9.20. Aggregate Functions

Aggregate functions compute a single result from a set of input values. The built-in normal aggregate functions are listed in Table 9-49 and Table 9-50. The built-in ordered-set aggregate functions are listed in Table 9-51 and Table 9-52. Grouping operations, which are closely related to aggregate functions, are listed in Table 9-53. The special syntax considerations for aggregate functions are explained in Section 4.2.7. Consult Section 2.7 for additional introductory information.

Table 9-49. General-Purpose Aggregate Functions

Function	Argument Type(s)	Return Type	Description
`array_agg(`***expression***`)`	any non-array type	array of the argument type	input values, including nulls, concatenated into an array
`array_agg(`***expression***`)`	any array type	same as argument data type	input arrays concatenated into array of one higher dimension (inputs must all have same dimensionality, and cannot be empty or null)
`avg(`***expression***`)`	**`smallint`**, **`int`**, **`bigint`**, **`real`**, **`double precision`**, **`numeric`**, or **`interval`**	**`numeric`** for any integer-type argument, **`double precision`** for a floating-point argument, otherwise the same as the argument data type	the average (arithmetic mean) of all non-null input values
`bit_and(`***expression***`)`	**`smallint`**, **`int`**, **`bigint`**, or **`bit`**	same as argument data type	the bitwise AND of all non-null input values, or null if none
`bit_or(`***expression***`)`	**`smallint`**, **`int`**, **`bigint`**, or **`bit`**	same as argument data type	the bitwise OR of all non-null input values, or null if none
`bool_and(`***expression***`)`	**`bool`**	**`bool`**	true if all input values are true, otherwise false
`bool_or(`***expression***`)`	**`bool`**	**`bool`**	true if at least one input value is true, otherwise false
`count(*)`		**`bigint`**	number of input rows
`count(`***expression***`)`	any	**`bigint`**	number of input rows for which the value of ***`expression`*** is not null
`every(`***expression***`)`	**`bool`**	**`bool`**	equivalent to `bool_and`
`json_agg(`***expression***`)`	**`any`**	**`json`**	aggregates values, including nulls, as a JSON array
`jsonb_agg(`***expression***`)`	**`any`**	**`jsonb`**	aggregates values, including nulls, as a JSON array
`json_object_agg(`***name***`, `***value***`)`	**`(any, any)`**	**`json`**	aggregates name/value pairs as a JSON object; values can be null, but not names
`jsonb_object_agg(`***name***`, `***value***`)`	**`(any, any)`**	**`jsonb`**	aggregates name/value pairs as a JSON object; values can be null, but not names
`max(`***expression***`)`	any numeric, string, date/time, network, or enum type, or arrays of these types	same as argument type	maximum value of ***`expression`*** across all non-null input values
`min(`***expression***`)`	any numeric, string, date/time, network, or enum type, or arrays of these types	same as argument type	minimum value of ***`expression`*** across all non-null input values
`string_agg(`***expression***`, `***delimiter***`)`	(**`text`**, **`text`**) or (**`bytea`**, **`bytea`**)	same as argument types	non-null input values concatenated into a string, separated by delimiter

Function	Argument Type(s)	Return Type	Description
sum(*expression*)	**smallint**, **int**, **bigint**, **real**, **double precision**, **numeric**, **interval**, or **money**	**bigint** for **smallint** or **int** arguments, **numeric** for **bigint** arguments, otherwise the same as the argument data type	sum of *expression* across all non-null input values
xmlagg(*expression*)	**xml**	**xml**	concatenation of non-null XML values (see also Section 9.14.1.7)

It should be noted that except for count, these functions return a null value when no rows are selected. In particular, sum of no rows returns null, not zero as one might expect, and array_agg returns null rather than an empty array when there are no input rows. The coalesce function can be used to substitute zero or an empty array for null when necessary.

Note: Boolean aggregates bool_and and bool_or correspond to standard SQL aggregates every and any or some. As for any and some, it seems that there is an ambiguity built into the standard syntax:

```
SELECT b1 = ANY((SELECT b2 FROM t2 ...)) FROM t1 ...;
```

Here ANY can be considered either as introducing a subquery, or as being an aggregate function, if the subquery returns one row with a Boolean value. Thus the standard name cannot be given to these aggregates.

