SQL

1. **SELECT Queries:** To retrieve data from database.

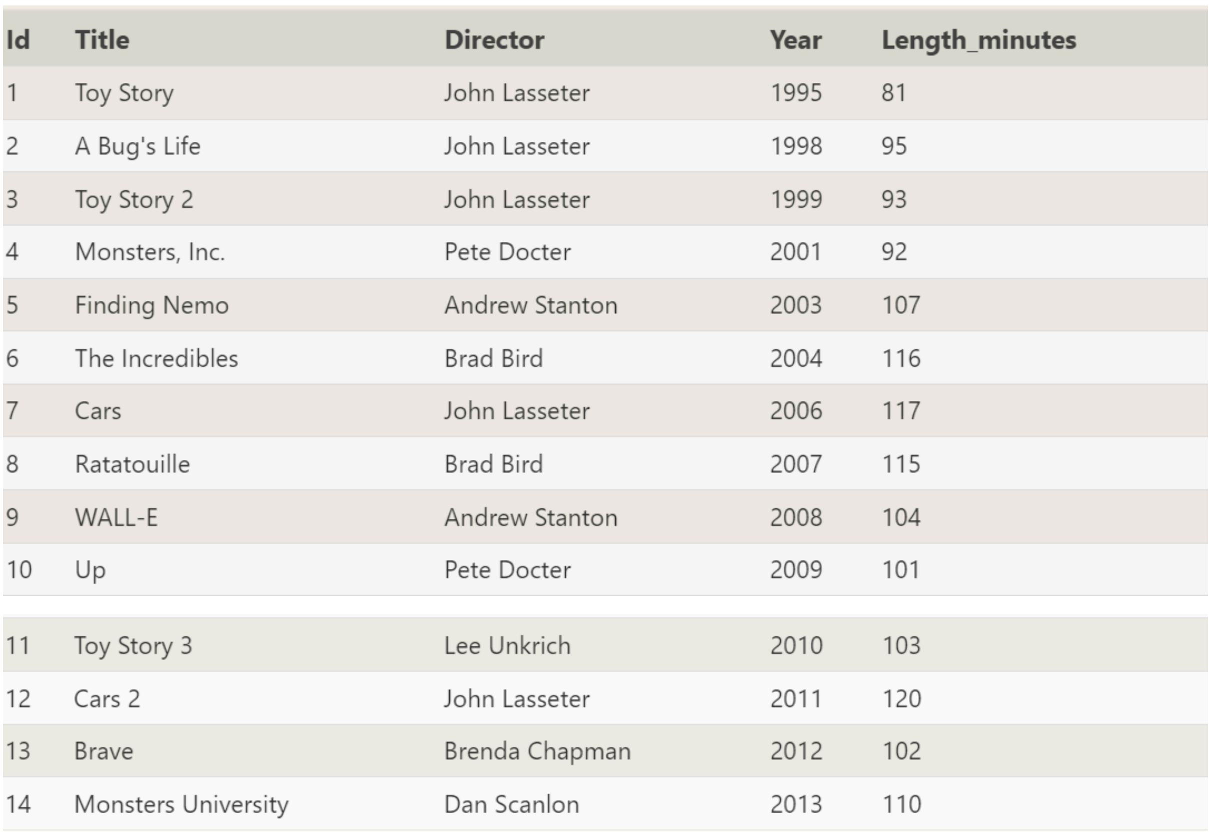
   - Specific columns:

     SELECT coln1_name, coln2_name FROM table_name;

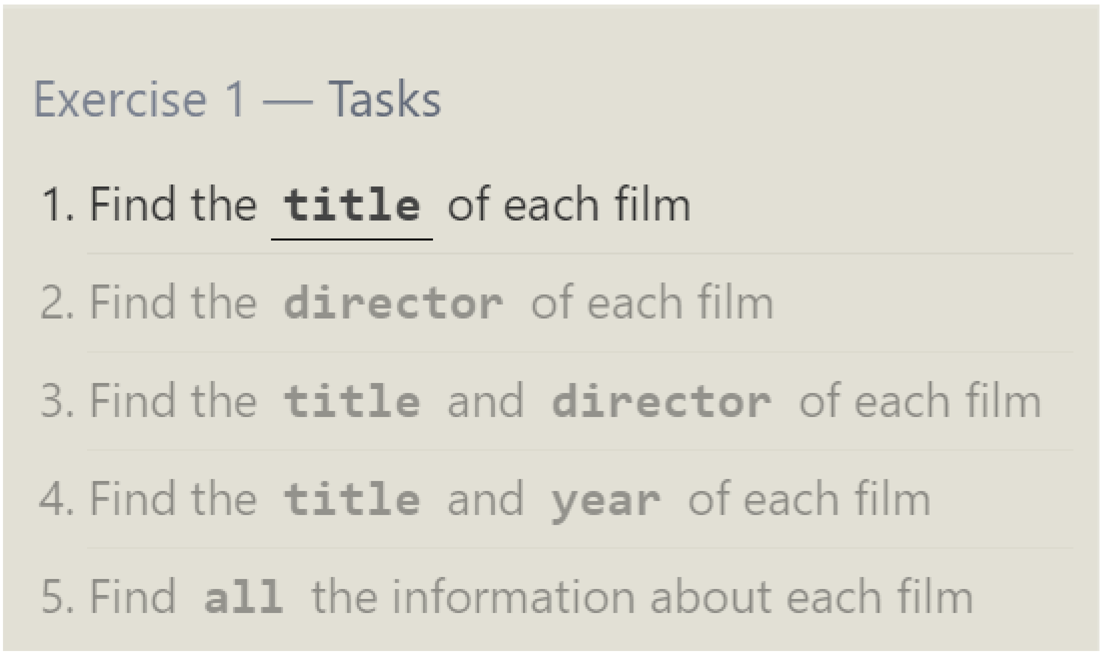
   - All columns:

     SELECT * FROM table_name;

Ex: Pixar's classic movies - Movies (table_name)



Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101
11	Toy Story 3	Lee Unkrich	2010	103
12	Cars 2	John Lasseter	2011	120
13	Brave	Brenda Chapman	2012	102
14	Monsters University	Dan Scanlon	2013	110


Exercise 1 — Tasks

1. Find the `title` of each film
2. Find the `director` of each film
3. Find the `title` and `director` of each film
4. Find the `title` and `year` of each film
5. Find `all` the information about each film

1. SELECT Title FROM movies;
2. SELECT Director FROM movies;
3. SELECT Title, Director FROM movies;
4. SELECT Title, Year FROM movies;
5. SELECT * FROM movies;

2. SELECT Queries with constraints - Numbers: To filter the records (WHERE clause).

SELECT coln1_name, coln2_name WHERE cond1 AND / OR cond2 … ;

