KOREA UNIVERSITY
DATABASE LAB

PostgreSQL for OLAP

GROUPING SETS

- GROUPING SETS generates a result set equivalent to which generated by the UNION ALL of the multiple GROUP BY clauses

```
((GROUP BY expr1)
UNION ALL
(GROUP BY expr2)
UNION ALL
(GROUP BY expr3))
```

↓

```
GROUPING SETS (expr1, expr2, expr3)
```

GROUPING SETS

```
SELECT customer_id, staff_id, sum(amount)
FROM payment GROUP BY customer_id, staff_id;
```

```
UNION ALL
```

```
SELECT customer_id, NULL, sum(amount)
FROM payment GROUP BY customer_id;
```

```
UNION ALL
```

```
SELECT NULL, staff_id, sum(amount)
FROM payment GROUP BY staff_id;
```

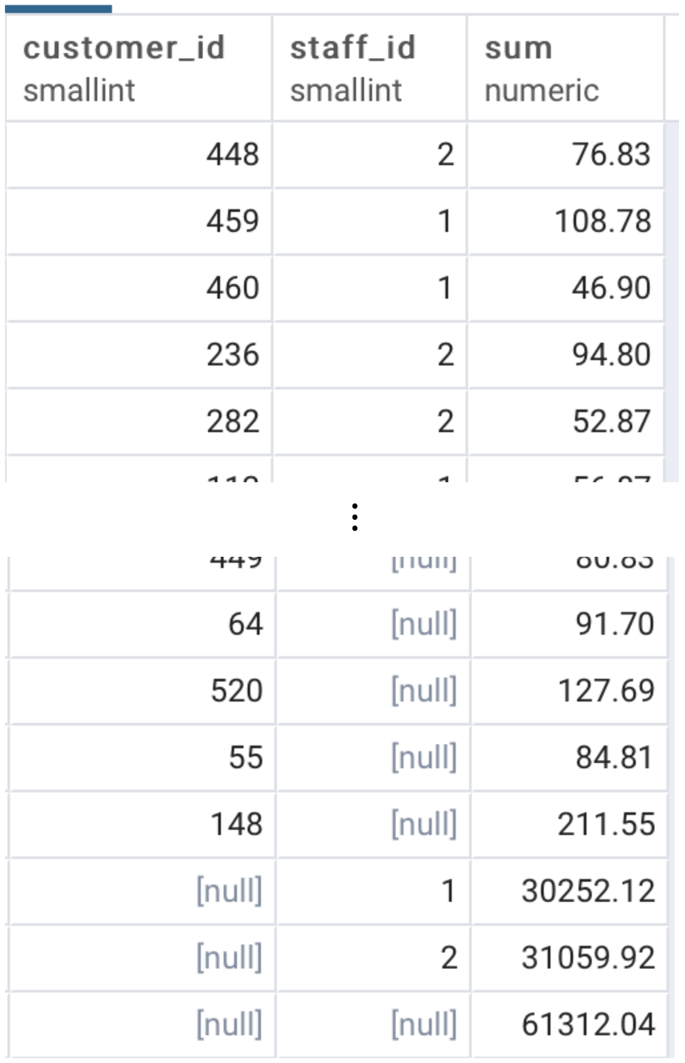
```
UNION ALL
```

```
SELECT NULL, NULL, sum(amount)
FROM payment;
```

```
SELECT customer_id, staff_id, sum(amount)
FROM payment
GROUP BY
   GROUPING SETS (
      (customer_id, staff_id),
      (customer_id),
      (staff_id),
      ()
   );
```

GROUPING SETS

```
SELECT customer_id, staff_id,
sum(amount)
FROM payment
GROUP BY
   GROUPING SETS (
      (customer_id, staff_id),
      (customer_id),
      (staff_id),
      ()
   );
```



customer_id smallint	staff_id smallint	sum numeric
448	2	76.83
459	1	108.78
460	1	46.90
236	2	94.80
282	2	52.87
⋮		
64	[null]	91.70
520	[null]	127.69
55	[null]	84.81
148	[null]	211.55
[null]	1	30252.12
[null]	2	31059.92
[null]	[null]	61312.04

KOREA UNIVERSITY
DATABASE LAB

CUBE

- Analyze all possible subsets with more columns

```
GROUPING SETS
(
    (c1,c2,c3),
    (c1,c2),
    (c1,c3),
    (c2,c3),
    (c1),
    (c2),
    (c3),
    ()
)
```

↓

```
CUBE(c1,c2,c3)
```

```
GROUPING SETS
(
    (c1,c2,c3,c4),
    (c1,c2,c3),
    …
    (c1)
    (c2),
    (c3),
    (c4),
    ()
)
```

↓

```
CUBE(c1,c2,c3,c4)
```

CUBE

- PostgreSQL allows you to perform a partial cube to reduce the number of aggregates calculated

```
SELECT c1, c2, c3, aggregate (c4)
FROM table_name
GROUP BY c1, CUBE(c2, c3);
```

```
GROUPING SETS (
    (c1,c2,c3),
    (c1,c2),
    (c1,c3),
    (c1),
)
```

ROLLUP

- ROLLUP(c1,c2,c3) generates only four grouping sets, **assuming the hierarchy c1 > c2 > c3** as follows

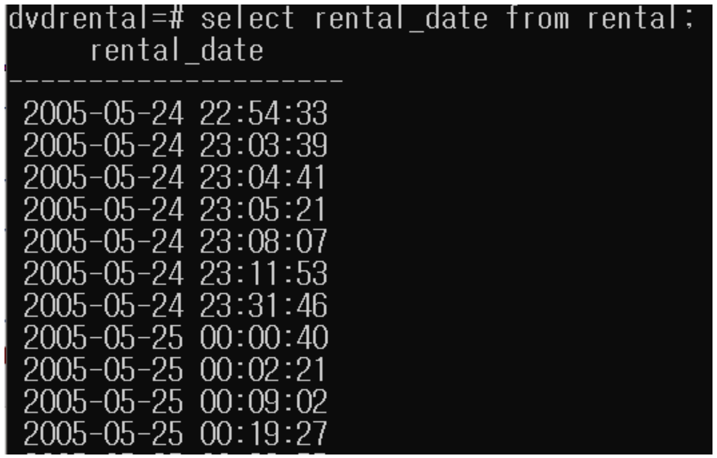
GROUPING SETS (
    (c1,c2,c3),
    (c1,c2),
    ~~(c1,c3)~~,
    ~~(c2,c3)~~,
    (c1),
    ~~(c2)~~,
    ~~(c3)~~,
    ()
)

$\Rightarrow$

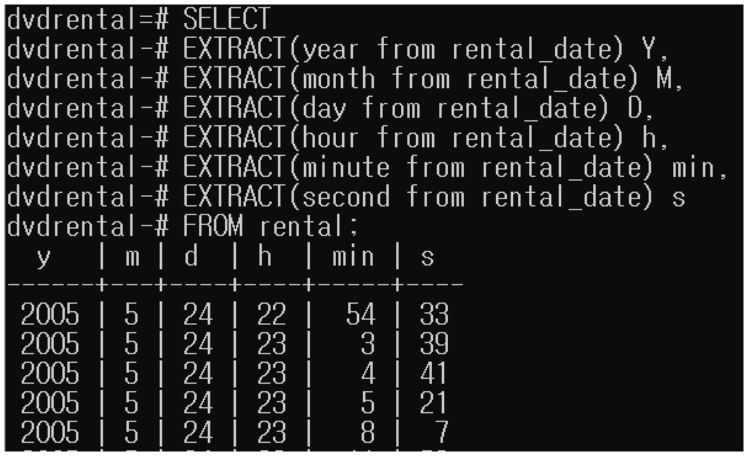
ROLLUP(c1,c2,c3)

ROLLUP

- A common use of ROLLUP is to calculate the aggregations of data by year, month, and date, considering the hierarchy year > month > date
- You can extract (year, month, day, hour, minute and second) from timestamp data type through EXTRACT



```
dvdrental=# select rental_date from rental;
     rental_date
---------------------
 2005-05-24 22:54:33
 2005-05-24 23:03:39
 2005-05-24 23:04:41
 2005-05-24 23:05:21
 2005-05-24 23:08:07
 2005-05-24 23:11:53
 2005-05-24 23:31:46
 2005-05-25 00:00:40
 2005-05-25 00:02:21
 2005-05-25 00:09:02
 2005-05-25 00:19:27
```



```
dvdrental=# SELECT
dvdrental-# EXTRACT(year from rental_date) Y,
dvdrental-# EXTRACT(month from rental_date) M,
dvdrental-# EXTRACT(day from rental_date) D,
dvdrental-# EXTRACT(hour from rental_date) h,
dvdrental-# EXTRACT(minute from rental_date) min,
dvdrental-# EXTRACT(second from rental_date) s
dvdrental-# FROM rental;
  y   | m | d  | h  | min | s
------+---+----+----+-----+----
 2005 | 5 | 24 | 22 |  54 | 33
 2005 | 5 | 24 | 23 |   3 | 39
 2005 | 5 | 24 | 23 |   4 | 41
 2005 | 5 | 24 | 23 |   5 | 21
 2005 | 5 | 24 | 23 |   8 |  7
```

Window Functions

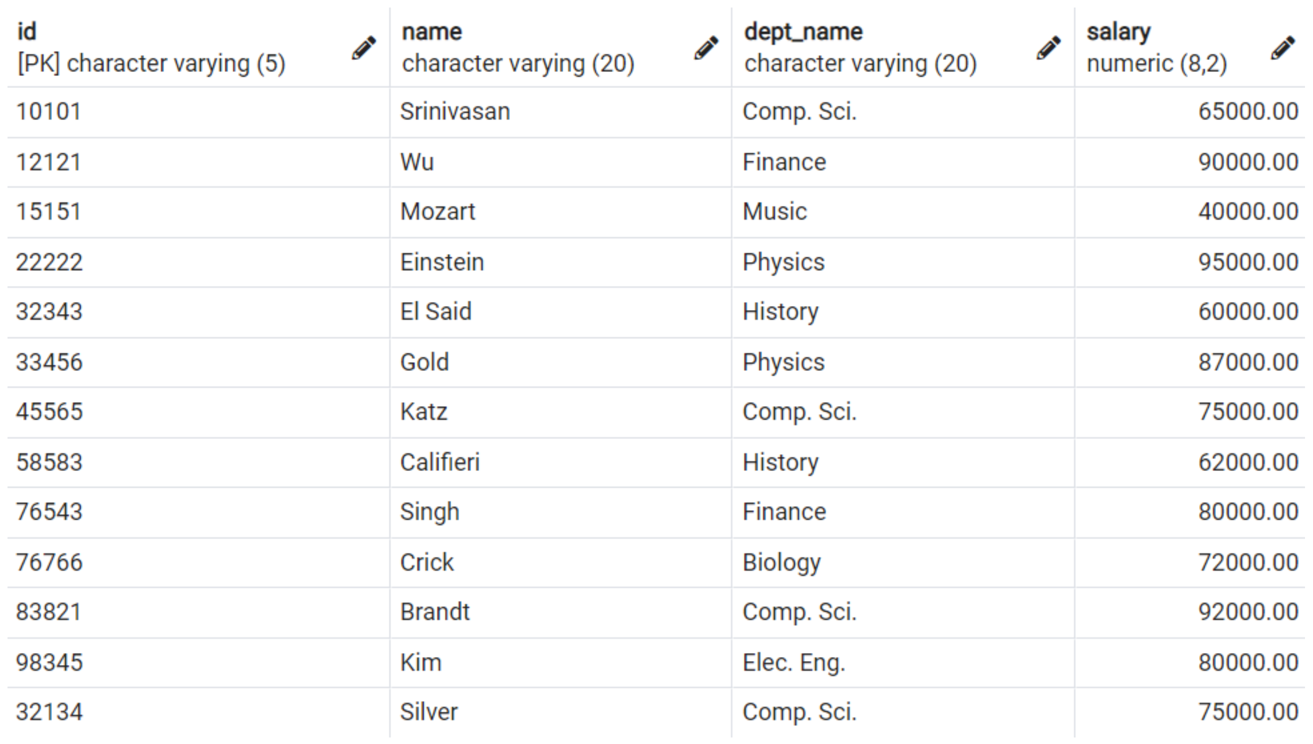
- A window function performs a calculation across a set of table rows that are somehow related to the current row
- This is comparable to the type of calculation that can be done with an aggregate function
- But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities

```
window_function(arg1, arg2,..) OVER (
   [PARTITION BY partition_expression]
   [ORDER BY sort_expression [ASC I DESC] [NULLS {FIRST I LAST }]
)
```

  - The PARTITION BY list within OVER specifies dividing the row into groups, or partitions, that share the same values of the PARTITION BY expression(s)

