Appendix D

Example Exams

This textbook is used for a variety of different courses and course formats. The exams included in this appendix reflect this diversity. In order to study from these exams, keep in mind that the amount of time given and material covered may be different.

1 2023 CAPP Databases Final: A

This exam was given to Master's level students in the University of Chicago CAPP program in the Spring of 2023. There were two versions of the exam – this is version "A".

The following tables contains information about Uber Eats drivers, their deliveries and reviews. Keep in mind that (a) not all deliveries will have reviews. A delivery can have, at most, one review. Not all drivers will have deliveries. When a driver first signs up they will not have any deliveries or reviews

- Only use syntax covered in class. Do not create any views.
- Interpret all inequalities as strict unless explicitly stated.
- If there is no specified return format (DataFrame/Series/etc.) than any format will be accepted.
- If you provide more than one answer, the lower of the two scores will be counted.
- Any two columns with the same name can be assumed to match.
- Columns in the *drivers* table / DataFrame:
  - **driver_id:** The ID of the driver (INT, UNIQUE, NOT NULL).
  - **state:** The state that the driver lives in (STRING, NOT NULL).
  - **age:** The age of the driver in years (INT, NOT NULL).
- Columns in the *reviews* table / DataFrame:
  - **del_id:** The ID of the delivery being reviewed (INT, NOT NULL).
  - **review:** The review score on a 1 (worst) to 5 (best) scale (INT, NOT NULL).
- Columns in the *delivery* table / DataFrame:
  - **del_id:** The unique ID associated with the delivery (INT, UNIQUE, NOT NULL).
  - **del_date:** The date that the delivery occurred (DATE, NOT NULL).
  - **driver_id:** The ID of the driver (INT, NOT NULL).
  - **car:** A flag (1/0) for if the driver used a car to make the delivery (INT, NOT NULL).
  - **length:** The distance that the delivery took, in km (FLOAT, NOT NULL).

driver_id	state	age
1	CA	27
2	MN	37
3	CA	28
4	CA	32

Drivers (1,234 Rows)

del_id	review
1	5
23	4
35	4
45	1

Reviews (980 Rows)

del_id	del_date	driver_id	car	length
1	1-1-2012	45	0	1.25
2	12-23-2012	45	1	23.45
3	7-6-2013	112	1	11.17
4	5-5-2014	1125	0	.75

Deliveries (14,365 Rows)

SQL Section

1. Write a query which returns the 7 oldest drivers (*driver_id* only) from the state of Michigan ("MI").

```
select driver_id
from drivers
where state = 'MI'
order by age desc
limit 7
```

2. Write a query which returns three columns: (1) the state, (2) the total number of *drivers* (count) from that state (only including drivers who have one or more deliveries) and (3) the total number of *deliveries* (count) from that state. This should return one row per state.

```
select
        drivers.state
        , count( distinct drivers.driver_id)
        , count( deliveries.del_id)
from
        drivers
join
        deliveries
using( driver_id)
group by state
```

3. Write a query which returns one row per delivery and four columns. The first column should be the state of the driver, the second should be the *driver_id*, the third should be *del_id* and the fourth should be the total length that the driver has travelled up to and including that delivery (cumulative sum of *length*). Make sure that the cumulative sum is calculated by the date of the delivery from earliest to latest. If a driver does not have any deliveries they should *not* be included in the results.

```
select
        driver_id
        , del_id
        , state
        , sum( length) over(
                partition by driver_id
                order by del_dt asc
                rows between unbounded preceding and current row
                ) as cum_sum
from
        drivers
join
        deliveries
using( driver_id )
```

4. Write a query which returns two rows and two columns. The first column should be state and the second should be the number of 5-star reviews for deliveries from that state. Only include Michigan ("MI") and Pennsylvania ("PA").

```
select
        state,
        count( case when review = 5 then 1 else null end ) as num_five_stars
from
        (select * from drivers where state in ('MI', 'PA')) as lhs
left join
        deliveries
        using( driver_id)
left join
        review
        using( del_id)
group by state
```

5. We want to return the average length of deliveries depending on if the driver used a car or not. Write a query which returns two rows and two columns. The first column should be if the delivery person used a car or not (the *car* column 1/0 flag) and the second column should be the average length of a delivery with that particular flag value. Only include those observations from the year 2012.

```
select
        car,
        avg( length)
from
        deliveries
where
        date_part('year', del_date) = 2012
group by 1
```

6. Please return one row and two columns. The first column should be the average length of deliveries when a car is used ($car = 1$) and the second column should be the average length of a delivery when a car is not used ($car = 0$). We want to calculate this on all deliveries from February in any year. Note that this is similar to the last problem, but the data shape and date filters are different.

```
select
        avg( case when car = 0 then length else null end) as car_0_avg
        , avg( case when car = 1 then length else null end) as car_1_avg
from
        deliveries
where
        date_part('month', del_date) = 2;
```

7. What state (*state* only) has the most drivers?

```
select state
from drivers
group by 1
order by count(1) desc
limit 1;
```

8. What was the longest (by *length*) non-car ($car = 0$) delivery (*del_id* only)?

```
select
        del_id
from
        deliveries
where car = 0
order by length desc
limit 1;
```

9. We call drivers who have ever done a delivery of more than 60 km a "long-driver". What is the average review score for "long-drivers"? This should include *all* deliveries from "long-drivers", even those less than 60 km. This should return only a single row and column.

```
select
        avg( review )
from
        deliveries
left join
        reviews
using(del_id)
where
        driver_id in (select distinct driver_id from deliveries where length > 60)
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that DataFrames named *drivers*, *deliveries* and *reviews* are already loaded. Unless otherwise specified you may return either a Series or DataFrame.

1. Return a DataFrame which has two columns and a row for each state. The first column should be the state and the second should be the number of deliveries completed by drivers from that state.

```
pd.merge( drivers, deliveries, on='did', how='left')
        .groupby('state', as_index=False)
        .agg({'del_id' : ['count']})

-- Since this is number of jobs it could be inner or left or outer join
```

2. Return a DataFrame with two rows and two columns. The first column should be a flag which takes one of two values: "LT10" and "MT15". The second column should be the average review for deliveries which are (strictly) "Less Than 10 km" and "More Than 15 km" , respectively. In other words, one row should contain "LT10" and the average review for deliveries which are less than 10 km and the other row should contain "MT15" and the average review for deliveries which are more than 15 km.

```
mrg = pd.merge( deliveries, reviews, on='del_id', how='inner')

mrg = (mrg
       .loc[(mrg.loc[:, 'length'] < 10) | (mrg.loc[:, 'length'] > 15), :]
      )

mrg.loc[:, 'flag'] = 'LT10'
mrg.loc[(mrg.loc[:, 'length'] >15), 'flag'] = 'MT15'

mrg.groupby('flag', as_index=False).agg({'review' : ['mean']})
```

3. We call drivers who have ever done a delivery of more than 60 km a "long-driver". What is the average review score for "long-drivers"? This should include *all* deliveries from "long-drivers", even those less than 60 km. This should return a single value (can be in a DataFrame, in a Series or as a number).

```
driver_id_lst = deliveries.loc[(deliveries.loc[:, 'length'] > 60), 'driver_id'].drop_duplicates()

mrg = pd.merge( reviews, deliveries, on='del_id', how='inner')
mrg.loc[ (mrg.loc[:, 'driver_id'].isin( driver_id_lst), 'review'].mean()
```

4. What was the longest (by length) non-car ($car = 0$) delivery (del_id only)?

```
deliveries.loc[ (deliveries.loc[:, 'car'] == 0), :].nlargest(1,'length').loc[:, 'del_id']

OR

(deliveries
        .loc[ (deliveries.loc[:, 'car'] == 0), :]
        .sort_values('length', ascending=False)
        .loc[:, 'del_id']
)
```

5. Which year had the largest number of deliveries (count)?

```
(deliveries
        .assign(yr = deliveries.loc[:, 'del_date'].dt.year)
        .groupby( yr, as_index=False)
        .agg( {'del_id' : ['count']})
        .nlargest(1, ('del_id', 'count'))
        .loc[:, 'yr']
)
```

6. How many drivers are from California ("CA")?

```
drivers.loc[ (drivers.loc[:, 'state'] == 'CA'), :].shape[0]

Lots of ways to do this one.
```

7. Which date (*del_date*) had the largest number of 3-star deliveries (count)?

```
mrg = pd.merge( deliveries, reviews, how='left', on='del_id')

(mrg
        .loc[ (mrg.loc[:, 'review'] == 3), :]
        .groupby( 'del_date', as_index=False)
        .agg( {'del_id' : ['count']})
        .nlargest(1, ('del_id', 'count'))
        .loc[:, 'del_date']
)
```

8. Return all drivers (*driver_id* only) from Texas ("TX") who are 32 years old as a *DataFrame*.

```
drivers.loc[ (drivers.loc[:, 'state'] == 'CA') & (drivers.loc[:, 'age'] == 32), ['driver_id']]
```

2 2023 CAPP Databases Final: B

This exam was given to Master's level students in the University of Chicago CAPP program in the Spring of 2023. There were two versions of the exam – this is version "B".

The following tables contains information about a large painting company. The company has many painters who do jobs and then get reviews on those jobs. A job can, at most, have **one** review. Not all painters will have jobs (when they first start there is some time before they are assigned a job). A painter can have multiple jobs as most jobs last only a day or two and every job will be in the database.

- Only use syntax covered in class. Do not create any views.
- Interpret all inequalities as strict unless explicitly stated.
- If there is no specified return format (DataFrame/Series/etc.) than any format will be accepted.
- If you provide more than one answer, the lower of the two scores will be counted.
- Any two columns with the same name can be assumed to match.
- Columns in the *painters* table / DataFrame:
  - **painter_id:** The ID of the painter (INT, UNIQUE, NOT NULL).
  - **state:** The state that the painter works in (STRING, NOT NULL).
  - **work_exp:** The number of years of work experience (INT, NOT NULL).
- Columns in the *jobs* table / DataFrame:
  - **job_id:** The unique ID associated with the job (INT, UNIQUE, NOT NULL).
  - **job_date:** The date that the job occurred (DATE, NOT NULL).
  - **painter_id:** The ID of the painter (INT, NOT NULL).
  - **sprayer:** A flag (1/0) for if the painter used a sprayer or not (INT, NOT NULL).
  - **paint:** The amount of paint used, in gallons (FLOAT, NOT NULL).
- Columns in the *reviews* table / DataFrame:
  - **job_id:** The ID of the job being reviewed (INT, NOT NULL).
  - **review:** The review score on a 1 (worst) to 5 (best) scale (INT, NOT NULL)

painter_id	state	work_exp
1	CA	0
2	MN	12
3	CA	4
4	CA	11

Painters (1,234 Rows)

job_id	job_date	painter_id	sprayer	paint
1	1-1-2012	45	0	1.25
2	12-23-2012	45	1	23.5
3	7-6-2013	112	1	11.25
4	5-5-2014	1125	0	.75

Jobs (14,365 Rows)

job_id	review
1	5
23	4
35	4
45	1

Reviews (980 Rows)

SQL Section

1. Write a query which returns the 11 painters (*painter_id*) with the most experience (largest *work_exp*) from Hawaii ("HI").

```
select painter_id
from painters
where state = 'HI'
order by work_exp desc
limit 11
```

2. Write a query which returns three columns: (1) the state, (2) the total number of *painters* (count) from that state (only including painters who have one or more jobs) and (3) the total number of *jobs* (count) from that state. This should return one row per state.

```
select
        state
        , count( distinct painters.painter_id)
        , count( jobs.job_id)
from
        painters
join
        jobs
using( painter_id )
group by state
```

3. Write a query which returns one row per job and four columns. The first column should be the state of the painter, the second should be the *painter_id*, the third should be *job_id* and the fourth should be the total amount of paint that the painter has used up to and including that job (cumulative sum of *paint*). Make sure that the cumulative sum is calculated by the date of the job from earliest to latest. If a painter does not have any jobs they should *not* be included in the results.

```
select
        painter_id
        , job_id
        , state
        , sum( paint ) over(
                partition by painter_id
                order by job_dt asc
                rows between unbounded preceding and current row
                ) as cum_sum
from
        painters
join
        jobs
using( painter_id )
```

4. Write a query which returns two rows and two columns. The first column should be state and the second should be the number of 5-star reviews for jobs from that state. Only include Alaska ("AK") and Hawaii ("HI").

```
select
        state,
        count( case when review = 5 then 1 else null end ) as num_five_stars
from
        (select * from painters where state in ('AK', 'HI')) as lhs
left join
        jobs
        using( painter_id)
left join
        review
        using( job_id)
group by state;
```

5. We want to return the average amount of paint used depending on if the painter used a sprayer or not. Write a query which return two rows and two columns. The first should be if the painter used a sprayer or not (the *sprayer* column 1/0 flag) and the second should be the average amount of paint used. Only include those observations from the year 2014.

```
select
        spray,
        avg( paint )
from
        jobs
where
        date_part('year', del_date) = 2014
group by 1
```

6. Please return one row with two columns. The first column should be the average amount of paint used when a sprayer is used ($sprayer = 1$) and the second column should be the average amount of paint used if a sprayer is not used ($sprayer = 0$). We want to calculate this on all jobs from July in any year. Note that this is similar to the last problem, but the data shape and date filters are different.

```
select
        avg( case when sprayer = 0 then paint else null end) as spray_0_avg
        , avg( case when sprayer = 1 then paint else null end) as spray_1_avg
from
        jobs
where
        date_part('month', del_date) = 7;
```

7. What state (state only) has the most painters?

```
select state
from painters
group by 1
order by count(1) desc
limit 1;
```

8. What was the largest (used the most paint) non-sprayer ($sprayer = 0$) paint job? Return the *job_id* only.

```
select
        job_id
from
        jobs
where sprayer = 0
order by paint desc
limit 1;
```

9. We call painters who have ever done a job of more than 25 gallons "large-scale" painters. What is the average review for "large-scale" painters? This should include *all* jobs from "large-scale" painters, even those jobs which are less 25 gallons. This should return only a single row and column.

```
select
        avg( review )
from
        jobs
left join
        reviews
using(job_id)
where
        painter_id in (select distinct painter_id from jobs where paint > 25)
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that DataFrames named *painters*, *jobs* and *reviews* are already loaded. Unless otherwise specified you may return either a Series or DataFrame.

1. Return a DataFrame which has two columns and a row for each state. The first column should be the state and the second should be the number of jobs completed by painters from that state.

```
pd.merge( painters, jobs, on='painter_id', how='left')
        .groupby('state', as_index=False)
        .agg({'job_id' : ['count']})

-- Since this is number of jobs it could be inner or left or outer join
```

2. Return a DataFrame with two rows and two columns. The first column should be a flag which takes one of two values: "less3" and "greater20". The second column should be the average review for jobs which are (strictly) "less than 3 gallons" and "greater than 20 gallons" , respectively. In other words, one row should contain "Less3" and the average review for jobs which are less than 3 gallons and the other row should contain "greater20" and the average review for jobs which are more than 20 gallons.

```
mrg = pd.merge( jobs, reviews, on='job_id', how='inner')

mrg = (mrg
       .loc[(mrg.loc[:, 'paint'] < 3) | (mrg.loc[:, 'length'] > 20), :]
      )

mrg.loc[:, 'flag'] = 'less3'
mrg.loc[(mrg.loc[:, 'paint'] > 20), 'flag'] = 'greater20'

mrg.groupby('flag', as_index=False).agg({'review' : ['mean']})
```

3. We call painters who have ever done a job of more than 25 gallons "large-scale" painters. What is the average review for "large-scale painters? This should include *all* jobs from "large-scale" painters, even those jobs which are less 25 gallons. This should return a single value (can be in a DataFrame, in a Series or as a number).

```
painter_id_list = jobs.loc[(jobs.loc[:, 'paint'] > 25), 'driver_id'].drop_duplicates()

mrg = pd.merge( reviews, jobs, on='driver_id', how='inner')
mrg.loc[ (mrg.loc[:, 'painter_id'].isin( painter_id_list), 'review'].mean()
```

4. What was the largest (most paint used) non-spray (*sprayer* = 0) job (job_id only)?

```
jobs.loc[ (jobs.loc[:, 'sprayer'] == 0), :].nlargest(1,'paint').loc[:, 'job_id']

OR

(job
        .loc[ (job.loc[:, 'sprayer'] == 0), :]
        .sort_values('paint', ascending=False)
        .loc[:, 'job_id']
)
```

5. Which year had the largest number of jobs (count)?

```
(job
        .assign(yr = job.loc[:, 'job_date'].dt.year)
        .groupby( yr, as_index=False)
        .agg( {'job_id' : ['count']})
        .nlargest(1, ('job_id', 'count'))
        .loc[:, 'yr']
)
```

6. How many painters are from New Mexico ("NM")?

```
painters.loc[ (painters.loc[:, 'state'] == 'CA'), :].shape[0]

Lots of ways to do this one.
```

7. Which date (*job_date*) had the largest number of 2-star jobs (count)?

```
mrg = pd.merge( jobs, reviews, how='left', on='job_id')

(mrg
        .loc[ (mrg.loc[:, 'review'] == 2), :]
        .groupby( 'job_date', as_index=False)
        .agg( {'job_id' : ['count']})
        .nlargest(1, ('job_id', 'count'))
        .loc[:, 'job_date']
)
```

8. Return all painters (*painter_id* only) from California ("CA") who have 12 years of experience as a *DataFrame*.

```
painters.loc[ (painters.loc[:, 'state'] == 'CA') & (painters.loc[:, 'age'] == 32), ['painter_id']]
```

3 2023 CAPP Databases Midterm: A

This exam was given to Master's level students in the University of Chicago CAPP program in the Spring of 2023. There were two versions of the exam – this is version "A".