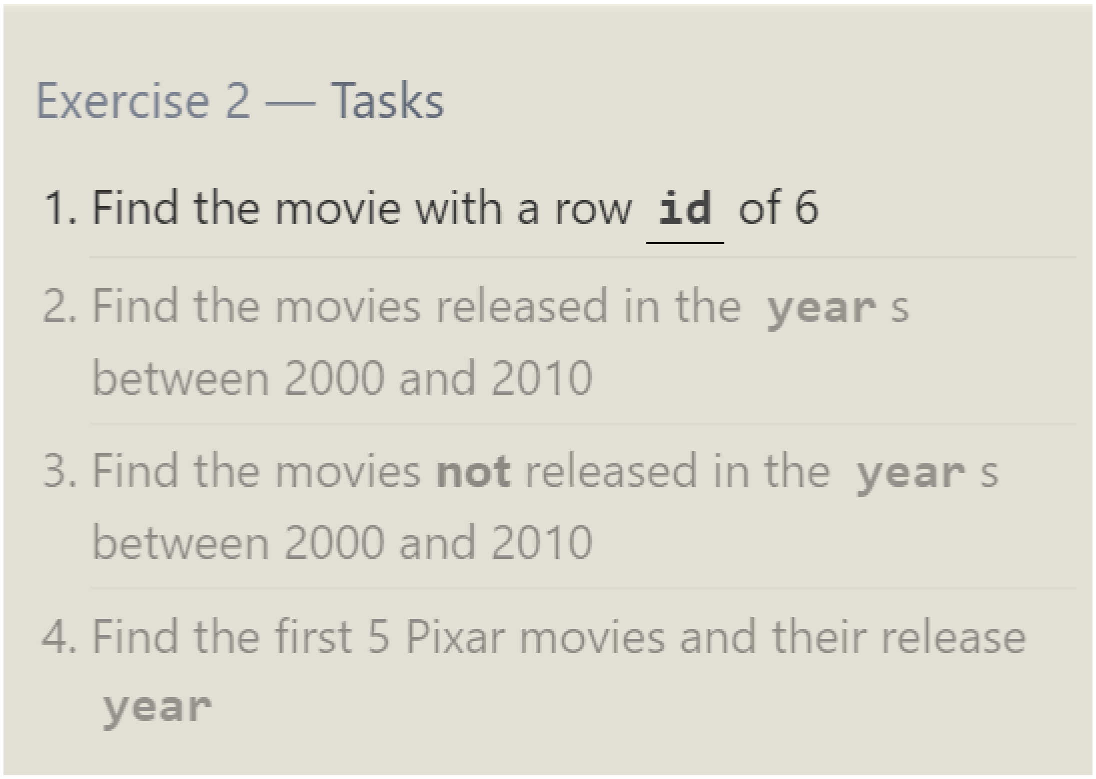
a) =, !=, >, <, >=, <=)) coln_name != num_value;

b) BETWEEN … AND …)) coln_name BETWEEN 5 AND 6;
[Within range of 2 values - inclusive]

c) NOT BETWEEN … AND …)) coln_name NOT BETWEEN 5 AND 6;
[Not within range of 2 values - inclusive]

d) IN)) coln_name IN (value1, value2, value3);
[Number exists in a list]

e) NOT IN)) coln_name NOT IN (value1, value2, value3);
[Number not exists in a list]



Exercise 2 — Tasks

1. Find the movie with a row `id` of 6
2. Find the movies released in the `year` s between 2000 and 2010
3. Find the movies **not** released in the `year` s between 2000 and 2010
4. Find the first 5 Pixar movies and their release `year`

1. SELECT * FROM movies WHERE Id = 6; (Or)
   SELECT id, title FROM movies WHERE id = 6;
2. SELECT * FROM movies WHERE Year BETWEEN 2000 AND 2010; (Or)
   SELECT title, year FROM movies WHERE year BETWEEN 2000 AND 2010;
3. SELECT * FROM movies WHERE Year NOT BETWEEN 2000 AND 2010; (Or)
   SELECT title, year FROM movies WHERE year < 2000 OR year > 2010;
4. SELECT * FROM movies WHERE Id <= 5; (Or)
   SELECT title, year FROM movies WHERE year <= 2003;

3. SELECT Queries with constraints - Strings: To filter the records (WHERE clause).

SELECT coln1_name, coln2_name WHERE cond1 AND / OR cond2 … ;

a) =)) coln_name = “abc”;
   [Case sensitive exact string comparison]

b) !=)) coln_name != “abc”;
   [Case sensitive exact string inequality comparison]

c) LIKE)) coln_name LIKE “abc”;
   [Case insensitive exact string comparison]

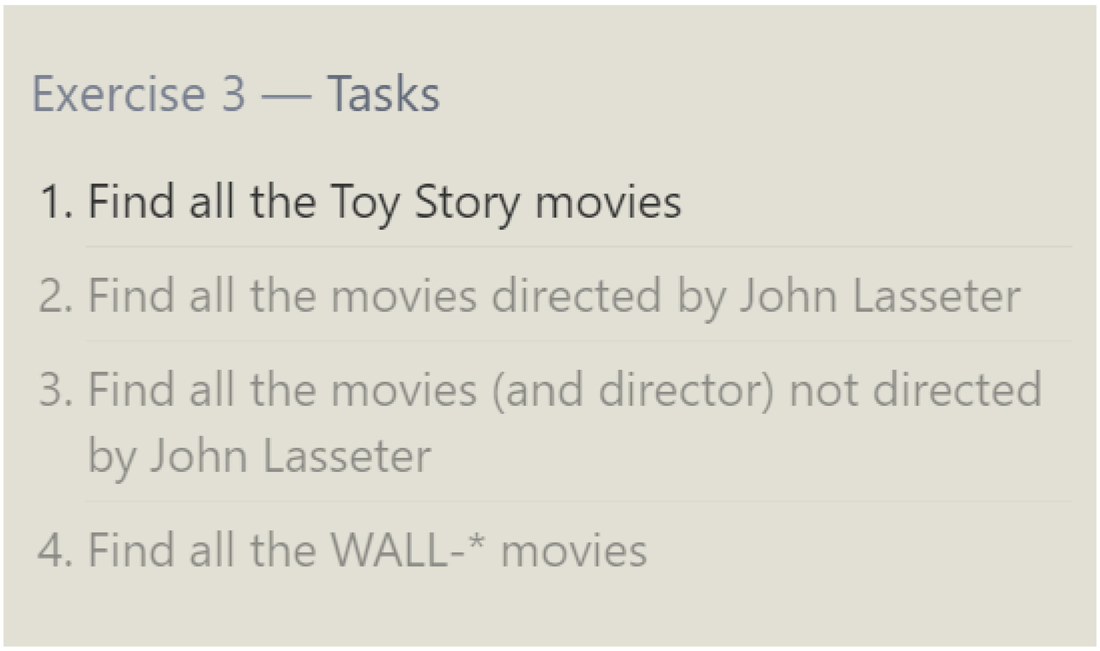
d) NOT LIKE)) coln_name NOT LIKE “abc”;
   [Case insensitive exact string inequality comparison]

e) %)) coln_name LIKE “%abc%”; | coln_name NOT LIKE “%abc%”;
   [Used anywhere in the string to match a sequence of characters]

f) _)) coln_name LIKE “ab_”; | coln_name NOT LIKE “ab_”;
   [Used anywhere in the string to match a single character]

g) IN)) coln_name IN (“a”, “b”, “c”);
   [String exists in a list]

h) NOT IN)) coln_name NOT IN (“a”, “b”, “c”);
   [String does not exists in a list]


Exercise 3 — Tasks

1. Find all the Toy Story movies
2. Find all the movies directed by John Lasseter
3. Find all the movies (and director) not directed by John Lasseter
4. Find all the WALL-* movies

1. SELECT * FROM movies WHERE Title LIKE "%Toy Story%"; (Or)
   SELECT title, director FROM movies WHERE title LIKE "Toy Story%";
2. SELECT * FROM movies WHERE Director LIKE "John Lasseter"; (Or)
   SELECT title, director FROM movies WHERE director = "John Lasseter";
3. SELECT * FROM movies WHERE Director NOT LIKE "John Lasseter"; (Or)
   SELECT title, director FROM movies WHERE director != "John Lasseter";
4. SELECT * FROM movies WHERE Title LIKE "WALL-_"; (Or)

4. Filtering & sorting query results:

a) DISTINCT - Discard rows that have duplicate column value.

   SELECT DISTINCT * FROM table_name WHERE condn(s);

b) ORDER BY - To sort the results by given column in ascending / descending order.

