Chapter 6

How to code summary queries

Objectives

Applied

1. Code summary queries that use the aggregate functions AVG, SUM, MIN, MAX, COUNT, and COUNT(*), including queries that use the WITH ROLLUP operator and the GROUPING and IF functions.
2. Code summary queries that use aggregate window functions, including functions that use frames and named windows.

Knowledge

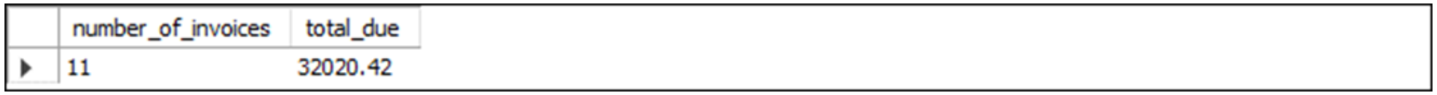
1. Describe summary queries.
2. Describe the differences between the HAVING clause and the WHERE clause.
3. Describe the use of the WITH ROLLUP operator.
4. Describe the use of the GROUPING and IF functions with the WITH ROLLUP operator.
5. Describe the use of the aggregate window functions.

The syntax of some common aggregate functions

```
AVG([ALL|DISTINCT] expression)
SUM([ALL|DISTINCT] expression)
MIN([ALL|DISTINCT] expression)
MAX([ALL|DISTINCT] expression)
COUNT([ALL|DISTINCT] expression)
COUNT(*)
```

A summary query

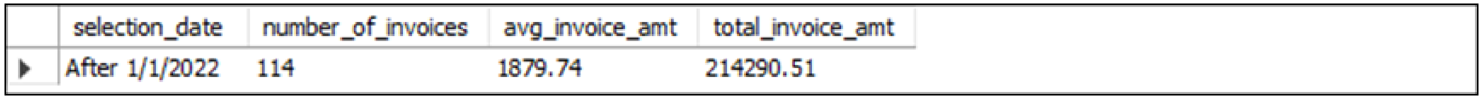
```
SELECT COUNT(*) AS number_of_invoices,
    SUM(invoice_total - payment_total - credit_total)
    AS total_due
FROM invoices
WHERE invoice_total - payment_total - credit_total > 0
```



	number_of_invoices	total_due
▶	11	32020.42

A summary query with COUNT(*), AVG, and SUM

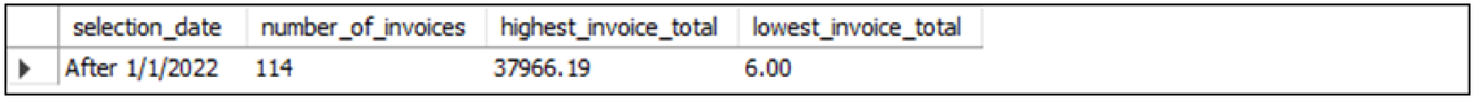
```
SELECT 'After 1/1/2022' AS selection_date,
    COUNT(*) AS number_of_invoices,
    ROUND(AVG(invoice_total), 2) AS avg_invoice_amt,
    SUM(invoice_total) AS total_invoice_amt
FROM invoices
WHERE invoice_date > '2022-01-01'
```



selection_date	number_of_invoices	avg_invoice_amt	total_invoice_amt
After 1/1/2022	114	1879.74	214290.51

A summary query with MIN and MAX

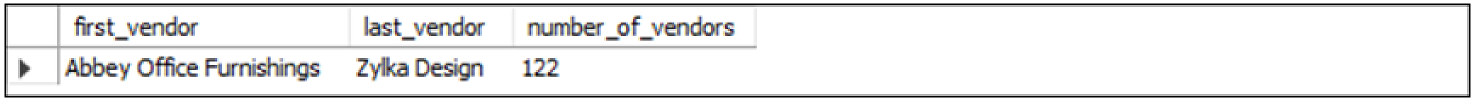
```
SELECT 'After 1/1/2022' AS selection_date,
    COUNT(*) AS number_of_invoices,
    MAX(invoice_total) AS highest_invoice_total,
    MIN(invoice_total) AS lowest_invoice_total
FROM invoices
WHERE invoice_date > '2022-01-01'
```



	selection_date	number_of_invoices	highest_invoice_total	lowest_invoice_total
▶	After 1/1/2022	114	37966.19	6.00

