column_types_constraints

```sql
-- Select the original value
SELECT profits_change,
    -- Cast profits_change to integer
    CAST(profits_change AS integer) AS profits_change_int
 FROM fortune500;
```

Explanation:

- This SQL query selects two columns from a table named fortune500. The first column is profits_change (presumably a column containing numeric values, possibly with decimal places). The second column, profits_change_int, is created by casting the profits_change column to an integer data type using the CAST function. This effectively truncates any decimal portion of the profits_change values.

```sql
-- Divide 10 by 3
SELECT 10/3,
    -- Cast 10 as numeric and divide by 3
    10::numeric/3;
```

Explanation:

- This SQL code snippet demonstrates the difference in integer division and floating-point division. The first part (10/3) performs integer division, resulting in a truncated integer value. The second part (10::numeric/3) explicitly casts 10 to a numeric data type before division, ensuring a floating-point result with decimal precision.

```sql
SELECT '3.2'::numeric,
    '-123'::numeric,
    '1e3'::numeric,
    '1e-3'::numeric,
    '02314'::numeric,
    '0002'::numeric;
```

Explanation:

- This SQL query demonstrates type casting in PostgreSQL. It explicitly converts various string literals into the numeric data type. The ::numeric syntax performs the type cast. The query will return a table with a single row and six columns, each containing the numeric representation of the respective string. Note that leading zeros in strings are ignored when converting to numeric.

```
-- Select the count of each value of revenues_change
SELECT revenues_change, COUNT(revenues_change)
 FROM fortune500
 GROUP BY revenues_change
 -- order by the values of revenues_change
 ORDER BY count DESC;
```

Explanation:

- This SQL query calculates the frequency of each distinct value in the revenues_change column of the fortune500 table. It groups the rows based on revenues_change, counts the occurrences within each group using COUNT(), and then orders the results in descending order based on the counts (most frequent first).

```
-- Select the count of each revenues_change integer value
SELECT revenues_change::integer, COUNT(revenues_change)
 FROM fortune500
 GROUP BY revenues_change::integer
 -- order by the values of revenues_change
 ORDER BY COUNT DESC;
```

Explanation:

- This SQL query calculates the frequency of each distinct integer value in the revenues_change column of the fortune500 table. It converts revenues_change to an integer using ::integer (assuming it might contain other data types), groups the results by the integer values, and then counts the occurrences of each value. Finally, it orders the results in descending order based on the count, showing the most frequent values first.

```
-- Count rows
SELECT COUNT(*)
 FROM fortune500
 -- Where revenues increased
 WHERE revenues_change > 0;
```

Explanation:

- This SQL query counts the number of rows in the fortune500 table where the revenues_change column has a value greater than 0. Essentially, it finds the number of companies in the Fortune 500 whose revenues increased.