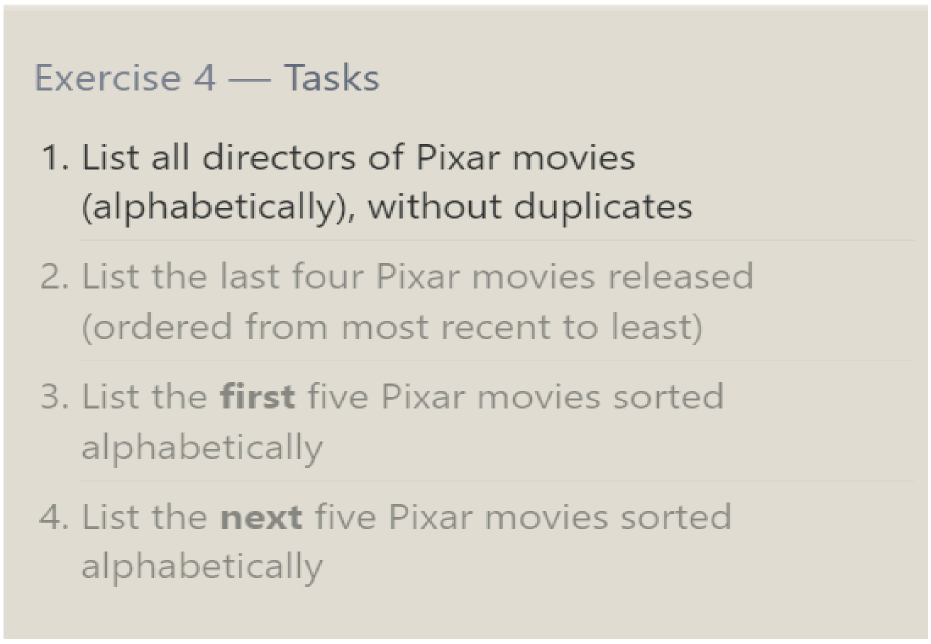
   SELECT * FROM table_name WHERE condn(s) ORDER BY coln_name ASC / DESC;

c) LIMIT - Reduce the number of rows return.

   SELECT * FROM table_name WHERE condn(s) ORDER BY coln_name ASC / DESC LIMIT limit_val;

d) OFFSET - Specify where to begin counting the number of rows from.

   SELECT * FROM table_name WHERE condn(s) ORDER BY coln_name ASC / DESC LIMIT limit_val OFFSET offset_val;


Exercise 4 — Tasks

1. List all directors of Pixar movies (alphabetically), without duplicates
2. List the last four Pixar movies released (ordered from most recent to least)
3. List the **first** five Pixar movies sorted alphabetically
4. List the **next** five Pixar movies sorted alphabetically

1. SELECT DISTINCT Director FROM movies ORDER BY Director ASC;
2. SELECT * FROM movies ORDER BY Year DESC LIMIT 4;
   (Or)
   SELECT title, year FROM movies ORDER BY year DESC LIMIT 4;
3. SELECT * FROM movies ORDER BY Title ASC LIMIT 5; (Or)
   SELECT title FROM movies ORDER BY title ASC LIMIT 5;
4. SELECT * FROM movies ORDER BY Title ASC LIMIT 5 OFFSET 5; (Or)
   SELECT title FROM movies ORDER BY title ASC LIMIT 5 OFFSET 5;

5. Review:

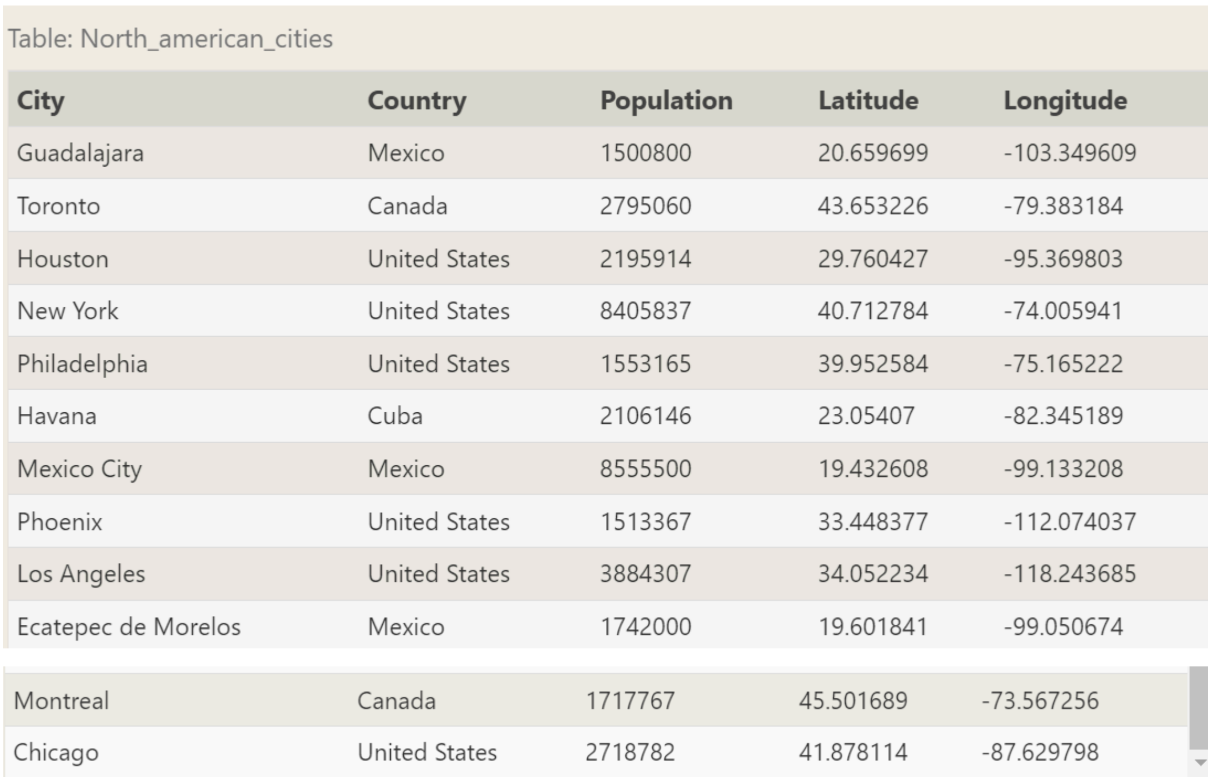
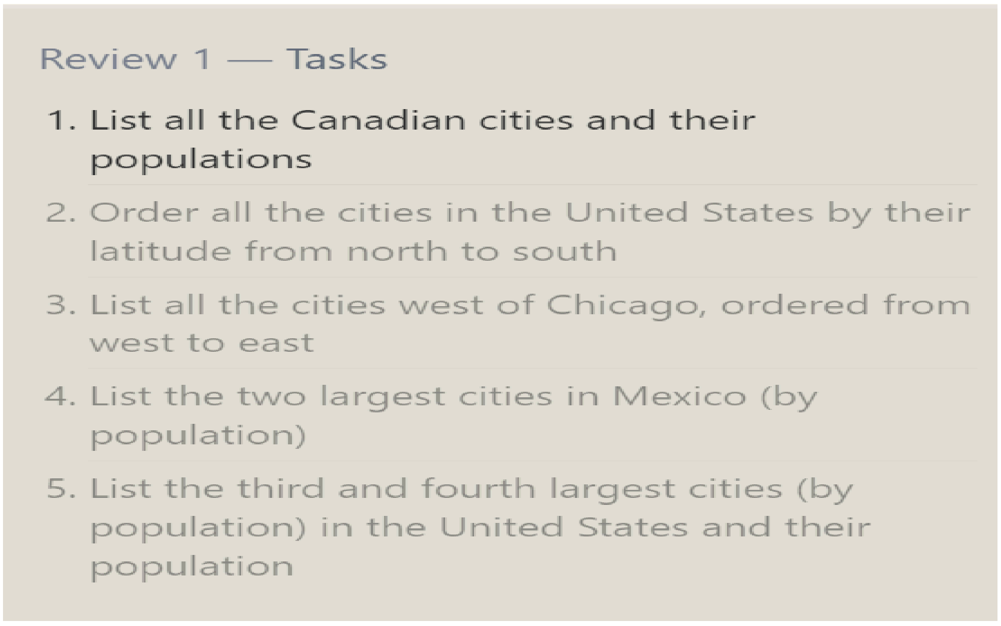