A summary query for non-numeric columns

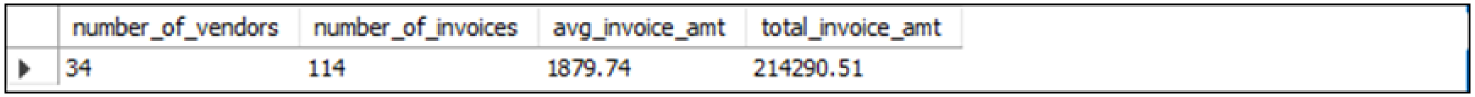
```
SELECT MIN(vendor_name) AS first_vendor,
    MAX(vendor_name) AS last_vendor,
    COUNT(vendor_name) AS number_of_vendors
FROM vendors
```



first_vendor	last_vendor	number_of_vendors
Abbey Office Furnishings	Zylka Design	122

A summary query with the DISTINCT keyword

```
SELECT COUNT(DISTINCT vendor_id) AS number_of_vendors,
    COUNT(vendor_id) AS number_of_invoices,
    ROUND(AVG(invoice_total), 2) AS avg_invoice_amt,
    SUM(invoice_total) AS total_invoice_amt
FROM invoices
WHERE invoice_date > '2022-01-01'
```



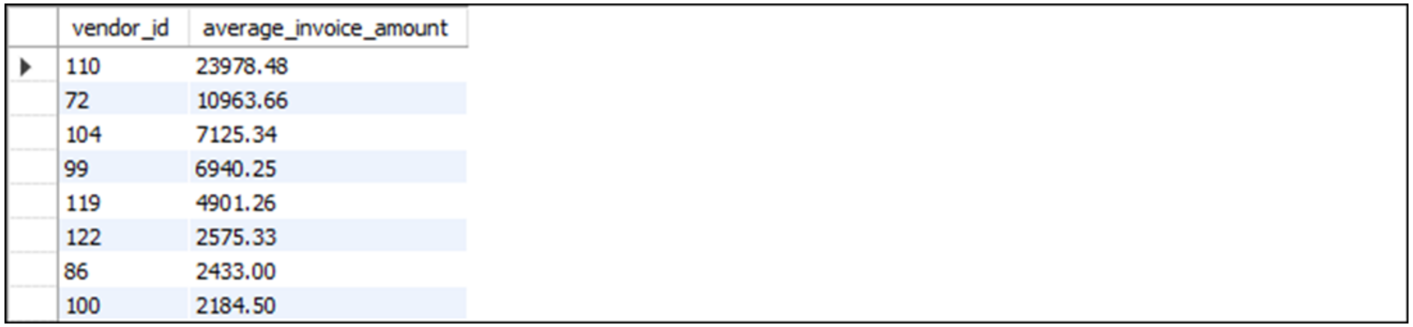
number_of_vendors	number_of_invoices	avg_invoice_amt	total_invoice_amt
34	114	1879.74	214290.51

The syntax of a SELECT statement with GROUP BY and HAVING clauses

```
SELECT select_list
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_list]
[HAVING search_condition]
[ORDER BY order_by_list]
```

A summary query that calculates the average invoice amount by vendor

```
SELECT vendor_id, ROUND(AVG(invoice_total), 2)
    AS average_invoice_amount
FROM invoices
GROUP BY vendor_id
HAVING AVG(invoice_total) > 2000
ORDER BY average_invoice_amount DESC
```



vendor_id	average_invoice_amount
110	23978.48
72	10963.66
104	7125.34
99	6940.25
119	4901.26
122	2575.33
86	2433.00
100	2184.50

