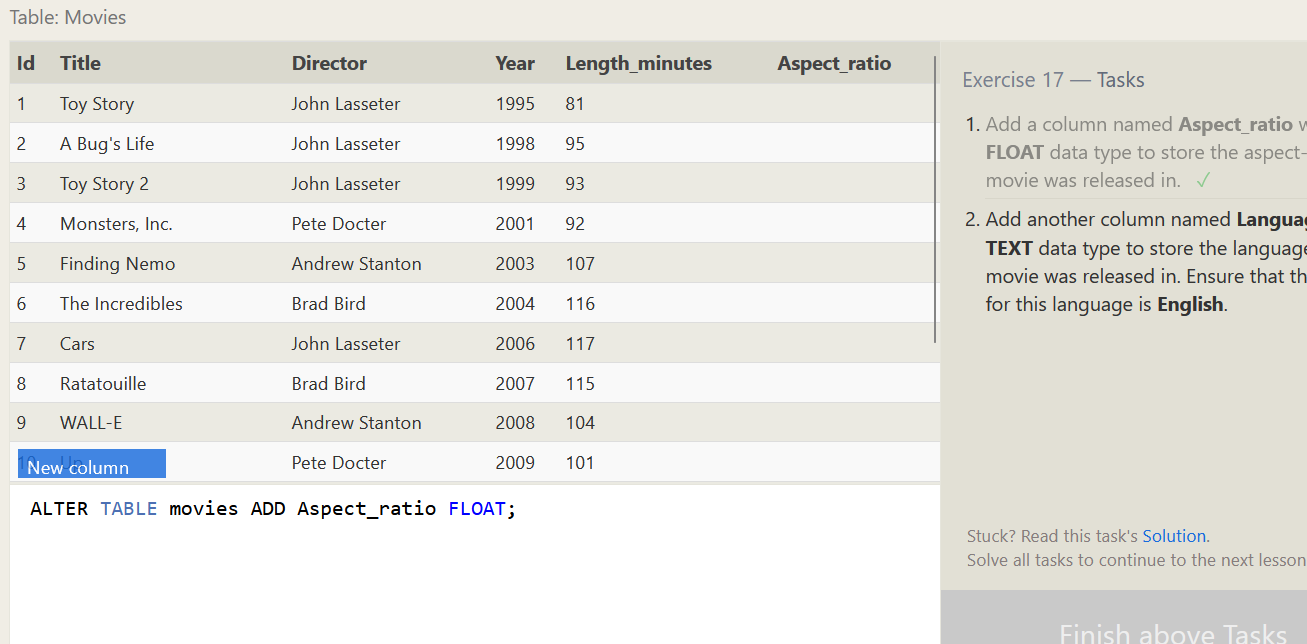
>>> CREATE TABLE IF NOT EXISTS database (id INTEGER PRIMARY KEY, Name TEXT, Version FLOAT, Download\_count INTEGER);



**SQL Lesson 17: Altering tables**

1. Add a column named Aspect\_ratio with a FLOAT data type to store the aspect-ratio each movie was released in.

>>> ALTER TABLE movies ADD Aspect\_ratio FLOAT;



2. Add another column named Language with a TEXT data type to store the language that the movie was released in. Ensure that the default for this language is English.

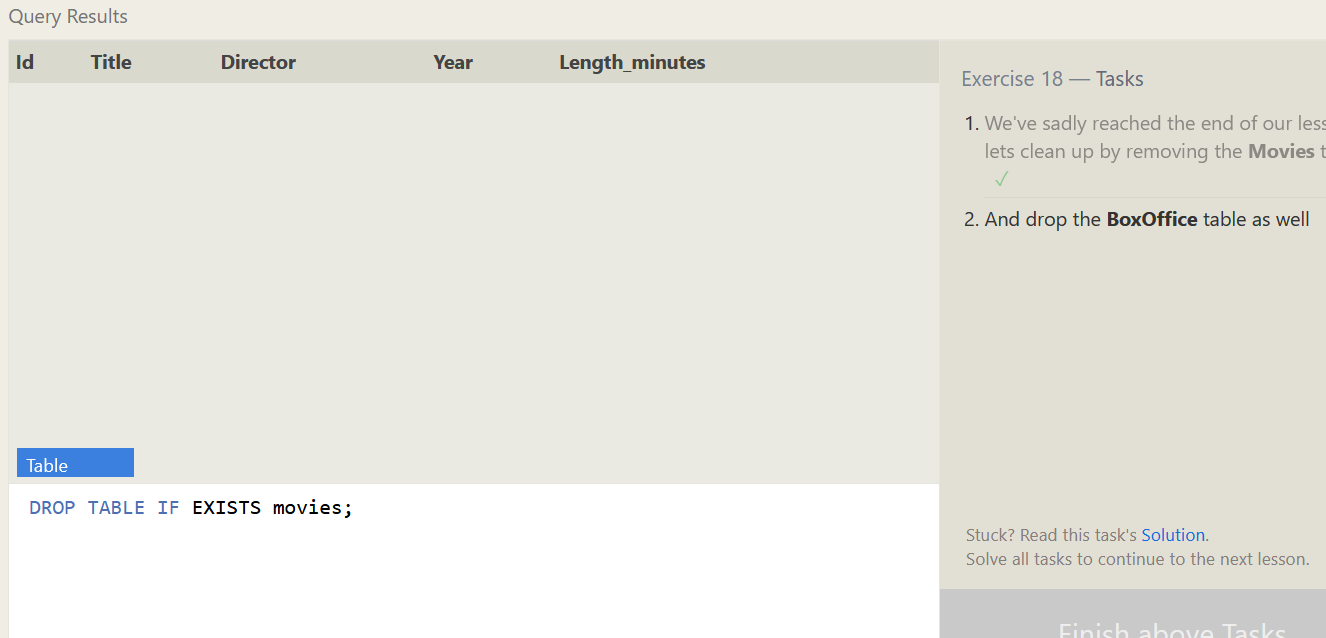
>>> ALTER TABLE movies ADD Language TEXT DEFAULT English;



**SQL Lesson 18: Dropping tables**

1. We've sadly reached the end of our lessons, lets clean up by removing the Movies table.

>>> DROP TABLE IF EXISTS movies;



2. And drop the BoxOffice table as well

>>> DROP TABLE IF EXISTS boxoffice;

