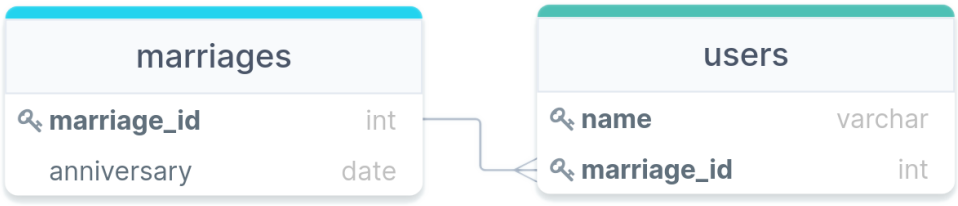
Topic Overview

The final is comprehensive, but will certainly include material from more recent chapters that you have not yet been tested on. As such, all previous study guides will still apply, and I'll just include the latest information here.

- Chapter 12
  - ☐ Extracting pieces of a timestamp, date, or time object
  - ☐ Creating date, time, or timestamp objects
  - ☐ Getting local system times or dates
  - ☐ Being able to convert a timestamp to another timezone
- Window Functions (partly Chapter 11)
  - ☐ Creating window functions using `OVER()`
  - ☐ Understanding basic special window functions like `RANK`/`DENSE_RANK` and `LAG`/`LEAD`
  - ☐ How ordering changes the default window
  - ☐ How partitioning works within a window
- Chapter 13
  - ☐ Using subqueries to generate a single value that can be compared against
  - ☐ Creating derived tables for use in other queries
  - ☐ Filtering tables based on the output of a subquery
  - ☐ Using subqueries in `SELECT` statements
  - ☐ Using derived tables within CTEs
  - ☐ Creating pivot tables or cross-tabulations
  - ☐ Using `CASE` statements
- Chapter 14
  - ☐ Using basic string manipulation functions
  - ☐ Matching patterns using regular expressions (regular expressions will be kept small: < 10 characters)
  - ☐ Extracting text from larger strings using regular expressions
  - ☐ Splitting text into arrays or (more likely) rows
  - ☐ Vectorizing a large block of text
  - ☐ Searching `tsvectors` for specific `tsquerys`
  - ☐ Constructing compound `tsquerys`
  - ☐ Creating indexes for `tsvector`-type columns
- Chapter 15
  - ☐ Creating points and lines using well-known text strings
  - ☐ Understand how to specify the used coordinate system
  - ☐ Compare and contrast the `geography` and `geometry` data types and known when to use each
  - ☐ Create `geography` or `geometry` objects corresponding to points or lines and a particular SRID
  - ☐ Creating indexes for spatial information
  - ☐ Compute distances between spatial objects

Practice Questions

1. Suppose you had the following tables containing information about individuals and their marriages:



Each unique marriage between two individuals gets a single marriage id and anniversary date, and the users table tracks the individuals name and the corresponding marriage id. Note that the users table has a compound primary key, and so an individual name can appear multiple times in the users table. This allows for the same user to be married multiple times, in case of divorce, death, etc. You can assume that no users otherwise have duplicate names: that is, if a name appears twice in the users table, it is because that individual has remarried.

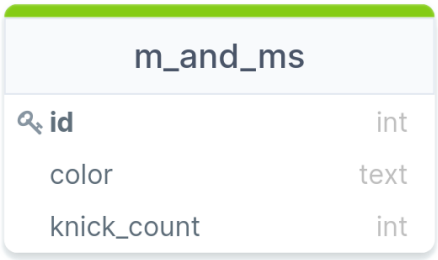
Write a query that would return to you all users who are celebrating the anniversary of their *latest* marriage **today** (whatever day the query happens to be run).

Solution: I'd generally see this as two questions: how can I match a given day of the year, and how can I determine the last marriage of an individual. For the first, I know I can use `date_part` to extract just the day and month to compare. For the second, I'm going to need a query that does some grouping by individual (to get all their possible marriages) and then selects the latest. Frequently, I find it useful to treat one of these problems as a CTE subquery. Since I need to know the marriage situation before doing any matching, I'll find the latest marriages as part of a CTE subquery and then do the matching in the main query. So one solution might look something like this:

```
WITH
current_marriages (name, anniversary) AS (
  SELECT
    u.name,
    max(m.anniversary) -- gets the latest from the group
  FROM users AS u
  JOIN marriages AS m
    ON m.marriage_id = u.marriage_id
  GROUP BY u.name
)

SELECT name
FROM current_marriages
WHERE date_part('day', anniversary) = date_part('day', now()) AND
      date_part('month', anniversary) = date_part('month', now())
;
```

2. I am testing the structural integrity of the color coating on the outside of M&M's. To do so, I drop each from a height of 1 meter and then inspect them for knicks or cracks in the coating. I'm keeping track of all this information in a simple table that has the form:



where the colors would be the classic M&M colors: red, orange, green, blue, yellow, and brown. Each row of the table hold information about another dropped M&M.

