Write queries for the following 3 problems in SQL, in Postgres. Submit your work in a file titled `morequeries.sql`. I have intentionally not supplied complete data for these problems, as your queries should work with any data that might occupy these tables and is consistent with the description presented. You should create your own data to comprehensively test these problems. Your `morequeries.sql` should have only the queries; your test data should be created elsewhere, which you do not need to submit.

Note carefully: This is a pair assignment. If you are working in a pair, this must be done with the two of you working side-by-side. Take turns typing; perhaps for each query you might swap the keyboard back and forth, or perhaps you might set a timer and swap every 15 minutes. Do *not* work on queries separately from each other and combine at the end.

You'll likely find it convenient to use views to help make your SQL easier to read.

Note for the last problem I've got some hints on the very last page; don't look at them without first reading the directions regarding them at the end of Problem 3.

Problem 1

Suppose you're trying to set up a telephone directory, and you have the following tables:

```
create table Employees (
    empid       integer primary key,
    firstname   varchar(20) not null,
    lastname    varchar (20) not null
);

create table Phones (
    empid       integer not null references Employees,
    phonetype   char(4) not null
      check     (phonetype in ('home','cell')),
    phonenumber char(10) not null,
    primary key (empid, phonetype)
);
```

We're assuming that each person can have a maximun of one phone number of each type, as illustrated by the primary key in Phones.

Write a query to produce a list of employees, one person per line, with one column for the land line phone number, and one column for the cell phone number. Use a NULL value if the employee is missing a phone number.

Problem 2

You've been hired by an auto dealership to work on their database system. You're hoping to get a car for yourself out of the arrangement, but time will tell. The database contains two tables: The first is a calendar of promotional events (e.g., "FREE FINANCING, NO MONEY DOWN!") that the dealership has had, which looks like:

```
create table Promotions (
    promo      varchar(35) not null primary key,
    startdate  date not null,
    enddate    date not null
);
```

The other table is a list of the sales that were made during the promotions. This table looks like:

```
create table Sales (
    salesperson      varchar(25) not null,
    saledate         date not null,
    amount           numeric not null
);
```

Write a query to determine which salesperson had the highest total amount of sales for each promotion. The output of your query should have one row for each promotion, and list the name of the promo, the sales person name, and the amount. In the event of a tie, all winning salespeople should be shown. If no one has any sales during a promotion, show it on the list, but with null values for the salesperson and amount. Your results should be sorted by promo name.

No two promotions will overlap with each other. (**This clarification was added on 4/18.**)

Problem 3

At a shopping mall drop-in babysitting center, parents drop off babies so that they can go shopping. Each baby is assigned to a particular babysitter for a block of time, as in the following table:

```
create table Sessions (
    baby            varchar(25) not null primary key,   -- baby name
    sitter          varchar(25) not null,               -- name of babysitter
    start_time      time not null,                      -- time of session start
    end_time        time not null                       -- time of session end
);
```

Here is some sample data:

baby	sitter	start_time	end_time
Alice	Rory	08:00	11:00
Ben	Rory	09:00	13:00
Cara	Amelia	09:00	15:30
Darren	Amelia	08:00	10:00
Eustace	Amelia	10:01	11:30
Merlin	Amelia	09:30	10:30

Notice that session times with a single babysitter overlap. Babysitters multitask.

As part of a new initiative in babysitting transparency, the center needs to share with parents when they pick up their babies how bad the multitasking got on the watch for their baby. Specifically, you wish to determine for each session the maximum number of simultaneous sessions (including the session itself) that the babysitter managed. For example, look at Alice. Babysitter Rory is only handling Alice for her first hour (so that's 1 overlapping session), but for the second hour of Alice's session, Rory is simultaneously also handling Ben. Therefore, the maximum number of simultaneous sessions during Alice's session is 2. Alternatively, consider baby Cara. Babysitter Amelia simultaneously handles babies Darren and Eustace, each of which overlap Cara, but Darren and Eustace don't occur at the same time. Baby Merlin, however, does occur at the same time as Cara, and Darren or Eustace (but not both). So the maximum number of simultaneous sessions for Cara's visit is 3.

You may assume that for any particular babysitter, no time appears more than once. In other words, I'm sparing you from having to worry about the detailed case where two babies start at 10:00, or one baby starts at 9:00 and another baby stops at 9:00. We can imagine that there is a punch clock being used that ensures that no time is recorded more than once. (**This paragraph was updated on 4/18 to simplify the case of potential overlapping times.**)

Write a query that produces a list of baby names, and the maximum number of overlapping sessions for the babysitter during that session. (You can assume that no two babysitters have the same name. You can also assume that no two babies have the same name.)

This is a pretty challenging query to write. **There are hints following after a blank page, but my recommendation is not to look at them unless you're really stuck.** Indicate with a comment in your submission if you used the hints. You'll lose no points for using the hints, but

I'm trying to impose a small penalty of guilt to encourage you to think about it first to see if you can do it on your own.

Here are some hints, that are still intended to be cryptic:

1. Create a view that, for every baby session, has a row for for the start time, and another row for the end time, for every other baby session including itself that overlaps it in some way. Think of each of these starts and stops as an "event." Add a variable called event_count that is either +1 or -1. +1 means that an overlapping event has started; -1 means that an overlapping event has ended.
2. Create another view that takes the above view, and adds a column containing a running sum of event_count, from the start of the first overlapping event to the current one. SQL doesn't really have a notion of tuple order, so you'll have to do this with inequalities relating to times.
3. Write a query that runs on the view from the previous step to pull the maximum running total for each event.