Table: North_american_cities

City	Country	Population	Latitude	Longitude
Guadalajara	Mexico	1500800	20.659699	-103.349609
Toronto	Canada	2795060	43.653226	-79.383184
Houston	United States	2195914	29.760427	-95.369803
New York	United States	8405837	40.712784	-74.005941
Philadelphia	United States	1553165	39.952584	-75.165222
Havana	Cuba	2106146	23.05407	-82.345189
Mexico City	Mexico	8555500	19.432608	-99.133208
Phoenix	United States	1513367	33.448377	-112.074037
Los Angeles	United States	3884307	34.052234	-118.243685
Ecatepec de Morelos	Mexico	1742000	19.601841	-99.050674
Montreal	Canada	1717767	45.501689	-73.567256
Chicago	United States	2718782	41.878114	-87.629798



Review 1 — Tasks

1. List all the Canadian cities and their populations
2. Order all the cities in the United States by their latitude from north to south
3. List all the cities west of Chicago, ordered from west to east
4. List the two largest cities in Mexico (by population)
5. List the third and fourth largest cities (by population) in the United States and their population

1. SELECT City, Population FROM north_american_cities WHERE Country = "Canada";
2. SELECT * FROM north_american_cities WHERE Country = "United States" ORDER BY Latitude DESC; (Or)

SELECT city, latitude FROM north_american_cities WHERE country = "United States" ORDER BY latitude DESC;

3. SELECT city, longitude FROM north_american_cities WHERE longitude < -87.629798 ORDER BY longitude ASC;
4. SELECT City FROM north_american_cities WHERE Country = "Mexico" ORDER BY Population DESC LIMIT 2; (Or)
   SELECT city, population FROM north_american_cities WHERE country LIKE "Mexico" ORDER BY population DESC LIMIT 2;
5. SELECT City, Population FROM north_american_cities WHERE Country = "United States" ORDER BY Population DESC LIMIT 2 OFFSET 2; (Or)
   SELECT city, population FROM north_american_cities WHERE country LIKE "United States" ORDER BY population DESC LIMIT 2 OFFSET 2;

6. Queries with Aggregates - Part 1:

- Aggregation: Summarize information about group of rows of data.

SELECT aggregate_func(coln_name or expression) AS aggregate_description FROM table_name WHERE condn(s);

a) COUNT(*) | COUNT(coln_name) - count the number of rows in the group if no column name is specified. O/W, count the number of rows in the group with non-null values in the column.

b) MIN(coln_name) - find the smallest value in the specified column for all rows in the group.

c) MAX(coln_name) - find the largest value in the specified column for all rows in the group.

d) AVG(coln_name) - find the average value in the specified column for all rows in the group.

e) SUM(coln_name) - find the sum of all values in the specified column for all rows in the group.

- GROUP BY: grouping rows that have the same value in the column specified.

  SELECT aggregate_func(coln_name or expression) AS aggregate_description FROM table_name WHERE condn(s) GROUP BY coln_name;

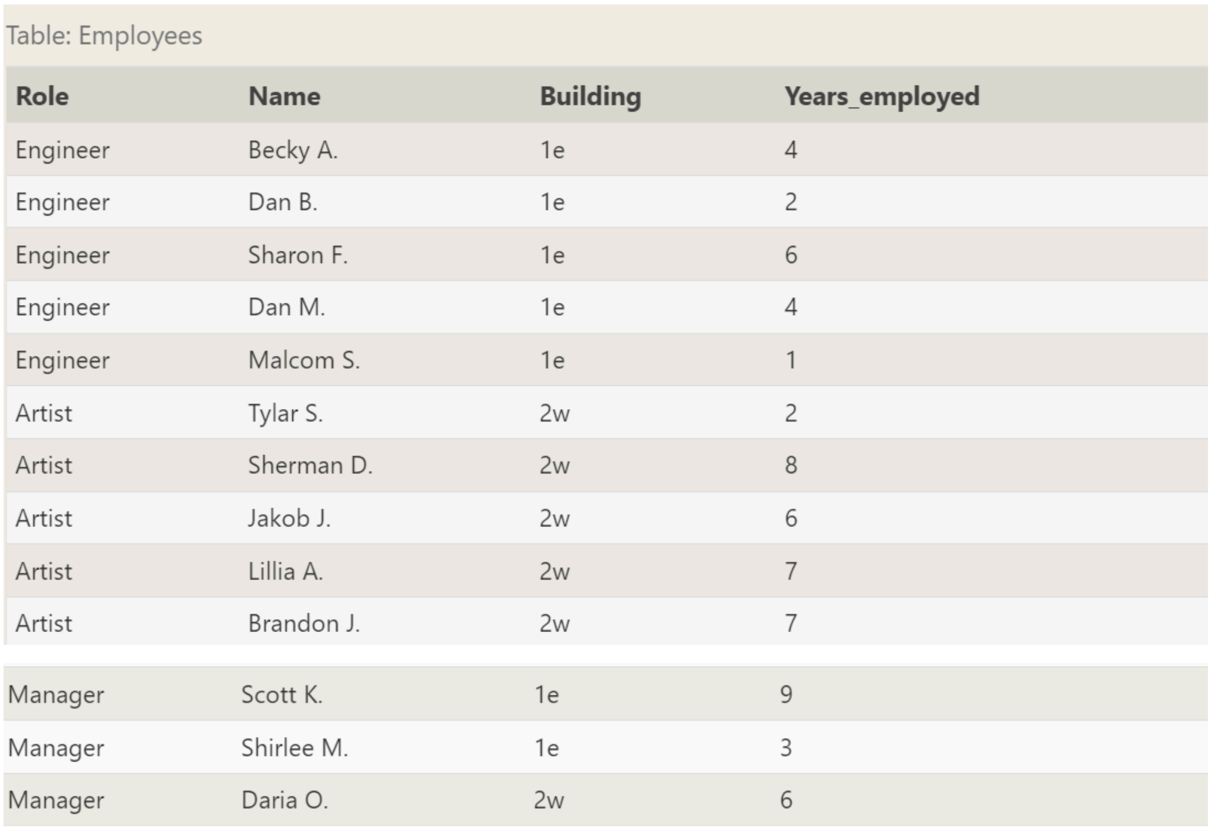
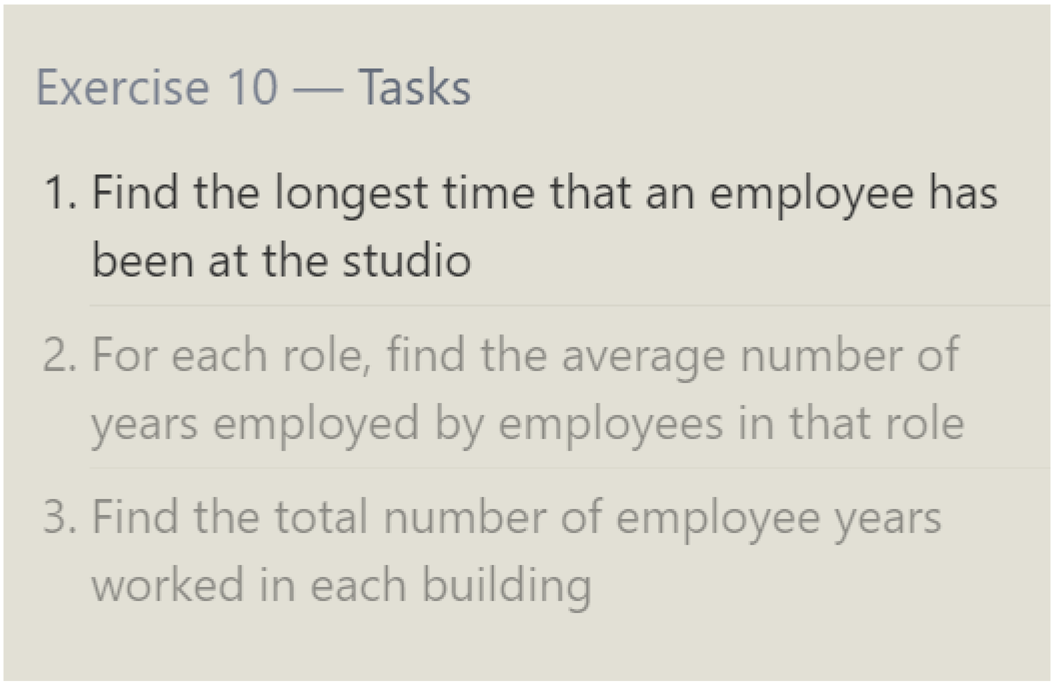