Example Relation

Instructor Table

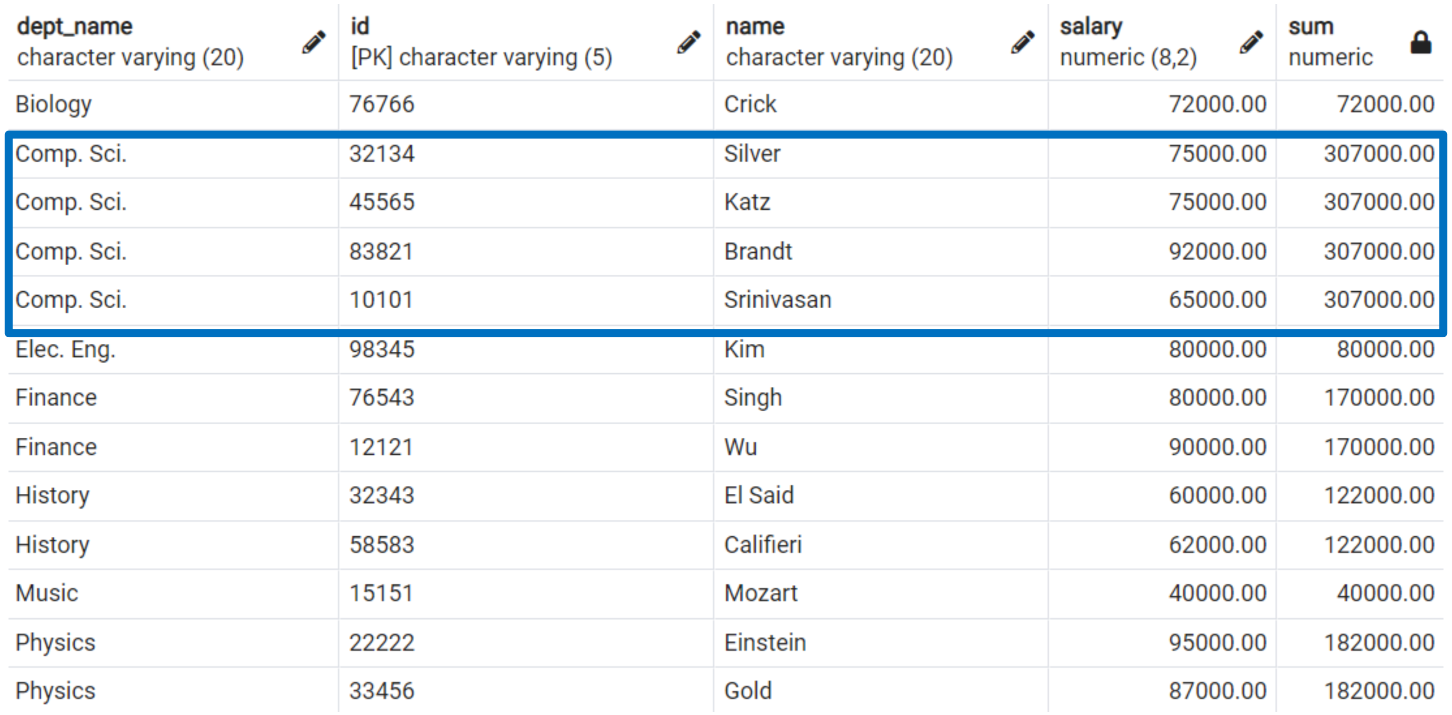


id [PK] character varying (5)	name character varying (20)	dept_name character varying (20)	salary numeric (8,2)
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00
32134	Silver	Comp. Sci.	75000.00

PARTITION BY

- For each row, the window function is computed across the rows that fall into the same partition as the current row

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```

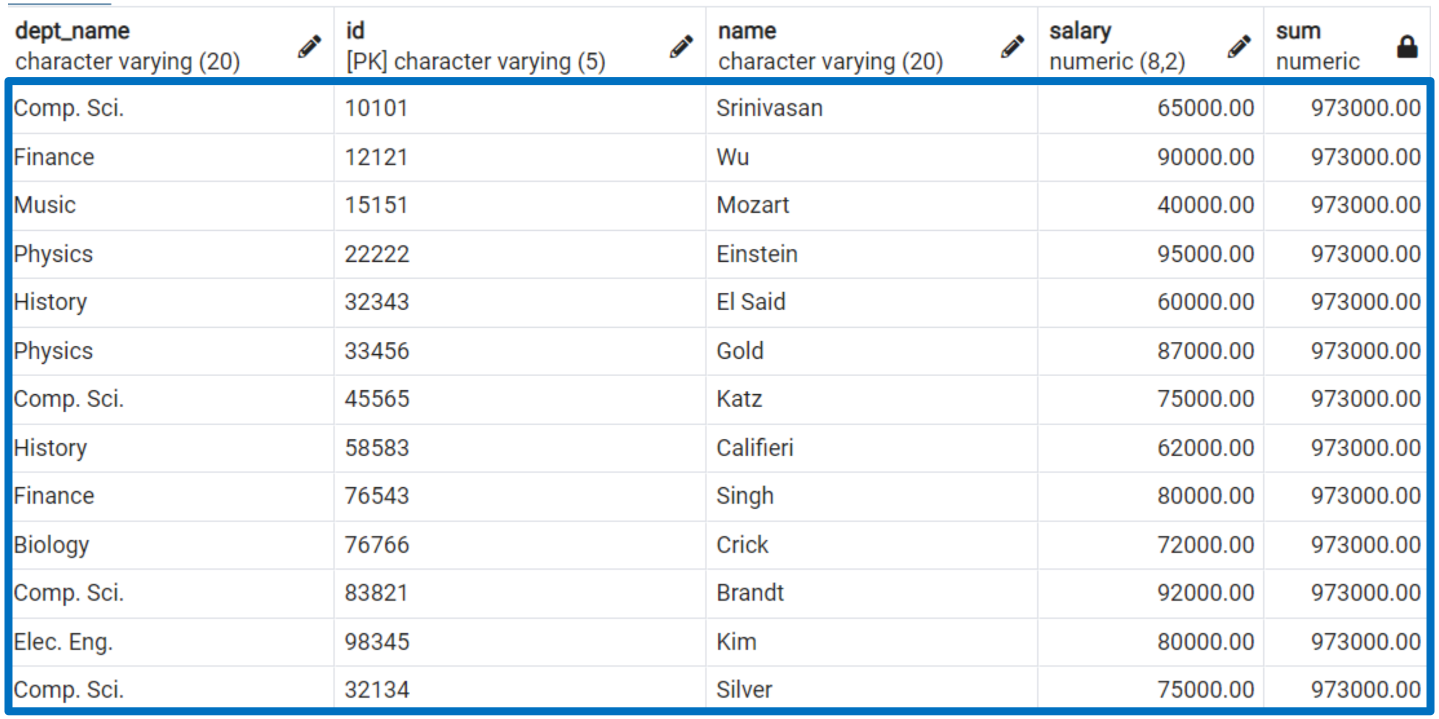


dept_name	id	name	salary	sum
character varying (20)	[PK] character varying (5)	character varying (20)	numeric (8,2)	numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	307000.00
Comp. Sci.	45565	Katz	75000.00	307000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Comp. Sci.	10101	Srinivasan	65000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	170000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	122000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	182000.00
Physics	33456	Gold	87000.00	182000.00

PARTITION BY

- The PARTITION BY clause is optional. If you skip the PARTITION BY clause, the window function will treat the whole result set as a single partition

```
SELECT dept_name, id, name, salary,
sum(salary) OVER ()
FROM instructor;
```

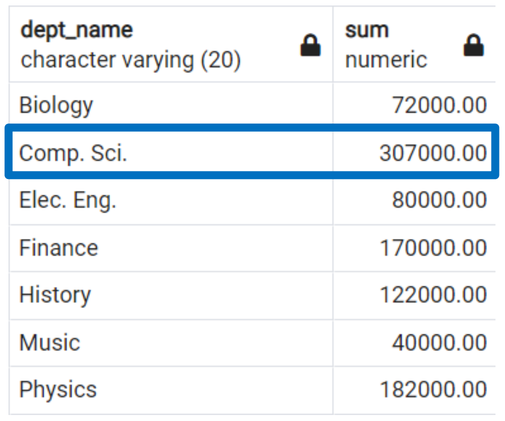


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Comp. Sci.	10101	Srinivasan	65000.00	973000.00
Finance	12121	Wu	90000.00	973000.00
Music	15151	Mozart	40000.00	973000.00
Physics	22222	Einstein	95000.00	973000.00
History	32343	El Said	60000.00	973000.00
Physics	33456	Gold	87000.00	973000.00
Comp. Sci.	45565	Katz	75000.00	973000.00
History	58583	Califieri	62000.00	973000.00
Finance	76543	Singh	80000.00	973000.00
Biology	76766	Crick	72000.00	973000.00
Comp. Sci.	83821	Brandt	92000.00	973000.00
Elec. Eng.	98345	Kim	80000.00	973000.00
Comp. Sci.	32134	Silver	75000.00	973000.00

Aggregation Functions vs Window Functions

Aggregation function

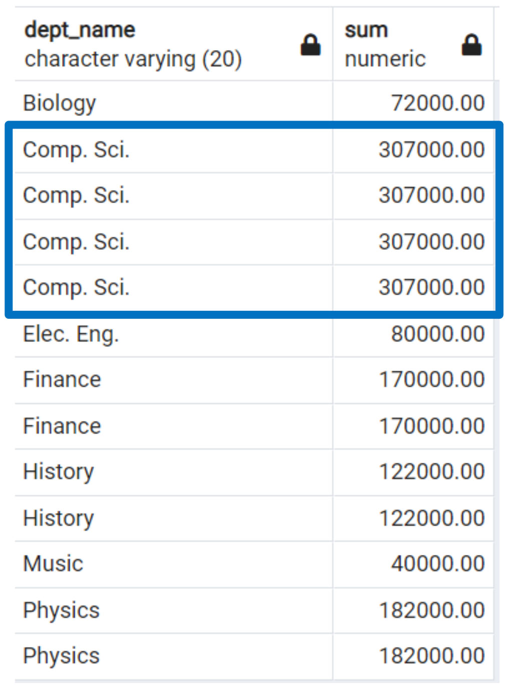
```
SELECT dept_name, sum(salary)
FROM instructor
GROUP BY dept_name
ORDER BY dept_name;
```



dept_name character varying (20)	sum numeric
Biology	72000.00
Comp. Sci.	307000.00
Elec. Eng.	80000.00
Finance	170000.00
History	122000.00
Music	40000.00
Physics	182000.00