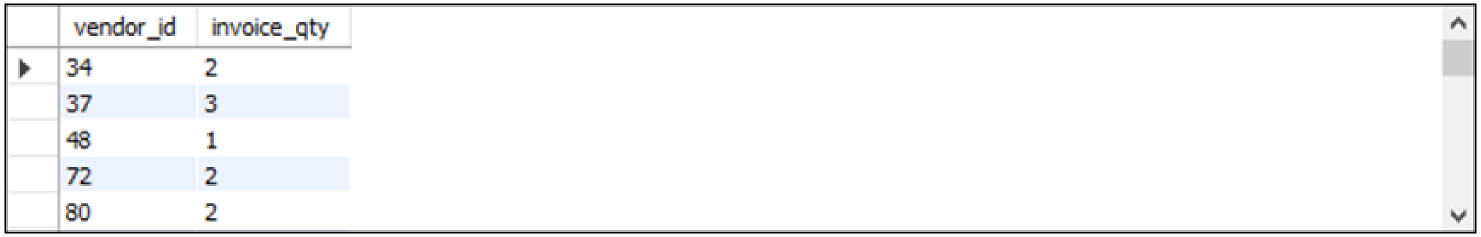
```
(8 rows)
```

A summary query that includes a functionally dependent column

```
SELECT vendor_name, vendor_state,
  ROUND(AVG(invoice_total), 2) AS average_invoice_amount
FROM vendors JOIN invoices
  ON vendors.vendor_id = invoices.vendor_id
GROUP BY vendor_name
HAVING AVG(invoice_total) > 2000
ORDER BY average_invoice_amount DESC
```

A summary query that counts the number of invoices by vendor

```
SELECT vendor_id, COUNT(*) AS invoice_qty
FROM invoices
GROUP BY vendor_id
```

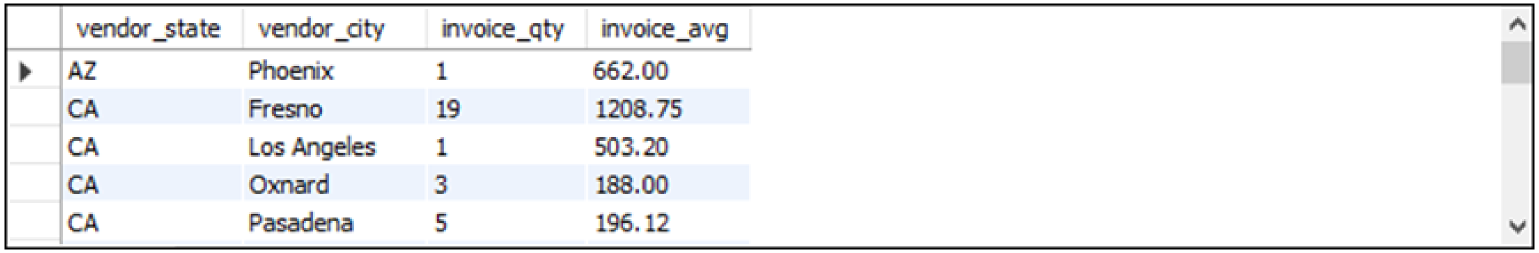


vendor_id	invoice_qty
34	2
37	3
48	1
72	2
80	2

```
(34 rows)
```

A summary query with a join

```
SELECT vendor_state, vendor_city, COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total), 2) AS invoice_avg
FROM invoices JOIN vendors
    ON invoices.vendor_id = vendors.vendor_id
GROUP BY vendor_state, vendor_city
ORDER BY vendor_state, vendor_city
```