The following table contains information about campers at a camp. You can assume each name uniquely defines a camper and that a camper only appears once in the table.

- **name:** The name of the camper (string, NOT NULL).
- **age:** The camper's age, in years (integer, NOT NULL).
- **hgt:** The height of the camper (in centimeters) (float, NOT NULL).
- **state:** The state that the camper is from (string, NOT NULL).
- **program:** The specific program (elective) that the camper choose. You can assume this is all lower case (string, NOT NULL).
- **amt_paid:** The amount they paid to attend camp (float, NOT NULL).
- **allergy:** If a camper has a food allergy (if Null that means no allergy). You can assume this is all lower case. There will be only a SINGLE allergy listed (string, HAS NULLS).
- The name of the table / DataFrame is **camper**. No need to use a schema or load the DataFrame.
- Only use syntax covered in class.
- Interpret all inequalities as strict unless explicitly stated.

Figure D.1: *camper* Table: 12,345 Rows

name	age	hgt	state	program	amt_paid	allergy
Ringly Roberson	7	121.0	NY	basketball	987.54	
Crash Bandicoot	11	144.5	MS	basket-weaving	1128.75	peanuts
Alligator Reynolds	8	129.0	PA	skateboarding	585.46	

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Using SQL, write a query which returns the names (name only) of the top-8 shortest campers who are 10 years old.

```
SELECT
        name
FROM
        campers
where age = 10
order by hgt asc limit 8;
```

2. Using SQL, write a query which returns all campers (name only) who are younger than 10 years old and are either from New Jersey ('NJ') or Wyoming ('WY'). Only include those campers who do NOT have a food allergy.

```
select name from campers where
        state in ('NY', 'AL')
        and age < 10
        and allergy is null;
```

3. Write an SQL query which returns the number of campers from each state who have food allergies. This should be two columns: one with the state and the other with the number of campers from that state who have food allergies.

```
SELECT
        state, count(1) as ct
from
        campers
where injury is not null
group by 1;
```

4. Write an SQL query which returns all rows and columns for campers who are taking "basketball" as their program (you can assume that all programs are lowercase). Sort them from tallest to shortest.

```
select * from campers
where program = 'basketball'
order by hgt desc;
```

5. We define an "allergy impacted" program as one with 10% (or more) of the campers in that program having a food allergy or any kind. Write an SQL query which returns a list of programs which are "allergy impacted". This should return 1 column with a list of "allergy impacted" programs.

```
select program from
        (select program
                , sum( case when allergy is not null then 1 else 0 end)::float / sum(1) as rat
        from campers
        group by 1 ) as innerQ
where rat >= .1
```

6. We calculate the age-adjusted height ("AAH") by taking a campers height and dividing it by their age squared ($\frac{height}{age^2}$). Write a query which returns all rows and three columns: age-adjusted height, name and program.

```
select hgt /age / age as aah, name, program
from campers;
```

7. Using SQL, write a query which returns three columns: name, program, and AAH_Flag. AAH_Flag should be equal to 0 if the AAH is less than or equal to 1, 1 if the AAH is greater than 1 and less than or equal to 3 and 2 otherwise. AAH is defined in the previous problem.

```
select
        name, program
        , case
                when hgt / age / age  <= 1 then 0
                when hgt / age / age <= 3 then 1
                else 2 end as AAH_Flag
from
        campers;
```

8. If a person's AAH is greater than or equal to 3 they are defined as "tall". Write a query which returns the *percentage* of campers of each program who are tall. This should have two columns: program and percentage of the campers in that program who are tall.

```
select
        program,
        sum( case when hgt / age / age >= 3 then 1 else 0 end )::float / count(1) as pctTall
from
        campers
group by 1;
```

9. Using SQL, write a query which returns one row and two columns. The first column should be the number of campers who are 10 years old (exactly) and signed up for the "basketball" program (call this column bb10). The second column should be the number of campers who are 7 years old (exactly) who are signed up for the "skateboarding" program (call this column sb7). You can assume that all programs are lower case.

```
select
        sum( case when program = 'basketball' and age = 10 then 1 else 0 end) as bb10
        , sum( case when program = 'skateboarding' and age = 7 then 1 else 0 end) as sb7
from campers;
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrame named *campers* is already loaded. If a specific output is not specified you can return anything (DataFrame/Series/List/Array/etc.)

1. Using Pandas, return the name (as a Series) of the top-8 shortest campers who are 10 years old.

```
campers.loc[(campers.loc[:, 'age'] == 10), :].nsmallest(8, 'hgt').loc[:, 'name']
```

   OR:

```
(campers
        .loc[(campers.loc[:, 'age']== 10), :]
        .sort_values( 'hgt', ascending=True)
        .head(8)
        .loc[:, 'name']
)
```

2. Using Pandas, return a DataFrame with two columns: name and state. The dataset should only contain campers that are either (a) over 10 years and from Virginia ("VA") or (b) under 8 years old and from Michigan ("MI").

```
campers.loc[((campers.loc[:, 'age'] > 10) & (campers.loc[:, 'state'] == 'VA') )
        | ((campers.loc[:, 'age'] < 8) & (campers.loc[:, 'state'] == 'MI') )
        , ['name', 'state']]
```

3. Using Pandas, return a **DataFrame** which contains all campers (name only) who are younger than 10 years old and are either from New Jersey ('NJ') or Wyoming ('WY'). Only include those campers who do NOT have a food allergy.

```
campers.loc[(campers.loc[:, 'age'] < 10)
        & (campers.loc[:, 'allergies'].isna())
        & (campers.loc[:, 'state'].isin( ['AL', 'NY']))
        , ['name']]
```

4. Using Pandas, return all programs (this should be without duplicates) which have a camper who has an allergy to "peanuts". You can assume that all allergies in the table are lower case.

```
campers.loc[ (campers.loc[:, 'allergy'] == 'peanuts'), 'program'].unique()
```

5. Return all rows and columns for campers who are taking "basketball" as their program (you can assume that all programs are lower case). Sort the resulting DataFrame first by state (alphabetically) and then, within state, from tallest to shortest.

```
(campers.loc[ (campers.loc[:, 'program'] == 'basketball'), :]
        .sort_values( ['state', 'hgt'], ascending=[True, False])
        )
```

6. Please return a DataFrame which has all the original data and adds a column called "AAH" which is the age-adjusted-height (this is height divided by age squared, as in the previous problems).

```
campers.loc[:, 'aah'] = campers.loc[:, 'hgt'] / campers.loc[:, 'age'] / campers.loc[:,  'age']
```

7. Please return a DataFrame which has all the original data as well as adds a column called "hgt_flag" which is equal to 0 if the camper is greater than or equal to 140cm, 1 if they are greater than or equal to 110 and less than 140 and 2 otherwise.

```
campers.loc[:, 'hgt_flag'] = 2
campers.loc[ (campers.loc[:, 'hgt'] >= 140), 'hgt_flag'] = 0
campers.loc[  (campers.loc[:, 'hgt'] >= 110) & (campers.loc[:, 'hgt'] < 140), 'hgt_flag'] = 0
```

8. There was an error and students who were 10 years old all had their height recorded as 10 centimeters too high. Please return an updated version of the campers DataFrame which has this error fixed. Specifically the DataFrame should have all rows and columns, but the `hgt` column should have this error fixed.

```
campers.loc[ (campers.loc[:, 'age'] == 10), 'hgt'] = campers.loc[ (campers.loc[:, 'age'] == 10), 'hgt'] -10
```

4 2023 CAPP Databases Midterm B

This exam was given to Master's level students in the University of Chicago CAPP program in the Spring of 2023. There were two versions of the exam – this is version "B".

The following table contains information about workers applying to a temp agency for data entry positions. You can assume that each name uniquely defines a person and that a person only appears once in the table.

- **name:** The name of the person (string, NOT NULL).
- **exp:** Years of experience (integer, NOT NULL).
- **wpm:** The number of words per minute (wpm) the person types (float, NOT NULL).
- **state:** The state that the worker is from (string, NOT NULL).
- **degree:** Highest educational attainment. You can assume this is all lower case (string, NOT NULL).
- **wage:** Preferred hourly wage (float, NOT NULL).
- **certificate:** If the person has a special certificate, such as for dealing with health care or bank data. If Null, this means that the person has no certificate. You can assume this is all lower case. A person can, AT MOST, have a single certificate (string, HAS NULLS).
- The name of the table / DataFrame is **agency**. No need to use a schema or load the DataFrame.
- Only use syntax covered in class.
- Interpret all inequalities as strict unless explicitly stated.

Figure D.2: *agency* Table: 12,345 Rows

name	exp	wpm	state	degree	wage	certificate
Ringly Roberson	7	68.5	NY	high-school	18.00	
Crash Bandicoot	14	88.0	MS	ba	22.00	healthcare
Alligator Reynolds	4	72.25	PA	ms	17.5	

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Using SQL, write a query which returns the names (name only) of the 6 slowest typers (smallest wpm) who are from Pennsylvania ("PA").

```
SELECT
        name
FROM
        agency
where state = 'PA'
order by wpm desc
limit 6;
```

2. Using SQL, write a query which returns all workers (name and state) who have less than 10 years of experience, have no certificate and are either from Idaho ("ID") or California ("CA").

```
select name, state from agency where
        state in ('ID', 'CA')
        and exp < 10
        and certificate is null.
```

3. Write an SQL query which returns the number of workers from each state who have a certificate. This should be two columns: one with the state and the other with the number of workers from that state who have a certificate.

```
SELECT
        state, count(1) as ct
from
        agency
where certificate is not null
group by 1;
```

4. Write an SQL query which returns all rows and columns for workers whose degree is equal to "high-school". You can assume that all degrees are lower case. Sort the data by words per minute from highest to lowest.

```
select * from agency
where degree = 'high-school'
order by wpm desc;
```

5. We are analyzing degrees and certificates. We want to figure out which degrees have more than 20% of their workers with a certificate. Write an SQL query which returns one column. In this column should be a list of degrees where more than 20% of the workers with that degree have a certificate.

```
select degree from
        (select degree
                , sum( case when certificate is not null then 1 else 0 end)::float / sum(1) as rat
        from agency
        group by 1 ) as innerQ
where rat >= .2
```

6. We calculate the dollar-adjusted wpm ("DAWPM") by taking a worker's WPM, squaring it and dividing it by their wage (in pennies) ($\frac{wpm^2}{100 \cdot wage}$). Write a query which returns three columns: name, degree and the DAWPM.

```
select (wpm * wpm) / (100 * wage) as DAWPM, name, degree
from agency;
```

7. Write a query which returns three columns: name, degree, and DAWPM_Flag. DAWPM_Flag should be equal to 0 if the DAWPM is less than or equal to 3, 1 if the DAWPM is greater than 3 and less than or equal to 10 and 2 otherwise. DAWPM is defined in the previous problem.

```
select
        name, degree
        , case
                when (wpm * wpm) / (100 * wage)   <= 3 then 0
                when (wpm * wpm) / (100 * wage)  <= 10 then 1
                else 2 end as DAWPM_Flag
from
        agency;
```

8. If a person's DAWPM is greater than or equal to 3 they are defined as "hyper-efficient". Write a query which returns the *percentage* of workers of each degree who are hyper-efficient. This should have two columns: degree and percentage of the workers with that degree who are hyper-efficient.

```
select
       degree,
       sum( case when (wpm * wpm) / (100 * wage) >= 3 then 1 else 0 end )::float / count(1) as pctEff
from
       agency
group by 1;
```

9. Using SQL, write a query which returns one row and two columns. The first column should be the number of workers who have exactly 3 years experience and have a "ba" degree (call this column ba3). The second column should be the number of workers who have exactly 7 years of experience and have an "ms" degree (call this column ms7).

```
select
        sum( case when degree = 'ba' and exp = 3 then 1 else 0 end) as ba3
        , sum( case when degree = 'ms' and exp = 7 then 1 else 0 end) as ms7
from agency;
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrame named *agency* is already loaded. If a specific output is not specified you can return anything (DataFrame/Series/List/Array/etc.)

1. Using Pandas, return the name (as a Series) of the top 6 fastest typers (largest wpm) who have 3 years of work experience.

```
agency.loc[(agency.loc[:, 'exp'] == 3), :].nlargest(6, wpm).loc[:, 'name']
```

   OR:

```
(agency
        .loc[(agency.loc[:, 'exp'] == 3), :]
        .sort_values( 'wpm', ascending=False)
        .head(6)
        .loc[:, 'name']
)
```

2. Using Pandas, return a DataFrame with two columns: name and state. The dataset should only contain workers that are either (a) over 10 years experience and from Georgia ("GA") or (b) under

8 years experience and from Nevada ("NV").

```
agency.loc[((agency.loc[:, 'exp'] > 10) & (agency.loc[:, 'state'] == 'GA') )
        | ((agency.loc[:, 'exp'] < 8) & (agency.loc[:, 'state'] == 'NV') )
        , ['name', 'state']]
```

3. Using Pandas, return a **DataFrame** which contains all workers (name only) who have less than 9 years experience and are either from New Mexico ('NM') or Texas ('TX'). Only include those workers who have a certificate (any certificate).

```
agency.loc[(agency.loc[:, 'exp'] < 9)
        & ~(agency.loc[:, 'certificate'].isna())
        & (agency.loc[:, 'state'].isin( ['NM', 'TX']))
        , ['name']]
```

4. Using Pandas, return all states (this should be without duplicates) which have a worker with a "healthcare" certificate. You can assume that all certificate names are lower case.

```
agency.loc[ (agency.loc[:, 'certificate'] == 'healthcare'), 'state'].unique()
```

5. Return all rows and columns for workers who have a "doctorate" degree (you can assume that all degrees are lower case). Sort the resulting DataFrame first by state (alphabetically) and then, within state, from fastest to slowest wpm.

```
(agency.loc[ (agency.loc[:, 'degree'] == 'doctorate'), :]
        .sort_values( ['state', 'wpm'], ascending=[True, False])
        )
```

6. Please return a DataFrame which has all the original data and adds a column called "DAWPM" which is the dollar-adjusted words per minute (as in the other problems it is defined as words per minute squared divided by wages in pennies: $\frac{wpm^2}{100 \cdot wage}$).

```
agency.loc[:, 'dawpm'] = (agency.loc[:, 'wpm'] * agency.loc[:, 'wpm']) / (100.0 * agency.loc[:,  'wage'])
```

7. Please return a DataFrame which has all the original data as well as adds a column called "exp_flag" which is equal to 0 if the worker has less than 10 years experience, 1 if they have greater than or equal to 10 years and less than 20 years experience and 2 otherwise.

```
agency.loc[:, 'exp_flag'] = 0
agency.loc[  (agency.loc[:, 'exp'] >= 10) & (agency.loc[:, 'exp'] < 20), 'exp_flag'] = 1
agency.loc[ (agency.loc[:, 'exp'] > 20), 'exp_flag'] = 2
```

8. There was an error and workers from Maryland ("MD") had their WPM recorded as 5 too large. Please return an updated DataFrame which has this error fixed. Specifically the DataFrame should have all rows and columns, but the `wpm` column should have this error fixed.

```
agency.loc[ (agency.loc[:, 'state'] == 'MD'), 'wpm'] = agency.loc[ (agency.loc[:, 'state'] == 'MD'), 'wpm'] - 5
```

5 2017 SQL Final

This exam was given at the masters level in 2017 for a course just covering SQL. Students had two hours to complete it.

Use the following tables when answering the questions. The tables below consist of information from a manufacturing company's internal database system. This company makes children's toys by assembling *parts* into *items*.

- Columns with the same name can be assumed to be the same.
- The company takes *parts* and turns them into *items* which are then sold.
- **Unless stated above, all values are not null.**
- Parts table:
  - **PID:** The ID for a particular part. (int)
  - **Price:** The cost associated with the part. (float)
  - **Desc:** A short part description. (string)
  - You can assume that PID is unique to each row.
- Item table:
  - **IID:** The Item ID for a particular item. (int)
  - **Desc:** A short item description. (string)
  - **TPlus:** This is a 1 or 0, depending on if the toy is designed for children who are "Twelve Plus" years old. (int)
  - You can assume that IID is unique to each row.
- Assembly Table:
  - This table contains the "recipe" for making each item. It is a map between the parts and the items.
  - **IID:** This is the item ID of the item that is being assembled. (int)
  - **PID:** This is the PID of the part that is going into the item. (int)
  - **NoPart:** This is the quantity of each part that is required to make the item. (int)
- Sales Table:
  - This table contains information regarding the company's sales.
  - **IID:** This is the item ID for the item sold. (int)
  - **Rev:** This is the amount that the item was sold for. (float)
  - **CustID:** This is the ID for the customer. (int)
  - **SalesID:** This is the ID of the salesperson. (int)
  - **DT:** This is the date that the sales took place. (date)
- SP Table:
  - This table contains information regarding the salespeople in the company.

– Each row represents a salesperson.

– **SalesID:** This the ID of the salesperson. (int)

– **ST:** This is the state that the salesperson lives. If a state is NULL, that means the salesperson lives internationally. (string)

– **Name:** This is the salesperson's name.

– **Bonus:** This is equal to "H" or "L" for "High" or "Low" bonus structure. (string)

– You can assume that SalesID is unique to each row.