Window function

```
SELECT dept_name,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```

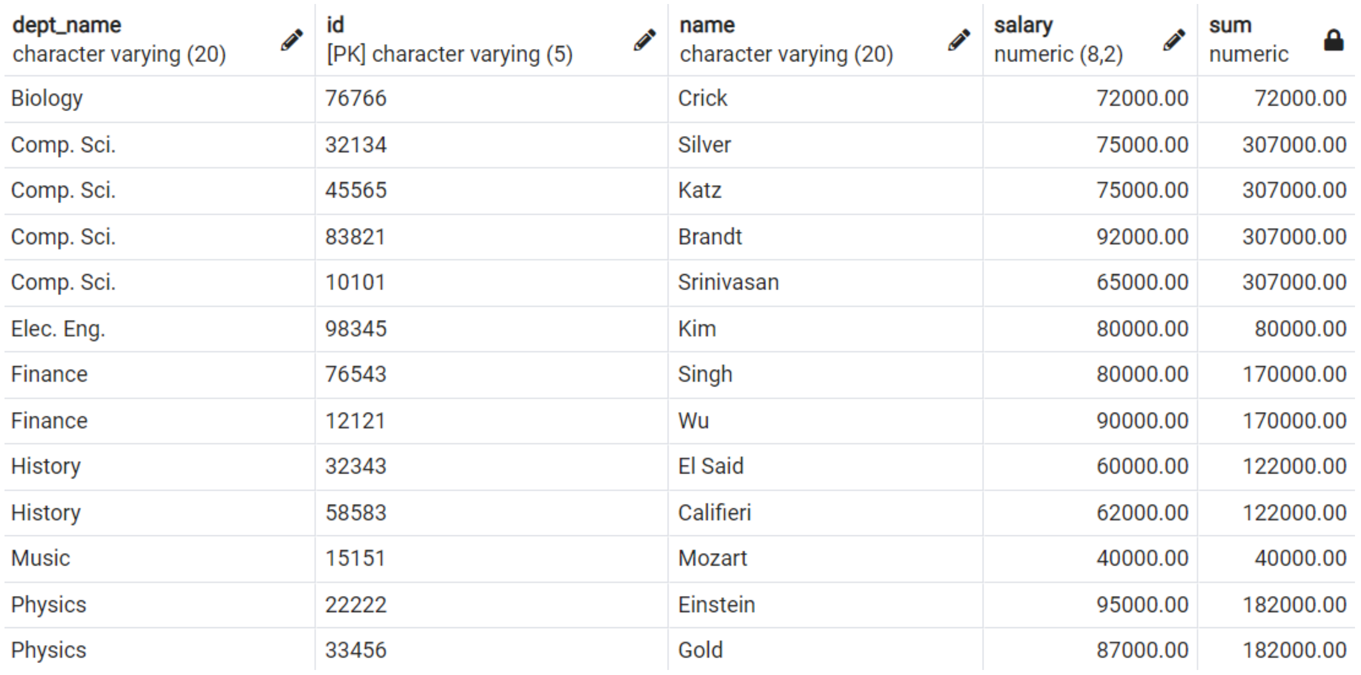


dept_name character varying (20)	sum numeric
Biology	72000.00
Comp. Sci.	307000.00
Comp. Sci.	307000.00
Comp. Sci.	307000.00
Comp. Sci.	307000.00
Elec. Eng.	80000.00
Finance	170000.00
Finance	170000.00
History	122000.00
History	122000.00
Music	40000.00
Physics	182000.00
Physics	182000.00

Why Window Functions?

- Window function is useful when you want to know the individual value and entire group value (sum, avg, …) at the same time

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	307000.00
Comp. Sci.	45565	Katz	75000.00	307000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Comp. Sci.	10101	Srinivasan	65000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	170000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	122000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	182000.00
Physics	33456	Gold	87000.00	182000.00

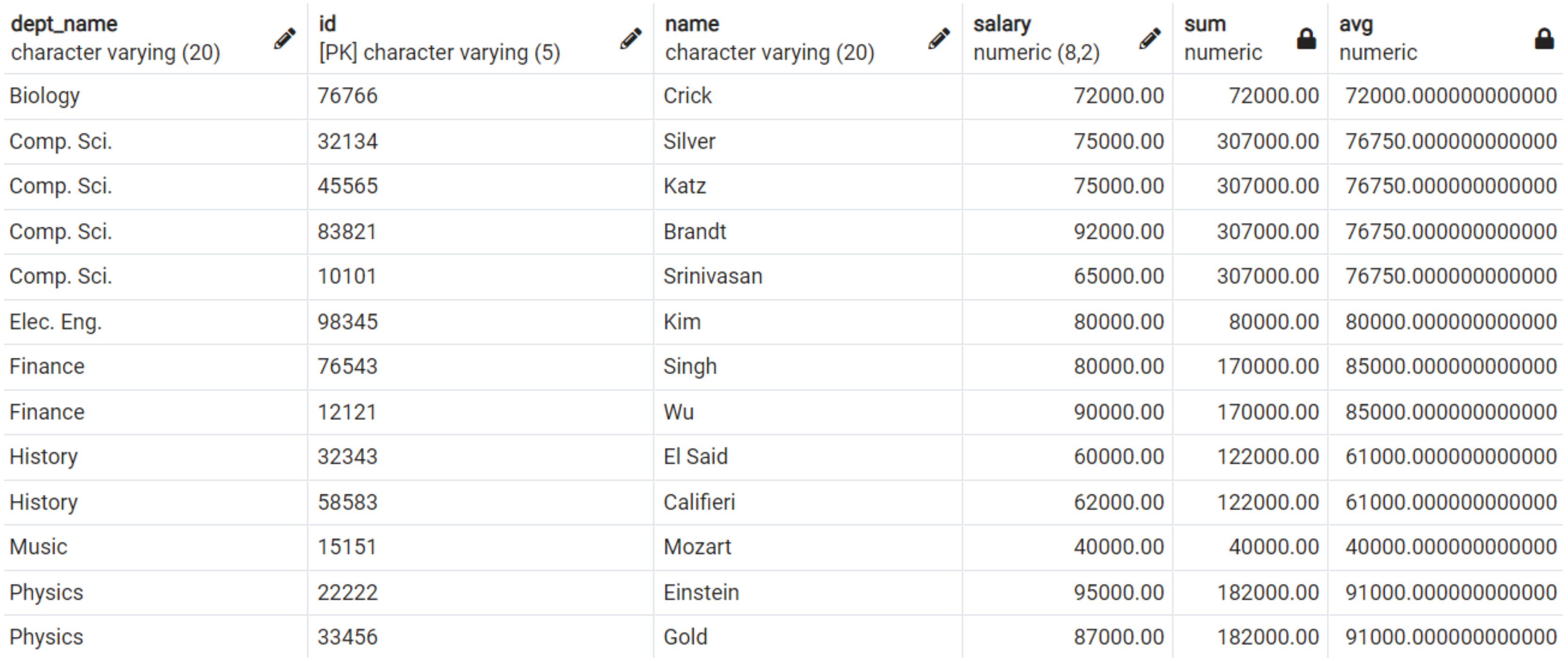
Multiple Window Functions

- You can use multiple window functions

```
SELECT
    wf1() OVER(PARTITION BY c1 ORDER BY c2),
    wf2() OVER(PARTITION BY c3 ORDER BY c4)
FROM table_name;
```

Multiple Window Functions

```sql
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name),
avg(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```

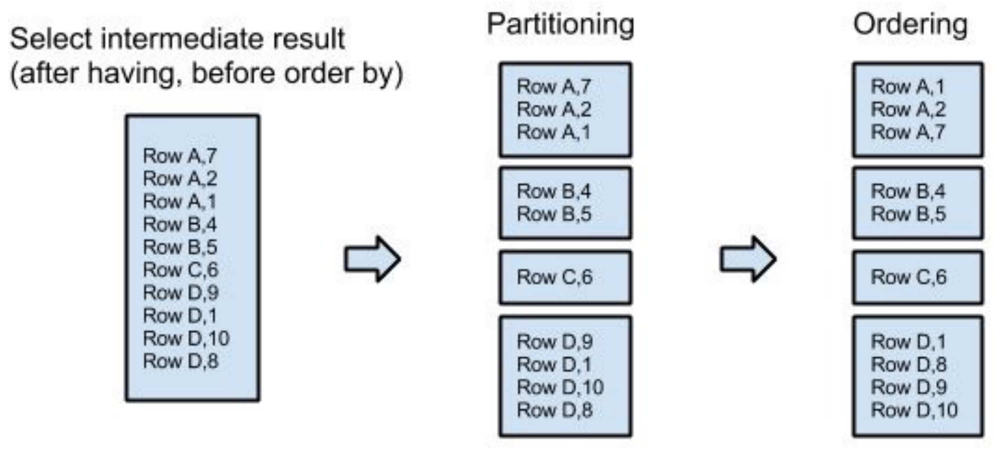


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric	avg numeric
Biology	76766	Crick	72000.00	72000.00	72000.000000000000
Comp. Sci.	32134	Silver	75000.00	307000.00	76750.000000000000
Comp. Sci.	45565	Katz	75000.00	307000.00	76750.000000000000
Comp. Sci.	83821	Brandt	92000.00	307000.00	76750.000000000000
Comp. Sci.	10101	Srinivasan	65000.00	307000.00	76750.000000000000
Elec. Eng.	98345	Kim	80000.00	80000.00	80000.000000000000
Finance	76543	Singh	80000.00	170000.00	85000.000000000000
Finance	12121	Wu	90000.00	170000.00	85000.000000000000
History	32343	El Said	60000.00	122000.00	61000.000000000000
History	58583	Califieri	62000.00	122000.00	61000.000000000000
Music	15151	Mozart	40000.00	40000.00	40000.000000000000
Physics	22222	Einstein	95000.00	182000.00	91000.000000000000
Physics	33456	Gold	87000.00	182000.00	91000.000000000000

ORDER BY

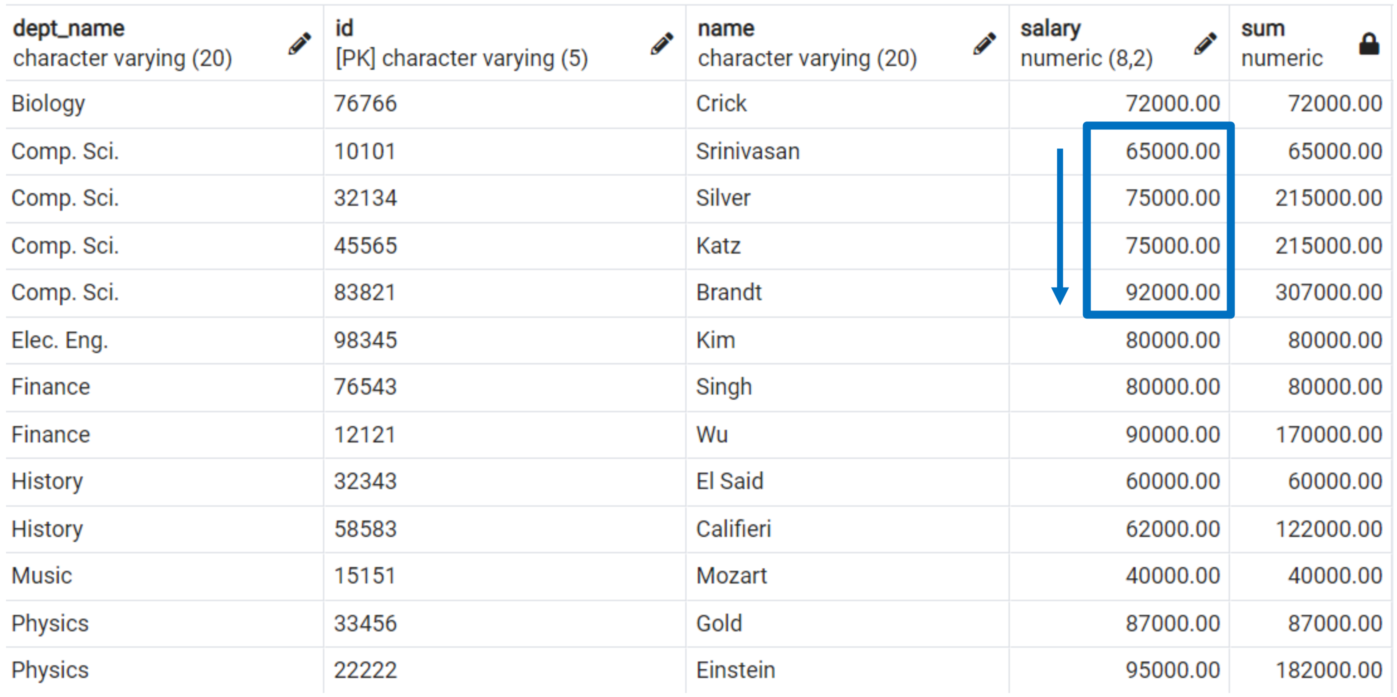
- You can also control the order in which rows are processed by window functions using ORDER BY within OVER

```
window_function(arg1, arg2,..) OVER (
  [PARTITION BY partition_expression]
  [ORDER BY sort_expression [ASC I DESC] [NULLS {FIRST I LAST }]
)
```