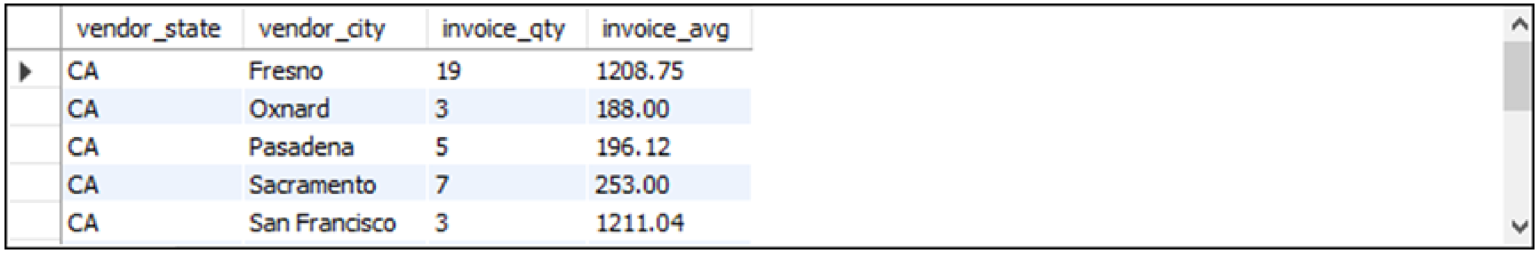


vendor_state	vendor_city	invoice_qty	invoice_avg
AZ	Phoenix	1	662.00
CA	Fresno	19	1208.75
CA	Los Angeles	1	503.20
CA	Oxnard	3	188.00
CA	Pasadena	5	196.12

```
(20 rows)
```

A summary query that limits the groups to those with two or more invoices

```
SELECT vendor_state, vendor_city, COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total), 2) AS invoice_avg
FROM invoices JOIN vendors
    ON invoices.vendor_id = vendors.vendor_id
GROUP BY vendor_state, vendor_city
HAVING COUNT(*) >= 2
ORDER BY vendor_state, vendor_city
```

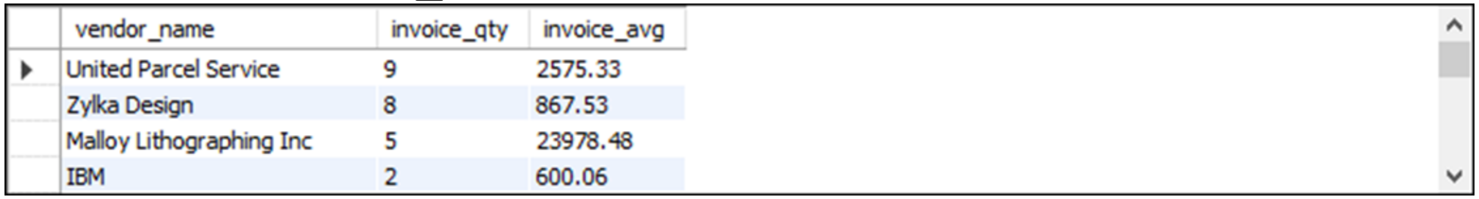


vendor_state	vendor_city	invoice_qty	invoice_avg
CA	Fresno	19	1208.75
CA	Oxnard	3	188.00
CA	Pasadena	5	196.12
CA	Sacramento	7	253.00
CA	San Francisco	3	1211.04

(12 rows)

A summary query with a search condition in the HAVING clause

```
SELECT vendor_name,
    COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total), 2) AS invoice_avg
FROM vendors JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
GROUP BY vendor_name
HAVING AVG(invoice_total) > 500
ORDER BY invoice_qty DESC
```

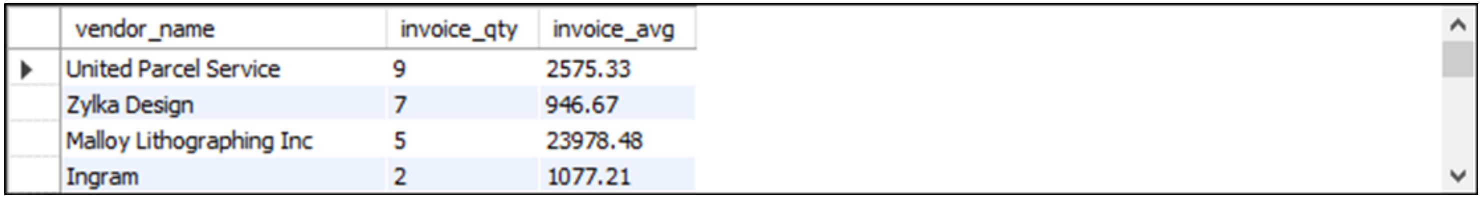


vendor_name	invoice_qty	invoice_avg
United Parcel Service	9	2575.33
Zylka Design	8	867.53
Malloy Lithographing Inc	5	23978.48
IBM	2	600.06

```
(19 rows)
```

A summary query with a search condition in the WHERE clause

```
SELECT vendor_name,
    COUNT(*) AS invoice_qty,
    ROUND(AVG(invoice_total), 2) AS invoice_avg
FROM vendors JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
WHERE invoice_total > 500
GROUP BY vendor_name
ORDER BY invoice_qty DESC
```



