# Views, CTEs, and Joins

This file contains several exercises on views, common table expressions (CTEs), and joins. These topics are a step more involved and confusing than in the previous two files. Before attempting these problems, you'll probably need to do some research into these topics. Here are a few tutorials:

\* [Tutorial on Views](https://www.sqlitetutorial.net/sqlite-create-view/)

\* [Tutorial on CTEs](https://www.essentialsql.com/introduction-common-table-expressions-ctes/)

\* [Tutorial on Joins](https://www.sqlitetutorial.net/sqlite-join/)

Here are a few other important notes I'd like you read before beginning:

\* Some of these problems can get pretty involved. Queries with adequate spacing can go longer than 15 lines in some problems.

\* Make sure you read each question thoroughly.

\* Don't skip problems, as some problems may rely on previous problems being done correctly.

\* Make sure you are saving your answers as you go, as some answers will simply be reworkings of previous answers.

\* Some problems in the `Views` section involve you making persistent changes to the database itself. Specifically, if you create a view, you cannot create another view with the same name if it already exists. You may want to learn the `DROP VIEW` command. Or maybe a little easier, you can recreate the entire database by simply running the `seed.py` script provided in this repo. If you're unsure, you can see your list of tables and views using the `.tables` dot-command from the `sqlite3` interface.

## Views

62) Look at the `yum` table. It is the stock data for Yum! Brands, Inc. from 2015 through 2019. Yum! is the company that owns Taco Bell, the best restaurant.

SELECT \* FROM yum;

63) Query the `yum` table, aggregating by \*\*both\*\* month and year, with the following resulting columns:

\* Year (4 digits)

\* Month

\* Average open, high, low, and close

\* Total volume

Finally, sort this data so it's in proper chronological order.

SELECT EXTRACT(year from date) AS year, EXTRACT(month from date) AS month, ROUND(AVG(open)), ROUND(AVG(high)), ROUND(AVG(low)), ROUND(AVG(close)), SUM(volume)

FROM yum

GROUP BY date

ORDER BY date;

64) Save the results of the previous query as a view named `yum\_by\_month`.

CREATE VIEW yum\_by\_month AS

SELECT EXTRACT(year from date) AS year, EXTRACT(month from date) AS month, ROUND(AVG(open)), ROUND(AVG(high)), ROUND(AVG(low)), ROUND(AVG(close)), SUM(volume)

FROM yum

GROUP BY date

ORDER BY date;

65) Create a view of `transactions` consisting of only three columns: year, month, and total sales in that month. Call this view `trans\_by\_month`.

CREATE VIEW trans\_by\_month AS

SELECT EXTRACT(year from orderdate) AS year, EXTRACT(month from orderdate) AS month, SUM(unit\_price \* quantity) AS total\_sales

FROM transactions

GROUP BY orderdate

ORDER BY orderdate ASC;

66) Create a view of `transactions` consisting of only two columns: `employee\_id` and the total sales corresponding to that employee. Call this view `trans\_by\_employee`.

CREATE VIEW trans\_by\_employee AS

SELECT employee\_id, SUM(unit\_price \* quantity) AS total\_sales

FROM transactions

GROUP BY employee\_id

ORDER BY employee\_id ASC;

## Common Table Expressions (CTEs)

CTEs are a convenient way of shortening SQL queries to keep your code DRY (\*\*d\*\*on't \*\*r\*\*epeat \*\*y\*\*ourself). You'll notice they're essentially the same in terms of the tasks they can accomplish, however CTEs are \_temporary\_. They vanish after the query has been called. Essentially, CTEs are single-use views.

Therefore, CTEs aren't needed to solve any of the following problems. You could use a view instead, however that would be wasteful since you'll never use them again. Additionally, for some problems, neither a view nor a CTE is truly needed, but the query would be very messy without one.

67) What's the most common first initial for pets in the `pets` table?

\* \_Hint:\_ Create a CTE that is simply the lowercased first letter of the pet's name. The solution is a simple `GROUP BY` from this CTE.

\* \_Hint 2:\_ You'll need the `SUBSTR()` and `LOWER()` functions.

WITH most\_common (firstletterlower, frequency\_of\_name) AS

(SELECT LOWER(SUBSTR(name, 1, 1)), COUNT(name)

FROM pets

GROUP BY SUBSTR(name, 1, 1)

ORDER BY COUNT(name) DESC)

SELECT firstletterlower, frequency\_of\_name FROM most\_common;

68) Create taglines for each employee in the `employees` table. As a template, the first row of the result should look like this:

```

Christine Thompson started in 2005 and makes $123,696 working in sales.

```

To do this easily, make a CTE featuring name (firstname + " " + lastname), job, salary (formatted), and year. Job title should be lowercased, \_unless\_ it is IT, in which case leave it capitalized. The solution is simple string concatenation off of this long CTE.

WITH tagline (fullname, job, salary, year) AS

(SELECT CONCAT(firstname, '', lastname) AS fullname,

CASE WHEN job = 'IT' THEN job ELSE lower(job) END,

to\_char(salary, 'fm$999,999,999') AS salary, EXTRACT(year from startdate)

FROM employees)

SELECT CONCAT(fullname, ' started in ', year, ' and makes ',

salary, ' working in ', job) AS tagline FROM tagline;

69) How many of our sales come from companies ending in each of "LLC", "Inc", "Ltd", or "PLC"? In a CTE, create a `company\_type` column of values `"LLC"`, `"Inc"`, `"Ltd"`, `"PLC"`, or `"Other"`. Outside the CTE, find the total revenue from these categories, as well as their respective counts.