ORDER BY

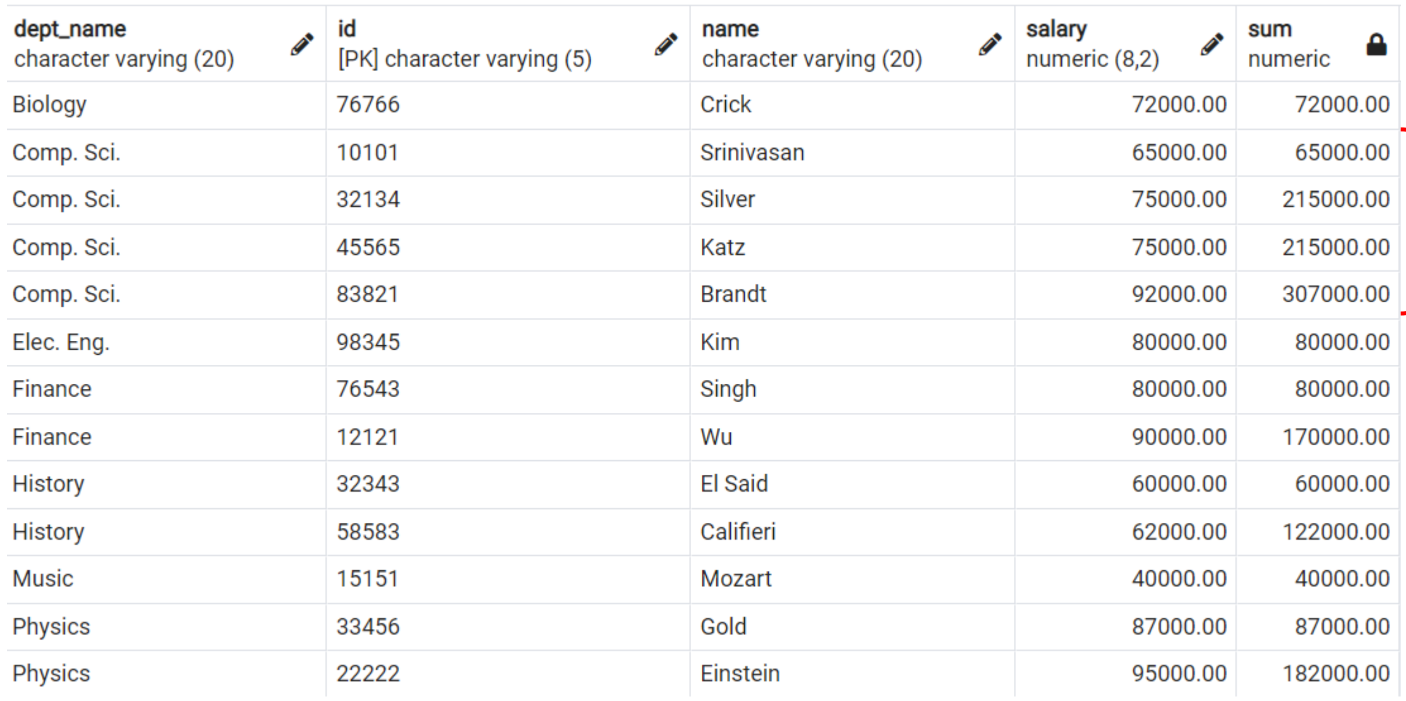
```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	215000.00
Comp. Sci.	45565	Katz	75000.00	215000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	182000.00

Window Frame

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```

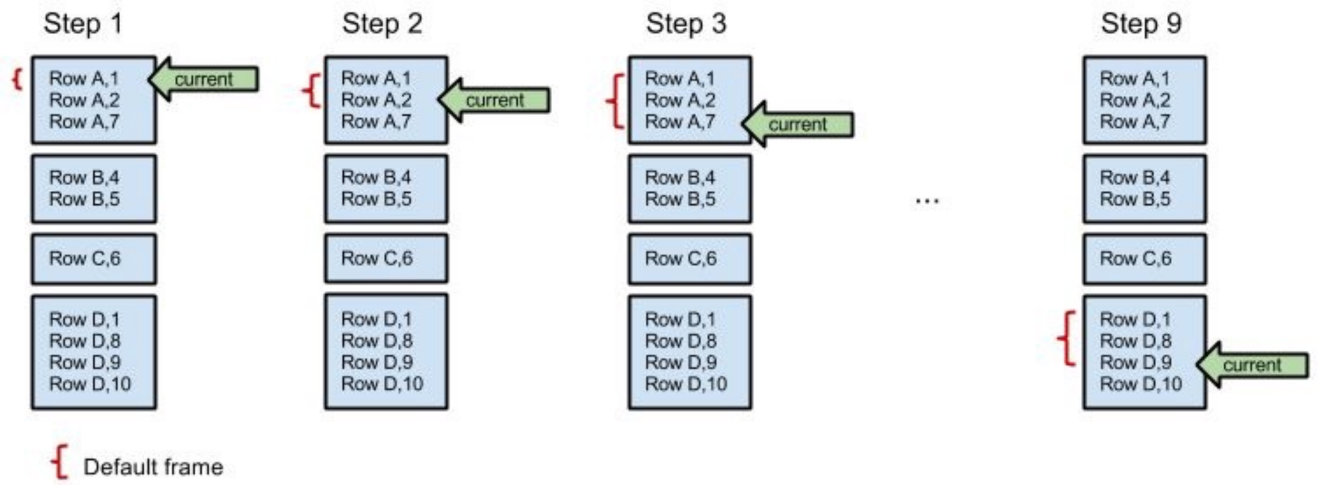


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	215000.00
Comp. Sci.	45565	Katz	75000.00	215000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	182000.00

Why sum is different in the same group?

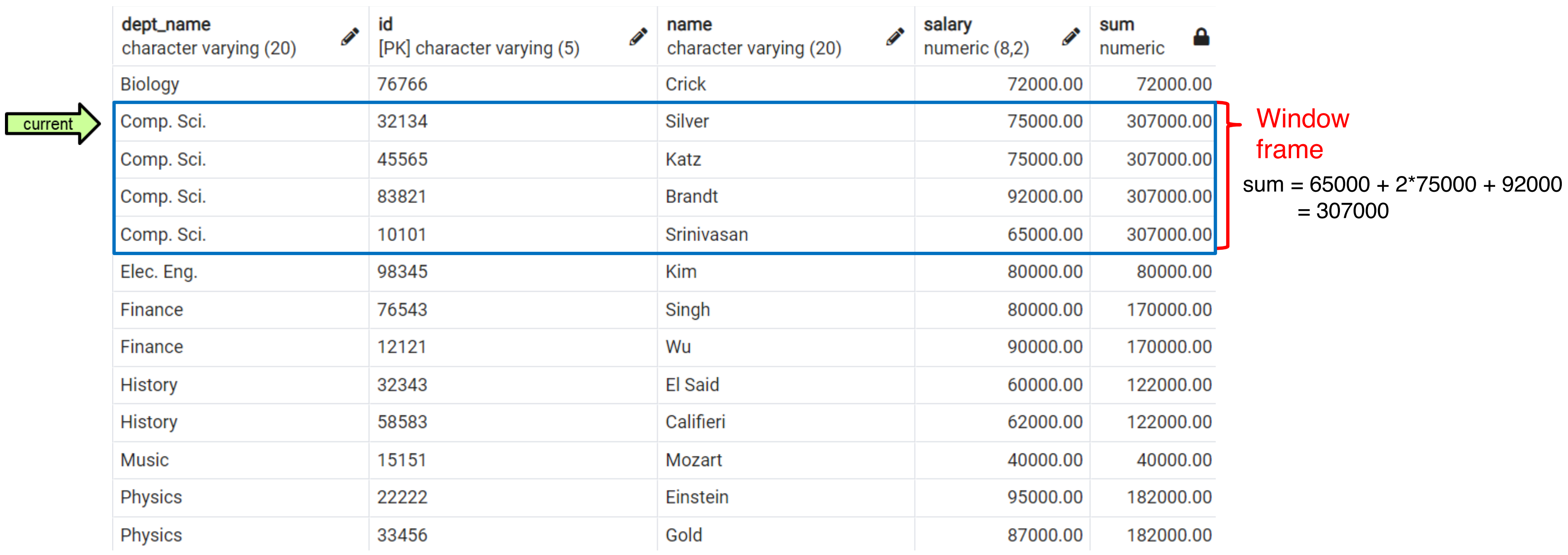
Window Frame

- There is another important concept associated with window functions: for each row, there is a set of rows within its partition called its window frame
  - Many (but not all) window functions act only on the rows of the window frame, rather than of the whole partition
  - avg( ), min( ), max( ), sum( ), count( ) etc
- When ORDER BY is omitted, the default frame consists of all rows in the partition
- When ORDER BY is supplied, the frame consists of all rows from the start of the partition up through the current row, plus any following rows that are equal to the current row according to the ORDER BY clause



Window Frame without ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```



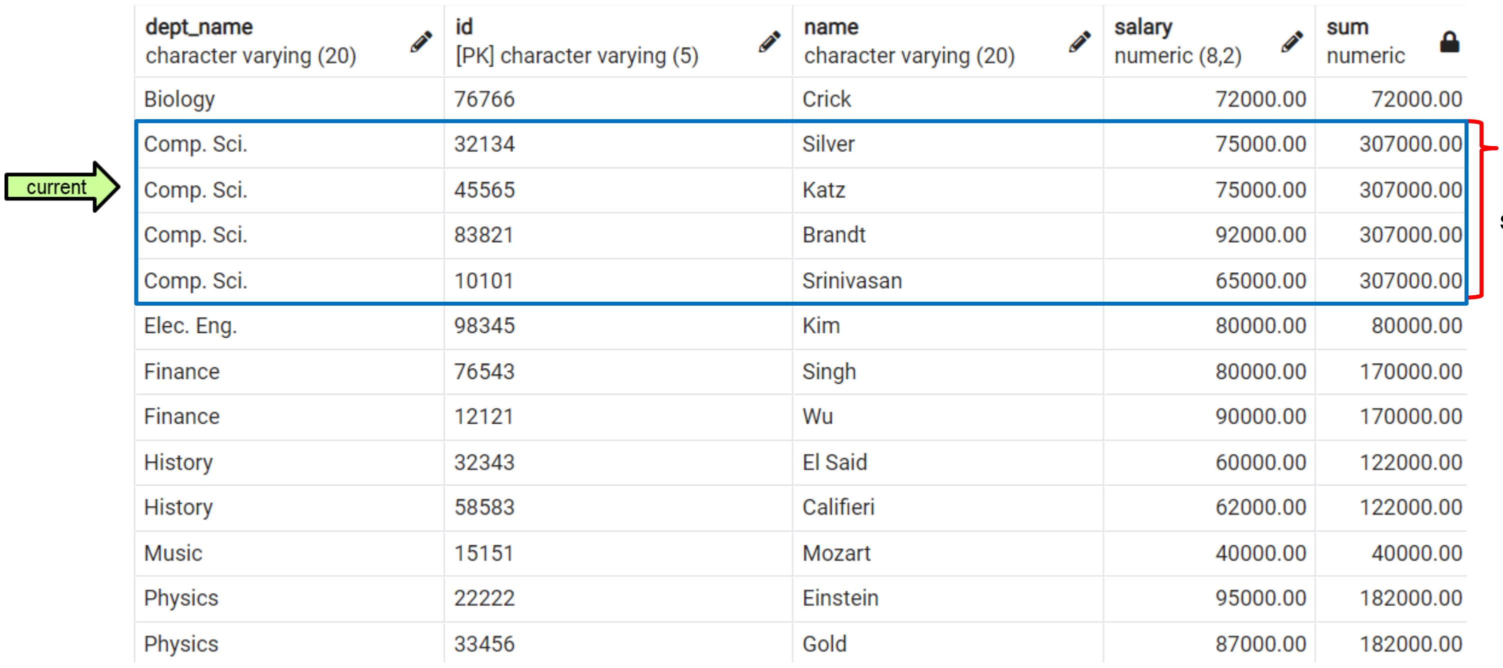
$sum = 65000 + 2*75000 + 92000$
$= 307000$

dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	307000.00
Comp. Sci.	45565	Katz	75000.00	307000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Comp. Sci.	10101	Srinivasan	65000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	170000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	122000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	182000.00
Physics	33456	Gold	87000.00	182000.00

KOREA UNIVERSITY
DATABASE LAB

Window Frame without ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```



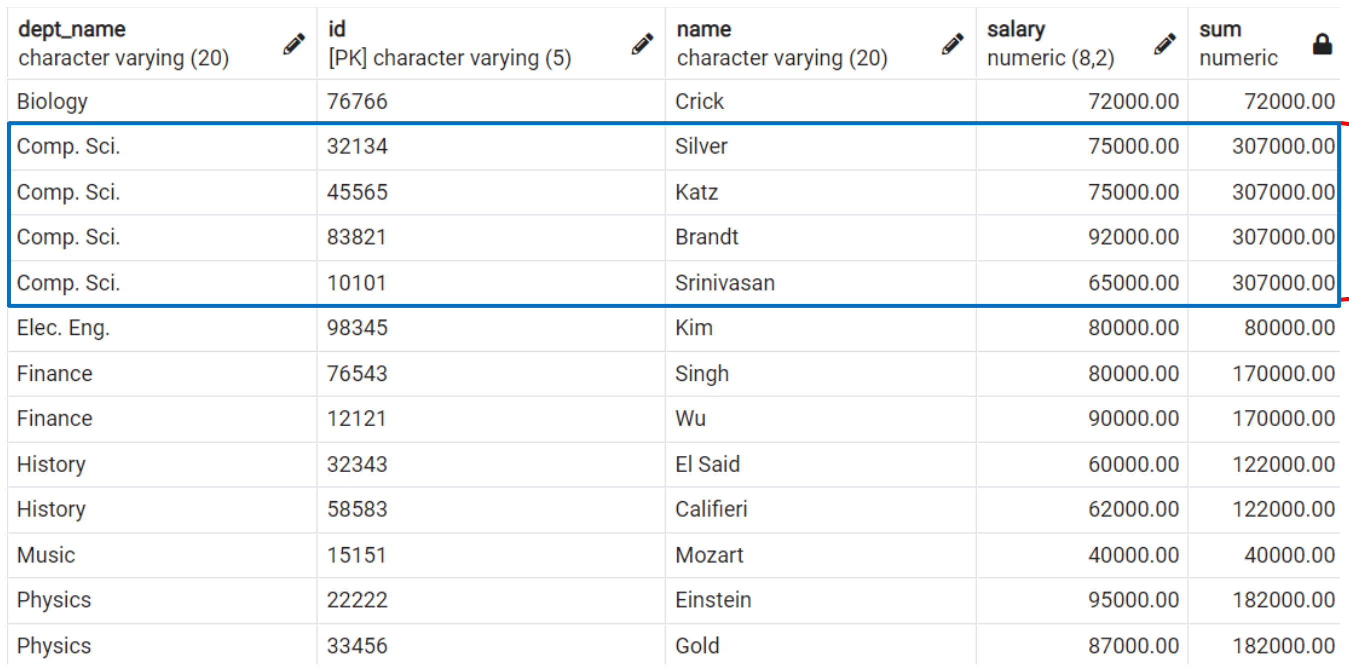
dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	307000.00
Comp. Sci.	45565	Katz	75000.00	307000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Comp. Sci.	10101	Srinivasan	65000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	170000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	122000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	182000.00
Physics	33456	Gold	87000.00	182000.00

Window frame

sum = 65000 + 2*75000 + 92000
= 307000

Window Frame without ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	307000.00
Comp. Sci.	45565	Katz	75000.00	307000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Comp. Sci.	10101	Srinivasan	65000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	170000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	122000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	182000.00
Physics	33456	Gold	87000.00	182000.00

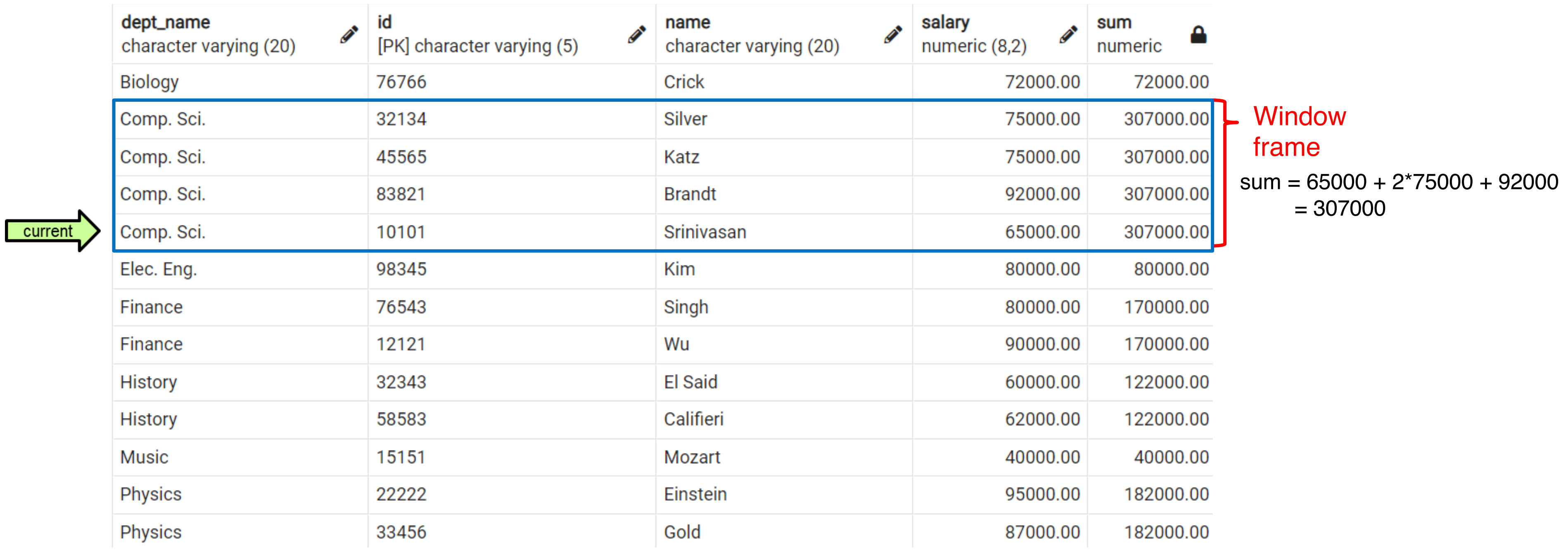
current

Window frame

sum = 65000 + 2*75000 + 92000
= 307000

Window Frame without ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```

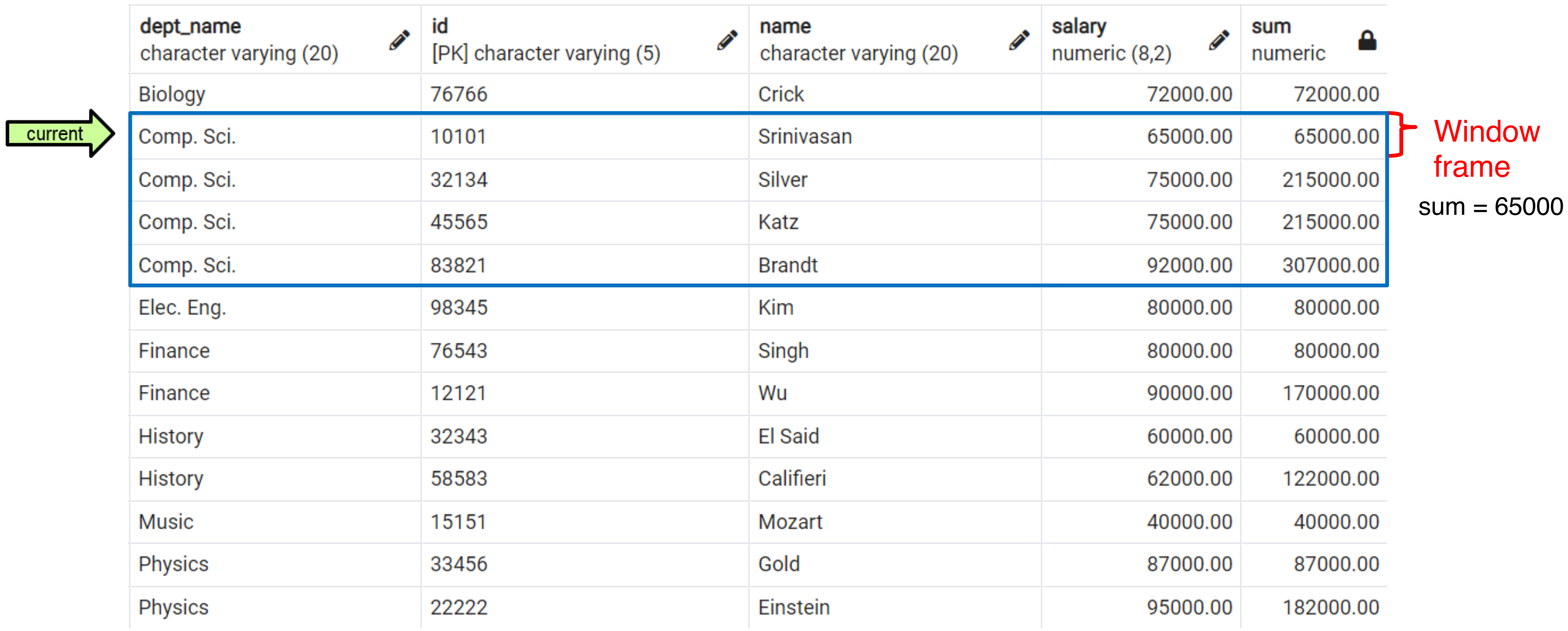


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	307000.00
Comp. Sci.	45565	Katz	75000.00	307000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Comp. Sci.	10101	Srinivasan	65000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	170000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	122000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	182000.00
Physics	33456	Gold	87000.00	182000.00

sum = 65000 + 2*75000 + 92000
= 307000

Window Frame with ORDER BY

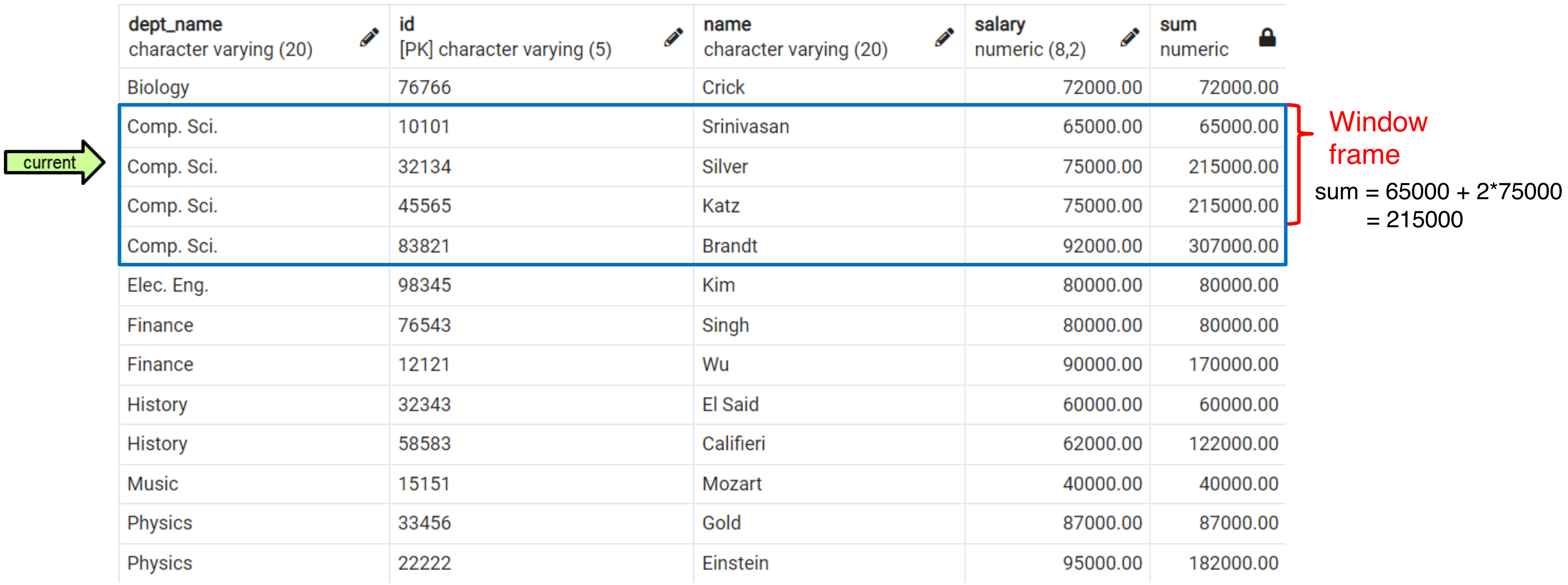
```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	215000.00
Comp. Sci.	45565	Katz	75000.00	215000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	182000.00