Table: Employees

Role	Name	Building	Years_employed
Engineer	Becky A.	1e	4
Engineer	Dan B.	1e	2
Engineer	Sharon F.	1e	6
Engineer	Dan M.	1e	4
Engineer	Malcom S.	1e	1
Artist	Tylar S.	2w	2
Artist	Sherman D.	2w	8
Artist	Jakob J.	2w	6
Artist	Lillia A.	2w	7
Artist	Brandon J.	2w	7
Manager	Scott K.	1e	9
Manager	Shirlee M.	1e	3
Manager	Daria O.	2w	6


Exercise 10 — Tasks

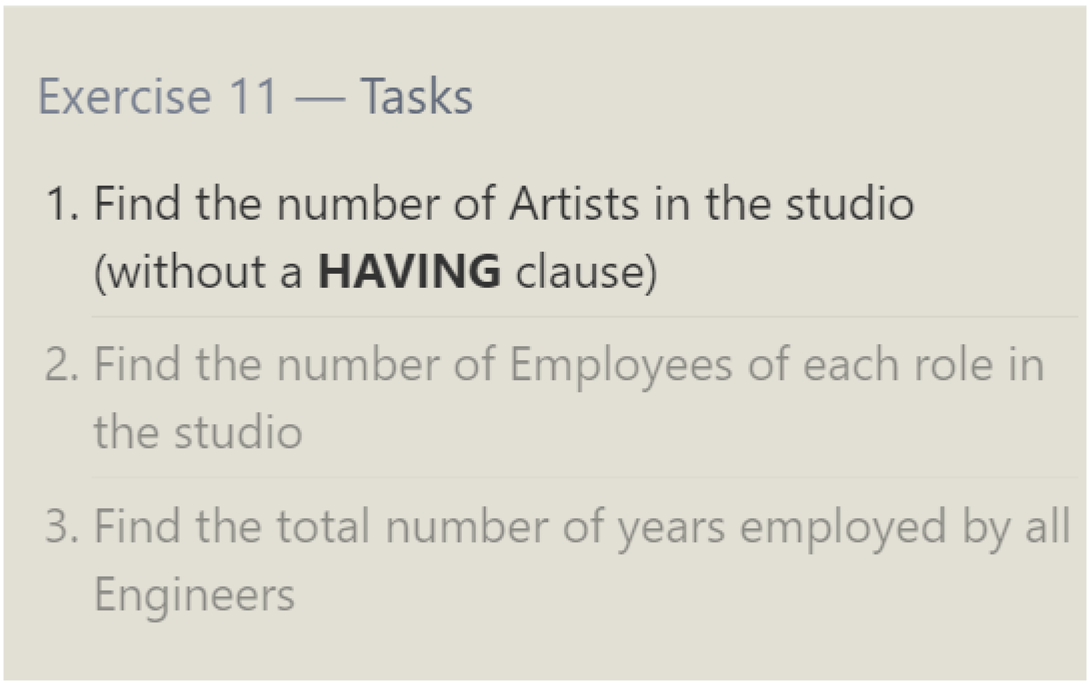
1. Find the longest time that an employee has been at the studio
2. For each role, find the average number of years employed by employees in that role
3. Find the total number of employee years worked in each building

1. SELECT MAX(Years_employed) FROM employees;
2. SELECT AVG(Years_employed) AS Average_YearsEmployes, Role FROM employees GROUP BY Role;
3. SELECT SUM(Years_employed) AS total_empYear, Building FROM employees GROUP BY Building;

- HAVING: used specifically with GROUP BY clause which will allow us to filter grouped rows.

  SELECT aggregate_func(coln_name or expression) AS aggregate_description FROM table_name WHERE condn(s) GROUP BY coln_name HAVING group_condition;

7. **Queries with Aggregates - Part 2:**



Exercise 11 — Tasks

1. Find the number of Artists in the studio (without a **HAVING** clause)
2. Find the number of Employees of each role in the studio
3. Find the total number of years employed by all Engineers

1. SELECT COUNT(Role) FROM employees GROUP BY Role HAVING Role = "Artist"; (Or)
   SELECT COUNT(Role) FROM employees WHERE Role = "Artist"; (Or)
   SELECT role, COUNT(*) as Number_of_artists FROM employees WHERE role = "Artist";
2. SELECT Role, COUNT(Name) AS No_of_emp_role FROM employees GROUP BY Role;
3. SELECT SUM(Years_employed) FROM employees GROUP BY Role HAVING Role = "Engineer"; (Or)
   SELECT SUM(Years_employed) FROM employees WHERE Role = "Engineer"; (Or)

8. Inserting rows:

- Insert statement with values for all columns:

  INSERT INTO table_name VALUES (value_or_expression1, …), (value_or_expression2, …), ... ;

- Insert statement with values for specific columns:

  INSERT INTO (coln1, coln2) VALUES (value_or_expression1, …), (value_or_expression2, …) ;

