

Topic Overview

The content that we have covered on this first test is fairly tightly focused, but here is a breakdown of the general topics you should be prepared for:

- ☐ Creating tables
 - ☐ Choosing proper data types
 - ☐ Importing and Exporting from/to CSV
 - ☐ Creating from a **SELECT** query
- ☐ Making selections
 - ☐ Choosing unique entries
 - ☐ Choosing desired columns
 - ☐ Filtering properly using **WHERE** and boolean operators
 - ☐ Sorting
- ☐ Calculations
 - ☐ Data type of outputs
 - ☐ Column operations
 - Basic arithmetic operations
 - Applying common, built-in functions
 - ☐ Aggregates
 - Simple aggregates like **avg()**, **sum()**, etc
 - Order dependent aggregates like **percentile_cont()** and **mode()**

Question Types

Questions will fall into 3-4 main divisions, of which I will include examples of each later in the study guide.

Qualitative: In general, these wouldn't deal with direct values in a table, but are more conceptual in understanding what a particular piece of SQL is doing.

- Given a general table and query, describe what the output would look like, or what properties it might have.
- Given a desired output, what properties might the query or initial table have needed to possess?
- Given a table and desired output, what would the query need to look like?

Quantitative: These will deal more directly with sample data in a table.

- For this particular query with this tabular data, what would the output be? (These will naturally be with small and simple tables, as you won't have a computer to aid you.)

Example Questions

1. You have a particular table in your database called `inventory` that follows the below schema and has at least one row of data.

Column Name	Data Type
<code>id</code>	<code>SERIAL</code>
<code>name</code>	<code>VARCHAR(20)</code>
<code>weight</code>	<code>REAL</code>
<code>price</code>	<code>NUMERIC(5,2)</code>
<code>stock</code>	<code>INT</code>

You then run the following query:

```
SELECT COUNT(weight) / COUNT(*) * 100::REAL
FROM inventory;
```

- (a) How many columns are returned in the output?
- A. 0
 - B. 1**
 - C. 5
 - D. Impossible to tell
- (b) How many rows are returned in the output?
- A. 0
 - B. 1**
 - C. The same as the number of rows in the `id` column
 - D. Impossible to tell
- (c) For each column that is returned, what would be its corresponding data type?

Solution: Since there will only be a single column returned, we just need to determine a single data type. `COUNT(weight)` will be an integer, as will `COUNT(*)`. Dividing them then will also result in an integer (which is probably *not* what was desired here, but is what the query will do). Then the 100 has been cast to be a floating-point value, and multiplying anything by a floating point value results in a floating point value. So the final value would be a `REAL` type value (and, most likely, equal to 0).

- (d) In a sentence or two, describe what this query is doing. I'm looking less for a line by line description of what is happening, and more an overall description of what the query is trying to achieve.

Solution: The query is computing the percentage of items that have a recorded weight.

2. Without any information about the table called `mystery`, you run the below query:

```
SELECT
  dim1 * dim2 * dim3 AS volume,
  |/(score::DECIMAL + 10) AS metric
FROM mystery
WHERE best_by + '3 days 10 minutes' < sold
ORDER BY score::DECIMAL
```

where any type conversions were **necessary** (not optional). The resulting table has the form:

Column Name	Data Type
volume	NUMERIC
metric	DOUBLE PRECISION

Write as *much detail as you can* about what you know about the table `mystery` from just this query and its results.

Solution: The table must have at least 6 columns, as determined by the number of different column identifiers mentioned. I can further narrow down what data types they might be according to how they are used and the resulting outputs:

- At least one of `dim1`, `dim2` or `dim3` must be a **NUMERIC** type. The others could also be **NUMERIC** or they could be **INTEGER**, but they can not be any sort of floating-point type, since the output in the end is **NUMERIC**.
- `score` is most likely a number in the form of a **TEXT** field (something like '15'). That seems to be the best explanation for why it would need to be converted to fixed-point in the calculation (you can't add text and numbers) and why it was converted in the ordering (text and numbers order differently). If it was already a number-type, then there would be no need to convert it for ordering.
- `best_by` needs to be some sort of **TIMESTAMP** or **DATE** type, since it is being added to an interval. Arguably I guess it could be **TEXT** that just gets added to the interval string, but the contents of the string would suggest a interval.
- `sold` would also need to be a sort of **TIMESTAMP** or **DATE** type so that it could be easily compared to the sum of `best_by` and the interval.

In general, `metric` being a **DOUBLE PRECISION** type at the end doesn't really tell us anything, since square roots always return a floating point type.

