**List all directors of Pixar movies (alphabetically), without duplicates**SELECT DISTINCT director FROM movies

ORDER BY director ASC;

**List the last four Pixar movies released (ordered from most recent to least)**  
SELECT title, year FROM movies

ORDER BY year DESC

LIMIT 4;

**List the first five Pixar movies sorted alphabetically**SELECT title FROM movies

ORDER BY title ASC

LIMIT 5;

**List the next five Pixar movies sorted alphabetically**SELECT title FROM movies

ORDER BY title ASC

LIMIT 5 OFFSET 5;  
  
**List all the Canadian cities and their populations ✓**  
SELECT city, population FROM north\_american\_cities

WHERE country = "Canada";

**Order all the cities in the United States by their latitude from north to south**SELECT city, latitude FROM north\_american\_cities

WHERE country = "United States"

ORDER BY latitude DESC;

**List all the cities west of Chicago, ordered from west to east**SELECT city, longitude FROM north\_american\_cities

WHERE longitude < -87.629798

ORDER BY longitude ASC;

**List the two largest cities in Mexico (by population)**  
SELECT city, population FROM north\_american\_cities

WHERE country LIKE "Mexico"

ORDER BY population DESC

LIMIT 2;

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

SELECT city, population FROM north\_american\_cities

WHERE country LIKE "UNITED STATES"

ORDER BY population DESC

LIMIT 2 OFFSET 2;  
  
**Find the domestic and international sales for each movie**SELECT title, domestic\_sales, international\_sales

FROM movies

INNER JOIN boxoffice

ON movies.id = boxoffice.movie\_id;

**Show the sales numbers for each movie that did better internationally rather than domestically**SELECT title, domestic\_sales, international\_sales

FROM movies

INNER JOIN boxoffice

ON movies.id = boxoffice.movie\_id;

WHERE international\_sales > domestic\_sales;

**List all the movies by their ratings in descending order**SELECT \* FROM movies;

SELECT title, rating

FROM movies

JOIN boxoffice

ON movies.id = boxoffice.movie\_id

ORDER BY rating DESC;  
  
**Find the list of all buildings that have employees**SELECT DISTINCT building FROM employees;

**Find the list of all buildings and their capacity**SELECT \* FROM buildings;

**List all buildings and the distinct employee roles in each building (including empty buildings)**SELECT DISTINCT role, building\_name

FROM buildings

LEFT JOIN employees

ON building\_name = building;

**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;

**Find the names of the buildings that hold no employees**  
SELECT DISTINCT building\_name

FROM buildings

LEFT JOIN employees

ON building\_name = building

WHERE role IS NULL;

**List all movies and their combined sales in millions of dollars**

SELECT title, (domestic\_sales + international\_sales) / 1000000 AS gross\_sales\_millions

FROM movies

JOIN boxoffice

ON movies.id = boxoffice.movie\_id;

**List all movies and their ratings in percent**SELECT title, rating \* 10 AS rating\_percent

FROM movies

JOIN boxoffice

ON movies.id = boxoffice.movie\_id;

**List all movies that were released on even number years**  
SELECT title, year

FROM movies

WHERE year % 2 = 0;

**Find the longest time that an employee has been at the studio**SELECT MAX(years\_employed) as Max\_years\_employed

FROM employees;

**For each role, find the average number of years employed by employees in that role**SELECT role, AVG(years\_employed) as Average\_years\_employed

FROM employees

GROUP BY role;

**Find the total number of employee years worked in each building**SELECT building, SUM(years\_employed) as Total\_years\_employed

FROM employees

GROUP by building;

**Find the number of Artists in the studio (without a HAVING clause) ✓**SELECT role, COUNT(\*) as Number\_of\_artists

FROM employees

WHERE role = "Artist";

**Find the number of Employees of each role in the studio**SELECT role, COUNT(\*) as Number\_of\_roles

FROM employees

GROUP BY role;

**Find the total number of years employed by all Engineers**SELECT role, SUM(Years\_employed) AS total\_years\_employed

FROM employees

GROUP BY role

HAVING role = "Engineer";  
  
**Find the number of movies each director has directed**SELECT director, COUNT(id) as Num\_movies\_directed

FROM movies

GROUP BY director;

**Find the total domestic and international sales that can be attributed to each director**SELECT director, SUM(domestic\_sales + international\_sales) as Cumulative\_sales\_from\_all\_movies

FROM movies

INNER JOIN boxoffice

ON movies.id = boxoffice.movie\_id

GROUP BY director;  
  
**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", "El Directore", 2015, 90);

**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.**INSERT INTO boxoffice VALUES (4, 8.7, 340000000, 270000000);

**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;

**The year that Toy Story 2 was released is incorrect, it was actually released in 1999**UPDATE movies

SET year = 1999

WHERE id = 3;

**Both the title and directory 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;  
  
**This database is getting too big, lets remove all movies that were released before 2005.**DELETE FROM movies

where year < 2005;

**Andrew Stanton has also left the studio, so please remove all movies directed by him**DELETE FROM movies

where director = "Andrew Stanton";  
  
**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 Database (

Name TEXT,

Version FLOAT,

Download\_count INTEGER

);

**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;

**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 COLUMN Language TEXT DEFAULT "English";  
  
**We've sadly reached the end of our lessons, lets clean up by removing the Movies table ✓**DROP TABLE Movies;

**And drop the BoxOffice table as well**DROP TABLE BoxOffice;  
  
RELATIONAL ALGEBRA  
SELECT ->σ (sigma)

PROJECT -> π(pi)

PRODUCT -> ×(times)

JOIN -> |×| (bow-tie)

UNION -> ∪ (cup)

INTERSECTION -> ∩(cap)

DIFFERENCE -> - (minus)

RENAME ->ρ (rho)

EXAMPLE  
**College(cname, state, enrollment)  
Student(SID, SName, GPA, sizeHS)  
Apply(SID, Cname, major, decision)  
GPA > 3.7**σGPA > 3.7 (GPA)  
**GPA > 3.7 HS< 1000**σgpa > 3.7 ^ hs 1000(student)  
**CS MAJOR**σcname = ‘stanford’ ^ major ‘cs’ (Apply)

**ID Name of student gpa >3,7**πsid,snameσgpa >3,7 (Student)  
**List of app majors & decisions**  
πmajor ^ dec (Apply)  
**Names, gpa student >1000 to CS Rejected**πname, GPA (σ student.SID = Apply.SID ^ HS > 1000 ^ major =’CS’ ^ dec = ‘R” (Student X Apply))  
**OR**  
πname, GPA (σ HS > 1000 ^ major =’CS’ ^ dec = ‘R” (Student |×| Apply)  
**Names, gpa student >1000 to CS with enr>20,000 & Rejected**πname, GPA (σ HS > 1000 ^ major =’CS’ ^ dec = ‘R’ ^ enr>20,000 (Student |×| Apply |×| College)  
  
**List of college & students name**πcname (College) U πsname (Student)  
**ID who didn’t apply anywhere**πSID (Student) – πSID(Apply)  
**ID & Name who didn’t apply anywhere**π sname (πSID (Student) – πSID(Apply)) |×| Student)  
**Name both college name and student name**πcname (college) ∩ πsname(Student)

**RENAME  
List all College and student names**ρc(name) ( πcname (College) U

ρc(name) (πsname (Student)  
**Pairs of college in same state**σ s1=s2 (ρc1(n1,s1,e1)(College) X ρc2(n2,s2,e2)(College))  
OR  
σ n1 < n2 (ρc1(n1,s,e1)(College) |×| ρc2(n2,s,e2)(College))