Table D.1: *Parts* Table, 4,525 Rows

PID	Price	Desc
1	11.99	Plastic Box (11 x 11)
2	12.85	8" Wheel
3	127.85	4" dowel

Table D.2: *Item* Table, 525 Rows

IID	Desc	TPlus
1	Small Wagon	0
2	Children's playhouse	0
3	Plastic Truck	1

Table D.3: *Assembly* Table, 15,225 Rows

IID	PID	NoPart
1	2	4
1	12	1
1	8	2

Table D.4: *Sales* Table, 45,258 Rows

IID	Rev	CustID	SalesID	Dt
12	65.73	1	12	01-03-2011
12	75.73	3	12	02-03-2011
48	265.04	2	17	01-05-2011
92	554.36	85	8	08-27-2011
115	18.18	92	22	08-11-2011

Table D.5: *SP* Table, 35 Rows

SalesID	ST	Name	Bonus
1	CA	Eldon Tyrell	H
2	PA	J.F. Sebastian	L
3		Roy Batty	L
4	CA	Rick Deckard	H

For the following questions write a query that returns the answer **and ONLY the answer.** All questions should be answered with a single SQL query, unless stated otherwise.

1. What are the top five salesperson (SalesID) in terms of *number* of items sold?

```
select SalesID
from sales
group by 1
order by sum(1) desc
limit 5;
```

2. How many salespeople are from California ("CA")?

```
select
        sum(1)
from
        SP
where st = 'CA';
```

3. Write a query which returns three columns and twelve rows. The first column should be month and should be the month that the sales took place, the second column should contain the number of items sold for the "Twelve Plus" audience and the third should contain the number of items sold for the not "Twelve Plus" audience. Assume that the sales table contains all information for the year 2011 (and no other year.

```
select
        date_part('month', dt) as mnth
        , sum( case when TPlus = 1 then 1 else 0 end ) as tplus
        , sum( case when TPlus = 0 then 1 else 0 end ) as tminus
from
        sales
left join
        item
using(iid)
group by 1;
```

4. Write a query which returns 5 columns. The first column is the state, the second column is the number of items sold to that state, the third is the number of items sold to that state which had a revenue greater than $1,000, the fourth column should the number of unique customers sold to that state and the final column should be the number of customers who spent more than $1,000 in a single transaction.

```
select
        lhs.st
        , count(1) as numitemsold
        , sum(case when Rev > 1000 then 1 else 0 end) as numitems1000
        , count(distinct CustID) as unique custs
        , count(distinct case when Rev > 1000 then custID else null end)
from
        SP
left join
        Sales
using( SalesID)
group by 1;
```

5. Write a query which returns IID and the average revenue generated for that item. Exclude items that have been sold less that 10 times.

```
select IID, avg( rev) as avgrev
from sales
group by 1
having count(1) > 10;
```

6. Items which are sold by salepeople in New York ("NY") have to add a tax of 5%. What is the total amount of tax that the company needs to add?

```
select
        .05 * sum( Rev ) as tax
from
        sales
where
        salesid in
                (select salesID from SP where st = 'NY')
```

7. Which item (IID) uses the most unique parts (PID)?

```
select
        IID
from
        Assembly
group by 1
order by count(1) desc
limit 1;
```

8. Which items uses the most *total* parts?

```
select
        IID
from
        Assembly
group by 1
order by sum(NoPart) desc
limit 1;
```

9. Return a list of the items (IID) that have never been sold:

```
select item.IID
from
        item
left join
        sales
using(IID)
where sales.IID is null
```

10. Return IID and the total cost of the parts used to make a one of them.

```
select
        IID, sum( price * noPart) as cost
from
        assembly
left join
        parts
using(PID)
group by IID;
```

11. Of all the salespeople from California ("CA") or Pennsylvania ("PA"), return the one (Name and SalesID) with the highest amount of revenue.

```
select
        SP.SalesID, Name
from
        SP
left join
        Sales
using( SalesID)
where SP.state in ('CA', 'PA')
group by 1,2
order by sum( Rev) desc
limit 1;
```

12. Of all the salespeople (SalesID) who have sold an item for more than $100.00, return the average revenue per item sold on all of their sales.

```
select
        avg( rev) as ar
from
        sales
where
        salesid in
                (select distinct salesid from sales where rev > 100);
```

13. For each bonus structure ("H" or "L") return the percentage of revenue generated from salespeople in that bonus structure. This should return two rows and two columns.

```
select
        bonus,
        sr / sum( sr) over() as pct
from
        (select
                Bonus, sum(rev) as sr
        from
                SP
        left join
                sales
        using( SalesID)
        group by 1) as IQ
```

14. For each Bonus Structure ("H" or "L") return the percentage of revenue generated from salespeople in that bonus structure. This should return one row and two columns (PctRevH and PctRevL). Note that this is the same data as the previous query, but shaped wide, rather than long.

```
select
        sum( case when Bonus = 'H' then rev else 0 end )/sum( Rev) as PctRevH
        , sum( case when Bonus = 'L' then rev else 0 end )/sum( Rev) as PctRevL
from
        SP
left join
        sales
using( SalesID)
```

15. Calculate the total profit (total revenue - total cost) for each item (IID).

```
select
        (SR - ct * totalCost) as profit, lhs.IID
from
        (select sum(NoPart * Price ) as totalCost, IID
                from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, count(1) as ct from sales group by 2) as lhs
using( IID )
group by 2;
```

16. Of all the items (IID) which use more than 5 unique PIDs, which two have the largest number of sales (in terms of count)?

```
select
        lhs.IID
from
        (select
                IID
        from
                Assembly
        group by 1
        having count(1) > 5 ) as lhs
left join
        Sales
using(IID)
group by 1
order by count(1) desc
limit 2;
```

17. An item is called "complex" if the ratio of unique parts (PIDs) to total parts is more than .90% and is called "simple" if the ratio is below .25%. Write a query which returns one row with three columns: the first column should be the total revenue generated by complex items, the second should be the total revenue generated by simple items and the third should be the revenue generated by items which are neither simple nor complex.

```
select
        sum( case when rat > .9 then amt else 0 end ) as TR_complext
        , sum( case when rat < .25 then amt else 0 end) as TR_Simple
        , sum( case when rat <=.9 and rat >= .25 then amt else 0 end) as TR_neither
from
        (select IID, count(1)::float / sum( NoPart) as rat from Assembly
        group by 1) as lhs
left join
        Sales
using(IID)
```

18. Return the long version of the query above. This time there will be two columns and three rows. An item is called "complex" if the ratio of unique parts (PIDs) to total parts is more than .90% and is called "simple" if the ratio is below .25%. The first column should be equal to "Complex", "Simple" or "Neither" and the the second column should be the revenue generated by that type.

```
select
        case
                when rat > .9 then 'Complex'
                when rat < .25 then 'Simple'
                else 'Neither'
        end as typ
        , sum( amt )
from
        (select IID, count(1)::float / sum( NoPart) as rat from Assembly) as lhs
left join
        Sales
using(IID)
group by 1;
```

19. Which salesperson (SalesID) generated the highest profit (total revenue - total cost)?

```
select
        SalesID
from
        (select sum(NoPart * Price ) as totalCost, IID
               from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, SalesID, count(1) as ct from sales group by 2,3) as lhs
using( IID )
group by 2
order by (SR - ct * totalCost) desc
limit 1;
```

20. The company believes that there is an assembly cost of roughly $1.00 per part. Note that if an item requires four of the same part, this means that the assembly cost is $4.00. Factoring in this cost, what item has the highest total profit?

```
select
        IID
from
        (select sum(NoPart * Price) + sum( NoPart)*1.0 as totalCost, IID
               from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, count(1) as ct from sales group by 2) as lhs
using( IID )
group by 2
order by (SR - ct * totalCost) desc
limit 1;
```

21. For each item (IID), return the average number of days between sales (note that subtracting two dates yields the number of days between those dates).

```
select
        IID, avg(diff) as avgdiff
from
        (select
                IID
                ,dt -  lag(dt) over(partition by IID order by dt asc) as diff
        from
                sales ) as IQ
group by 1;
```

22. Using a cross join return a dataset which contains 3 columns: IID, month and the number of days that that IID was *not* sold that month. Assume that (a) the Sales table contains all information from 2011 and (b) that every possible sales date is represented in the Sales table. Note that if an item is not sold at all during the year it *should* still be in this result.

```
select
        IID, date_part('month', dt ) as mnth
        , sum( case when sales.dt is null then 1 else 0 end ) as daysmissing
from
        (select distinct IID from item) as lhs
cross join
        (select distinct dt from sales) as rhs1
left join
        sales
on lhs.IID = sales.IID and rhs1.dt = sales.dt
group by 1,2;
```

23. Write a query which returns 3 columns: date, the total revenue on that date and the average revenue from the last three days, but *not* including the current day. In other words if today is 1/5, then it should be the average revenue from 1/2, 1/3 and 1/4.

```
select
        dt, sr
        , avg( sr ) over (order by dt asc rows between 4 preceding and 1 preceding) as MA
from
        (select
                sum( rev) as SR, dt
        from
                sales
        group by 2 ) as iq
```

24. Which salesperson (name) sold the most number of *parts*. Note that this is not asking for number of *items* that each salesperson sold, but the number of parts contained within those items.

```
select
        Name
from
        SP
left join
        Sales
        using( salesid)
left join
        Assembly
        using( IID )
group by 1
order by sum(NoPart) desc
limit 1;
```

25. **Bonus** The salespeople in the table are characters from a movie. Which one?

6 2017 SQL Final

This exam was given at the masters level in 2017 for a course just covering SQL. Students had two hours to complete it.

Use the following tables when answering the questions. The tables below consist of information from a manufacturing company's internal database system. This company makes children's toys by assembling *parts* into *items*.

- Columns with the same name can be assumed to be the same.
- The company takes *parts* and turns them into *items* which are then sold.
- **Unless stated above, all values are not null.**
- Parts table:
  - **PID:** The ID for a particular part. (int)
  - **Price:** The cost associated with the part. (float)
  - **Desc:** A short part description. (string)
  - You can assume that PID is unique to each row.
- Item table:
  - **IID:** The Item ID for a particular item. (int)
  - **Desc:** A short item description. (string)
  - **TPlus:** This is a 1 or 0, depending on if the toy is designed for children who are "Twelve Plus" years old. (int)
  - You can assume that IID is unique to each row.
- Assembly Table:
  - This table contains the "recipe" for making each item. It is a map between the parts and the items.
  - **IID:** This is the item ID of the item that is being assembled. (int)
  - **PID:** This is the PID of the part that is going into the item. (int)
  - **NoPart:** This is the quantity of each part that is required to make the item. (int)
- Sales Table:
  - This table contains information regarding the company's sales.
  - **IID:** This is the item ID for the item sold. (int)
  - **Rev:** This is the amount that the item was sold for. (float)
  - **CustID:** This is the ID for the customer. (int)
  - **SalesID:** This is the ID of the salesperson. (int)
  - **DT:** This is the date that the sales took place. (date)
- SP Table:
  - This table contains information regarding the salespeople in the company.

– Each row represents a salesperson.

– **SalesID:** This the ID of the salesperson. (int)

– **ST:** This is the state that the salesperson lives. If a state is NULL, that means the salesperson lives internationally. (string)

– **Name:** This is the salesperson's name.

– **Bonus:** This is equal to "H" or "L" for "High" or "Low" bonus structure. (string)

– You can assume that SalesID is unique to each row.

Table D.6: *Parts* Table, 4,525 Rows

PID	Price	Desc
1	11.99	Plastic Box (11 x 11)
2	12.85	8" Wheel
3	127.85	4" dowel

Table D.7: *Item* Table, 525 Rows

IID	Desc	TPlus
1	Small Wagon	0
2	Children's playhouse	0
3	Plastic Truck	1

Table D.8: *Assembly* Table, 15,225 Rows

IID	PID	NoPart
1	2	4
1	12	1
1	8	2

Table D.9: *Sales* Table, 45,258 Rows

IID	Rev	CustID	SalesID	Dt
12	65.73	1	12	01-03-2011
12	75.73	3	12	02-03-2011
48	265.04	2	17	01-05-2011
92	554.36	85	8	08-27-2011
115	18.18	92	22	08-11-2011

Table D.10: *SP* Table, 35 Rows

SalesID	ST	Name	Bonus
1	CA	Eldon Tyrell	H
2	PA	J.F. Sebastian	L
3		Roy Batty	L
4	CA	Rick Deckard	H

For the following questions write a query that returns the answer **and ONLY the answer.** All questions should be answered with a single SQL query, unless stated otherwise.

1. What are the top five salesperson (SalesID) in terms of *number* of items sold?

```
select SalesID
from sales
group by 1
order by sum(1) desc
limit 5;
```

2. How many salespeople are from California ("CA")?

```
select
        sum(1)
from
        SP
where st = 'CA';
```

3. Write a query which returns three columns and twelve rows. The first column should be month and should be the month that the sales took place, the second column should contain the number of items sold for the "Twelve Plus" audience and the third should contain the number of items sold for the not "Twelve Plus" audience. Assume that the sales table contains all information for the year 2011 (and no other year.

```
select
        date_part('month', dt) as mnth
        , sum( case when TPlus = 1 then 1 else 0 end ) as tplus
        , sum( case when TPlus = 0 then 1 else 0 end ) as tminus
from
        sales
left join
        item
using(iid)
group by 1;
```

4. Write a query which returns 5 columns. The first column is the state, the second column is the number of items sold to that state, the third is the number of items sold to that state which had a revenue greater than $1,000, the fourth column should the number of unique customers sold to that state and the final column should be the number of customers who spent more than $1,000 in a single transaction.

```
select
        lhs.st
        , count(1) as numitemsold
        , sum(case when Rev > 1000 then 1 else 0 end) as numitems1000
        , count(distinct CustID) as unique custs
        , count(distinct case when Rev > 1000 then custID else null end)
from
        SP
left join
        Sales
using( SalesID)
group by 1;
```

5. Write a query which returns IID and the average revenue generated for that item. Exclude items that have been sold less that 10 times.

```
select IID, avg( rev) as avgrev
from sales
group by 1
having count(1) > 10;
```

6. Items which are sold by salepeople in New York ("NY") have to add a tax of 5%. What is the total amount of tax that the company needs to add?

```
select
        .05 * sum( Rev ) as tax
from
        sales
where
        salesid in
                (select salesID from SP where st = 'NY')
```

7. Which item (IID) uses the most unique parts (PID)?

```
select
        IID
from
        Assembly
group by 1
order by count(1) desc
limit 1;
```

8. Which items uses the most *total* parts?

```
select
        IID
from
        Assembly
group by 1
order by sum(NoPart) desc
limit 1;
```

9. Return a list of the items (IID) that have never been sold:

```
select item.IID
from
        item
left join
        sales
using(IID)
where sales.IID is null
```

10. Return IID and the total cost of the parts used to make a one of them.

```
select
        IID, sum( price * noPart) as cost
from
        assembly
left join
        parts
using(PID)
group by IID;
```

11. Of all the salespeople from California ("CA") or Pennsylvania ("PA"), return the one (Name and SalesID) with the highest amount of revenue.

```
select
        SP.SalesID, Name
from
        SP
left join
        Sales
using( SalesID)
where SP.state in ('CA', 'PA')
group by 1,2
order by sum( Rev) desc
limit 1;
```

12. Of all the salespeople (SalesID) who have sold an item for more than $100.00, return the average revenue per item sold on all of their sales.

```
select
        avg( rev) as ar
from
        sales
where
        salesid in
                (select distinct salesid from sales where rev > 100);
```

13. For each bonus structure ("H" or "L") return the percentage of revenue generated from salespeople in that bonus structure. This should return two rows and two columns.

```
select
        bonus,
        sr / sum( sr) over() as pct
from
        (select
                Bonus, sum(rev) as sr
        from
                SP
        left join
                sales
        using( SalesID)
        group by 1) as IQ
```

14. For each Bonus Structure ("H" or "L") return the percentage of revenue generated from salespeople in that bonus structure. This should return one row and two columns (PctRevH and PctRevL). Note that this is the same data as the previous query, but shaped wide, rather than long.

```
select
        sum( case when Bonus = 'H' then rev else 0 end )/sum( Rev) as PctRevH
        , sum( case when Bonus = 'L' then rev else 0 end )/sum( Rev) as PctRevL
from
        SP
left join
        sales
using( SalesID)
```

15. Calculate the total profit (total revenue - total cost) for each item (IID).

```
select
        (SR - ct * totalCost) as profit, lhs.IID
from
        (select sum(NoPart * Price ) as totalCost, IID
                from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, count(1) as ct from sales group by 2) as lhs
using( IID )
group by 2;
```

16. Of all the items (IID) which use more than 5 unique PIDs, which two have the largest number of sales (in terms of count)?

```
select
        lhs.IID
from
        (select
                IID
        from
                Assembly
        group by 1
        having count(1) > 5 ) as lhs
left join
        Sales
using(IID)
group by 1
order by count(1) desc
limit 2;
```

17. An item is called "complex" if the ratio of unique parts (PIDs) to total parts is more than .90% and is called "simple" if the ratio is below .25%. Write a query which returns one row with three columns: the first column should be the total revenue generated by complex items, the second should be the total revenue generated by simple items and the third should be the revenue generated by items which are neither simple nor complex.

```
select
        sum( case when rat > .9 then amt else 0 end ) as TR_complext
        , sum( case when rat < .25 then amt else 0 end) as TR_Simple
        , sum( case when rat <=.9 and rat >= .25 then amt else 0 end) as TR_neither
from
        (select IID, count(1)::float / sum( NoPart) as rat from Assembly
        group by 1) as lhs
left join
        Sales
using(IID)
```

18. Return the long version of the query above. This time there will be two columns and three rows. An item is called "complex" if the ratio of unique parts (PIDs) to total parts is more than .90% and is called "simple" if the ratio is below .25%. The first column should be equal to "Complex", "Simple" or "Neither" and the the second column should be the revenue generated by that type.

```
select
        case
                when rat > .9 then 'Complex'
                when rat < .25 then 'Simple'
                else 'Neither'
        end as typ
        , sum( amt )
from
        (select IID, count(1)::float / sum( NoPart) as rat from Assembly) as lhs
left join
        Sales
using(IID)
group by 1;
```

19. Which salesperson (SalesID) generated the highest profit (total revenue - total cost)?

```
select
        SalesID
from
        (select sum(NoPart * Price ) as totalCost, IID
               from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, SalesID, count(1) as ct from sales group by 2,3) as lhs
using( IID )
group by 2
order by (SR - ct * totalCost) desc
limit 1;
```

20. The company believes that there is an assembly cost of roughly $1.00 per part. Note that if an item requires four of the same part, this means that the assembly cost is $4.00. Factoring in this cost, what item has the highest total profit?

```
select
        IID
from
        (select sum(NoPart * Price) + sum( NoPart)*1.0 as totalCost, IID
               from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, count(1) as ct from sales group by 2) as lhs
using( IID )
group by 2
order by (SR - ct * totalCost) desc
limit 1;
```

21. For each item (IID), return the average number of days between sales (note that subtracting two dates yields the number of days between those dates).

```
select
        IID, avg(diff) as avgdiff
from
        (select
                IID
                ,dt -  lag(dt) over(partition by IID order by dt asc) as diff
        from
                sales ) as IQ
group by 1;
```

22. Using a cross join return a dataset which contains 3 columns: IID, month and the number of days that that IID was *not* sold that month. Assume that (a) the Sales table contains all information from 2011 and (b) that every possible sales date is represented in the Sales table. Note that if an item is not sold at all during the year it *should* still be in this result.

```
select
        IID, date_part('month', dt ) as mnth
        , sum( case when sales.dt is null then 1 else 0 end ) as daysmissing
from
        (select distinct IID from item) as lhs
cross join
        (select distinct dt from sales) as rhs1
left join
        sales
on lhs.IID = sales.IID and rhs1.dt = sales.dt
group by 1,2;
```

23. Write a query which returns 3 columns: date, the total revenue on that date and the average revenue from the last three days, but *not* including the current day. In other words if today is 1/5, then it should be the average revenue from 1/2, 1/3 and 1/4.

```
select
        dt, sr
        , avg( sr ) over (order by dt asc rows between 4 preceding and 1 preceding) as MA
from
        (select
                sum( rev) as SR, dt
        from
                sales
        group by 2 ) as iq
```

24. Which salesperson (name) sold the most number of *parts*. Note that this is not asking for number of *items* that each salesperson sold, but the number of parts contained within those items.

```
select
        Name
from
        SP
left join
        Sales
        using( salesid)
left join
        Assembly
        using( IID )
group by 1
order by sum(NoPart) desc
limit 1;
```

25. **Bonus** The salespeople in the table are characters from a movie. Which one?