Window Frame with ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```

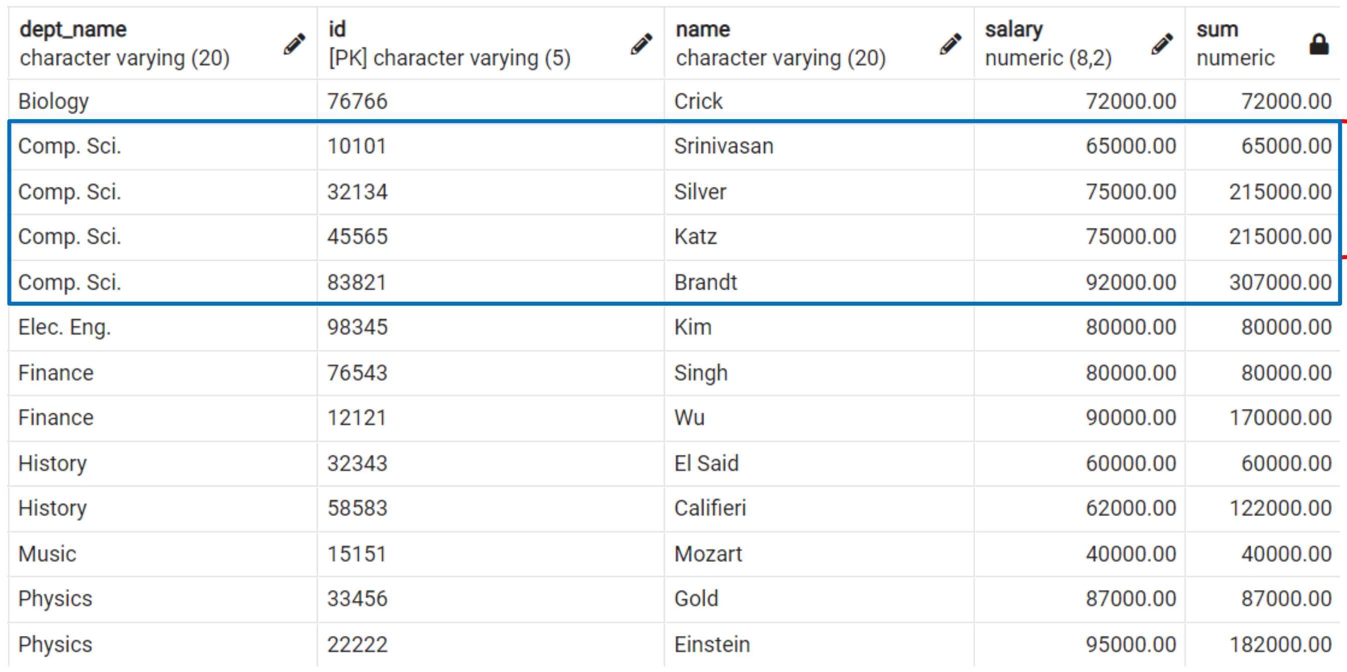


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	215000.00
Comp. Sci.	45565	Katz	75000.00	215000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	182000.00

sum = 65000 + 2*75000
= 215000

Window Frame with ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	215000.00
Comp. Sci.	45565	Katz	75000.00	215000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	182000.00