I'm interested in seeing if certainly colors of M&M's have a coating that is more resistant to damage. I will call any M&M's with 1 or fewer knicks as "undamaged" and any with more than 1 knick as "damaged". Write a query that would construct a pivot table that contains the counts of tested M&M's that fall into different categories, where the colors of the M&M's are across the horizontal axis and condition of the M&M (damaged or undamaged) is down the vertical axis. So a possible output might look like:

Condition	blue	brown	green	orange	red	yellow
undamaged	10	14	7	34	20	17
damaged	14	8	12	22	10	24

Solution: I know I will need a cross tabulation here, so I'll need to figure out my three subqueries. The first subquery will need to give me a condition column, a color colum, and then the counts. Most likely, this means I need to group based on condition and color, which raises a potential issue: I need to have done my cases to sort `knick_count` into damage or undamage groups before that grouping happens. I *could* just put the case statement directly in the `GROUP BY` but I find it a bit more clear to just use a CTE to precompute it. Keep in mind though that that CTE must go *inside* the first subquery string.

```
SELECT *
FROM crosstab(

  WITH
    -- Just getting a color and condition table
    mm_condition (color, condition) AS (
      SELECT
        color,
        CASE
          WHEN knick_count <= 1 THEN 'undamaged' ELSE 'damaged'
        END
      FROM m_and_ms
    )
```

```
  -- My three column table is easy now
  SELECT
    condition,
    color,
    COUNT(*) as m_and_m_count
  FROM mm_condition
  GROUP BY condition, color
  ORDER BY condition, color -- don't forget to sort!
,

  -- No need for a CTE here, as I can get colors directly
  SELECT DISTINCT color
  FROM m_and_ms
  ORDER BY color

) AS (
  condition text,
  -- Colors are sorted alphabetically, so hence my below ordering
  blue int,
  brown int,
  green int,
  orange int,
  red int,
  yellow int
);
```

3. You are doing analysis on criminal behavior in the city of Salem. You have managed to get your hands on a pile of crime reports, and you have imported them into a table (called `crime_reports`) in a raw format, where each line of the table contains information from a single report, but the entirety of each criminal report exists as text within a single column (called `raw_text`). The text of the criminal report has a varying format, but contains information describing the criminal activity, who reported the crime, who responded to the call, and where the crime occurred (in a consistent (latitude, longitude) format).

   Your task is to determine, for crimes involving murder or homicide but *not* robbery, what the average distance from the police headquarters (located at (44.947138680439984, -123.03630819427937)) was. You can use multiple queries if you like.

   This is definitely more complicated than anything that would show up on the test, but is good practice bringing in ideas from both full text searching and geospatial information. If you can reason your way through this, you should be in a great place for any test questions about those concepts.

   Solution: Again, approach these by identifying the smaller problems that you will need to solve along the way. Here, I'm going to need some way of filtering based on some slightly complicated search query, which makes me think of `ts_vectors`. Once I have filtered, I'm going to need to find the actual location in the latitude, longitude form, which I could probably do with some kind of regex. And then in the end I'll need to work with PostGIS to take those locations and turn them into geographic objects so that I could compute distances (and thus an average distance). So implementing those things, I arrived at something like this:

```
WITH
-- CTE1: extract coordinates from desired reports
event_coordinates (latitude, longitude) AS (
  SELECT
    -- number with a decimal that follows a ( and goes to a ,
    (regexp_match(raw_text, '\((\d*\.\d*),'))[1]::real as latitude,
    -- possible neg number with a decimal that follows a , and goes to a )
    (regexp_match(raw_text, ',\s*(-?\d*\.\d*)\)'))[1]::real as longitude
  FROM crime_reports
  -- Just making the tsvector on the fly
  WHERE to_tsvector(raw_text) @@ to_tsquery(
    'murder | homicide & !robbery'
  )
),
-- CTE2: compute coordinate geographies
event_locations (geog_pt) AS (
  SELECT
    ST_SetSRID(ST_MakePoint(longitude, latitude), 4326)::geography
  FROM event_coordinates
)

SELECT
  avg(ST_Distance(
    geog_pt,
    ST_SetSRID(
      ST_MakePoint(-123.03630819427937, 44.947138680439984),
      4326)::geography
    )
  ) as avg_distance_in_meters
FROM event_locations ;
```