7 2017 SQL Final

This exam was given at the masters level in 2017 for a course just covering SQL. Students had two hours to complete it.

Use the following tables when answering the questions. The tables below consist of information from a manufacturing company's internal database system. This company makes children's toys by assembling *parts* into *items*.

- Columns with the same name can be assumed to be the same.
- The company takes *parts* and turns them into *items* which are then sold.
- **Unless stated above, all values are not null.**
- Parts table:
  - **PID:** The ID for a particular part. (int)
  - **Price:** The cost associated with the part. (float)
  - **Desc:** A short part description. (string)
  - You can assume that PID is unique to each row.
- Item table:
  - **IID:** The Item ID for a particular item. (int)
  - **Desc:** A short item description. (string)
  - **TPlus:** This is a 1 or 0, depending on if the toy is designed for children who are "Twelve Plus" years old. (int)
  - You can assume that IID is unique to each row.
- Assembly Table:
  - This table contains the "recipe" for making each item. It is a map between the parts and the items.
  - **IID:** This is the item ID of the item that is being assembled. (int)
  - **PID:** This is the PID of the part that is going into the item. (int)
  - **NoPart:** This is the quantity of each part that is required to make the item. (int)
- Sales Table:
  - This table contains information regarding the company's sales.
  - **IID:** This is the item ID for the item sold. (int)
  - **Rev:** This is the amount that the item was sold for. (float)
  - **CustID:** This is the ID for the customer. (int)
  - **SalesID:** This is the ID of the salesperson. (int)
  - **DT:** This is the date that the sales took place. (date)
- SP Table:
  - This table contains information regarding the salespeople in the company.

– Each row represents a salesperson.

– **SalesID:** This the ID of the salesperson. (int)

– **ST:** This is the state that the salesperson lives. If a state is NULL, that means the salesperson lives internationally. (string)

– **Name:** This is the salesperson's name.

– **Bonus:** This is equal to "H" or "L" for "High" or "Low" bonus structure. (string)

– You can assume that SalesID is unique to each row.

Table D.11: *Parts* Table, 4,525 Rows

PID	Price	Desc
1	11.99	Plastic Box (11 x 11)
2	12.85	8" Wheel
3	127.85	4" dowel

Table D.12: *Item* Table, 525 Rows

IID	Desc	TPlus
1	Small Wagon	0
2	Children's playhouse	0
3	Plastic Truck	1

Table D.13: *Assembly* Table, 15,225 Rows

IID	PID	NoPart
1	2	4
1	12	1
1	8	2

Table D.14: *Sales* Table, 45,258 Rows

IID	Rev	CustID	SalesID	Dt
12	65.73	1	12	01-03-2011
12	75.73	3	12	02-03-2011
48	265.04	2	17	01-05-2011
92	554.36	85	8	08-27-2011
115	18.18	92	22	08-11-2011

Table D.15: *SP* Table, 35 Rows

SalesID	ST	Name	Bonus
1	CA	Eldon Tyrell	H
2	PA	J.F. Sebastian	L
3		Roy Batty	L
4	CA	Rick Deckard	H

For the following questions write a query that returns the answer **and ONLY the answer.** All questions should be answered with a single SQL query, unless stated otherwise.

1. What are the top five salesperson (SalesID) in terms of *number* of items sold?

```
select SalesID
from sales
group by 1
order by sum(1) desc
limit 5;
```

2. How many salespeople are from California ("CA")?

```
select
        sum(1)
from
        SP
where st = 'CA';
```

3. Write a query which returns three columns and twelve rows. The first column should be month and should be the month that the sales took place, the second column should contain the number of items sold for the "Twelve Plus" audience and the third should contain the number of items sold for the not "Twelve Plus" audience. Assume that the sales table contains all information for the year 2011 (and no other year.

```
select
        date_part('month', dt) as mnth
        , sum( case when TPlus = 1 then 1 else 0 end ) as tplus
        , sum( case when TPlus = 0 then 1 else 0 end ) as tminus
from
        sales
left join
        item
using(iid)
group by 1;
```

4. Write a query which returns 5 columns. The first column is the state, the second column is the number of items sold to that state, the third is the number of items sold to that state which had a revenue greater than $1,000, the fourth column should the number of unique customers sold to that state and the final column should be the number of customers who spent more than $1,000 in a single transaction.

```
select
        lhs.st
        , count(1) as numitemsold
        , sum(case when Rev > 1000 then 1 else 0 end) as numitems1000
        , count(distinct CustID) as unique custs
        , count(distinct case when Rev > 1000 then custID else null end)
from
        SP
left join
        Sales
using( SalesID)
group by 1;
```

5. Write a query which returns IID and the average revenue generated for that item. Exclude items that have been sold less that 10 times.

```
select IID, avg( rev) as avgrev
from sales
group by 1
having count(1) > 10;
```

6. Items which are sold by salepeople in New York ("NY") have to add a tax of 5%. What is the total amount of tax that the company needs to add?

```
select
        .05 * sum( Rev ) as tax
from
        sales
where
        salesid in
                (select salesID from SP where st = 'NY')
```

7. Which item (IID) uses the most unique parts (PID)?

```
select
        IID
from
        Assembly
group by 1
order by count(1) desc
limit 1;
```

8. Which items uses the most *total* parts?

```
select
        IID
from
        Assembly
group by 1
order by sum(NoPart) desc
limit 1;
```

9. Return a list of the items (IID) that have never been sold:

```
select item.IID
from
        item
left join
        sales
using(IID)
where sales.IID is null
```

10. Return IID and the total cost of the parts used to make a one of them.

```
select
        IID, sum( price * noPart) as cost
from
        assembly
left join
        parts
using(PID)
group by IID;
```

11. Of all the salespeople from California ("CA") or Pennsylvania ("PA"), return the one (Name and SalesID) with the highest amount of revenue.

```
select
        SP.SalesID, Name
from
        SP
left join
        Sales
using( SalesID)
where SP.state in ('CA', 'PA')
group by 1,2
order by sum( Rev) desc
limit 1;
```

12. Of all the salespeople (SalesID) who have sold an item for more than $100.00, return the average revenue per item sold on all of their sales.

```
select
        avg( rev) as ar
from
        sales
where
        salesid in
                (select distinct salesid from sales where rev > 100);
```

13. For each bonus structure ("H" or "L") return the percentage of revenue generated from salespeople in that bonus structure. This should return two rows and two columns.

```
select
        bonus,
        sr / sum( sr) over() as pct
from
        (select
                Bonus, sum(rev) as sr
        from
                SP
        left join
                sales
        using( SalesID)
        group by 1) as IQ
```

14. For each Bonus Structure ("H" or "L") return the percentage of revenue generated from salespeople in that bonus structure. This should return one row and two columns (PctRevH and PctRevL). Note that this is the same data as the previous query, but shaped wide, rather than long.

```
select
        sum( case when Bonus = 'H' then rev else 0 end )/sum( Rev) as PctRevH
        , sum( case when Bonus = 'L' then rev else 0 end )/sum( Rev) as PctRevL
from
        SP
left join
        sales
using( SalesID)
```

15. Calculate the total profit (total revenue - total cost) for each item (IID).

```
select
        (SR - ct * totalCost) as profit, lhs.IID
from
        (select sum(NoPart * Price ) as totalCost, IID
                from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, count(1) as ct from sales group by 2) as lhs
using( IID )
group by 2;
```

16. Of all the items (IID) which use more than 5 unique PIDs, which two have the largest number of sales (in terms of count)?

```
select
        lhs.IID
from
        (select
                IID
        from
                Assembly
        group by 1
        having count(1) > 5 ) as lhs
left join
        Sales
using(IID)
group by 1
order by count(1) desc
limit 2;
```

17. An item is called "complex" if the ratio of unique parts (PIDs) to total parts is more than .90% and is called "simple" if the ratio is below .25%. Write a query which returns one row with three columns: the first column should be the total revenue generated by complex items, the second should be the total revenue generated by simple items and the third should be the revenue generated by items which are neither simple nor complex.

```
select
        sum( case when rat > .9 then amt else 0 end ) as TR_complext
        , sum( case when rat < .25 then amt else 0 end) as TR_Simple
        , sum( case when rat <=.9 and rat >= .25 then amt else 0 end) as TR_neither
from
        (select IID, count(1)::float / sum( NoPart) as rat from Assembly
        group by 1) as lhs
left join
        Sales
using(IID)
```

18. Return the long version of the query above. This time there will be two columns and three rows. An item is called "complex" if the ratio of unique parts (PIDs) to total parts is more than .90% and is called "simple" if the ratio is below .25%. The first column should be equal to "Complex", "Simple" or "Neither" and the the second column should be the revenue generated by that type.

```
select
        case
                when rat > .9 then 'Complex'
                when rat < .25 then 'Simple'
                else 'Neither'
        end as typ
        , sum( amt )
from
        (select IID, count(1)::float / sum( NoPart) as rat from Assembly) as lhs
left join
        Sales
using(IID)
group by 1;
```

19. Which salesperson (SalesID) generated the highest profit (total revenue - total cost)?

```
select
        SalesID
from
        (select sum(NoPart * Price ) as totalCost, IID
               from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, SalesID, count(1) as ct from sales group by 2,3) as lhs
using( IID )
group by 2
order by (SR - ct * totalCost) desc
limit 1;
```

20. The company believes that there is an assembly cost of roughly $1.00 per part. Note that if an item requires four of the same part, this means that the assembly cost is $4.00. Factoring in this cost, what item has the highest total profit?

```
select
        IID
from
        (select sum(NoPart * Price) + sum( NoPart)*1.0 as totalCost, IID
               from Assembly left join Part using(PID) group by 2) as lhs
left join
        (select sum( rev ) as SR, IID, count(1) as ct from sales group by 2) as lhs
using( IID )
group by 2
order by (SR - ct * totalCost) desc
limit 1;
```

21. For each item (IID), return the average number of days between sales (note that subtracting two dates yields the number of days between those dates).

```
select
        IID, avg(diff) as avgdiff
from
        (select
                IID
                ,dt -  lag(dt) over(partition by IID order by dt asc) as diff
        from
                sales ) as IQ
group by 1;
```

22. Using a cross join return a dataset which contains 3 columns: IID, month and the number of days that that IID was *not* sold that month. Assume that (a) the Sales table contains all information from 2011 and (b) that every possible sales date is represented in the Sales table. Note that if an item is not sold at all during the year it *should* still be in this result.

```
select
        IID, date_part('month', dt ) as mnth
        , sum( case when sales.dt is null then 1 else 0 end ) as daysmissing
from
        (select distinct IID from item) as lhs
cross join
        (select distinct dt from sales) as rhs1
left join
        sales
on lhs.IID = sales.IID and rhs1.dt = sales.dt
group by 1,2;
```

23. Write a query which returns 3 columns: date, the total revenue on that date and the average revenue from the last three days, but *not* including the current day. In other words if today is 1/5, then it should be the average revenue from 1/2, 1/3 and 1/4.

```
select
        dt, sr
        , avg( sr ) over (order by dt asc rows between 4 preceding and 1 preceding) as MA
from
        (select
                sum( rev) as SR, dt
        from
                sales
        group by 2 ) as iq
```

24. Which salesperson (name) sold the most number of *parts*. Note that this is not asking for number of *items* that each salesperson sold, but the number of parts contained within those items.

```
select
        Name
from
        SP
left join
        Sales
        using( salesid)
left join
        Assembly
        using( IID )
group by 1
order by sum(NoPart) desc
limit 1;
```

25. **Bonus** The salespeople in the table are characters from a movie. Which one?

8 2018 SQL Final

This exam was given at the masters level in 2018 for a course just covering SQL. Students had two hours to complete it.

The information below comes from an insurance company's database, which operates under the following model: Agents sell insurance to *households*. A household may purchase multiple types of insurance (e.g. fire, car, earthquake). If a household makes a claim on a policy, then the insurance company can either pay the claim or fight it. If the insurance company fights a claim, then there are expenses associated with it. This database contains only active policies.

- Columns with the same name can be assumed to be the same.
- **Unless stated above, all values are not null.**
- Agent Table:
  - This table contains information regarding the agents in the company.
  - **AgentID:** This the ID of the agent, it is unique to each row. (int)
  - **St:** This is the agent's state. NULL values mean that the agent is international. (string)
  - **Name:** This is the agent's name.
  - **Bonus:** This is equal to "H" or "L" for "High" or "Low" bonus structure. (string)
- Household table:
  - This table contains information about the households that are insured.
  - **HHID:** The unique ID for the household, it is unique to each row. (int)
  - **AgentID:** This is the ID of the agent who manages the household. (int)
  - **Name:** This is the head of household's name. (string)
  - **MultiFam:** Are there multiple people in the household (1 = "Yes", 0 = "No"). (int)
  - **zip:** The zip for the household. (string)
- Policy table:
  - This table contains information on each policy. A household may *multiple* policies.
  - **PolicyID:** The unique ID for each policy. You can assume it is unique to each row. (int)
  - **HHID:** The unique ID for the household. (int)
  - **PolicyType:** The type of insurance policy. (string)
  - **EndDate:** The date the policy expires. (date)
  - **Cost**: The cost charged for the policy. (float)
- Claims Table:
  - This table contains information on claims made against insurance policies. A policy *may* have multiple claims or *no* claims against it.
  - **ClaimID:** The ID associated with the claim, it is unique to each row. (int)
  - **PolicyID:** The policy the claim is against. (int)

  - **Amt:** The amount of money being asked by the policy holder for the claim. (float)
  - **dt:** This is the date that the claim was received. (date)

- Expense Table:
  - This table contains information on legal expenses for disputing a claim. A single claim may have multiple expenses.
  - **EID:** Expense ID, assume it is unique for each row. (int)
  - **ClaimID:** The claim that the expense is against. (int)
  - **EAmt:** The amount of the expense. (float)
  - **dt:** This is the ID of the salesperson. (int)
  - **type:** The type of expense (legal, private investigator, etc.). (float)

Table D.16: *Agent* Table, 228 Rows

AgentID	ST	Name	Bonus
1	CA	Miles Quaritch	H
2	PA	Jake Sully	L
3		Parker Selfridge	L
4	CA	Neytiri	H

Table D.17: *Household* Table, 4,525 Rows

HHID	AgentID	Name	Zip	MultiFam
1	1	John Smith	11217	1
2	1	James McWright	99924	1
3	37	Elrod Lee	12345	0

Table D.18: *Policy* Table, 5,587 Rows

PolicyID	HHID	PolicyType	EndDate	Cost
1	1	Fire	01-01-2019	1,245.76
2	1	Car	01-01-2019	2,247.05
3	5	Flood	07-08-2020	287.56
4	37	Earthquake	03-01-2019	22,476.18

Table D.19: *Claims* Table, 1,225 Rows

ClaimID	PolicyID	Amt	dt
1	22	254.85	09-03-2018
2	22	212.87	09-05-2018
3	188	12,285.96	07-07-2017

Table D.20: *Expense* Table, 2,258 Rows

EID	EAmt	ClaimID	dt	type
1	65.73	1	01-03-2011	Legal
2	75.73	12	02-03-2011	Private Invest.
3	265.04	17	01-05-2011	Filing Fees
4	554.36	8	08-27-2011	Court Fees
5	18.18	22	08-11-2011	Legal

For the following questions write a query that returns the answer **and ONLY the answer.** All questions should be answered with a single SQL query, unless stated otherwise. Do not use CTE's.