\* \_Hint:\_ You'll need the `INSTR()` function.

WITH company (c\_type) AS

(SELECT CASE WHEN RIGHT(customer, 3) = 'LLC' THEN 'LLC'

WHEN RIGHT(customer, 3 )= 'Inc' THEN 'Inc'

WHEN RIGHT(customer, 3)= 'Ltd' THEN 'Ltd'

WHEN RIGHT(customer, 3) = 'PLC' THEN 'PLC'

ELSE 'other' END AS company\_type, order\_id

FROM transactions

ORDER BY company\_type)

SELECT TO\_CHAR(SUM(unit\_price \* quantity), 'fm&999,999,999') AS total\_revenue, c.c\_type, COUNT(c\_type)

FROM transactions t

JOIN company c

ON c.order\_id = t.order\_id

GROUP BY c\_type;

## Joins

No, we're not done talking about views and CTEs! We're just going to start intermingling them in with further examples on joins, where the real power of these techniques becomes clearer.

70) Which employee made which sale? Join the `employees` table onto the `transactions` table by `employee\_id`. You only need to include the employee's first/last name from `employees`.

SELECT (e.firstname, e.lastname) AS employee\_name, t.order\_id

FROM employees e

JOIN transactions t

ON e.id = t.employee\_id;

71) What is the name of the employee who made the most in sales? Find this answer by doing a join as in the previous problem. Your resulting query will be difficult for someone else to read.

SELECT CONCAT(e.firstname, ' ', e.lastname) AS employee\_name, SUM(DISTINCT t.order\_id), SUM(t.unit\_price \* t.quantity) AS total

FROM employees e

JOIN transactions t

ON e.id = t.employee\_id

GROUP BY 1

ORDER BY 3 DESC

72) Solve the previous problem by joining `employees` onto the `trans\_by\_employee` view you made earlier.

SELECT CONCAT(e.firstname, ' ', e.lastname) AS employee\_name, t.employee\_id, t.total\_cost

FROM employees e

JOIN trans\_by\_employee t

ON t.employee\_id = e.id

ORDER BY total\_cost DESC;

73) Solve the previous problem by joining `employees` onto a CTE.

WITH most\_successful (employee, revenue, id) AS

(SELECT t.employee\_id, t.total\_cost, CONCAT(e.firstname, ' ', e.lastname) AS employee

FROM trans\_by\_employee t

JOIN employees e

ON e.id = t.employee\_id)

SELECT \* FROM most\_successful

ORDER BY revenue DESC;

74) Next, the company will try to give bonuses based on performance. Show all employees who've made more in sales than 1.5 times their salary. (You may use whatever technique you'd like to do the join: view, CTE, or even a subquery!)

WITH over\_achiever (revenue, salary, over\_salary, employee, o\_a\_id) AS

(SELECT t.total\_cost, e.salary, (e.salary \* 1.5) AS over\_total, CONCAT(e.firstname, ' ', e.lastname) AS fullname, e.id

FROM employees e

JOIN trans\_by\_employee t

ON e.id = t.employee\_id)

SELECT \* FROM over\_achiever

WHERE revenue > over\_salary

ORDER BY salary;

75) Do we have potentially erroneous rows? Find all transactions which occurred \_before\_ the employee was even hired! (Make sure each transaction only occupies one row).

SELECT t.orderdate, e.startdate, CONCAT(e.firstname, ' ', e.lastname) as fullname

FROM transactions t

JOIN employees e

ON e.id = t.employee\_id

GROUP BY t.orderdate, e.startdate, fullname

HAVING orderdate < startdate

ORDER BY fullname;

76) Among all transactions that occurred from 2015 to 2019, create a table that is the monthly revenue of our company versus the total trading volume of Yum! in that month. Format the columns nicely. That is, a sample row of your result might look like this:

```

| year | month | company\_revenue | yum\_trade\_volume |

|------|-------|-----------------|------------------|

| 2017 | 03 | $100,000 | 125,000,000 |

```

\* \_Hint:\_ You don't need any `WHERE` statements here. You can get the right answer simply by changing what kind of join you do!

SELECT y.year, y.month, to\_char(SUM(t.total\_cost), 'fm$999,999,999') AS company\_revenue, to\_char(y.tot\_volume, 'fm999,999,999') AS yum\_trade\_volume

FROM yum\_by\_month y

JOIN trans\_by\_month t

ON y.month = t.month

GROUP BY y.year, y.month, y.tot\_volume

ORDER BY y.year, y.month;

77) Repeat the previous problem, but in addition to the total volume, include:

\* The lowest price that month (ie, lowest low)

\* The highest price that month (ie, highest high)

SELECT EXTRACT(year from y.date) AS year, EXTRACT(month from y.date) AS month, MIN(y.low), MAX(y.high),

to\_char(SUM(t.unit\_price \* t.quantity), 'fm$999,999,999') AS revenue, to\_char(SUM(y.volume), 'fm999,999,999,999') AS volume

FROM yum y

JOIN transactions t

ON EXTRACT(month from y.date) = EXTRACT(month from t.orderdate) AND EXTRACT(year from y.date) = EXTRACT(year from t.orderdate)

GROUP BY EXTRACT(year from y.date), EXTRACT(month from y.date)

ORDER BY EXTRACT(year from y.date), EXTRACT(month from y.date);