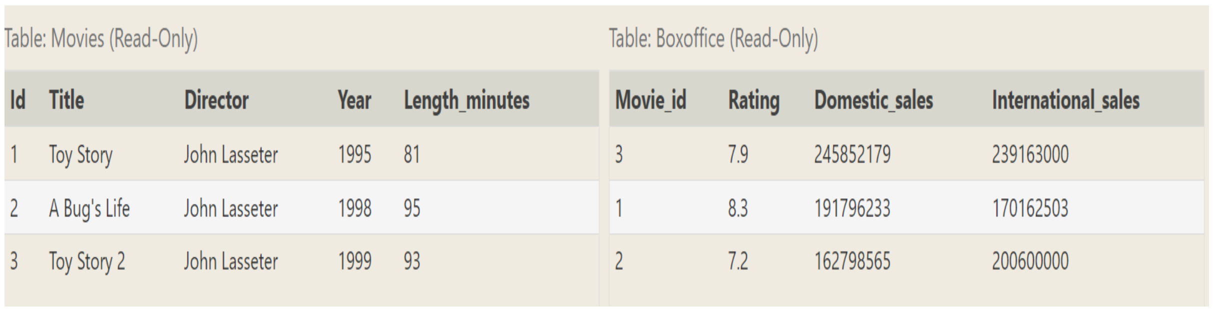
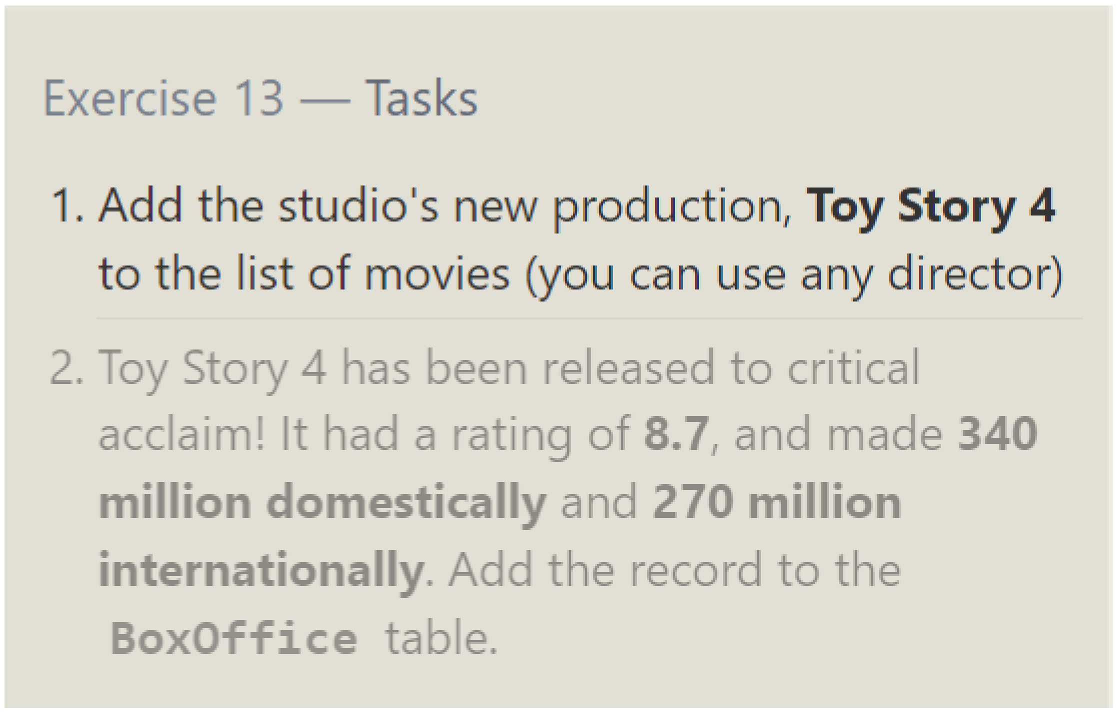


Table: Movies (Read-Only)

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93

Table: Boxoffice (Read-Only)

Movie_id	Rating	Domestic_sales	International_sales
3	7.9	245852179	239163000
1	8.3	191796233	170162503
2	7.2	162798565	200600000



Exercise 13 — Tasks

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

1. INSERT INTO Movies (Id, Title, Director, Year, Length_minutes) VALUES (15, "Toy Story 4", "John Lasseter", 2014, 124); (Or) INSERT INTO movies VALUES (15, "Toy Story 4", "John Lasseter", 2014, 124);
2. INSERT INTO Boxoffice(Movie_id, Rating, Domestic_sales, International_sales) VALUES (15, 8.7, 340, 270); (Or) INSERT INTO boxoffice VALUES (15, 8.7, 340, 270);

9. Updating rows: To update the existing data.

- UPDATE table_name SET coln_name1 = value_or_expression, coln_name2 = value_or_expression, … WHERE condition;

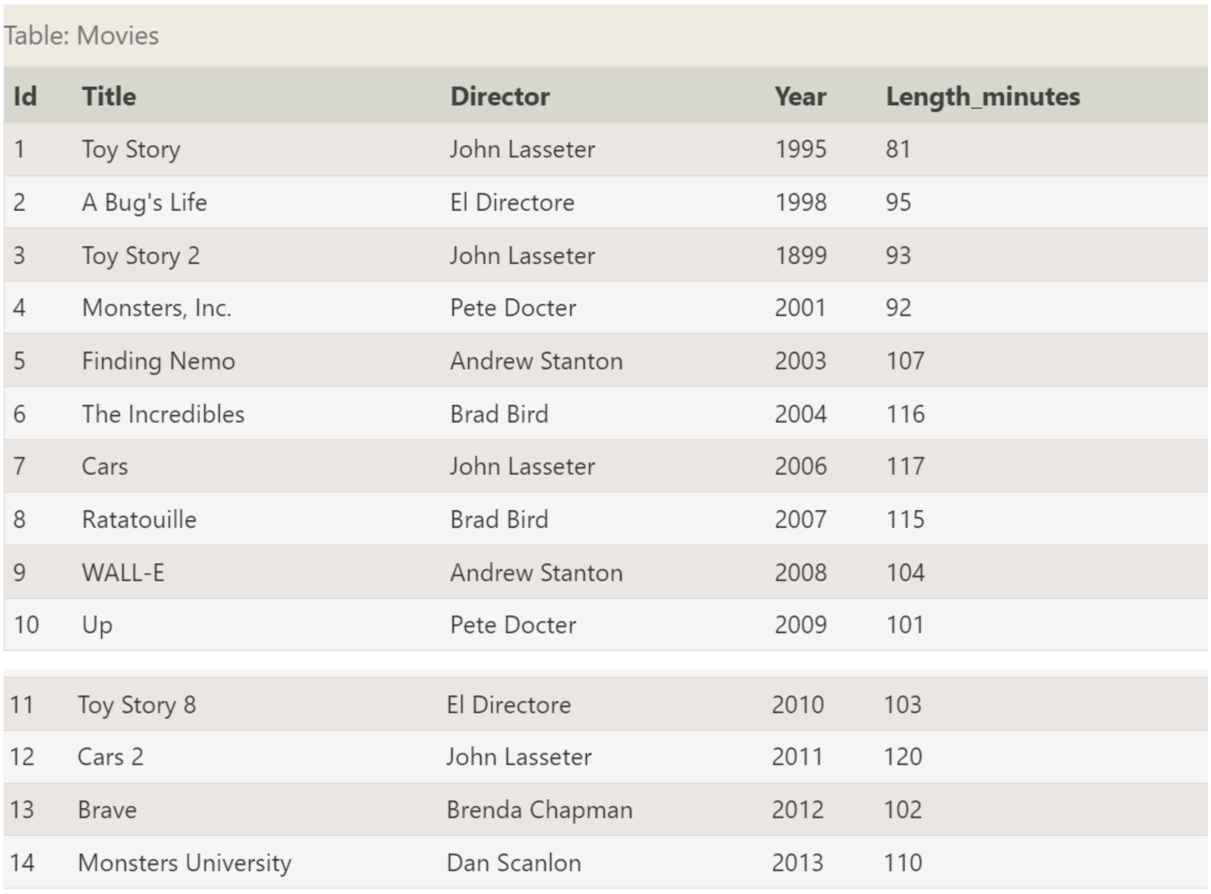
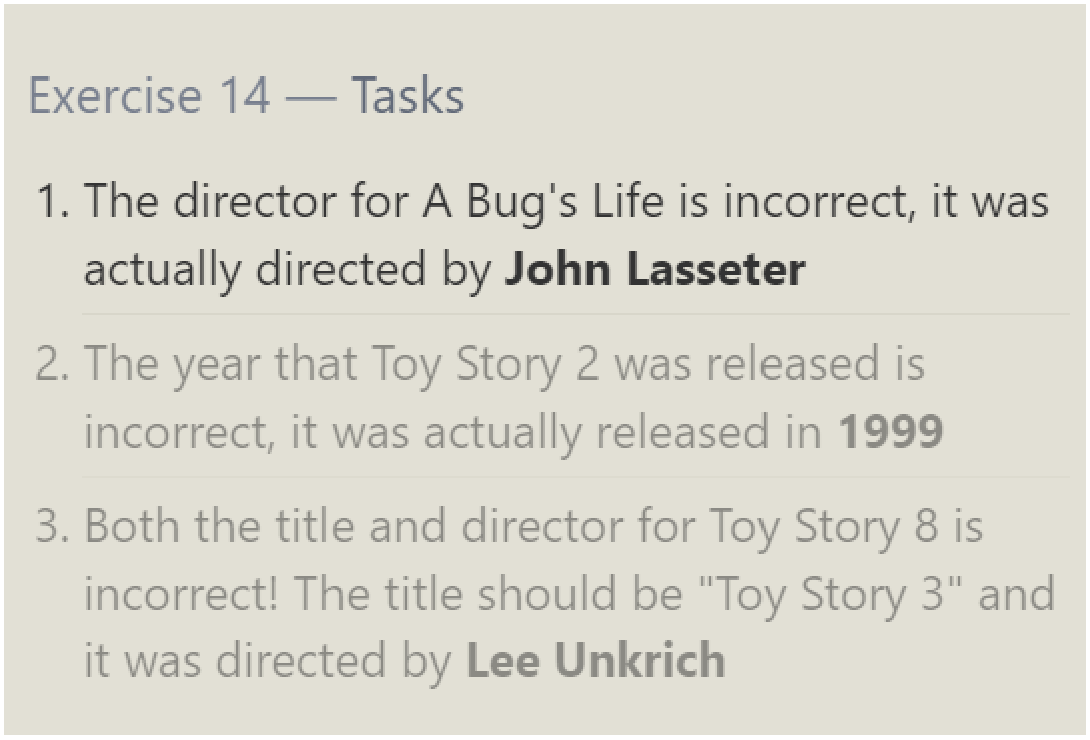