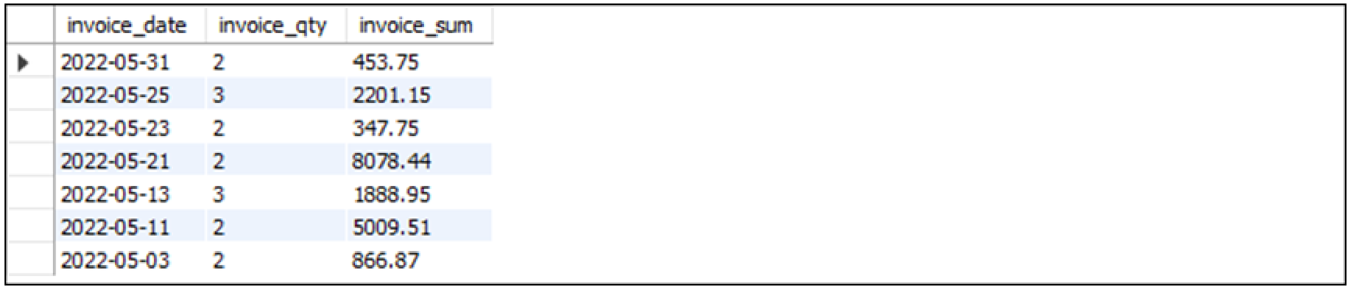
vendor_name	invoice_qty	invoice_avg
United Parcel Service	9	2575.33
Zylka Design	7	946.67
Malloy Lithographing Inc	5	23978.48
Ingram	2	1077.21

```
(20 rows)
```

A summary query with a compound condition in the HAVING clause

```
SELECT
    invoice_date,
    COUNT(*) AS invoice_qty,
    SUM(invoice_total) AS invoice_sum
FROM invoices
GROUP BY invoice_date
HAVING invoice_date BETWEEN '2018-05-01' AND '2018-05-31'
    AND COUNT(*) > 1
    AND SUM(invoice_total) > 100
ORDER BY invoice_date DESC
```

The result set



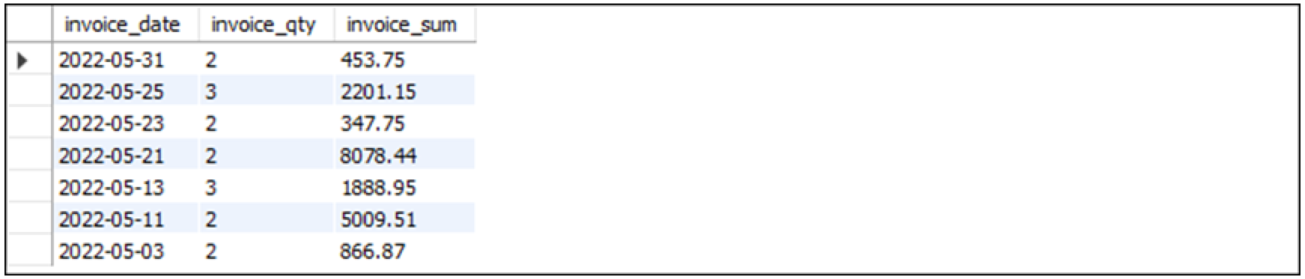
invoice_date	invoice_qty	invoice_sum
2022-05-31	2	453.75
2022-05-25	3	2201.15
2022-05-23	2	347.75
2022-05-21	2	8078.44
2022-05-13	3	1888.95
2022-05-11	2	5009.51
2022-05-03	2	866.87

(7 rows)

The same query coded with a WHERE clause

```
SELECT
    invoice_date,
    COUNT(*) AS invoice_qty,
    SUM(invoice_total) AS invoice_sum
FROM invoices
WHERE invoice_date BETWEEN '2018-05-01' AND '2018-05-31'
GROUP BY invoice_date
HAVING COUNT(*) > 1
    AND SUM(invoice_total) > 100
ORDER BY invoice_date DESC
```