1. What are the top five agents (AgentID) in terms of *number* of Households sold to?

```
select AgentID
from Household
group by 1
order by sum(1) desc
limit 5;
```

2. How many Agent's have sold policy's to households in zipcode 11217?

```
select
        count(distinct agentID)
from
        Household
where zipcode = 11217;
```

3. Write a query which returns four columns. The first column should be the year (int), the second should be the month (int) that a policy expires, the third column should be the total costs of all policies that expire during that month-year and the final column should be the number of "Fire" policies that expire that month.

```
select
       date_part('year', EndDate) as yr
       , date_part('month', EndDate) as mnth
       , sum( Cost) as totalCost
       , sum( case when PolicyType = 'Fire' then 1else 0 end) as numberFire
from
       Policy
group by 1,2;
```

4. For each household (HHID), return (a) the HHID, (b) the number of policies it has (c) the total costs associated with those policies.

```
select
        hhid, count(1), sum(Cost)
from
        policy
group by 1;
```

5. Similar to the above question, return the (a) HHID, (b) the number of policies for that household and (c) the number of claims for that household.

```
select
        lhs.hhid, count(distinct rhs1.policyID), count(distinct rhs2.claimID)
from
        household as lhs
left join
        policy as rhs1
        using(hhid)
left join
        claims as rhs2
        using(policyid)
group by 1;
```

6. International Agents need to pay an additional tax on their policy's of 7.5% of the cost. For those policies which expire in 2019, what is the total international tax bill?

```
select
        sum(cost) *.075 as taxpaid
from
        agent
left join
        household
        using( agentID)
left join
        policy
        using( policyID)
where agent.st is null
and date_part('year', enddate) = 2019;
```

7. Write a query which returns the most common policy type (e.g. "Fire") for MultiFam households. Note that most common is the number of policies, NOT cost.

```
select
        policy.policytype
from
        household
        using( agentID)
left join
        policy
        using( policyID)
where multifam = 1
group by 1
order by count(1) desc. limit 1;
```

8. Write a query which returns two rows and two columns. The first column should be "Bonus" type ("H" or "L") and the second should be the number of agents who have that bonus type (number of rows).

```
select bonus, count(1)
from agent group by 1;
```

9. Write a query which returns one rows and two columns, which is the transpose of the previous

question. The first column should be "HighBonus" and contain the number of agents who have Bonus type of High and the second column should be the number of agents who have a Bonus type of Low ("LowBonus")

```
select
        sum( case when bonus = 'H' then 1 else 0 end) as HighBonus
        , sum( case when bonus = 'H' then 1 else 0 end) as LowBonus
from agent;
```

10. What is the total claim (*not policy cost*) amount by policy type?

```
select
        policytype
        , sum(amt) as claimamt
from
        policy
left join
        claims
        using(policyID)
group by 1
```

11. The company is interested in learning if policies from "H" bonus agents have more claims than those of "L" bonus agent. Please write query which returns the total policy costs as well as the total amount paid for claims, by Agent bonus type. This should return two rows and three columns (bonustype, policycosts, claimamt).

```
select
        bonus
        , sum( policy.cost) as policycost
        , sum( rhs2.amt) as claimamt
from
        agent
left join
        household
        using(AgentID)
left join
        policy
        using(HHID)
left join
        (select sum(amt) as amt, policyid from claims group by 2) as rhs2
        using(policyID)
group by 1;
```

12. Write a query which returns the following information: (a) AgentID, (b) The number of policies that they are in charge of, (c) the total number of claims against those policies and (d) the percentage of policies that the agent is in charge of that have a claim against it. Specifically, if a policy has two claims against it, then it should only be counted once in the numerator.

```
select
        AgentID
        , count(distinct policyID) as numPolicies
        , count(distinct claimID) as numclaims
        , count(distinct claimID)::float / count(distinct policyID) as pct
from
        household
left join
        policy
        using( HHID)
left join
        claims
        using( PolicyID)
group by 1;
```

13. Write a query which has three columns: year of policy expiration (as an integer), month of policy expiration (as an integer) and a running sum of the dollar amount of policies expiring over time. There should be one row for each year-month combination.

```
select yr, mon
        , sum( cost) over( order by yr asc, mon asc
                  rows between unbounded preceding and current row)
from
(select
        date_trunc('year', enddate) as yr
        , date_trunc('month', enddate) as mon
        sum( cost) as cost
from
        policy
group by 1,2) as iq
```

14. We define a "house" risk zip code to be one where the total cost of "Fire", "Flood" and "Earthquake" is more than twice as large as the cost of "Car" policies within that zip code. What percentage of zip codes are high risk (return a single number, the percent of zip codes are "house" risk)? You may assume that there is at least one policy of each policy type within each zip code.

```
select
        avg( case when 2* hr > car then 1 else 0 end)  as pct
from
        (select
                zip
                , sum( case when PolicyType in ('Fire', 'Earthquake', 'Flood)
                        then cost else 0 end) as hr
                , sum( case when PolicyType = 'Car'
                        then cost else 0 end) as car
        from
                household
        left join
                policy
        using(policyID)
        group by 1) as innerq;
```

15. Write a query which returns the total expense amount of each "type" of Expense as well as the type of expense, making sure to sort from the largest total amount to the smallest total amount.

```
select
        type
        , sum(Eamt) as totalamt
from
        expenses
group by 1
order by 2 desc;
```

16. Write a query which returns the state with the largest number of "H" bonus agents. This should return 1 row and 1 column

```
select
        st
from
        agent
        where bonus = 'H'
group by 1
order by count(1) desc limit 1;
```

17. Of all the agents (agentID) who have ever sold a policy to zip code to 11217, which one had the highest average policy cost over all their policies that they sold?

```
select household.agentID
from
        household
left join
        policy
using( hhid)
where agentid in
        (select distinct agentid from household where zip = 11217)
group by agentID
order by avg( cost) desc limit 1;
```

18. Which household (HHID) has the most number of policies (assume that this is unique and that there exists a household with more policies than any other household)

```
select
        HHID
from
        policy
group by HHID
order by count(1) desc limit 1;
```

19. We are interested in the cash flow associated with claims, by policy type. Write a query which returns the 3 day moving average of the total claims (amt) received *per day* by policy type as well as the day-over-day percent change. There should be one row per day-policy type combination and four columns in the table (dt, policy type, moving average and percent change). To compute the percent

change, take today's total amount and divide by yesterday's. If today is 1/5/2018, then the three days in the moving average should be 1/3, 1/4 and 1/5. You can assume that every day of interest is represented in the policy table for all policies and that there are no days without claims for every policy type.

```
select
        dt, policytype
        , avg( amt) over( partition by policytype
                order by dt asc rows between 2 preceding and current row)
        , amt / lag( amt) over( partition by policytype
                order by dt asc )
from
(select
        sum( amt) as amt
        , dt
        , policytype
from
        policy
left join
        claims
        using( PolicyID)
        group by 2,3) as IQ
```

20. What month (just month, e.g. 12 for "December") is the most common end date for policies (by number of policies)?

```
select date_part('month', enddate)
from policy group by 1 order by count(1) desc limit 1;
```

21. It turns out that California Agents (those with state = "CA") were lying on costs associated with "Fire" Policies. In particular, every policy which was more than $500 had an additional $100 of fraud added to it. Write a query which returns two columns and one row. The first column should be the number of affected policies and the second should be the total amount (in dollars) of fraud.

```
select
        sum(1) as fclaims, 100 * sum(1) as dollarsfraud
from
        (select HHID, policyID, cost from policy
                where policytype = 'Fire' and cost > 1000) as lhs
left join
        household
        using(HHID)
where agentID in (select agentID from agent where st = 'CA') ;
```

22. What percentage of expenses have a claim amount associated with them larger than $1000? Return a single number which is this percent.

```
select
        sum( case when claims.amt > 1000 then 1 else 0 end) / sum(1) as pct
from
        expense
left join
        claims
        using(claimID);
```

23. Write a query which returns three columns. The first column should be policy type, the second should be the number of Multifam households which purchased that policy type and the third should be the number of NON Multifam households (Multifam = 0) that purchased it. There should be one row per policy type.

```
select
        policytype
        , sum( case when multifam = 1 then 1 else 0 end)
        , sum( case when multifam = 0 then 1 else 0 end)
from household
left join policy using( HHID )
group by 1;
```

24. Which policy type has the highest average cost? Return one row and one column.

```
select policytype
from policy
group by 1
order by avg( cost) desc
limit 1;
```

25. Agent's are paid based on the following formula: If they have a "H" type bonus plan, then they receive $100 for each policy they sell + 10% of the cost of that policy. If an agent has a low type ("L") bonus plan, they receive $200 for each policy they sell + 5% of the cost of the policy. Compute the total wages paid to each agent. This should return two columns: agentID and their wages.

```
select
        agentID
        , case
                when bonus = 'H' then 100 * numpolicies + .1 * totalcost
                else 200 * numpolicies + .05 * totalcost
from
        (select
                agentID
                , max( bonus ) as bonus
                , count(1) as numpolicies
                , sum( cost) as totalcost
        from
                agent
                left join
                household
                using( agentID)
        left join
                policy
                using(HHID)
        group by 1) as innerQ
```

26. **Bonus**: All the agent's in the agent table are from a movie. What movie?

9 2019 Exams

In 2019, a joint Pandas/SQL five week course was taught which had four exams. These are all four exams. Note that each exam contained both Pandas and SQL questions.

Exam #1

The following table contains information about athletes at a college. You can assume each name uniquely defines a person and that a person only plays a single sport. There are no two rows with the same name.

- **name:** The name of the applicant (string)
- **wgt:** The athlete's weight (in kg) (float)
- **hgt:** The athlete's height (in meters) (float)
- **state:** The state that the athlete is from (string)
- **mdt:** The date that the measurement was taken (date type)
- **sport:** The sport (all lowercase) that the person plays (string)
- **sex:** Is the athlete male or female ("M" or "F") (string)
- **injury:** Injuries (all lowercase) are listed here (if Null that means no injury). There will be only a SINGLE injury listed (string)
- The name of the table / DataFrame is **ath**. No need to use a schema or load the DataFrame.
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.

Figure D.3: *Ath* Table: 12,435 Rows

name	wgt	hgt	state	mdt	sport	sex	injury
Ringly Roberson	94.25	1.75	NY	8-1-2012	basketball	M	
Crash Bandicoot	88.25	1.62	MS	1-1-2012	rugby	M	shoulder
alligator reynolds	66.1	1.88	PA	8-5-2012	softball	F	

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Using SQL, write a query which returns the names (name only) of the top-6 tallest athletes who play basketball.

```
SELECT
        name
FROM
        ath
where sport = 'basketball'
order by hgt desc limit 6;
```

2. Using SQL, write a query which returns all basketball players (name only) shorter than 1.65 m from either New York ('NY') or Alabama ('AL'). Only include those athletes who are *not* injured.

```
select name from ath where
        state in ('NY', 'AL')
        and hgt < 1.65
        and sport = 'basketball'
        and injury is null;
```

3. Write an SQL query which returns the number of athletes from each state who *are* injured. This should be two columns by fifty rows.

```
SELECT
        state, count(1) as ct
from
        ath
where injury is not null
group by 1;
```

4. Write an SQL query which returns all rows and columns for female athletes.

```
select * from ath where sex = 'F';
```

5. We say that a sport is injury prone if 10% or more of the sport has injuries. Write an SQL query which returns a list of sports which are injury prone.

```
select sport from ath
group by 1
having sum( case when injury is not null then 1 else 0 end) ::float / sum(1) >= .10;
```

OR:

```
select sport from
(select sport
       , sum( case when injury is not null then 1 else 0 end)::float / sum(1) as rat
from ath
group by 1 ) as innerQ
where rat >= .1
```

6. We calculate the BMI ("Body Mass Index") of a person by taking their weight and dividing it by the height *squared*. Write a query which returns all rows and three columns: BMI, name and sport.

```
select wgt / hgt / hgt as bmi, name, sport
from ath;
```

7. Return a table with three columns: name, sport, and BMIFlag. BMIFlag should be equal to "0" if the BMI is less than or equal to 20, "1" if the BMI is greater than 20 and less than or equal to 30 and "2" otherwise.

```
select
        name, sport, wgt / hgt / hgt as BMI
        , case
                when wgt / hgt / hgt <= 20 then 0
                when wgt / hgt / hgt <= 30 then 1
                else 2 end as bmiflag
from
        ath;
```

8. If a person's BMI is greater than or equal to 30 they are defined as obese. Write a query which returns the *percentage* of athletes of each sport who are obese.

```
select
        sport,
        sum( case when wgt / hgt / hgt > 30 then 1 else 0 end )::float / count(1) as pctObese
from
        ath
group by 1;
```

9. Write a query which returns the sport, average height and average weight (by sport) for everyone whose name begins with the letter "A". Do not assume that the first letter of the person's name is always uppercase (there could be an "ann mitchell" in the table).

```
select
        sport
        , avg( hgt) as agh
        , avg(wgt) as agw
from ath
where upper(left(name,1)) = 'A'
group by 1;
```

10. Volleyball and basketball are known to be hard on the knees. Write a query which returns the percent of athletes who have "knee" injuries who play "volleyball" or "basketball" (combined) vs. the percent of knee injuries for sports which are NOT "volleyball" or "basketball". In other words, this should return two rows (one for volleyball / basketball and one for other) and two columns (one with a label for the sports included and one for the percent).

```
select
        case
                when sport = 'basketball' or sport = 'volleyball' then 'VB'
                else 'not VB'
        end as sportsType
, sum( case when injury = 'knee' then 1 else 0 end)::float / sum(1) as pctKnee
from ath
group by 1;
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrame named *ath* is already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Using Pandas, return an object (Series, DataFrame or array) of the names, and names only, of the top-6 tallest athletes who play basketball.

```
ath.loc[(ath.sport == 'basketball')].nlargest(6, hgt)['name']
```

2. Using Pandas, return all basketball players (name only) shorter than 1.65 m from either New York ('NY') or Alabama ('AL'). Only include those athletes who are not injured.

```
ath.loc[
(ath.sport=='basketball') & (ath.hgt < 1.65) & ((ath.state == 'AL') | (ath.state == 'NY'))
& (ath.injury.isna() == True), 'name']
```

3. Using Pandas, return an object which contains the names of sports (this should be without duplicates) which have a player with an injury to their "shoulder". You may assume that all the injuries in the table are lowercase.

```
ath.loc[(ath.injury=='shoulder')].sports.unique()
```

4. Female soccer players who have had a knee injury are going to be put on a special training program. Please return an object which contains *their names and only their names* sorted by state (A to Z).

```
ath.loc[ (ath.sport == 'soccer' ) & (ath.injury == 'knee') & (ath.sex == 'F')]
        .sort_values('state')['name']
```

5. We calculate the BMI ("Body Mass Index") of a person by taking their weight and dividing it by the height squared. In Pandas, return a DataFrame with three columns: BMI, name and sport for all rows.

```
ath = ath.assign(bmi = ath.wgt / ath.hgt / ath.hgt).loc[:, ['name', 'sport', 'bmi']]
```

6. Return a DataFrame with three columns: name, sport and BMIFlag. BMIFlag should be equal to "0" if the BMI is less than or equal to 20, "1" if the BMI is greater than 20 and less than or equal to 30 and "2" otherwise.

```
ath = ath.assign(bmi=ath.wgt / ath.hgt / ath.hgt).loc[:, ['name', 'sport', 'bmi']]
ath.loc[ (ath.bmi >= 30), 'bmiflag'] = 2
ath.loc[ (ath.bmi >= 20) & (ath.bmi < 30), 'bmiflag'] = 1
ath.loc[ (ath.bmi < 20), 'bmiflag'] = 0
```

7. There was an error with the weight machine and all weight-ins done in the month of March of 2012 were 10% too high. Please return an updated DataFrame with the information corrected. Note that this should include all rows and columns from the original dataset with wgt set 10% lower for miss-measured observations.

```
ath.loc[(ath.mdt.dt.year == 2012) & (ath.mdt.dt.month == 3), 'wgt']
        = .9* ath.loc[(ath.mdt.dt.year == 2012) & (ath.mdt.dt.month == 3), 'wgt']
```

8. Return a DataFrame which contains all information on anyone from Rhode Island ("RI") or who has a "knee" injury. Return this data sorted first by sport (A to Z), then by state (A to Z) and then by name (*Z to A*). Finally, upper case all returned names.

```
ath = ath.loc[(ath.state == "RI") | (ath.injury == "knee")]
        .assign(name=ath.name.str.upper())
        .sort_values(['sport', 'state', 'name'], ascending=[True, True, False])
```

9. Return a DataFrame which contains name, state and a flag which is equal to 1 if they play soccer and weight less than 70 kg or play basketball and weight less than 80 kg. The flag should be zero otherwise.

```
ath = ath.assign(flag = 0)
ath.loc[((ath.sport == "soccer") & (ath.wgt < 70))
        | ((ath.sport == "basketball") & (ath.wgt < 80)), 'flag'] =1
ath[['name', 'state', 'flag']]
```

Exam 2

The following table contains information about customer service interactions at a company. In particular, this has information about customers coming in and asking questions about their computer laptops.

- **serviceid:** This is an incrementing integer (int)
- **custid:** This is the ID for the customer (int)
- **pid:** This is the ID of the reported problem (e.g. battery problems are when pid = 2) (int)
- **servicedt:** This is the date that the service took place (date)
- **location:** This is the city and state of the service center (string)
- **result:** This is the diagnosis code for the device (e.g. result = 1 means solved) (int)
- **followup:** This contains information about if there was a follow up to the customer service (string)
- The name of the table / DataFrame is **cust**. No need to use a schema or load the DataFrame.
- The only column with Null values is "followup".
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.

Figure D.4: *cust* Table: 12,435 Rows

serviceid	custid	pid	servicedt	location	result	followup
1	21	12	1-1-2012	Livermore, CA	1	
2	21	23	1-5-2014	Livermore, CA	1	
3	26	11	1-15-2018	Livermore, CA	110	Refund
4	53	18	3-3-2011	Yuma, AZ	1	

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Write a query which returns all rows and columns for services which occur in March or December of *any* year.

