7.3.5

Create Additional Lists

So far, we've completed several joins, become far more familiar with aliases, and we've also started incorporating other functions into our queries. This is awesome because we're able to generate summary data for Bobby's supervisor. All of those rows of data compacted into a few lines, and presented in descending order? That's really great work and we've accomplished a lot.

Bobby's supervisor has been really impressed with the work completed so far. He's even thought of a few more lists he'd like to see. These lists will require queries that filter and order the data and even join more than one table together.

With these lists, we'll continue to work with all of the SQL skills we've been developing and, like a puzzle, put them all together to build the exact queries we need to create the requested lists.

Because of the number of people leaving each department, the boss has requested three lists that are more specific:

1. **Employee Information:** A list of employees containing their unique employee number, their last name, first name, gender, and salary
2. **Management:** A list of managers for each department, including the department number, name, and the manager's employee number, last name, first name, and the starting and ending employment dates
3. **Department Retirees:** An updated current_emp list that includes everything it currently has, but also the employee's departments

Get started with the first list.

List 1: Employee Information

The first requested list is general employee information, but with their current salaries included. Let's look at our ERD again.

The Employees table has all of the information we need and uses the emp_no column as the primary key. The Salaries table has the additional information we need as well as the same primary key. The only problem is that the Employees table holds data for all employees, even the ones who are not retiring. If we use this table, we will have a far bigger list to present than expected.

To include all of the information Bobby's manager wants, we'll need to create an entirely new table from the beginning. Here's everything we need:

1. Employee number
2. First name
3. Last name
4. Gender
5. to_date
6. Salary

According to our ERD, the Salaries table has a to_date column in it. Let's make sure it aligns with the employment date or something else. Run a `SELECT` statement in the query editor to take a look.

```
SELECT * FROM salaries;
```

These dates are all over the place. We want to know what the most recent date on this list is, so let's sort that column in descending order. Back in the query editor, modify our select statement as follows:

```
SELECT * FROM salaries
ORDER BY to_date DESC;
```

	emp_no integer	salary integer	from_date date	to_date date
1	57279	84427	2000-02-01	2000-01-...
2	54420	66835	2000-02-01	2000-01-...
3	27628	40000	2000-02-01	2000-01-...
4	40179	56806	2000-02-01	2000-01-...
5	100364	40000	2000-02-01	2000-01-...

That looks a little better, but what's wrong with the date? It's certainly not the most recent date of employment, so it must have something to do with salaries. Looks like we'll need to pull employment dates from the dept_emp table again.

Now that we know what data we need from which tables, we can get started. It doesn't have to be from scratch, though. We've already created code to filter the Employees table to show only employes born and hired at the correct time, so let's look at our query editor to see if we can reuse that code.

```
SELECT emp_no, first_name, last_name
INTO retirement_info
FROM employees
WHERE (birth_date BETWEEN '1952-01-01' AND '1955-12-31')
AND (hire_date BETWEEN '1985-01-01' AND '1988-12-31');
```

This looks promising because the age and hiring filters are already in place, the only thing we're missing is the gender. Add that to our `SELECT` statement and also format it to fit within the best practices guidelines.

```
SELECT emp_no,
    first_name,
last_name,
    gender
```

We won't want to save this query into the same table we used before. Not only would it be confusing, but Postgres wouldn't allow it anyway. We'll want to update the `INTO` portion. The rest of the code looks good, as we want the same filters to be in place, so leave it as-is.

```
INTO emp_info
FROM employees
WHERE (birth_date BETWEEN '1952-01-01' AND '1955-12-31')
AND (hire_date BETWEEN '1985-01-01' AND '1988-12-31');
```

Now that our employees table has been filtered again and is being saved into a new temporary table (emp_info), we need to join it to the salaries table to add the to_date and Salary columns to our query. This will require a join, so let's get started. First, update the `SELECT` statement by adding the two columns we need from the Salaries table. Remember to use aliases to make it easier to read.

```
SELECT e.emp_no,
    e.first_name,
e.last_name,
    e.gender,
    s.salary,
    de.to_date
```

All columns are accounted for. We already know we're naming our new table emp_info, so we can leave the `INTO` statement as-is. Let's move on to the joins. In this case, we'll use inner joins in our query. This is because we want only the matching records. Back in the query editor, update the `INTO` and `FROM` lines, then add the first join directly below.

```
INTO emp_info
FROM employees as e
INNER JOIN salaries as s
ON (e.emp_no = s.emp_no)
```