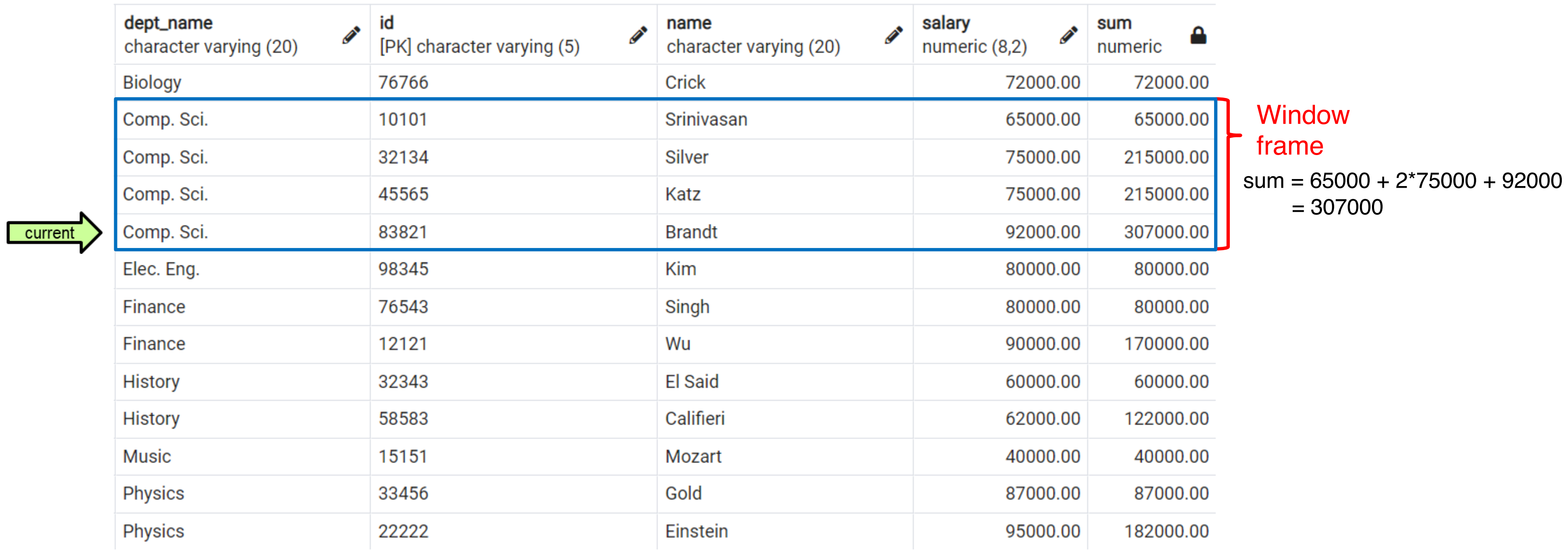
current

Window frame

sum = 65000 + 2*75000
= 215000

Window Frame with ORDER BY

```
SELECT dept_name, id, name, salary,
sum(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```

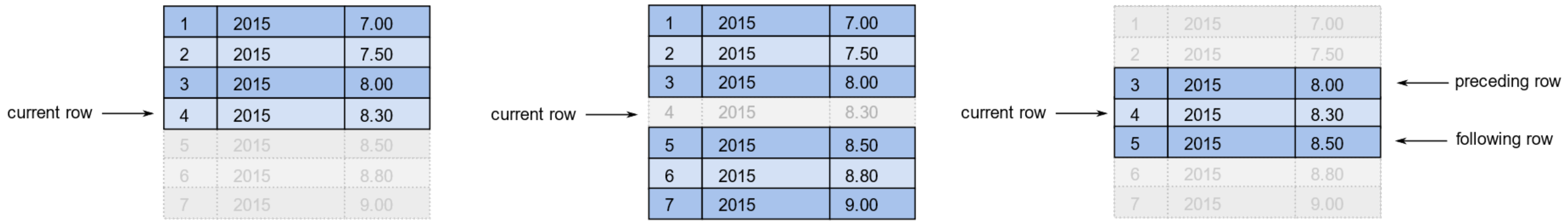


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	sum numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	215000.00
Comp. Sci.	45565	Katz	75000.00	215000.00
Comp. Sci.	83821	Brandt	92000.00	307000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	170000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	122000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	182000.00

sum = 65000 + 2*75000 + 92000
= 307000

Window Frame

- You can also create your own window frame
- But out of our range…



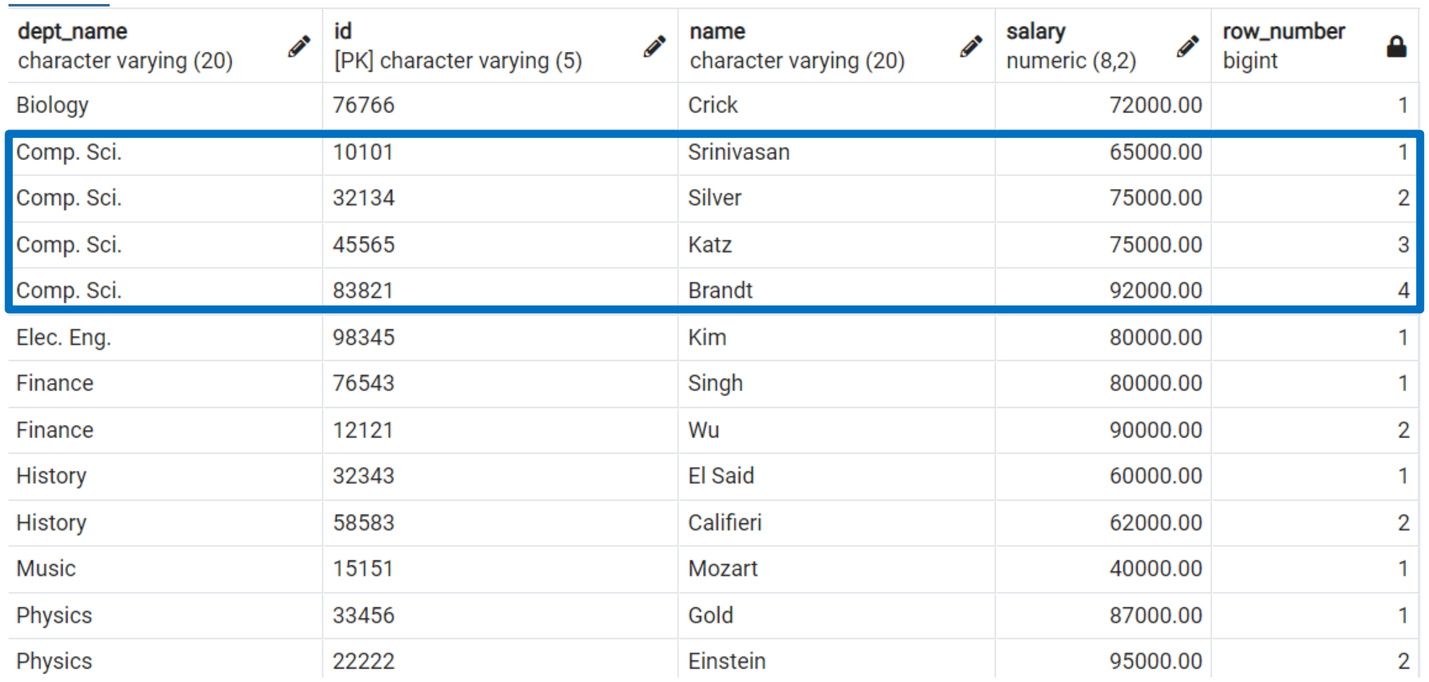
Window Functions

- AVG(), MIN(), MAX(), SUM() and COUNT()
- ROW_NUMBER()
  - Number the current row within its partition starting from 1
- FIRST_VALUE()
  - Return a value evaluated against the first row within its partition
- LAST_VALUE()
  - Return a value evaluated against the last row within its partition
- NTH_VALUE()
  - Return a value evaluated against the nth row in an ordered partition
- RANK()
  - Rank the current row within its partition with gaps
- DENSE_RANK()
  - Rank the current row within its partition without gaps

Window Functions

- The ROW_NUMBER() function assigns a sequential number to each row in each partition

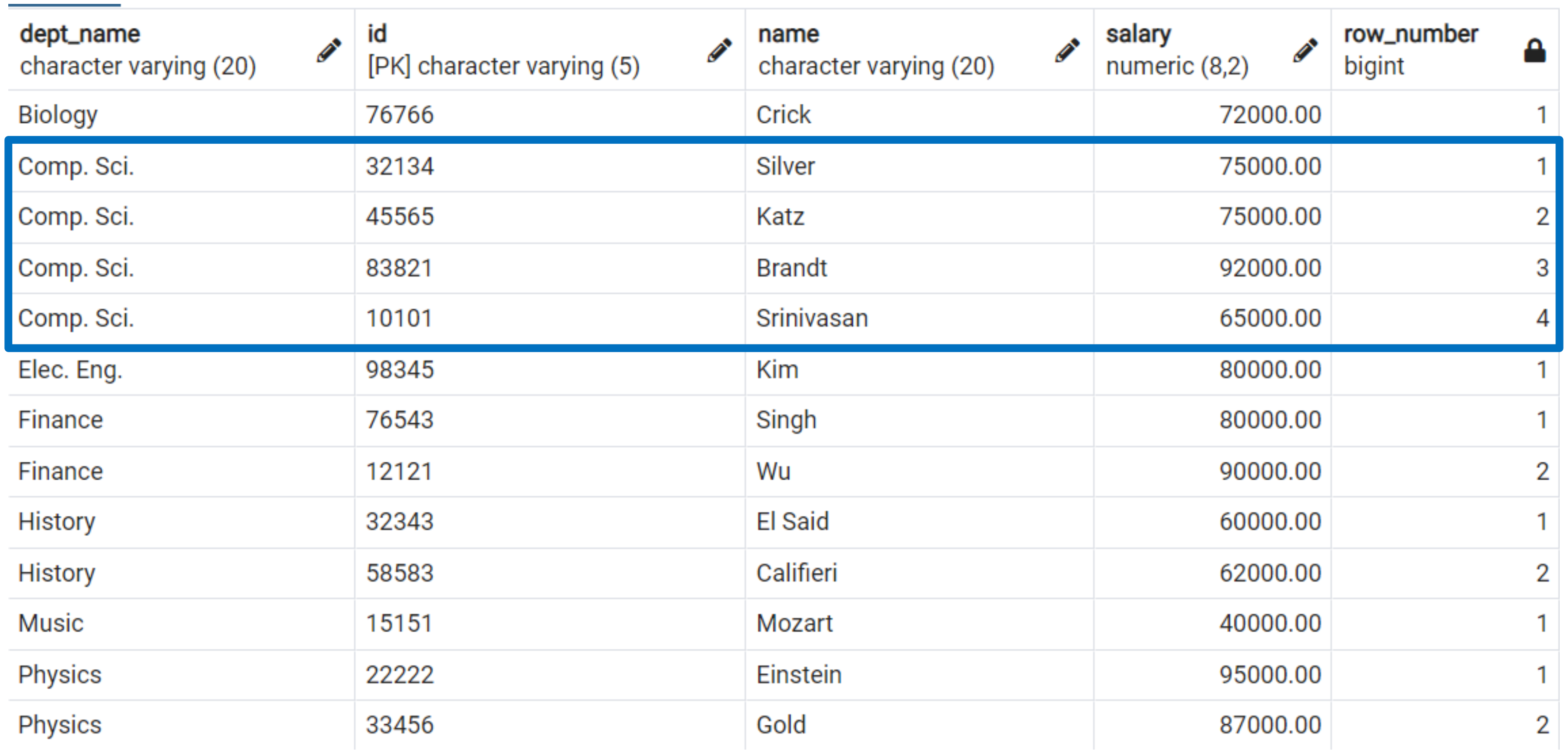
```
SELECT dept_name, id, name, salary,
ROW_NUMBER() OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	row_number bigint
Biology	76766	Crick	72000.00	1
Comp. Sci.	10101	Srinivasan	65000.00	1
Comp. Sci.	32134	Silver	75000.00	2
Comp. Sci.	45565	Katz	75000.00	3
Comp. Sci.	83821	Brandt	92000.00	4
Elec. Eng.	98345	Kim	80000.00	1
Finance	76543	Singh	80000.00	1
Finance	12121	Wu	90000.00	2
History	32343	El Said	60000.00	1
History	58583	Califieri	62000.00	2
Music	15151	Mozart	40000.00	1
Physics	33456	Gold	87000.00	1
Physics	22222	Einstein	95000.00	2