```
select
        *
from
        cust
where date_part('month', servicedt ) = 3
        or date_part('month', servicedt) = 12;
```

2. All customers with problem 22 (pid = 22) in California (location ends with 'CA') had the wrong result. Please write a query which returns a list of customers (no duplicates, just their IDs) who need to be notified.

```
select distinct custid
from cust
where right(location,2) = 'CA'
and pid = 22;
```

3. Write a query which returns a time series of the number of services which have *no* followup. This should be aggregated to the month/year level, so that all no-followup service of the same month/year are combined. Make sure to return the data sorted from earliest to latest date. In other words, there should be two columns: one indicating the year / month (as a date) and one with the number of services without a followup.

```
select
        date_trunc('month', servicedt) as monyear
        , sum(1) as ct
from
        cust
where followup is null
group by 1
order by 1 asc;
```

4. We say that a service request is solved if there is no followup *and* the result is equal to 1. For each problem type (pid), report the total number of solved service requests.

```
select
        pid
        , count(1) as solved
from
        cust
where followup is null and result = 1
group by 1;
```

5. We say that a service request is solved if there is no followup *and* the result is equal to 1. Generate a dataset which has one row per location and three columns. The first column is location, the second is the *total* number of solved interactions at that location (over all time) and the third is the total number of all customer service interactions in August of 2014 at that location.

```
select
        location
        , sum( case when result = 1 and followup is null then 1 else 0 end) as tst2
        , sum( case when date_trunc('month', servicedt)::date = '08-01-2014'
                then 1 else 0 end ) as tstaug
from
        cust
group by 1;
```

6. Write a query which returns the locations which have more than 10 different types of problems (unique pid). For those locations, return two columns: one with the original location and one *with just the state abbreviation*. You can assume that all locations are of the form "cityname, state abbreviation" and that all state abbreviations are TWO characters long.

```
select
        location, right(location, 2) as state
from
        cust
group by 1
having count(distinct pid) > 10
```

NOTE THE ABOVE CAN BE GROUP BY 1 OR GROUP BY 1,2

7. We are trying to figure out how effective each service center is at solving different problems. Create a dataset with 3 columns: the first should be location, the second should be problem (pid) and the third should be the *percent* of problems of that type and at that location which are "solved" (result = 1 and no followup). Make sure to exclude any problem/location group with less than or equal to 10 rows.

```
select
        pid
        , location
        , sum( case when result = 1 and followup is null
                then 1 else 0 end)::float / sum(1) as pct
from
        cust
group by 1,2
having count(1) > 10;
```

8. Write a query which returns the frequency distribution of different problem types. This should return two columns. The first, called num, should be the number of times a problem appears in the dataset and the second, called "val" should be number of times this frequency occurs. For example, let's say that problems (pid) 27, 35 and 115 each appear 8 times in the table (and no other problem appears exactly 8 times in the table), then there should be a row which is (8,3). Note that each problem (pid) should only be tallied once.

```
select ct as num, count(1) as val
from
        (select count(1) as ct
        from cust group by pid) as innerq
group by 1;
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrame named *cust* is already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Generate a DataFrame which contains all columns and only those rows which have problem #3 (pid = 3) and are solved (result = 1 and there is no followup).

```
cust.loc[ (cust.result == 1 ) & (cust.pid == 3) & (cust.followup.isna())]
```

2. Generate a dataset which contains (a) location (b) the earliest date that a service occurred for that location, (c) the latest date that a service occurred at that location, (d) the number of unique problem's (pid) that the location experienced and (d) the total number of services that occurred at that location. Don't worry about column names, but make sure that location is a *column*.

```
(cust
        .groupby(['location'])
        .agg( {'servicedt' : ['max', 'min'], 'pid' : ['nunique', 'sum']})
        .reset_index()
)
```

3. Generate a DataFrame with three columns: location, day of the week ("dow", as an integer) and the number of rows with *any, non-null* followup that were at that location on that day-of-the-week. Note that it doesn't matter if this returns columns or indexes for any value.

```
cust['dow'] = cust.servicedt.dt.dayofweek
cust['flag'] = 0
cust.loc[ ~(cust.followup.isna), 'flag'] = 1
cust.groupby(['location', 'dow']).agg({ 'flag' : 'sum'})
```

4. Generate a dataset which has one row per location and three columns. The first column is location, the second is the *total* number of solved (result = 1 and followup is empty) interactions at that location (over all time) and the third is the total number of customer service interactions in August of 2014 at that location. Make sure that this is three *columns*.

```
cust.assign(succ=0, Aug2014=0)
cust.loc[(cust.result == 1) & (cust.followup.isna()), 'succ'] =  1
cust.loc[(cust.servicedt.dt.month == 8) & (cust.servicedt.dt.year == 2014), 'Aug2014'] = 1
cust.groupby('location').agg({'succ' : ['sum'], 'Aug2014' : ['sum']}).reset_index()
```

5. We are trying to figure out how effective each service center is at each location at solving different problems. Create a DataFrame with three *columns*: the first should be location ("location"), the second should be problem ("pid") and the third ("succ") should be equal to 1 or 0, depending on if the outcome was solved (result = 1 and no followup) or not. This should have the same number of rows as the original DataFrame. Name this DataFrame "tst".

```
cust['succ'] = 0
cust.loc[(cust.result == 1) & (cust.followup.isna()), 'succ'] = 1
tst = cust.loc[ :, ['location', 'pid', 'succ']]
```

6. Assume that you have the "tst" DataFrame from the problem above. We now want to calculate

the percent of customer interactions which are solved, aggregated to the problem (pid) and location level. In other words, using the DataFrame from the previous problem, generate a new DataFrame consisting of three *columns*: location, pid and the percent of interactions which were solved. Specifically this is the sum of "succ" divided by the number of rows. Make sure to remove any row which has less than 10 observations in the original DataFrame (as there is not enough data to conclude anything from them).

```
tst = tst.groupby(['location', 'pid']).agg({'succ' : ['sum', 'count']})
tst.columns = ['s1', 'c1']
tst = tst.loc[ (tst.c1 > 10) ]
tst.assign(pct=tst.s1/tst.c1)[['pct']].reset_index()
```

Exam #3

The following table contains information about doctors and their patients. At most, each patient has one doctor.

- Columns in the patients table
  - **patientid:** This is an auto-incrementing integer for the patient (int)
  - **doctorid:** This is the ID for the doctor that they see (int)
  - **hgt:** This the height of the patient in meters (float)
  - **birthdt:** This is the date of birth of the patient (date)
  - **wgt:** This is the weight of the patient in kg (float)
  - **city:** This the city that the person lives in (string)
  - **state:** This is the state that the person lives in (string)
  - **sex:** The sex of the patient (M/F) (string)
- Columns in the doctors table
  - **doctorid:** This is an auto-incrementing integer for the doctor (int)
  - **speciality:** This is the type of doctor (string)
  - **surgeon:** Is the doctor a surgeon (Y/N) (string)
  - **sex:** The sex of the doctor (M/F) (string)
- The names of each table are "patients" and "doctors". No need to refer to any schema or load a DataFrame.
- **There are some patients who have not yet been assigned doctors, so doctorid could be Null in the patients table.**
- **There are some doctors who were just hired who have not been assigned patients yet.**
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.
- Any two columns with the same name can be assumed to match.

Figure D.5: *Patients* Table: 12,435 Rows

patientid	doctorid	hgt	birthdt	wgt	city	state	sex
1	2	1.7	1-1-1997	58.2	Livermore	CA	M
2	18	1.65	1-5-1975	56.5	Livermore	CA	F
3	18	1.8	1-15-1994	67.3	Livermore	CA	M
4	7	1.93	3-3-1964	66.0	Yuma	AZ	M

Figure D.6: *Doctor* Table: 277 Rows

doctorid	speciality	surgeon	sex
1	Oncology	Y	M
2	ENT	N	F
3	General	N	M
4	Pediatric	Y	F

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Write a query which returns two columns. The first should be the doctorid and the second should be the number of patients that that doctor sees. This should be sorted from most to least patients seen. Make sure to *not* include doctors who have no patients and patients who have no doctors.

```
select
        doctorid, count(1) as ct
        patients
where doctorid is not null
group by 1
order by 2 desc;
```

2. A high-usage doctor is one that sees strictly more than 50 patients. Write a query which returns four columns. The first should be state, the second and third should be the average height and average weight of patients from that state and the fourth should be the number of patients from that state. Note that this should *only* include patients who have a high-usage doctor. There should be one row returned per state.

```
select
        state
        , avg( wgt) as aw
        , avg(hgt) as ah
        , count(1) as ct
from
        (select doctorid from patients
        group by doctorid
        having count(1) > 50) as lhs
left join
        patients
using(doctorid)
group by 1 ;
```

OR

```
select
        state
        , avg( wgt) as aw
        , avg(hgt) as ah
        , count(1) as ct
from
        patients
where doctorid in
        (select doctorid from patients
                group by doctorid
                having count(1) > 50)
group by 1 ;
```

3. Write a query which returns three columns. The first should be the speciality, the second should be the number of surgeons (surgeon = 'Y') within that speciality and the third should be the number of non-surgeons (surgeon = 'N') of that speciality.

```
select
        speciality
        , sum( case when surgeon = 'Y' then 1 else 0 end) as numsurg
        , sum( case when surgeon = 'N' then 1 else 0 end) as numnonsurg
from
        doctor
group by 1;
```

4. We say that a doctor is the same-sex as their patient if they are the same-sex as the patient (e.g. both Male or both Female). Write a query which returns two columns. The first should be the patientid and the second should be a flag which is equal to 1 if the patient and doctor are the same sex, zero otherwise. If a patient does not yet have a doctor the flag should be set to -1. This should have the same number of rows as the patients table.

```
select
        patientid
        , case
                when lhs.sex = rhs.sex then 1
                when rhs.sex is null then -1
                else 0
        end as flag
from
        patients as lhs
left join
        doctor as rhs
using(doctorid);
```

5. Create a dataset with five columns: year, speciality, number of patients who were born that year and have a doctor with that speciality, the average weight of patients who were born that year and have a doctor of that speciality and the average height of patients who were born that year and have a doctor of that speciality. Note that year should be returned as a number, not a date. Only include those patients who have been assigned doctors.

```
select
        date_part('year', birthdt) as yr
        , speciality
        , count(1) as numpatients
        , avg( hgt ) as avghgt
        , avg( wgt) as avgwgt
from
        patients
inner join
        doctor
using(doctorid)
group by 1,2;
```

6. We calculate the Body Mass Index of a person (BMI) as weight divided by height *squared*. Generate a table which has two columns: speciality and the max BMI of patients who see doctors of that speciality. Be careful to exclude doctors without patients and patients without doctors.

```
select
        speciality
        , max( wgt / hgt / hgt ) as maxbmi
from
        patients
inner join
        doctor
using(doctorid)
group by 1;
```

7. Write a query which returns four columns. The first should be speciality, the second should be sex (of the doctor), the third should be surgeon (the 'Y'/'N' flag) and the fourth should be the number of doctors of that type (where type is defined as speciality, sex and surgeon combination). If there is no doctor of that type then the count should be set to zero.

```
select lhs1.sex, lhs2.speciality, lhs3.surgeon, count(rhs.doctorid)
from
        (select distinct sex from doctor) as lhs1
cross join
        (select distinct speciality from doctor) as lhs2
cross join
        (select distinct surgeon from doctor) as lhs3
left join
        doctor as rhs
on lhs1.sex = rhs.sex
        and lhs2.speciality = rhs.speciality
        and lhs3.surgeon = rhs.surgeon
group by 1,2,3
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrames named *patients* and *doctor* are already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Generate a DataFrame with two columns. The first should be the doctorid and the second should be the number of patients that the doctor sees. This should be sorted from most to least patients seen. Make sure to *not* include doctors who have no patients and that it returns two *columns*.

```
(pd.merge(doctor, patients, on='doctorid', how='inner')
 .groupby( 'doctorid' )
 .agg({'patientid' : 'count'})
 .sort_values('patientid', ascending=False)
 .reset_index())

 This can be done without a merge, but you need to filter out the rows which don't match
```

2. Return a DataFrame which contains patientid, doctorid, birthdate and the speciality of the that patient's doctor. Only include those patients from California "CA" who have been assigned doctors.

```
lhs = patients.loc[ (patients.state == 'CA'), ['patientid', 'doctorid', 'birthdate'] ]
rhs = doctors.loc[ :, ['doctorid' , 'mtype']]
mrg = pd.merge( lhs, rhs, on='doctorid', how='inner')
```

3. A high-usage doctor is one that sees strictly more than 50 patients. Create a DataFrame which returns four *columns*. The first should be state, the second and third should be the average height and average weight of patients from that state and the fourth should be the number of patients from that state. Note that this should *only* include patients who have a high-usage doctor. There should be one row returned per state.

```
p1 = patients[['doctorid']].groupby('doctorid').agg({'doctorid' : ['count']}).reset_index()
p1.columns = ['doctorid', 'ct']
p1 = p1.loc[(p1.ct > 50)]

mrg = (pd.merge(p1, patients, on = ['doctorid'], how = 'inner')
        .groupby('state')
        .agg({'wgt' : ['mean'], 'hgt' : ['mean'], 'doctorid' : ['count']})
        .reset_index()
        )
```

OR

```
lst = patients[['doctorid']].groupby('doctorid').agg({'doctorid' : ['count']}).reset_index()
lst.columns = ['doctorid', 'ct']
lst = lst.loc[(lst.ct > 50), 'doctorid']

p1 = (patients.loc[(patients.doctorid.isin(lst)), :]
        .groupby('state')
        .agg({'wgt' : ['mean'], 'hgt' : ['mean'], 'doctorid' : ['count']})
        .reset_index()
        )
```

4. We calculate the Body Mass Index of a person (BMI) as weight divided by height *squared*. Generate a DataFrame which has two columns: speciality and the max BMI of patients who see doctors of that speciality. Be careful to exclude doctors without patients and patients without doctors. Make sure to return two *columns*

```
patients['bmi'] = patients.wgt / patients.hgt / patients.hgt

mrg = (pd.merge( patients, doctor, on='doctorid', how = 'inner')
       .groupby('speciality')
       .agg({'bmi' : ['max']})
       .reset_index()
      )
```

5. Create a DataFrame with three *columns*. The first should be state, the second should be number of patients from that state (total), the third should be the number of patients from that state which do NOT have doctors.

```
mrg = pd.merge( patients, doctor, on ='doctorid', how='left').assign(nodoc=0)
mrg.loc[ mrg.doctorid.isna(), 'nodoc'] = 1
mrg = mrg.groupby('state').agg({'doctorid' : ['count'], 'nodoc' : ['sum']}).reset_index()
```

6. How many patients do not have a doctor assigned?

```
patients.loc[patients.doctorid.isna(), 'patientid'].count()
```

Exam #4

The following tables contains information about Uber drivers, their rides and reviews.

- Columns in the *drivers* table:
  - **did:** This is an auto-incrementing integer ID for the driver (int)
  - **state:** This is the state that the driver lives in (string)
  - **prom:** Is the driver on a promotion? (Y/N) (string)
- Columns in the *rides* table:
  - **rid:** This is an auto-incrementing integer for the ride (int)
  - **ridets:** This is the date and time that the ride occurred (date)
  - **did:** This is ID for the driver (int)

  - **air:** This is a flag (Y/N) for if the trip went to the airport (string)
  - **length:** This is the ride length in km (float)
- Columns in the *reviews* table:
  - **rid:** This is the ride which was reviewed (int)
  - **review:** This is the review (star scale: 1 to 5) (int)
- The names of the tables and DataFrames are *drivers, rides* and *reviews*.
- Assume that there are no Null values in any of the tables.
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.
- Do not create any views.
- Any two columns with the same name can be assumed to match.
- **Not all rides will have reviews. A ride can have, at most, one review.**
- **Not all Drivers may have rides. When a driver first signs up they will not have any rides.**
- **DO NOT USE CTE ("with"), but you CAN use any analytic / window function.**

did	state	prom
1	CA	Y
2	MN	N
3	CA	N
4	CA	Y

rid	ridets	did	air	length
1	1-1-2012 10:24 AM	45	N	1.25
2	12-23-2012 12:22 PM	45	N	23.45
3	7-6-2013 4:13 AM	112	Y	11.17
4	5-5-2014 1:23 PM	1125	N	.75

rid	review
1	5
23	4
35	4
45	1

Figure D.7: *driver* table (27,777 rows), *rides* table (454,123 rows) and *reviews* table (137,145)

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Write a query which returns two columns and a row for each state in the table. The first column should be the state and the second should be the number of rides completed by drivers from that state.

```
select
        state
        , count(1)
from
        drivers
join
        rides
using( did )
group by 1;
```

2. Write a query which returns two columns and one row per driver. The first column should be the driver's ID number (did) and the second should be a Y/N flag if the driver's first ride was to the airport or not.