Up to this point, we have updated and added code to:

- Select columns from three tables
- Create a new temp table
- Add aliases
- Join two of the three tables

Adding a third join seems tricky, but thankfully, the syntax is exactly the same. All we need to do is add the next join right under the first. Back in the query editor, add the following:

```
INNER JOIN dept_emp as de
ON (e.emp_no = de.emp_no)
```

Almost there! We have all of the joins, but we still need to make sure the filters are in place correctly. The birth and hire dates are still resting right under our joins, so update that with the proper aliases. For more on how to join two or more tables, refer to Joining More than Two Tables

(http://www.postgresqltutorial.com/postgresql-inner-join/) .

```
WHERE (e.birth_date BETWEEN '1952-01-01' AND '1955-12-31')
     AND (e.hire_date BETWEEN '1985-01-01' AND '1988-12-31')
```

Okay, now we have joined all three tables and have updated the birth and hire date filters to reference the correct table using an alias. We have one more filter to add, then we're ready to check and export the data.

The last filter we need is the to_date of 999-01-01 from the dept_emp table. To add another filter to our current `WHERE` clause, we will use `AND` again. In the query editor, add this last line:

```
     AND (de.to_date = '9999-01-01');
```

Before we create a temporary table using this code, comment out the `INTO` line so that we don't run it with the rest. This way, we'll be able to see the results of our code immediately. This is useful because if there is a mistake, we won't need to backtrack and delete the table.

Highlight the `INTO` emp_info line and press Command + forward slash, / (for Mac), or CTRL + forward slash, / (for Windows). This will automatically add the double hyphen to indicate a comment. Now highlight the entire block and run the code.

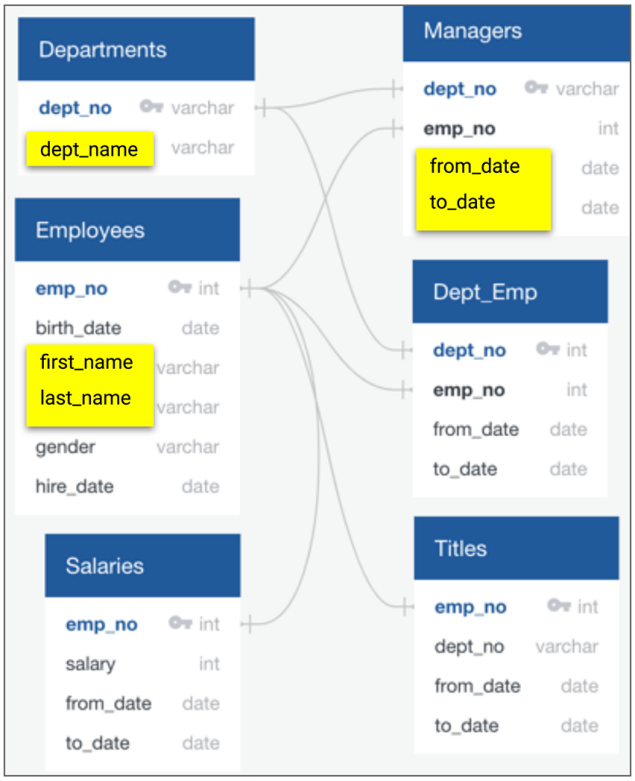
The results are looking good and the list contains everything Bobby's boss requested. Let's uncomment `INTO` and run the code again to save the temporary table. Remember to export this list as a CSV into your current project folder.

Those salaries still look a little strange, though. Bobby will need to ask his manager about the lack of employee raises.

List 2: Management

The next list to work on involves the management side of the business. Many employees retiring are part of the management team, and these positions require training, so Bobby is creating this list to reflect the upcoming departures.

This list includes the manager's employee number, first name, last name, and their starting and ending employment dates. Look at the ERD again and see where the data we need resides.



We can see that the information we need is in three tables: Departments, Managers, and Employees. Remember, we're still using our filtered Employees table, current_emp, for this query.