3. Suppose I wanted to import the below CSV file (saved at `C:\Data\important.csv`) into a PostgreSQL database. Write out the necessary commands to create the table and import the data.

```
id,name,p1,p2,p3,total,submitted
1,Bill,7,8,2,17,2022-01-25 18:00
2,Nancy,7,7,7,21,2022-01-26 15:15
3,Jacob,5,10,5.6,20.6,2022-01-25 23:47
4,Sebastian,9.5,10,10,29.5,2022-01-29 19:34
```

Solution: I would probably write something like this:

```
CREATE TABLE important (
  "id" BIGSERIAL,
  "name" TEXT,
  p1 NUMERIC(4,2),
  p2 NUMERIC(4,2), -- see note below
  p3 NUMERIC(4,2),
  total NUMERIC(5,2),
  submitted TIMESTAMP
);

COPY important
FROM 'C:\Data\important.csv'
WITH (FORMAT CSV, HEADER);
```

All the current values of p2 are integers, but given the similarity in name to the others that *do* have fractional values, it doesn't seem inconceivable that this column could occasionally get fractional values as well.

4. You have a table named `special` in your database, that looks as can be seen below:

id	name	cola	colb	colc
<i>SERIAL</i>	<i>TEXT</i>	<i>INT</i>	<i>NUMERIC(4,2)</i>	<i>INT</i>
1	Bob	3	4.50	9
2	Bob	2	2.00	5
3	Bob	NULL	4.10	4
4	Bob	5	12.40	10
5	Bob	8	NULL	7

- (a) What would be the output of the below query?

```
SELECT
  name,
  colb / (colc / cola) AS o1,
  2 * colc + colb AS o2
FROM special
WHERE colb IS NOT NULL
ORDER BY o1
```

Solution: Initially, I'd just look at the table before ordering. The filter that `colb` is not `NULL` means we are only looking at the first 4 rows. So those would look like:

name	o1	o2
<i>TEXT</i>	<i>NUMERIC</i>	<i>NUMERIC</i>
Bob	1.50	22.50
Bob	1.00	12.00
Bob	NULL	12.10
Bob	6.20	32.40

So then ordering, with `NULLs` at the start, would give us:

name	o1	o2
<i>TEXT</i>	<i>NUMERIC</i>	<i>NUMERIC</i>
Bob	NULL	12.10
Bob	1.00	12.00
Bob	1.50	22.50
Bob	6.20	32.40

- (b) What would be the output of the below query?

```
SELECT
  min(colc - cola) AS mind,
  percentile_disc(0.5) WITHIN GROUP (ORDER BY name) AS midname,
  sum(colb + colc) AS summy
FROM special
WHERE id % 2 > 0;
```

Solution: It is clear that we have aggregate functions here, so I'll figure out what the table would look like *before* those were applied, and then compute them accordingly. So just applying the filter, we will just keep the odd id rows:

id <i>SERIAL</i>	name <i>TEXT</i>	cola <i>INT</i>	colb <i>NUMERIC(4,2)</i>	colc <i>INT</i>
1	Bob	3	4.50	9
3	Bob	NULL	4.10	4
5	Bob	8	NULL	7

Then $\text{colc} - \text{cola}$ will only be non-null in the first and 3rd rows, of which -1 would be the smallest. The median name is clearly Bob, because I was an idiot and forgot to change the other names. But even if they were different, it would just be the middle one when ordered alphabetically. And then, again, $\text{colb} + \text{colc}$ is only non-null for the first two rows, which if we add them we have $(4.5 + 9) + (4.10 + 4) = 21.6$. So our final table would be:

mind <i>INT</i>	midname <i>TEXT</i>	summy <i>NUMERIC</i>
-1	Bob	21.6

5. You have the table (named `teachers`) of teachers in your local area with a schema given below, where I have also added a quick description of each column.

Column Name	Data Type	Description
<code>id</code>	<code>SERIAL</code>	Unique identifying integer
<code>name</code>	<code>TEXT</code>	Full name of the teacher
<code>gender</code>	<code>CHAR(1)</code>	Gender of teacher: M or F
<code>grade</code>	<code>INT</code>	Grade level taught. Kindergarten is 0.
<code>yr_exp</code>	<code>INT</code>	Years of teaching experience
<code>salary</code>	<code>NUMERIC(8,2)</code>	Yearly salary in US dollars
<code>peak_deg</code>	<code>VARCHAR(3)</code>	Peak degree obtained: HS,BS/BA,MS,PhD

Write out queries to answer the following questions.

- (a) What is the average salary of high school (grades 9-12) teachers with graduate degrees?

Solution:

```
SELECT
  avg(salary)
FROM teachers
WHERE grade BETWEEN 9 AND 12 AND peak_deg IN ('MS', 'PhD');
```

There are definitely multiple ways you could do the filtering on this, but this is probably the most concise I'm currently seeing.

- (b) What Ms. or Mrs. Johnson has been teaching for the longest?

Solution:

```
SELECT
  name
FROM teachers
WHERE name ILIKE '%Johnson%' AND gender = 'F'
ORDER BY yr_exp DESC
LIMIT 1;
```

If I was really worried about some female with a first name of “Johnson”, I could probably remove the trailing wildcard character from my pattern match, thereby forcing the “Johnson” to be at the end of the full name.