The same result set

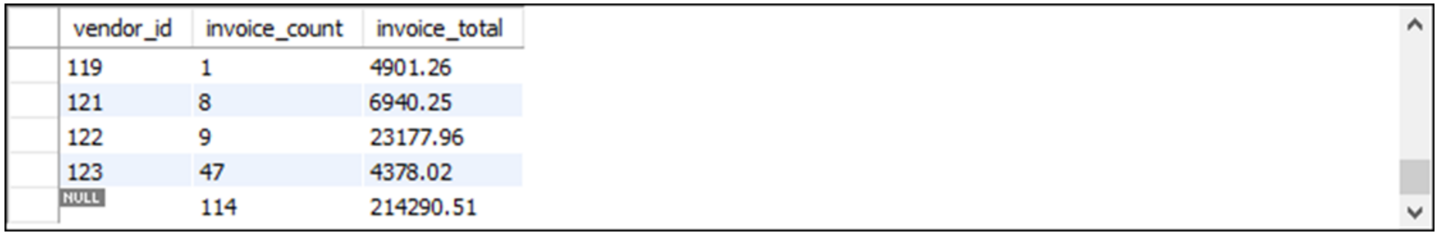


invoice_date	invoice_qty	invoice_sum
2022-05-31	2	453.75
2022-05-25	3	2201.15
2022-05-23	2	347.75
2022-05-21	2	8078.44
2022-05-13	3	1888.95
2022-05-11	2	5009.51
2022-05-03	2	866.87

```
(7 rows)
```

A summary query with a final summary row

```
SELECT vendor_id, COUNT(*) AS invoice_count,
    SUM(invoice_total) AS invoice_total
FROM invoices
GROUP BY vendor_id WITH ROLLUP
```

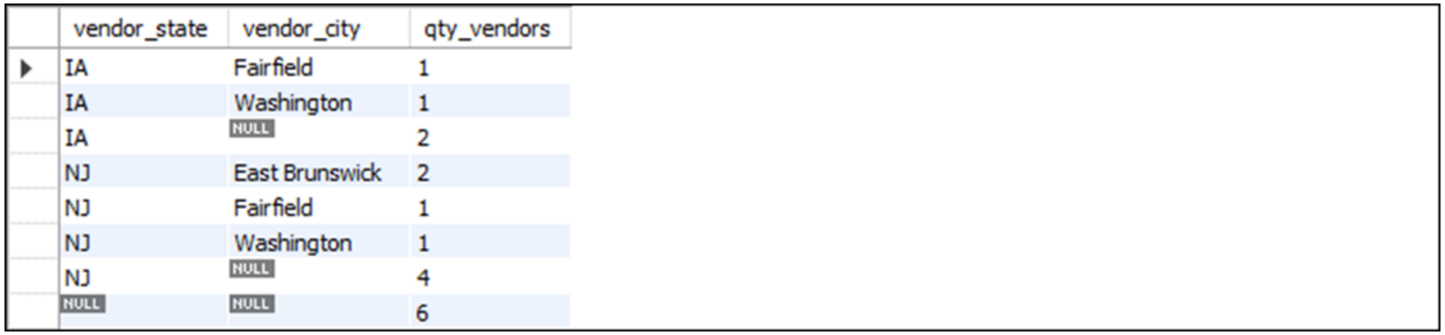


vendor_id	invoice_count	invoice_total
119	1	4901.26
121	8	6940.25
122	9	23177.96
123	47	4378.02
NULL	114	214290.51

```
(35 rows)
```

A summary query with a summary row for each grouping level

```
SELECT vendor_state, vendor_city, COUNT(*) AS qty_vendors
FROM vendors
WHERE vendor_state IN ('IA', 'NJ')
GROUP BY vendor_state, vendor_city WITH ROLLUP
```