Table: Movies

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	El Directore	1998	95
3	Toy Story 2	John Lasseter	1899	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101
11	Toy Story 8	El Directore	2010	103
12	Cars 2	John Lasseter	2011	120
13	Brave	Brenda Chapman	2012	102
14	Monsters University	Dan Scanlon	2013	110


Exercise 14 — Tasks

1. The director for A Bug's Life is incorrect, it was actually directed by **John Lasseter**
2. The year that Toy Story 2 was released is incorrect, it was actually released in **1999**
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**

1. UPDATE Movies SET Director = "John Lasseter" WHERE Title = "A Bug's Life"; (Or)
   UPDATE movies SET director = "John Lasseter" WHERE id = 2;
2. UPDATE Movies SET Year = 1999 WHERE Title = "Toy Story 2"; (Or)
   UPDATE movies SET year = 1999 WHERE id = 3;
3. UPDATE Movies SET Title = "Toy Story 3", Director = "Lee Unkrich" WHERE Id = 11;

10. Deleting rows:

- DELETE FROM table_name WHERE condn(s);

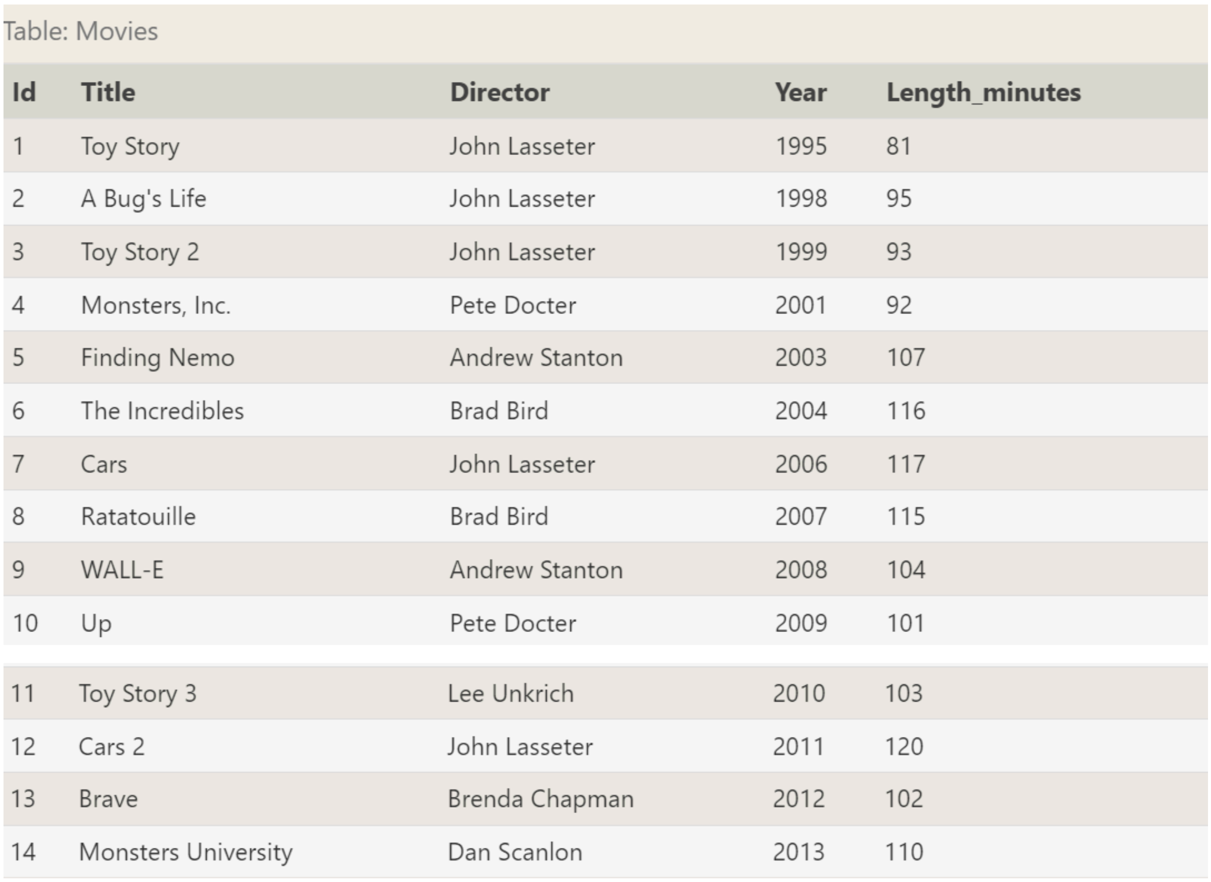
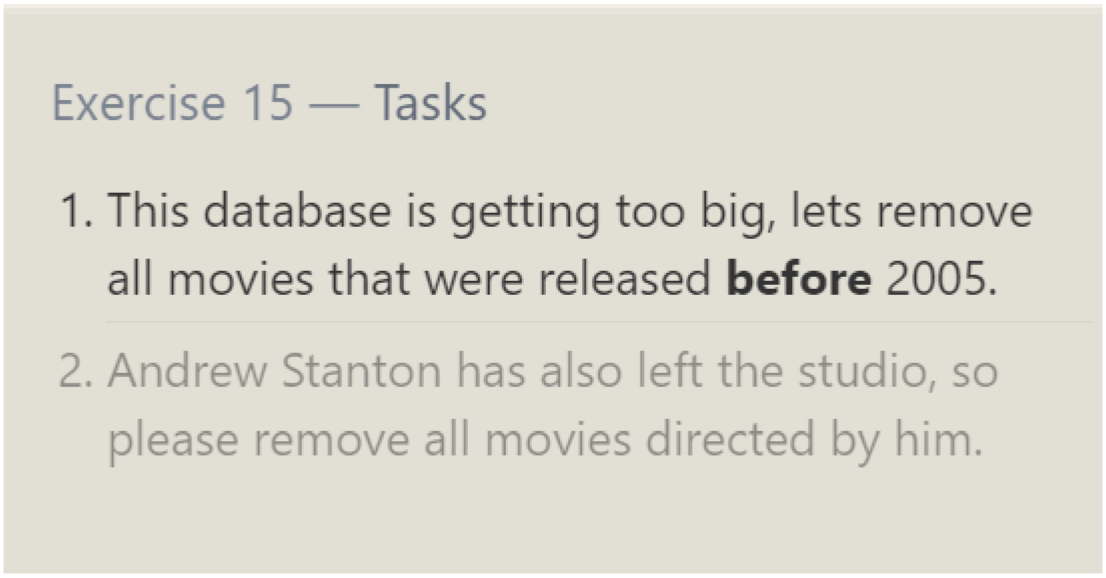


Table: Movies

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101
11	Toy Story 3	Lee Unkrich	2010	103
12	Cars 2	John Lasseter	2011	120
13	Brave	Brenda Chapman	2012	102
14	Monsters University	Dan Scanlon	2013	110


Exercise 15 — Tasks

1. This database is getting too big, lets remove all movies that were released **before** 2005.
2. Andrew Stanton has also left the studio, so please remove all movies directed by him.