Let's do this one together. At the bottom of the query editor, type the following:

```
-- List of managers per department
SELECT  dm.dept_no,
        d.dept_name,
        dm.emp_no,
        ce.last_name,
        ce.first_name,
        dm.from_date,
        dm.to_date
INTO manager_info
FROM dept_manager AS dm
    INNER JOIN departments AS d
        ON (dm.dept_no = d.dept_no)
    INNER JOIN current_emp AS ce
        ON (dm.emp_no = ce.emp_no);
```

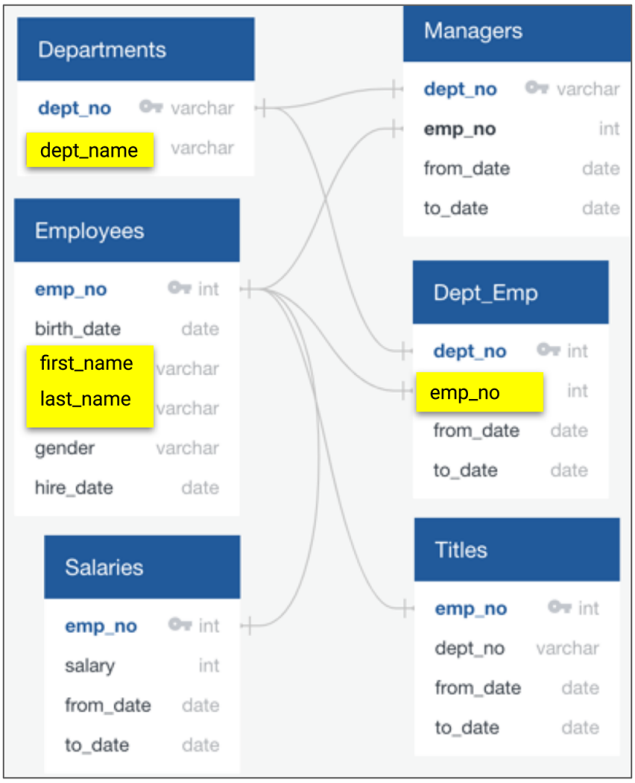
This is pretty similar to the last set of joins we completed. Like last time, comment out the `INTO` line before we run the code for the first time.

	dept_no character varying (4)	**dept_name** character varying (40)	**emp_no** integer	**last_name** character varying	**first_name** character varying	**from_date** date	**to_date** date
1	d003	Human Resources	110183	Ossenbruggen	Shirish	1985-01-01	1992-03-...
2	d004	Production	110386	Kieras	Shem	1992--08-02	1996-08-...
3	d007	Sales	111133	Zhang	Hauke	1991-03-07	9999-01-...
4	d008	Research	111534	Kambil	Hilary	1991-04-08	9999-01-...
5	d009	Customer Service	111692	Butterworth	Tonny	1985-01-01	1988-10-...

The result of this query looks even more strange than the salaries. How can only five departments have active managers? This is another question Bobby will need to ask his manager.

List 3: Department Retirees

The final list needs only to have the departments added to the current_emp table. We've already consolidated most of the information into one table, but let's look at the department names and numbers we'll need.



The Dept_Emp and Departments tables each have a portion of the data we'll need, so we'll need to perform two more joins in the next query.

We'll use inner joins on the current_emp, departments, and dept_emp to include the list of columns we'll need to present to Bobby's manager:

1. emp_no
2. first_name
3. last_name
4. dept_name

In the query editor, begin with the `SELECT` statement. Type the following:

```
SELECT ce.emp_no,
ce.first_name,
ce.last_name,
d.dept_name
```

Notice we have only selected four columns from two tables, yet there are three tables in the ERD that we need. That's because we don't need to see a column from each table in a join, but we do need the foreign and primary keys to link them together. Continue with the `INTO` statement, this time naming the temporary table dept_info.

```
-- INTO dept_info
```

We should go ahead and comment it out now because we know we'll want to test the code before saving it as a table.

Next, start defining the aliases with `FROM` and the joins. In the query editor, type the following:

```
FROM current_emp as ce
INNER JOIN dept_emp AS de
ON (ce.emp_no = de.emp_no)
INNER JOIN departments AS d
ON (de.dept_no = d.dept_no);
```

After executing the code and checking the results, a few folks are are appearing twice. Maybe they moved departments? It's interesting how each list has given Bobby a question to ask his manager. So far, Bobby would like to know the following:

1. What's going on with the salaries?
2. Why are there only five active managers for nine departments?
3. Why are some employees appearing twice?

To help Bobby find these answers, we're going to create tailored lists.