```
select
        distinct did, airflag
from
        (select
                did
                , first_value( air ) over(partition by did order by ridets asc) as airflag
        from
                rides ) as innerQ;

Note that you could use an aggregate function in the outer query, but there has to be an
inner query.
```

3. Write a query which returns the following: state of the driver, total rides completed by drivers from that state and the average review of drivers from that state. Make sure to sort this from highest to lowest average review. Exclude any state with strictly less than 1,000 riders served. This should have one row per state.

```
select
        state
        , sum(1) as totalrides
        , avg( review ) as avgreview
from
        drivers
join
        rides
        using( did )
left join
        reviews
        using( rid )
group by 1
having count(distinct rid ) >= 1000
order by 3 desc;
```

4. Write a query which returns three columns. The first column should be year (as an integer), the second column should be total rides from that year and the third should be the *running or cumulative total of all rides, excluding the current year*. There should be one row per year.

```
select
       yr
       , numrides
       , sum(numrides) over(order by yr asc rows between unbounded preceding and 1 preceding) as cumsum
from
       (select
              date_part('year', ridets) as yr
              , sum(1) as numrides
       from
              rides
       group by 1 ) as iq;
```

5. Write a query which returns four columns: The first should be the driver id ('did'), the second should be the total number of rides, the third should be the number of their rides with a review and the fourth should be the number of rides with 5-star reviews (review = 5). Only include rides from 2018

and make sure to sort the drivers from most to least rides with reviews. Exclude drivers without any rides.

```
select
        did
        , count(1) as numrides
        , sum( case when review is not null then 1 else 0 end) as num_with_reviews
        , sum( case when review = 5 then 1 else 0 end) as five_star_reviews
from
        rides
left join
        reviews
using(rid)
where date_part('year', ridets) = 2018
group by 1
order by 3 desc;
```

6. Write a query which returns two rows and two columns. One row contains the phrase "LT10" and have the average review for rides that were (strictly) less than 10 km and the other row should be "MT15" and should be the average review for rides which are (strictly) more than 15 km. There should only be two rows.

```
select
        case
                when length < 10 then 'LT10'
                when length > 15 then 'MT15'
        end as flag
        , avg( review)
from
        rides
join
        reviews
using(rid)
where length < 10 or length > 15
group by 1;
```

7. *Without* using an analytic function, return one row and two columns. This should be the transpose of the data in the previous question. The first column should be the average review for rides of less than (strictly) 10 km and the second should be the average review for rides of more than (strictly) 15 km. It should only have one row.

```
select
        avg( case
                when length < 10 then review
                else null
        end ) as lt10
        , avg( case
                when length > 15 then review
                else null
        end        ) as mt15
from
        rides
join
        reviews
using(rid);
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that DataFrames named *drivers*, *rides* and *reviews* are already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Return a DataFrame which has two columns and a row for each state. The first column should be the state and the second should be the number of rides completed by drivers from that state.

```
pd.merge( rides, drivers, on='did', how='left')
        .groupby('state')
        .agg({'rid' : ['count']})
```

2. We are interested in studying the effect of driver promotion (prom = 'Y') on long rides (strictly greater than 10 km). Return a dataset which contains two rows (one for prom='Y' and one for prom = 'N') and has three *columns*. The first should be prom, the second should the number of long rides to the airport (air='Y') the third should be the number of long rides *not* to the airport (air='N').

```
mrg = pd.merge( drivers, rides, on='did', how='left')
mrg = mrg.loc[(mrg.loc[:, 'length'] > 10), :]

mrg.loc[ : , 'airflag' ]  = 0
mrg.loc[ (mrg.loc[:, 'air']) =='Y' , 'airflag' ]  = 1

mrg.loc[ : , 'Nairflag' ]  = 0
mrg.loc[ mrg.loc[:, 'air']=='N' , 'Nairflag' ]  = 1

mrg = (mrg
        .groupby('prom')
        .agg( { 'airflag' : ['sum'], 'Nairflag' : ['sum']})
        .reset_index()
        )
```

3. Return a DataFrame with two rows and two *columns*. One row contain the phrase "LT10" and then has the average review for rides that were (strictly) less than 10 km and the other row should be

"MT15" and should be the average review for rides which are (strictly) more than 15 km. There should only be two rows.

```
mrg = pd.merge( rides, reviews, on='rid', how='inner')

mrg = (mrg
       .loc[(mrg.loc[:, 'length'] < 10) | (mrg.loc[:, 'length'] > 15), :]
      )

mrg.loc[:, 'flag'] = 'LT10'
mrg.loc[(mrg.loc[:, 'length'] >15), 'flag'] = 'MT15'

mrg.groupby('flag').agg({'review' : ['mean']}).reset_index()
```

4. There is a worry that there is a relationship between ride length and the percentage of rides with reviews. To analyze this we will create a flag called "lflag", which is equal to 1 if the ride length < 2 km, 2 if the length is $>= 2$ and < 5 km and 3 if the length $>= 5$ km. Create a dataset which has the following columns: lflag, state, number of *rides* which are of that flag-state combination, the number of those rides with reviews and the average review from rides of that lflag-state combination.

```
mrg = pd.merge( drivers, rides, on ='did', how='inner')
mrg = pd.merge( mrg, reviews, on = 'rid', how='left')

mrg.loc[:, 'lflag'] = 1
mrg.loc[(mrg.loc[:, 'length']) >=2) & (mrg.loc[:, 'length'] <5) , 'lflag'] = 2
mrg.loc[(mrg.loc[:, 'length'] >5)  , 'lflag'] = 3

mrg.groupby( ['state', 'lflag'])
        .agg({ 'review' : ['mean', 'count'], 'rid' : ['count'] })
        .reset_index()
```

5. What is the average review for all rides from drivers which have *ever* had a ride over 60km? This should return a single number.

```
lst = rides.loc[(rides.loc[:, 'length'] > 60), 'did'].drop_duplicates()

rds = rides.loc[rides.loc[:, 'did'].isin(lst), :]

pd.merge(rds, reviews, on='rid', how='inner')['review'].mean()
```

10 USF's student table

Use the following tables when answering the questions. The tables below consistent of information from USF's undergraduate student system.

1. All columns with the same name (such as CID, SID, etc.) can be assumed to represent the same thing and join easily.
   - **SID** is an integer and is unique for each student.
   - **CID** is an integer and is unique for each class.
   - **PID** is an integer and is unique to each professor.
2. Student Table: Each row is a unique student currently enrolled at USF.
   - **BirthDate** is a date while **ParentCity** and **ParentState** represent the city and state where their parent's live while **CurrentCity** and **CurrentState** are where the student currently resides. Both are varchar
3. Classes Table: Maps currently enrolled students to what classes they enrolled in over the course of their studies.
   - **EnrollDate** is the date they they enrolled in the class.
   - **Semester** is the a varchar(10) which is equal to "Summer", "Spring" or "Fall." while **Yr** is the year, in integer form.
   - **Grade** is the grade they received. **If the student is currently taking the class or if the student withdraws the course, then grade is null.**
4. Catalog Table: Represents the list of classes available at USF for all currently enrolled undergraduates to take.
   - **Department**, **ClassName** and **Units** represent the department, name and number of units of each class.
5. WithDraw Table: Represents information about currently enrolled students who withdraw from a class.
   - **DropDate** is the date the student withdrew from the class.
6. Some other assumptions:
   - A student cannot repeat a class and a student can only withdraw from a class a single time. In other words, students get one shot at taking each class.
   - **All currently enrolled students are in the Class table.**

Figure D.8: *Student* Table, 4,525 Rows

SID	BirthDate	ParentCity	ParentState	CurrentCity	CurrentState	major
1	01-01-99	Oakland	CA	San Francisco	CA	Math
3	01-21-99	Fremont	CA	San Francisco	CA	Undeclared
2	11-08-98	Philadelphia	PA	Burlingame	CA	Comp. Sci.

Figure D.9: *Classes* Table, 72,485 Rows

SID	CID	EnrollDate	Semester	Yr	Grade
12	101	01-11-2015	Spring	2015	3.7
12	109	01-11-2015	Spring	2015	
12	800	01-10-2016	Spring	2016	0.0
12	1923	01-10-2016	Spring	2016	4.0
12	111	08-15-2016	Fall	2016	
12	546	08-15-2016	Fall	2016	
12	999	08-15-2016	Fall	2016	

Figure D.10: *Catalog* Table, 1,288 Rows

CID	PID	Department	ClassName	units
800	12	Math	Calculus I	4
801	22	Math	Calculus I	4
1118	102	English	Intro. to Shakespeare	2
45	888	Physics	Freshmen Seminar	2

Figure D.11: *Withdraw* Table, 14,888 Rows

SID	CID	DropDate
12	109	03-11-2015
1114	888	03-18-2015
765	2345	10-22-2015
9022	891	05-21-2015

1. Draw a picture showing the four tables and how they connect. Make sure to pay attention to which columns match with which table.
2. How many students are there currently enrolled?

```
select
        count(distinct sid)
from
        students;
```

3. What are the top 10 currently enrolled students (SID only) in terms of number of classes ever enrolled? E.g. which SID's enrolled in the most classes.

```
select
        sid
from
        classes
group by sid
order by count(1) desc
limit 10;
```

4. For each student (SID only), report the number of distinct departments that they have ever taken classes from.

```
select
        sid, count(distinct department) as numDepts
from
        students
left join
        catalog
using(cid)
group by 1;
```

5. What are the top five currently enrolled students (SID only) in terms of number of classes withdrawn?

```
select
        sid
from
        withdraw
group by sid
order by count(1) desc
limit 5;
```

6. Which department has the most popular major?

```
select
        major
from
        student
group by 1
order by count(1) desc
limit  1;
```

7. Which department has the largest number of currently enrolled students withdrawing from their classes?

```
select
        department
from
        withdraw
left join
        catalog
using(cid)
group by department
order by count(1) desc
limit 1;
```

8. For each student (SID only), report both how many classes they are enrolled in and how many they have withdrawn from.

```
select
        sid, count(classes.sid) as numEnrolled, count(withdraw.cid) as numWithDrawn
from
        classes
left join
        withdraw
using(sid, cid)
group by 1;
```

9. How many currently enrolled students have never withdrawn from a class?

```
select
        count( distinct sid)
from
        students
where sid not in
        (select distinct sid from withdraw);
```

We could also use a JOIN to answer this question:

```
select
        students.sid
from
        students
left join
        withdraw
on students.sid = withdraw.sid
group by 1
having count(withdraw.sid) = 0;
```

10. Which currently enrolled student (SID only) has the highest percentage of their classes withdrawn?

```
select
        sid
from
        classes
left join
        withdraw
using(sid, cid)
group by 1
order by count( withdraw.dropdate)::float / count( classes.sid) desc
limit 1;
```

11. Which department (department name) had the highest average time between date enrolled and date withdrawn (in number of days), for those currently enrolled students who withdrew from a class in that department?

```
select
        department
from
        classes left join withdraw using( sid, cid)
        left join catalog using( cid )
where dropdate is not null
group by 1
order by avg( dropdate - enrollate)  desc
limit 1;
```

12. Report, for each student the number of classes that they have taken in each department. Make sure to include rows (with a count of zero) for those departments from which a student has never taken a class.

```
select
        lhs.sid
        , rhs.department
        , count(classes.sid) as numclasses
from
        (select distinct sid from classes) as lhs
cross join
        (select distinct department from catalog) as rhs
left join
        classes
        on lhs.sid = classes.sid and rhs.sid = classes.sid
group by 1;
```

13. Of those currently enrolled students who withdraw from at least one class, what is the average number of classes that they withdrew from?

```
select avg( ct )
from
        (select count(1) as ct
        from withdraws group by sid)
        as innerQ;
```

14. Which professor (PID only) had the average highest percentage of their students withdraw from a class? This should be an average over classes as professors can teach multiple courses.

```
select pid, avg( pct) as apct
from (
        select pid, cid, sum(coalesce( wd, 0))::float / count(1) as pct
        from
                (select sid, cid from classes ) as lhs
        left join
                (select sid, cid, 1 as wd from withdraw) as rhs
                using(sid, cid)
        left join
                catalog
        using(cid)
        ) as innerQ
group by 1
order by 2 desc
limit 1;
```

15. Calculate each currently enrolled student's GPA, making sure to weigh it by the number of units.

```
select
        SID, sum( units * grade ) / sum( units) as GPA
from
        classes
left join
        catalog
using( cid )
where grade is not null
group by 1;
```

16. Of all classes with more than 15 currently enrolled students, which ones have an average grade given of more than 3.5? Make sure to not include withdraws.

```
select cid
from
classes
where grade is not null
group by 1
having count(1) > 15 and avg(grade) > 3.5;
```

17. Write a single query which calculates the average GPA of currently enrolled students who (1) withdraw from more than 10% of their classes and (2) withdraw from less than 10% of their classes.

```
select avg( GPA) as avgGPA, more_than10
from
(select
        sid
        , sum( units * grade ) / sum( units) as GPA
        , case when wd::float/ccount > .1 then 1 else 0 end as more_than10
from
        (select sid, sum( withdraw.dropdate) as wd, sum(1) as ccount
                from classes left join withdraw using( sid, cid)
                where grade is not null or dropdate is not null
                group by 1 ) as studentInfo
        left join classes using(sid)
        left join catalog using(cid)
        group by 1)
as innerQ
group by more_than10;
```

11 F&F Example

- In this section we are going to use the following set of hypothetical tables about a company (called F&F). When writing queries about this data you can assume that all tables are within the same schema.
- For this company, each transaction (or sale) has a single item and sales person attached to it.
- All columns with the same name can be assumed to match and merge.
- Transaction Table:
  - SID, ItemID and TID are all integers, Amount is a float and TransTS is a timestamp[1]
  - TID is unique per transaction and stands for "Transaction ID"
  - SID is unique per sales person and stands for "SalesPersonID"
  - ItemID is an ID that is unique to an item.
- Refund Table:
  - RefundTS is a timestamp
  - RefundAmount is a float, it is always less than or equal to the transaction amount
  - A transaction can only have a single refund, but not all transactions will have refunds.

Table D.21: *Transaction* Table, 12,525 Rows

SID	TransTS	ItemID	TID	Amount
1	01-01-14 08:10:25 PST	124	1	15000.18
3	01-21-14 18:10:25 PST	888	2	25000.45
2	11-08-14 12:09:25 PST	125	12	1854.65

Table D.22: *Refund* Table, 385 Rows

TID	RefundTS	RefundAmount
12	03-14-14 14:12:18 PST	1,854.65

Table D.23: *SalesPerson* Table, 50 Rows

SID	Name	MobilePhone	State	BonusStructure
1	Brian O'Conner	111-222-3333	CA	High
2	Dominic Torretto	444-555-6666	CA	High
3	Letty	777-888-9999	CA	High
4	Lightning McQueen	111-333-5555	AZ	Low
5	Tow Mater	222-444-6666	AZ	Low

[1] You can assume that the date functions introduced in class work on this data.

Table D.24: *Item* Table, 50 Rows

ItemID	BaseCost	Name
1	4.99	Washer Fluid
2	14.89	Brake Fluid
3	56.78	Brake Pads (Generic)

1. What are the top five sales people (SID only) in terms of number of sales?

```
select
    count(1) as numsales
    , sid
from
    transaction
group by 2
order by 1 desc
limit 5;
```

2. What are the top five sales people (Name) in terms of number of sales?

```
select
    count(1) as numsales
    , name
from
    transaction
left join
    salesperson
using( sid )
group by name
order by 1 desc
limit 5;
```

3. What are the top 10 sales people (Name) in terms of dollars of sales?

```
select
    name
from
    transaction
left join
    salesperson
using(sid)
group by 1
order by sum( amount) desc
limit 10;
```

4. Which mobile phone area code (first three digits) has the highest number of sales?

```
select
    left( mobilePhone, 3) as areaCode
from
    transaction
left join
    salesperson
    using(sid)
group by 1
order by sum( amount) desc
limit 10;
```

5. Calculate the total of revenue from each state.

```
select
    state
    , sum( amount) as state_amt
from
    transaction
left join
    salesperson
using(sid)
group by 1
```

6. Calculate the total revenue from all states.

```
select sum(amount) as totalsales from transaction;
```

7. Calculate the *percentage* of revenue from each state.

```
select lhs.state, lhs.state_amt / rhs.totalsales
from
    (select
        state
        , sum( amount) as state_amt
    from
        transaction
    left join
        salesperson
    using(sid)
    group by 1) as lhs
cross join
    (select sum( amount) as totalsales from transaction ) as rhs;
```

8. What was the total refunded amount for each sales person (SID only)?

```
select
    SID
    , sum( refundamount) as refamt
from
    transactions
left join
    refunds
using( TID )
group by 1;
```

9. How many sales people had no refunds? When thinking about this problem remember that a sales person has multiple transactions and each transaction *may* have a refund. We need to make sure that there are no refunds for any of the transactions for a sales person.

```
select
    sid
from
    transactions
left join
    refunds
using( TID )
group by 1
having count( refunds.refundamount ) = 0;
```

10. Which sales person (name only) had the highest percentage of refunds, based on number of transactions?

```
select
    sid
from
    transaction
left join
    refund
on transactions.tid = refund.tid
left join
    salesperson
on transaction.sid = salesperson.sid
group by 1
order by sum(refundamount)
```

11. For each salesperson (Name), what percentage of their sales were refunded?

```
select
    name
    , sum(refundamount)/sum(amount) as pct_refund
from
    transactions
left join
    salesperson
    using(sid)
left join
    refunds
    using(tid)
group by 1;
```

12. What is the average percentage refunded, on those transactions with refunds?

```
select
    avg( refundamount / amount ) as avg_ref_pct
from
    transactions
inner join
    refunds
    using( tid );
```

13. For each month, report the percentage of sales refunded by both number of refunds and dollars. Assume that a refund can occur in any month after a sale, but that all refunds are in these tables.

```
select
    date_part('month', transTS) as sales_month
    , count( refunds.refundamount )::float
        / count(transactions.amount) as pct_ref
    , sum( refunds.refundamount )
        / sum(transactions.amount) as pct_dol_ref
from
    transactions
left join
    refunds
using( tid )
group by 1;
```

14. What percentage of sales had (1) returns above 20% (by dollar) and (2) returns above 50% (by dollar) of their value? Write a single query that returns two values.

```
select
    sum( case when refundamount > .2 * amount then 1 else 0 end)
        / count(1) as pct_above_20
    , sum( case when refundamount > .5 * amount then 1 else 0 end)
        / count(1) as pct_above_20
FROM
    transactions
left join
    refunds
using(tid)
```

15. Let's calculate which item (Name) is the most returned, by percent of returns:

```
select
    item.name
from
    transactions
left join
    refunds
    using(tid)
left join
    item
    using( itemID)
group by item.name
order by count(refunds.tid)::float / count( transactions.tid) desc
limit 1;
```

16. Calculate the total amount of BaseCost returned, by item name.

```
select
    item.name
    , sum( BaseCost) as amtReturned
from
    refunds
left join
    transactions
    using(tid)
left join
    item
    using( itemID)
group by item.name;
```

12 The Sales Rollup

In this exercise we are going to write a ton of queries about an interesting sales dataset.