1. DELETE FROM Movies WHERE Year < 2005;
2. DELETE FROM Movies WHERE Director = "Andrew Stanton";

10. Altering tables: To add, remove, or modify columns in the table.

- Adding columns:

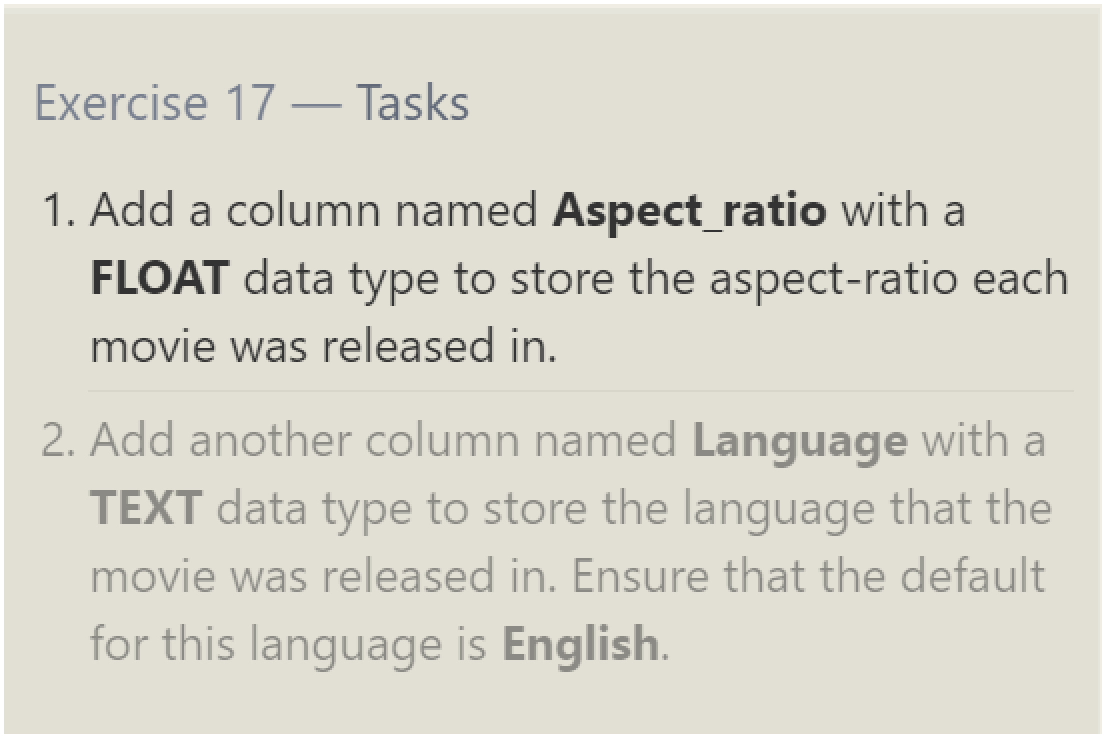
  ALTER TABLE table_name ADD coln_name DATATYPE DEFAULT default_value;

- Removing columns:

  ALTER TABLE table_name DROP coln_name_to_be_deleted;

- Renaming the table:

  ALTER TABLE old_table_name RENAME TO new_table_name;



Exercise 17 — Tasks

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

1. ALTER TABLE Movies ADD Aspect_ratio FLOAT DEFAULT 6.5;
2. ALTER TABLE Movies ADD Language TEXT DEFAULT "English";

11. Dropping tables:

- DROP TABLE IF EXISTS table_name;

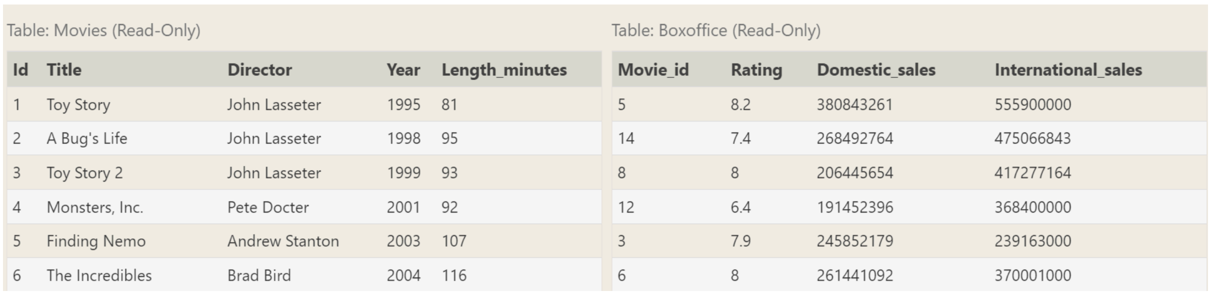
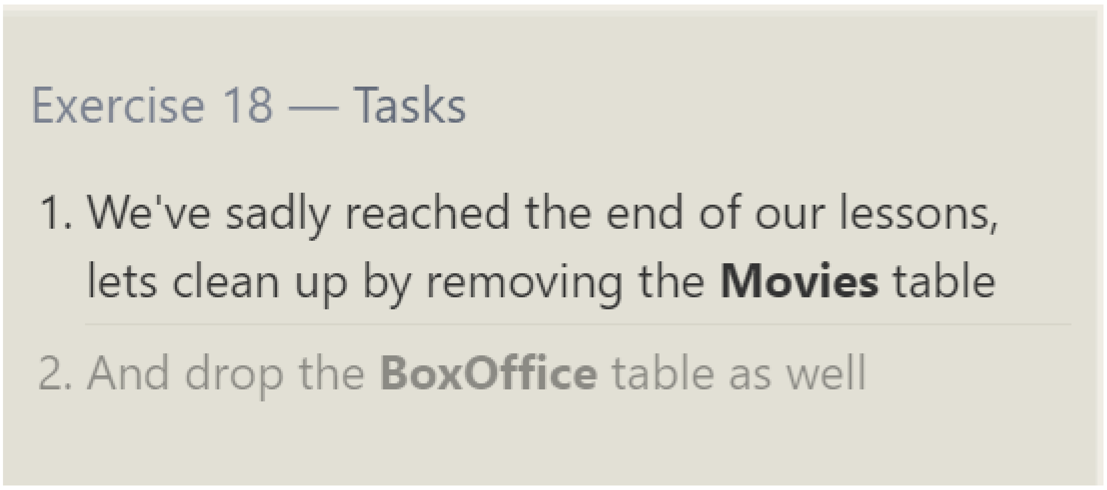


Table: Movies (Read-Only)

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116

Table: Boxoffice (Read-Only)

Movie_id	Rating	Domestic_sales	International_sales
5	8.2	380843261	555900000
14	7.4	268492764	475066843
8	8	206445654	417277164
12	6.4	191452396	368400000
3	7.9	245852179	239163000
6	8	261441092	370001000



Exercise 18 — Tasks

1. We've sadly reached the end of our lessons, lets clean up by removing the **Movies** table
2. And drop the **BoxOffice** table as well

1. DROP TABLE IF EXISTS Movies;
2. DROP TABLE IF EXISTS BoxOffice;