Aggregates

- Aggregates
- User-defined Aggregates

❖ Aggregates

Aggregates reduce a collection of values into a single result.

Examples: **`count(`***Tuples***`)`**, **`sum(`***Numbers***`)`**, **`max(`***AnyOrderedType***`)`**

The action of an aggregate function can be viewed as:

```
State = initial state
for each item T {
    # update State to include T
    State = updateState(State, T)
}
return makeFinal(State)
```

❖ Aggregates (cont)

Aggregates are commonly used with **`GROUP BY`**.

In that context, they "summarise" each group.

Example:

```
R                       select a,sum(b),count(*)
 a | b | c              from R group by a
---+---+---
 1 | 2 | x               a | sum | count
 1 | 3 | y              ---+-----+-------
 2 | 2 | z               1 |   5 |     2
 2 | 1 | a               2 |   6 |     3
 2 | 3 | b
```

❖ User-defined Aggregates

SQL standard does not specify user-defined aggregates.

But PostgreSQL provides a mechanism for defining them.

To define a new aggregate, first need to supply:

- *BaseType* ... type of input values
- *StateType* ... type of intermediate states
- state mapping function: *sfunc(state,value)* → *newState*
- [optionally] an initial state value (defaults to null)
- [optionally] final function: *ffunc(state)* → *result*

❖ User-defined Aggregates (cont)

New aggregates defined using **`CREATE AGGREGATE`** statement:

```
CREATE AGGREGATE AggName(BaseType) (
    sfunc     = UpdateStateFunction,
    stype     = StateType,
    initcond  = InitialValue,
    finalfunc = MakeFinalFunction,
    sortop    = OrderingOperator
);
```

- **`initcond`** (type *StateType*) is optional; defaults to **`NULL`**
- **`finalfunc`** is optional; defaults to identity function
- **`sortop`** is optional; needed for min/max-type aggregates

❖ User-defined Aggregates (cont)

Example: defining the **count** aggregate (roughly)

```
create aggregate myCount(anyelement) (
    stype    = int,     -- the accumulator type
    initcond = 0,       -- initial accumulator value
    sfunc    = oneMore  -- increment function
);

create function
    oneMore(sum int, x anyelement) returns int
as $$
begin return sum + 1; end;
$$ language plpgsql;
```

❖ User-defined Aggregates (cont)

Example: **sum2** sums two columns of integers

```
create type IntPair as (x int, y int);

create function
   addPair(sum int, p IntPair) returns int
as $$
begin return sum + p.x + p.y; end;
$$ language plpgsql;

create aggregate sum2(IntPair) (
   stype     = int,
   initcond  = 0,
   sfunc     = addPair
);
```

make sure sum = 0 firstly

eg

	x	y
1	10	20
2	-5	15
3	30	-10

0 + (10+20) = 30

30 - 5 + 15 = 40

40 + 30 - 10 = 60

❖ User-defined Aggregates (cont)

PostgreSQL has many aggregates (e.g. **`sum`**, **`count`**, ...)

But it doesn't have a product aggregate.

Implement a **`prod`** aggregate that

- computes the product of values in a column of numeric data

Usage:

```
select prod(*) from iota(5);
 prod
------
  120
```

❖ User-defined Aggregates (cont)

Example: product aggregate

```
create function
   mult(soFar numeric, next numeric) returns numeric
as $$
begin return soFar * next; end;
$$ language plpgsql;

create aggregate prod(numeric) (
   stype    = numeric,
   initcond = 1,
   sfunc    = mult
);
```

❖ User-defined Aggregates (cont)

Define a **concat** aggregate that

- takes a column of string values
- returns a comma-separated string of values

Example:

```
select count(*), concat(name) from Employee;
-- returns e.g.
  count |          concat
 -------+--------------------------
      4 | John,Jane,David,Phil
```

❖ User-defined Aggregates (cont)

Example: string concatenation aggregate

```
create function
    join(s1 text, s2 text) returns text
as $$
begin
   if (s1 = '') then
      return s2;
   else
      return s1||','||s2;
   end if;
end;
$$ language plpgsql;

create aggregate concat(text) (
   stype    = text,
   initcond = '',
   sfunc    = join
);
```

s1 is null

combine text.

Produced: 15 Oct 2020

Produced: 15 Oct 2020