Note: Users accustomed to working with other SQL database management systems might be disappointed by the performance of the count aggregate when it is applied to the entire table. A query like:

```
SELECT count(*) FROM sometable;
```

will require effort proportional to the size of the table: PostgreSQL will need to scan either the entire table or the entirety of an index which includes all rows in the table.

The aggregate functions array_agg, json_agg, jsonb_agg, json_object_agg, jsonb_object_agg, string_agg, and xmlagg, as well as similar user-defined aggregate functions, produce meaningfully different result values depending on the order of the input values. This ordering is unspecified by default, but can be controlled by writing an ORDER BY clause within the aggregate call, as shown in Section 4.2.7. Alternatively, supplying the input values from a sorted subquery will usually work. For example:

```
SELECT xmlagg(x) FROM (SELECT x FROM test ORDER BY y DESC) AS tab;
```

But this syntax is not allowed in the SQL standard, and is not portable to other database systems.

Table 9-50 shows aggregate functions typically used in statistical analysis. (These are separated out merely to avoid cluttering the listing of more-commonly-used aggregates.) Where the description mentions ***N***, it means the number of input rows for which all the input expressions are non-null. In all cases, null is returned if the computation is meaningless, for example when ***N*** is zero.

Table 9-50. Aggregate Functions for Statistics

Function	Argument Type	Return Type	Description
corr(***Y***, ***X***)	**double precision**	**double precision**	correlation coefficient
covar_pop(***Y***, ***X***)	**double precision**	**double precision**	population covariance
covar_samp(***Y***, ***X***)	**double precision**	**double precision**	sample covariance
regr_avgx(***Y***, ***X***)	**double precision**	**double precision**	average of the independent variable (sum(***X***) / ***N***)
regr_avgy(***Y***, ***X***)	**double precision**	**double precision**	average of the dependent variable (sum(***Y***) / ***N***)
regr_count(***Y***, ***X***)	**double precision**	**bigint**	number of input rows in which both expressions are nonnull
regr_intercept(***Y***, ***X***)	**double precision**	**double precision**	y-intercept of the least-squares-fit linear equation determined by the (***X***, ***Y***) pairs
regr_r2(***Y***, ***X***)	**double precision**	**double precision**	square of the correlation coefficient
regr_slope(***Y***, ***X***)	**double precision**	**double precision**	slope of the least-squares-fit linear equation determined by the (***X***, ***Y***) pairs

Function	Argument Type	Return Type	Description
`regr_sxx(Y, X)`	`double precision`	`double precision`	`sum(X^2) - sum(X)^2/N` ("sum of squares" of the independent variable)
`regr_sxy(Y, X)`	`double precision`	`double precision`	`sum(X*Y) - sum(X) * sum(Y)/N` ("sum of products" of independent times dependent variable)
`regr_syy(Y, X)`	`double precision`	`double precision`	`sum(Y^2) - sum(Y)^2/N` ("sum of squares" of the dependent variable)
`stddev(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, or `numeric`	`double precision` for floating-point arguments, otherwise `numeric`	historical alias for `stddev_samp`
`stddev_pop(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, or `numeric`	`double precision` for floating-point arguments, otherwise `numeric`	population standard deviation of the input values
`stddev_samp(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, or `numeric`	`double precision` for floating-point arguments, otherwise `numeric`	sample standard deviation of the input values
`variance(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, or `numeric`	`double precision` for floating-point arguments, otherwise `numeric`	historical alias for `var_samp`
`var_pop(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, or `numeric`	`double precision` for floating-point arguments, otherwise `numeric`	population variance of the input values (square of the population standard deviation)
`var_samp(expression)`	`smallint`, `int`, `bigint`, `real`, `double precision`, or `numeric`	`double precision` for floating-point arguments, otherwise `numeric`	sample variance of the input values (square of the sample standard deviation)

Table 9-51 shows some aggregate functions that use the *ordered-set aggregate* syntax. These functions are sometimes referred to as "inverse distribution" functions.