Window Functions

```
SELECT dept_name, id, name, salary,
ROW_NUMBER() OVER (PARTITION BY dept_name)
FROM instructor;
```

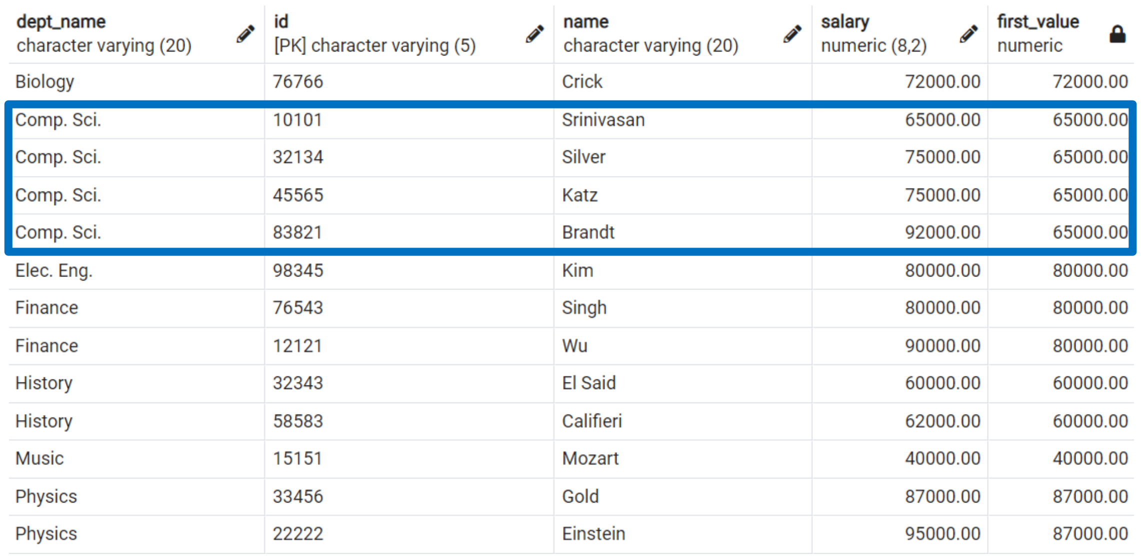


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	row_number bigint
Biology	76766	Crick	72000.00	1
Comp. Sci.	32134	Silver	75000.00	1
Comp. Sci.	45565	Katz	75000.00	2
Comp. Sci.	83821	Brandt	92000.00	3
Comp. Sci.	10101	Srinivasan	65000.00	4
Elec. Eng.	98345	Kim	80000.00	1
Finance	76543	Singh	80000.00	1
Finance	12121	Wu	90000.00	2
History	32343	El Said	60000.00	1
History	58583	Califieri	62000.00	2
Music	15151	Mozart	40000.00	1
Physics	22222	Einstein	95000.00	1
Physics	33456	Gold	87000.00	2

Window Functions

- FIRST_VALUE() function returns a value evaluated against the first row within its partition, whereas the LAST_VALUE() function returns a value evaluated against the last row in its partition
- The following statement uses the FIRST_VALUE() to return the lowest price for every product group

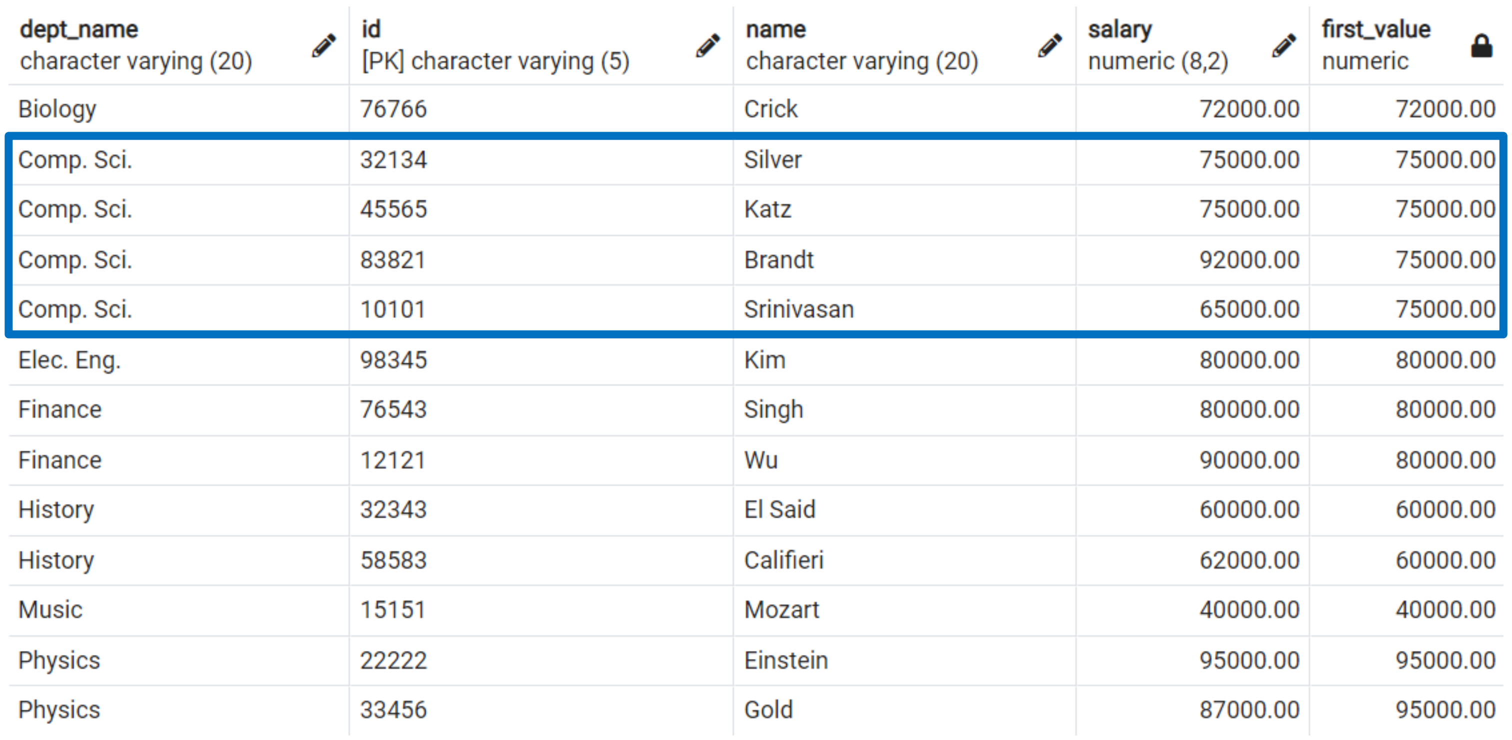
```
SELECT dept_name, id, name, salary,
FIRST_VALUE(salary) OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	first_value numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	10101	Srinivasan	65000.00	65000.00
Comp. Sci.	32134	Silver	75000.00	65000.00
Comp. Sci.	45565	Katz	75000.00	65000.00
Comp. Sci.	83821	Brandt	92000.00	65000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	80000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	60000.00
Music	15151	Mozart	40000.00	40000.00
Physics	33456	Gold	87000.00	87000.00
Physics	22222	Einstein	95000.00	87000.00

Window Functions

```
SELECT dept_name, id, name, salary,
FIRST_VALUE(salary) OVER (PARTITION BY dept_name)
FROM instructor;
```

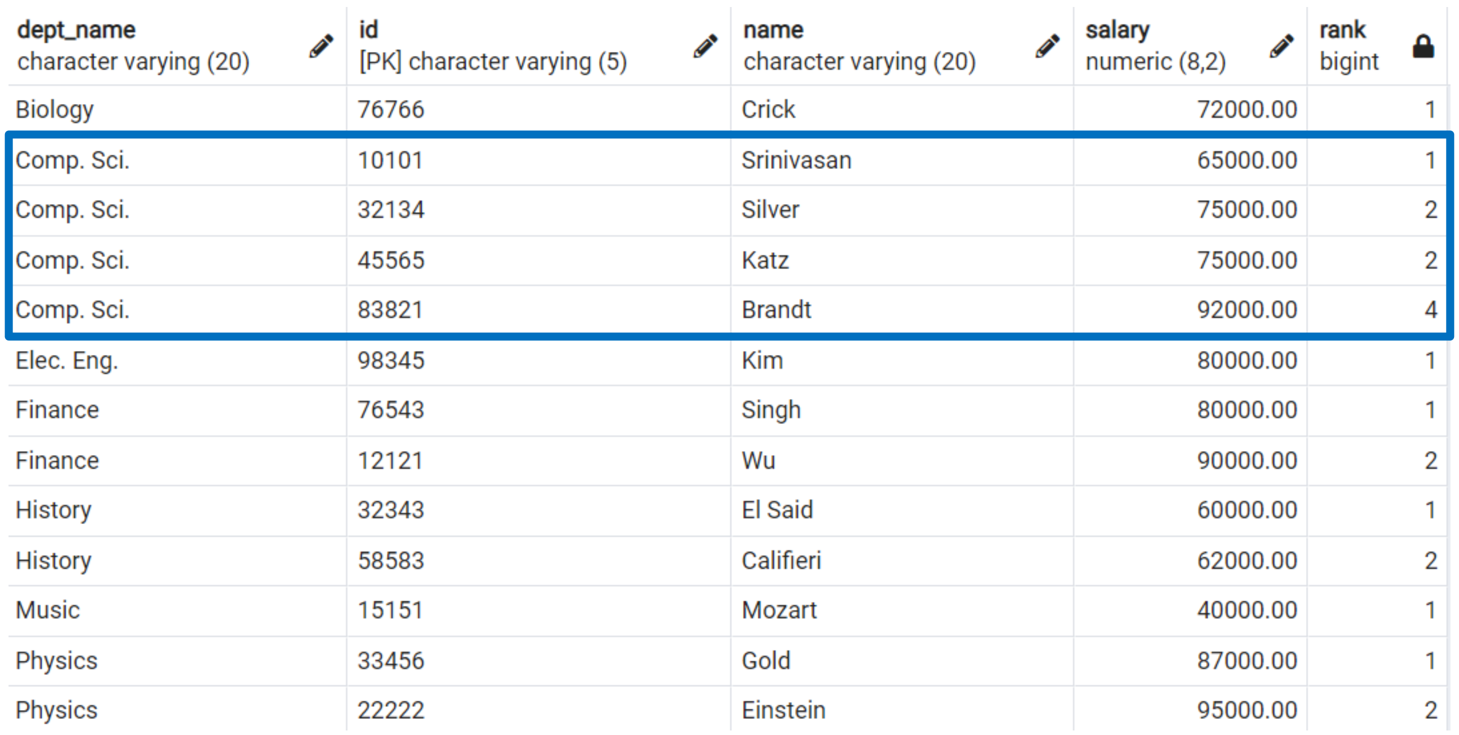


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	first_value numeric
Biology	76766	Crick	72000.00	72000.00
Comp. Sci.	32134	Silver	75000.00	75000.00
Comp. Sci.	45565	Katz	75000.00	75000.00
Comp. Sci.	83821	Brandt	92000.00	75000.00
Comp. Sci.	10101	Srinivasan	65000.00	75000.00
Elec. Eng.	98345	Kim	80000.00	80000.00
Finance	76543	Singh	80000.00	80000.00
Finance	12121	Wu	90000.00	80000.00
History	32343	El Said	60000.00	60000.00
History	58583	Califieri	62000.00	60000.00
Music	15151	Mozart	40000.00	40000.00
Physics	22222	Einstein	95000.00	95000.00
Physics	33456	Gold	87000.00	95000.00

Window Functions

- RANK() function assigns a rank to each row within an ordered partition

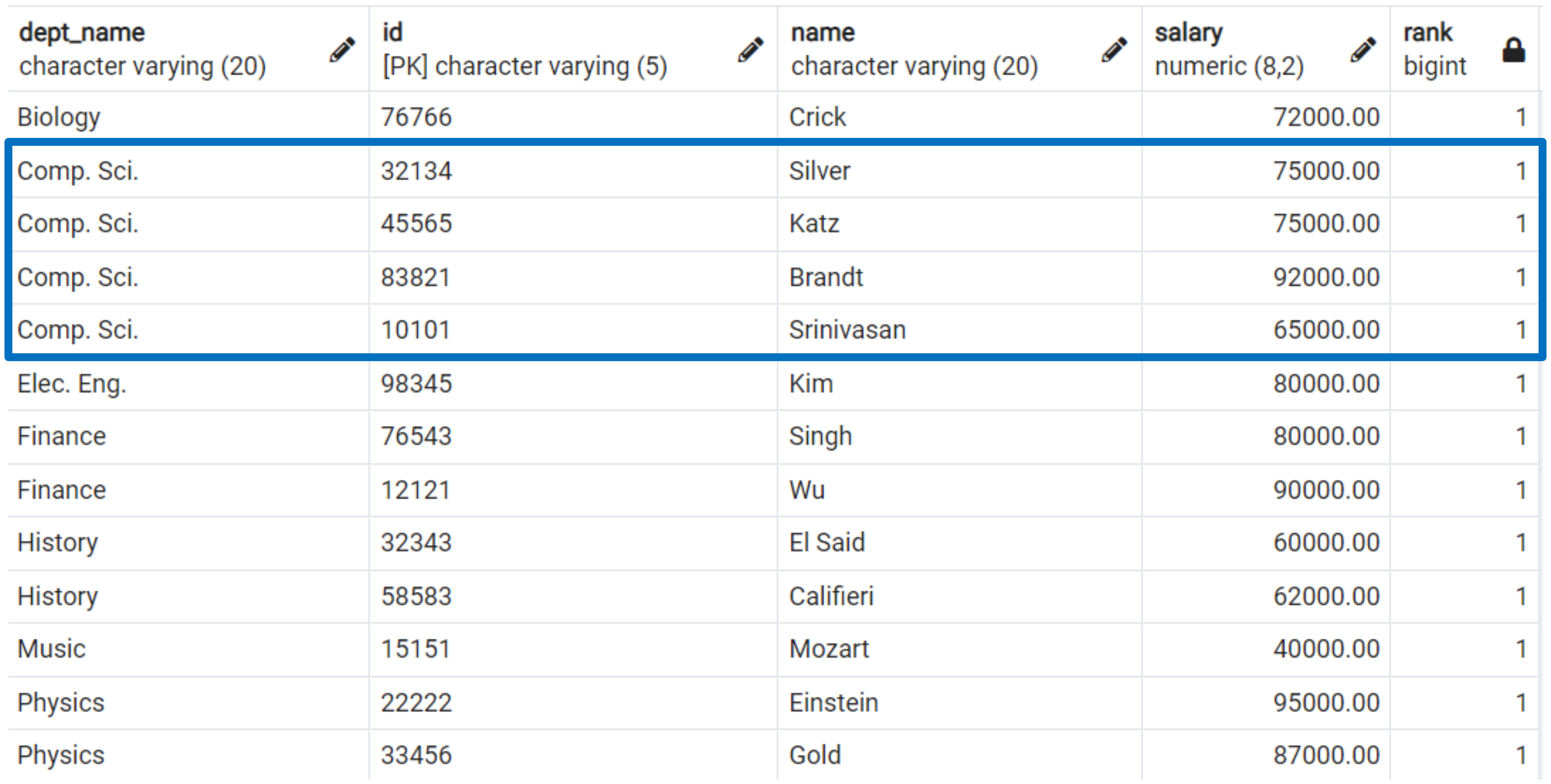
```
SELECT dept_name, id, name, salary,
RANK() OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	rank bigint
Biology	76766	Crick	72000.00	1
Comp. Sci.	10101	Srinivasan	65000.00	1
Comp. Sci.	32134	Silver	75000.00	2
Comp. Sci.	45565	Katz	75000.00	2
Comp. Sci.	83821	Brandt	92000.00	4
Elec. Eng.	98345	Kim	80000.00	1
Finance	76543	Singh	80000.00	1
Finance	12121	Wu	90000.00	2
History	32343	El Said	60000.00	1
History	58583	Califieri	62000.00	2
Music	15151	Mozart	40000.00	1
Physics	33456	Gold	87000.00	1
Physics	22222	Einstein	95000.00	2