The Data

Table D.25: *Sales* Table, 10,250 Rows

SID	CID	amount	TID	sales_dt
1	4	13.55	1	1/11/2011
2	12	18.99	2	12/22/2012
2	12	22.01	3	1/12/2013

Table D.26: *Expenses* Table, 425 Rows

SID	CID	exp_dt	amount	e_type
12	18	5/4/2012	112.24	Dinner
2	44	10/10/2010	112.24	Sport

Table D.27: *Transactions* Table, 25,254 Rows

TID	IID	num
1	12	1
1	18	1
2	45	1
3	18	1

Table D.28: *Client* Table, 152 Rows

CID	Name	Address	City	st	Zip
1	John Smith	12 Blue Bell street	Hayward	CA	94552
2	Julia Xue	14 Howard Street	Danville	VA	24543

Table D.29: *SalesPerson* Table, 22 Rows

SID	Name	Address	City	st	Zip	StartDt	EndDt
1	Rachel Adams	27796 Hanover Hill	Hanover	NY	14081	12-12-2011	1-3-2014
2	Julia Xue	60 Darwin Court	Mobile	AL	36602	01-11-2012	7-7-2013

Table D.30: *Items* Table, 440 Rows

IID	ItemName	Cost
1	10 lbs. Concrete	12.95
2	20 lbs. Concrete	19.95
3	30 lbs. Concrete	27.95

Questions

Main Questions

1. How many sales people are there from CA?

Table D.31: *Regions* Table, 50 Rows

ST	Region
CA	West
PA	East
AL	South
VT	North

2. In how many different states does this company have clients?
3. What is the average, min and max cost of an item which is for sale?
4. What is the average, min and max cost of items which have concrete in their name?
5. What is the average, min and max cost of items which have concrete in their name vs. those which do not?
6. Write a query which returns the total sales and number of sales per month.
7. How many sales did each sales person have?
8. How many states are in each region?
9. How many transactions contained item 12?
10. What was the average number of items per transaction? (Think about what to do with the number of items column)
11. What are the top-5 items sold in units-sold?

A bit harder

1. The total dollars sold and number of items, per salesperson.
2. How many currently employed sales people?
3. Write a query which returns the total number of items that each salesperson sold.
4. What was the average amount expensed per-client and per-salesperson? (Need to do two queries?)
5. How many sales people are in each region?
6. How many sales are from each region? (How would you measure this?)
7. For each client, compute the Revenue - expenses.
8. For each client, compute the profit (revenue - expenses - costs).
9. For each client region, compute the total revenue.
10. For each salesperson region, compute the total revenue.
11. How many clients are in each region?
12. What is our profit per region? (How would you measure this?)
13. **How would you calculate profit per transaction?**
14. How many different sales people sold each item?
15. How many distinct items has each sales person sold?

16. For each region (based on client), return the # of salespeople servicing the region, the number of items sold in that region, the number of transactions that occurred in that region and the total cost of the items sold in that region.

17. Calculate, for each client region, the number of sales that occurred in the same region as the salesperson.

18. Because of weird tax rules, we need to collect a tax of 3% of revenue for those transactions where the client is in the western region. How much tax do we owe per month?

13 Sales Work Through

This assignment contains information up to and including aggregate functions and dates. We will be using the table sp.mast which contains the following columns:

Column Name	Description
SID	This is the Salesperson's ID number.
spname	This is the Salesperson's name.
daysworked	The number of total days that the salesperson has worked.
itemssold	The total number of items that the salesperson sold.
bonus	If the salesperson was under the high- or low- bonus plan.
region	What region they worked in.
startdate	The date that they started.
salesdt	The date that the particular sales occurred.
descr	A description of the item sold.
cost	The cost of the item (in cents).
prc	The price of the item (in cents).

Figure D.12: Information regarding SP.MAST

1. How many total sales are in the database?

```
select sum(1) from sp.mast;
```

2. How many sales were completed each month?

```
select count(1), date_trunc('month', salesdt)
from sp.mast group by 2 order by 2;
```

3. How many sales were completed by region?

```
select count(1), region
from sp.mast group by 2 order by 2;
```

4. Using another tool (such as Excel or Google Docs) prepare a graph which contains the following information:
   (a) Month
   (b) Number of sales for that month
   (c) Total Revenue from sales that month
   (d) Total cost of items from that month

```
select
        count(1) as ct
        , date_trunc('month', salesdt)
        , sum(prc) / 100.0 as totalRev
        , sum(cost) / 100.0 as totalCost
from sp.mast group by 2 order by 2;
```

5. Using another tool (such as Excel or Google Docs) prepare a graph which shows, by region and month, the amount of *profit* generated. This should have four lines – one for each region.

```
select
        date_trunc('month', salesdt) as mnt
        , region
        , sum(prc - cost) / 100.0 as prft
from sp.mast
group by 1,2 order by 2,1;
```

6. Identify the top 7 sales people (name and SID) in terms of total revenue generated.

```
select spname, sid
from sp.mast
group by 1,2
order by sum( prc) desc limit 7;
```

7. Plot the monthly revenue (combined) for the top 7 salespeople.

```
select sum(prc) , date_trunc('month', salesdt) as mnt
from sp.mast
where sid in
        (select sid
        from sp.mast
        group by 1
        order by sum( prc) desc limit 7)
group by 2
order by 2;
```

8. Create a pie chart which breaks down all revenue into one of four categories: (a) the salesperson worked less than 10 days (b) the salesperson worked between 10 and 20 days (c) the salesperson worked between 20 and 50 days and (d) the salesperson worked more than 50 days.

```
select sum(prc) as rev,
        case
                when daysworked < 10 then 1
                when daysworked < 20 then 2
                when daysworked < 50 then 3
                else 4
        end
from
        sp.mast
group by 2;
```

9. Calculate the average profit *per region*. In particular, calculate the profit per region and then find the average over the regions.

```
select
        avg(prft) as prft
from
        (select sum(prc - cost) as prft
                from sp.mast
                group by region) as innerq;
```

10. We want to understand where we should concentrate our business – high margin items (which are those where the profit margin (price - cost)/cost >= 23%) or mid margin items (those where the profit margin is between 23% and 18%) or low-margin items (profit margin less than 18%). Create a dataset which identifies, for each distinct item, what margin group (high-, mid-, or low-) it is in.

```
select
        iid
        ,case
                when (prc-cost)::float / cost >= .23 then 'high'
                when (prc-cost)::float / cost >= .18 then 'mid'
                else 'low' end as margin
from
(select distinct prc, cost, iid from sp.mast) as innerQ
```

11. What was the average number of days that a salesperson worked? In particular, identify the number of days that each salesperson worked and then calculate the average of it.

```
select avg( days) as avgdays
from
        (select sid, max(daysworked) as days
        from sp.mast group by 1) as innerQ;
```

12. Salespeople are paid based on one of two plans: The "H" bonus plan which means that they are paid $130 per day, but receive a 10% commission or the "L" bonus plan which they are paid $150 per day, but receive a 5% commission. Calculate the amount of money that each salesperson made *on the non-commission* part.

```
select sid,
case when max(bonus) = 'H' then 130.0*max(daysworked)
else 150*max(daysworked) end as totalcomp
from sp.mast group by 1;
```

13. Calculate the total compensation paid to each salesperson, including both the commission and non-commission portion.

```
select
       sid
       , case
               when max(bonus) = 'H' then 130.0*max(daysworked) + sum(prc/100.0)*.1
               else 150*max(daysworked) + sum(prc/100.0)*.05 end as totalcomp
from sp.mast group by 1;
```

14. The company is thinking about changing the bonus plan so that the "H" bonus plan would be $100 per day, but 20% commission and the "L" bonus Play would be $175 per day with no commission.

Calculate the number of salespeople, within each bonus plant, that would be better off under the new vs. the old plan.

```
select count(1), bh, case when totalcomp > totalcomp2 then 1 else 0 end
from (
select
        sid, max(bonus) as bh
        , case
                when max(bonus) = 'H' then 130.0*max(daysworked) + sum(prc)/100.0*.1
                else 150*max(daysworked) + sum(prc)/100.0*.05 end as totalcomp
        , case
                when max(bonus) = 'H' then 100.0*max(daysworked) + sum(prc)/100.0*.2
                else 175*max(daysworked) end as totalcomp2

        from sp.mast

        group by 1) as iq group by 2,3;
```

15. Write a query which returns the following data per region:
    (a) The total profit (price - cost)
    (b) The average profit per salesperson
    (c) The total number of items sold

```
select
        region
        , sum(prc - cost) as prft
        , sum(prc - cost)::float / count(distinct sid) as pftPerPerson
        , count(1) as numSol
from
        sp.mast
group by 1;
```

16. Write a query which returns the following data, this time by margin – high margin items (which are those where the profit margin (price - cost)/cost >= 23%) or mid margin items (those where the profit margin is between 23% and 18%) or low-margin items (profit margin less than 18%).
    (a) The total profit (price - cost)
    (b) The average profit per item
    (c) The total number of items sold

```
select
        case
                when (prc-cost)::float / cost >= .23 then 'high'
                when (prc-cost)::float / cost >= .18 then 'mid'
                else 'low' end as margin

        , sum(prc - cost) as prft
        , sum(prc - cost)::float / count(1) as pftPeritem
        , count(1) as numSol
from
        sp.mast
group by 1;
```

14 Sales Work Through, part 2

This assignment is a follow-up to the previous Sales Walk Through. This is the same company and data as the previous case, but now there are multiple tables, rather than a single combined table. The following is a data dictionary:

Column Name	Description
iid	This is the ID number of an item.
descr	A description of the item sold.
cost	The cost of the item (in cents).
prc	The price of the item (in cents).

Figure D.13: Information regarding sp.itemlist, which contains information about each item.

Column Name	Description
SID	This is the Salesperson's ID number.
iid	This is the ID number of an item.
salesdt	The date that the particular sales occurred.

Figure D.14: Information regarding sp.itemmap, which contains a map between salesperson, the date of the sale and what was sold.

Column Name	Description
SID	This is the Salesperson's ID number.
spname	This is the Salesperson's name.
daysworked	The number of total days that the salesperson has worked.
bonus	If the salesperson was under the high- or low- bonus plan.
region	What region they worked in.
startdate	The date that they started.

Figure D.15: Information regarding sp.sp, which contains information on each salesperson.

1. How many total sales are in the database?

```
select sum(1) from sp.itemmap;
```

2. How many sales were completed each month?

```
select count(1), date_trunc('month', salesdt)
from sp.itemmap group by 2 order by 2;
```

3. How many sales were completed by region?

```
select count(1), region
from
        sp.sp
left join
        sp.itemmap
using(SID)
group by 2 order by 2;
```

4. Using another tool (such as Excel or Google Docs) prepare a graph which contains the following information:

   (a) Month

   (b) Number of sales for that month

   (c) Total Revenue from sales that month

   (d) Total cost of items from that month

```
select
        count(1) as ct
        , date_trunc('month', salesdt)
        , sum(prc)/100.0 as totalRev
        , sum(cost)/100.0 as totalCost
from
        sp.itemmap
left join
        sp.itemlist
using(iid)
group by 2
order by 2;
```

5. Using another tool (such as Excel or Google Docs) prepare a graph which shows, by region and month, the amount of *profit* generated. This should have four lines – one for each region.

```
select
        date_trunc('month', salesdt) as mnt
        , region
        , sum(prc - cost)/ 100.0
from
        sp.itemmap
join
        sp.sp
using(sid)
join
        sp.itemlist
using(iid)
group by 1,2
order by 2,1;
```

   OR:

```
select
    date_trunc('month', salesdt)::date as mnt
    ,sum (case when region = 'N' then prc-cost else 0 end ) as Npft
    ,sum (case when region = 'S' then prc-cost else 0 end ) as Spft
    ,sum (case when region = 'E' then prc-cost else 0 end ) as Epft
    ,sum (case when region = 'W' then prc-cost else 0 end ) as Wpft
from
    sp.itemmap
join
    sp.sp
    using(sid)
join
    sp.itemlist
    using(iid)
group by 1
order by 1;
```

6. Identify the top 7 sales people (name and SID) in terms of total revenue generated.

```
select
        spname, sid
from
        sp.itemlist
join
        sp.itemmap
using(iid)
join
        sp.sp
using(sid)
group by 1,2
order by sum( prc) desc limit 7;
```

7. Plot the monthly revenue (combined) for the top 7 salespeople.

```
select sum(prc)/100.0 , date_trunc('month', salesdt) as mnt
from
        sp.itemmap
join
        sp.itemlist
        using(iid)
where sid in
        (select
                sid
        from
                sp.itemlist
        join
                sp.itemmap
        using(iid)
        join
                sp.sp
        using(sid)
        group by 1
        order by sum( prc) desc limit 7)
group by 2
order by 2;
```

8. Create a pie chart which breaks down all revenue into one of four categories: (a) the salesperson worked less than 10 days (b) the salesperson worked between 10 and 20 days (c) the salesperson worked between 20 and 50 days and (d) the salesperson worked more than 50 days.

```
select sum(prc)::float/100 as rev,
        case
                when daysworked < 10 then 1
                when daysworked < 20 then 2
                when daysworked < 50 then 3
                else 4
        end
from
        sp.sp
join
        sp.itemmap
        using(sid)
join
        sp.itemlist
        using(iid)
group by 2;
```

OR

```
select
    sum( case when daysworked < 10 then prc else 0 end)/100.0 as C1
    ,sum( case when daysworked >= 10 and daysworked < 20 then prc else 0 end)/100.0 as C2
    ,sum( case when daysworked >= 20 and daysworked < 50 then prc else 0 end)/100. as C3
    ,sum( case when daysworked >= 50 then prc else 0 end)/100.0 as C4
from
    sp.sp
left join
    sp.itemmap
    using(sid)
left join
    sp.itemlist
    using(iid);
```

9. Calculate the average profit *per region*.

```
select
        avg(prft)/100.0 as prft
from
        (select sum(prc - cost)
                from
                        sp.sp
                join
                        sp.itemmap
                        using(sid)
                join
                        sp.itemlist
                        using(iid)

                group by region) as innerq;
```

10. We want to understand where we should concentrate our business – high margin items (which are those where the profit margin (price - cost)/cost >= 23%) or mid margin items (those where the profit margin is between 23% and 18%) and low-margin items (profit margin less than 18%). Create a dataset which identifies, for each distinct item, what margin group (high-, mid-, or low-) it is in.

```
select
        iid
        ,case
                when (prc-cost)::float / cost >= .23 then 'high'
                when (prc-cost)::float / cost >= .18 then 'mid'
                else 'low' end as margin
from
        sp.itemlist;
```

11. What was the average number of days that a salesperson worked?

```
select avg( daysworked)
from sp.sp;
```

12. Salespeople are paid based on one of two plans: The "H" bonus plan which means that they are paid $130 per day, but receive a 10% commission or the "L" bonus plan which they are paid $150 per

day, but receive a 5% commission. Calculate the amount of money that each salesperson made *on the non-commission* part.

```
select
         sid
         , case
                  when bonus = 'H' then 130.0*daysworked
                          else 150.0*daysworked
                  end as totalcomp
from sp.sp;
```

13. Calculate the total compensation paid to each salesperson, including both the commission and non-commission portion.

```
select
        sid
        , case
                when max(bonus) = 'H' then 130 *max(daysworked) + .1*sum(prc/100)
                else 150* max(daysworked) + .05*sum(prc/100)
        end as totalcomp
from
        sp.sp
left join
        sp.itemmap
        using(sid)
left join
        sp.itemlist
        using(iid)
group by 1;
```

14. The company is thinking about changing the bonus plan so that the "H" bonus plan would be $100 per day, but 20% commission and the "L" bonus plan would be $175 per day with no commission. Calculate the number of salespeople, within each bonus plant, that would be better off under the new vs. the old plan.

```
select count(1), bh, case when totalcompT <= totalcompN then 1 else 0 end as BetterOffFlag
from (
select
        sid, max(bonus) as bh
        , case
                when max(bonus) = 'H' then 130 *max(daysworked) + .1*sum(prc/100)
                else 150* max(daysworked) + .05*sum(prc/100)
        end as totalcompT
        , case
                when max(bonus) = 'L' then 130 *max(daysworked) + .1*sum(prc/100)
                else 150* max(daysworked) + .05*sum(prc/100)
        end as totalcompN
from
        sp.sp
left join
        sp.itemmap
        using(sid)
left join
        sp.itemlist
        using(iid)
group by 1) as iq
group by 2,3;
```