Table 9-51. Ordered-Set Aggregate Functions

Function	Direct Argument Type(s)	Aggregated Argument Type(s)	Return Type	Description
`mode() WITHIN GROUP (ORDER BY sort_expression)`		any sortable type	same as sort expression	returns the most frequent input value (arbitrarily choosing the first one if there are multiple equally-frequent results)
`percentile_cont(fraction) WITHIN GROUP (ORDER BY sort_expression)`	`double precision`	`double precision` or `interval`	same as sort expression	continuous percentile: returns a value corresponding to the specified fraction in the ordering, interpolating between adjacent input items if needed
`percentile_cont(fractions) WITHIN GROUP (ORDER BY sort_expression)`	`double precision[]`	`double precision` or `interval`	array of sort expression's type	multiple continuous percentile: returns an array of results matching the shape of the `fractions` parameter, with each non-null element replaced by the value corresponding to that percentile
`percentile_disc(fraction) WITHIN GROUP (ORDER BY sort_expression)`	`double precision`	any sortable type	same as sort expression	discrete percentile: returns the first input value whose position in the ordering equals or exceeds the specified fraction
`percentile_disc(fractions) WITHIN GROUP (ORDER BY sort_expression)`	`double precision[]`	any sortable type	array of sort expression's type	multiple discrete percentile: returns an array of results matching the shape of the `fractions` parameter, with each non-null element replaced by the input value corresponding to that percentile

All the aggregates listed in Table 9-51 ignore null values in their sorted input. For those that take a ***fraction*** parameter, the fraction value must be between 0 and 1; an error is thrown if not. However, a null fraction value simply produces a null result.

Each of the aggregates listed in Table 9-52 is associated with a window function of the same name defined in Section 9.21. In each case, the aggregate result is the value that the associated window function would have returned for the "hypothetical" row constructed from ***args***, if such a row had been added to the sorted group of rows computed from the ***sorted_args***.

Table 9-52. Hypothetical-Set Aggregate Functions

Function	Direct Argument Type(s)	Aggregated Argument Type(s)	Return Type	Description

Function	Direct Argument Type(s)	Aggregated Argument Type(s)	Return Type	Description
`rank(`***`args`***`) WITHIN GROUP (ORDER BY` ***`sorted_args`***`)`	`VARIADIC "any"`	`VARIADIC "any"`	**`bigint`**	rank of the hypothetical row, with gaps for duplicate rows
`dense_rank(`***`args`***`) WITHIN GROUP (ORDER BY` ***`sorted_args`***`)`	`VARIADIC "any"`	`VARIADIC "any"`	**`bigint`**	rank of the hypothetical row, without gaps
`percent_rank(`***`args`***`) WITHIN GROUP (ORDER BY` ***`sorted_args`***`)`	`VARIADIC "any"`	`VARIADIC "any"`	**`double precision`**	relative rank of the hypothetical row, ranging from 0 to 1
`cume_dist(`***`args`***`) WITHIN GROUP (ORDER BY` ***`sorted_args`***`)`	`VARIADIC "any"`	`VARIADIC "any"`	**`double precision`**	relative rank of the hypothetical row, ranging from 1/***N*** to 1

For each of these hypothetical-set aggregates, the list of direct arguments given in ***`args`*** must match the number and types of the aggregated arguments given in ***`sorted_args`***. Unlike most built-in aggregates, these aggregates are not strict, that is they do not drop input rows containing nulls. Null values sort according to the rule specified in the `ORDER BY` clause.

Table 9-53. Grouping Operations

Function	Return Type	Description
`GROUPING(`***`args...`***`)`	**`integer`**	Integer bit mask indicating which arguments are not being included in the current grouping set

Grouping operations are used in conjunction with grouping sets (see Section 7.2.4) to distinguish result rows. The arguments to the `GROUPING` operation are not actually evaluated, but they must match exactly expressions given in the `GROUP BY` clause of the associated query level. Bits are assigned with the rightmost argument being the least-significant bit; each bit is 0 if the corresponding expression is included in the grouping criteria of the grouping set generating the result row, and 1 if it is not. For example:

```
=>  SELECT * FROM items_sold;
 make  | model | sales
-------+-------+-------
 Foo   | GT    |  10
 Foo   | Tour  |  20
 Bar   | City  |  15
 Bar   | Sport |  5
(4 rows)

=>  SELECT make, model, GROUPING(make,model), sum(sales) FROM items_sold GROUP BY ROLLUP(make,model);
 make  | model | grouping | sum
-------+-------+----------+-----
 Foo   | GT    |        0 | 10
 Foo   | Tour  |        0 | 20
 Bar   | City  |        0 | 15
 Bar   | Sport |        0 | 5
 Foo   |       |        1 | 30
 Bar   |       |        1 | 20
       |       |        3 | 50
(7 rows)
```