Window Functions

```
SELECT dept_name, id, name, salary,
RANK() OVER (PARTITION BY dept_name)
FROM instructor;
```

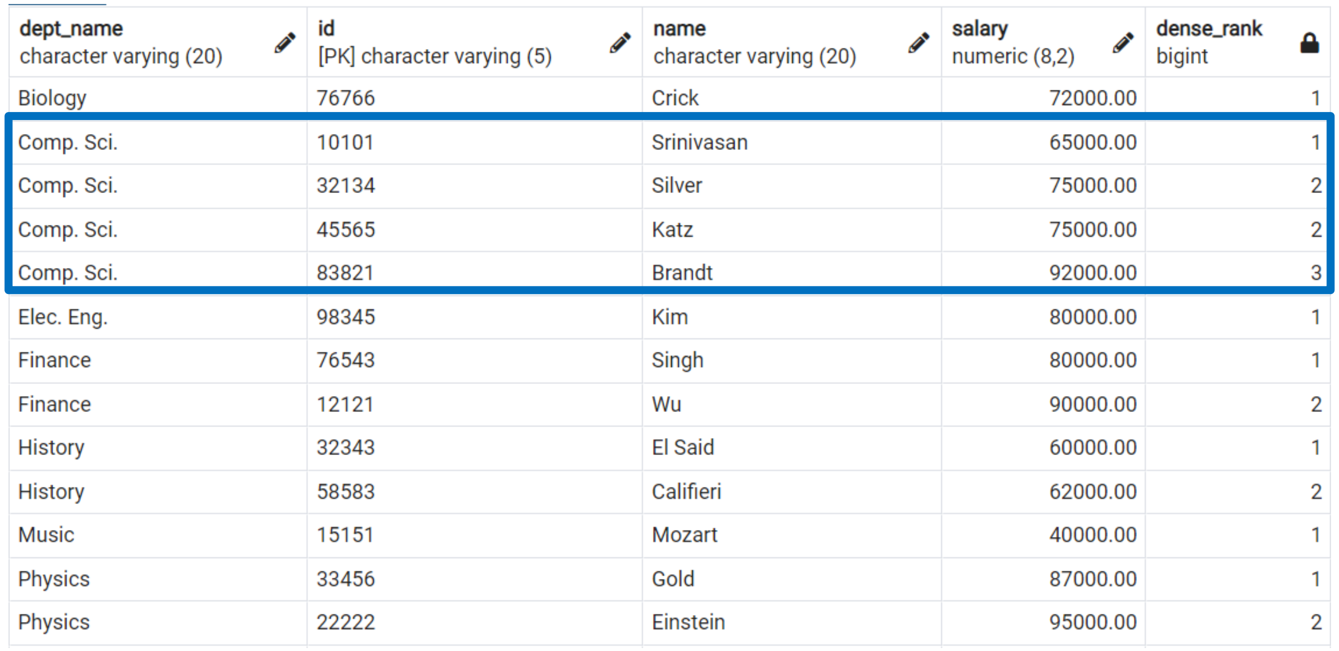


dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	rank bigint
Biology	76766	Crick	72000.00	1
Comp. Sci.	32134	Silver	75000.00	1
Comp. Sci.	45565	Katz	75000.00	1
Comp. Sci.	83821	Brandt	92000.00	1
Comp. Sci.	10101	Srinivasan	65000.00	1
Elec. Eng.	98345	Kim	80000.00	1
Finance	76543	Singh	80000.00	1
Finance	12121	Wu	90000.00	1
History	32343	El Said	60000.00	1
History	58583	Califieri	62000.00	1
Music	15151	Mozart	40000.00	1
Physics	22222	Einstein	95000.00	1
Physics	33456	Gold	87000.00	1

Window Functions

- DENSE_RANK() function assigns a rank to each row within an ordered partition, but the ranks have no gap. In other words, the same ranks are assigned to multiple rows and no ranks are skipped

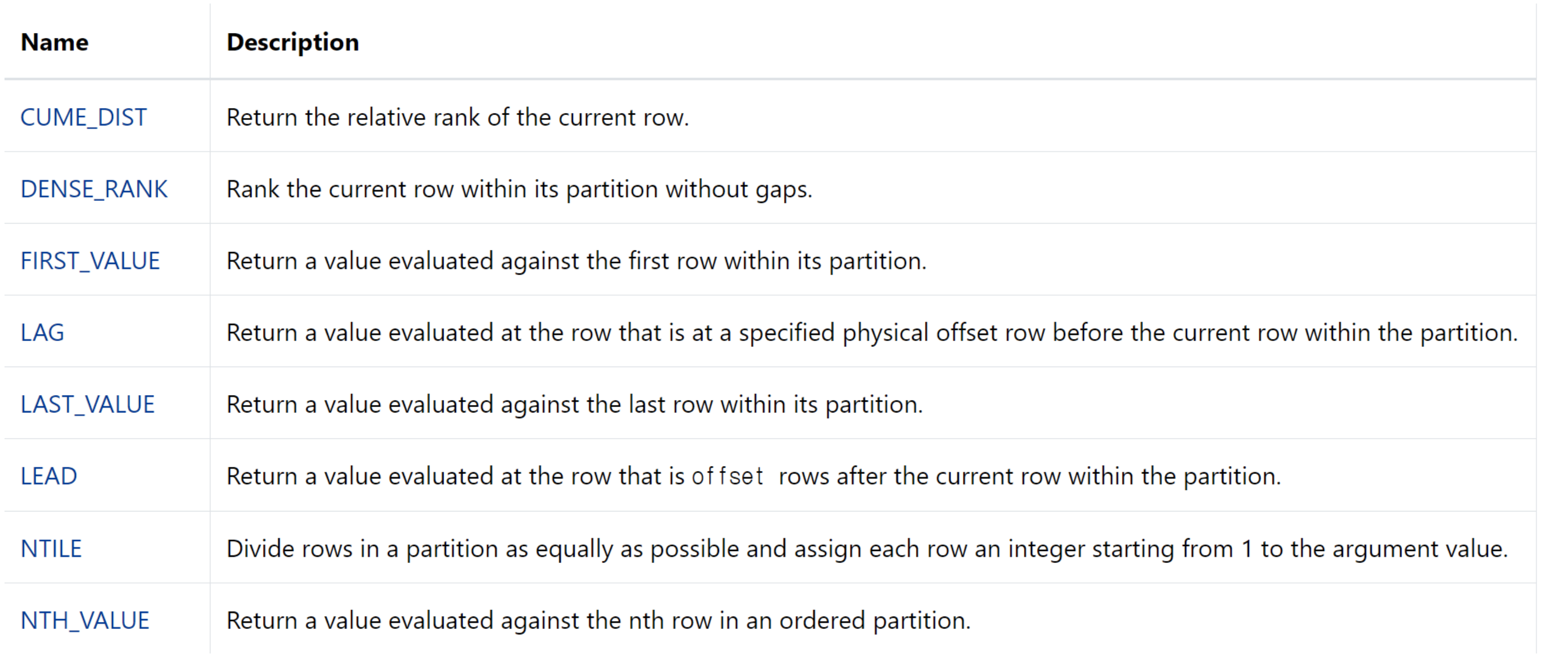
```
SELECT dept_name, id, name, salary,
DENSE_RANK() OVER (PARTITION BY dept_name ORDER BY salary)
FROM instructor;
```



dept_name character varying (20)	id [PK] character varying (5)	name character varying (20)	salary numeric (8,2)	dense_rank bigint
Biology	76766	Crick	72000.00	1
Comp. Sci.	10101	Srinivasan	65000.00	1
Comp. Sci.	32134	Silver	75000.00	2
Comp. Sci.	45565	Katz	75000.00	2
Comp. Sci.	83821	Brandt	92000.00	3
Elec. Eng.	98345	Kim	80000.00	1
Finance	76543	Singh	80000.00	1
Finance	12121	Wu	90000.00	2
History	32343	El Said	60000.00	1
History	58583	Califieri	62000.00	2
Music	15151	Mozart	40000.00	1
Physics	33456	Gold	87000.00	1
Physics	22222	Einstein	95000.00	2

Other Window Functions

- http://www.postgresqltutorial.com/postgresql-window-function/



Name	Description
CUME_DIST	Return the relative rank of the current row.
DENSE_RANK	Rank the current row within its partition without gaps.
FIRST_VALUE	Return a value evaluated against the first row within its partition.
LAG	Return a value evaluated at the row that is at a specified physical offset row before the current row within the partition.
LAST_VALUE	Return a value evaluated against the last row within its partition.
LEAD	Return a value evaluated at the row that is `offset` rows after the current row within the partition.
NTILE	Divide rows in a partition as equally as possible and assign each row an integer starting from 1 to the argument value.
NTH_VALUE	Return a value evaluated against the nth row in an ordered partition.