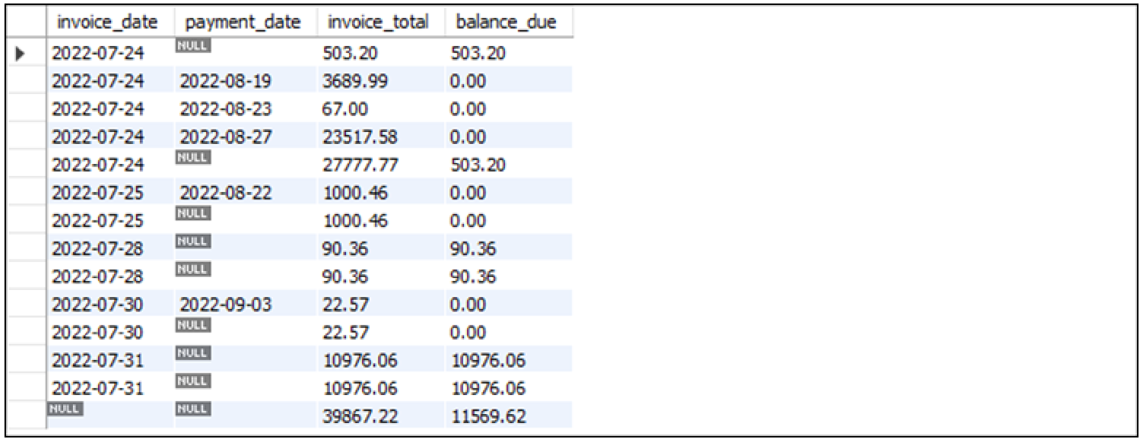
vendor_state	vendor_city	qty_vendors
IA	Fairfield	1
IA	Washington	1
IA	NULL	2
NJ	East Brunswick	2
NJ	Fairfield	1
NJ	Washington	1
NJ	NULL	4
NULL	NULL	6

The basic syntax of the GROUPING function

```
GROUPING(expression)
```

A summary query that uses WITH ROLLUP on a table with null values

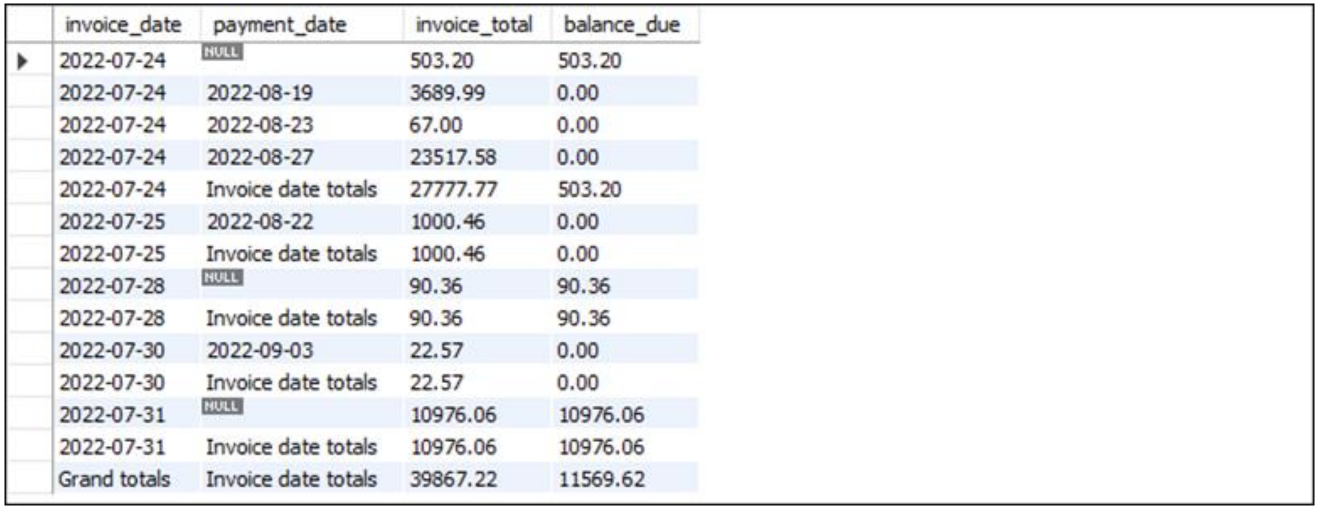
```
SELECT invoice_date, payment_date,
       SUM(invoice_total) AS invoice_total,
       SUM(invoice_total - credit_total - payment_total)
           AS balance_due
FROM invoices
WHERE invoice_date BETWEEN '2022-07-24' AND '2022-07-31'
GROUP BY invoice_date, payment_date WITH ROLLUP
```



invoice_date	payment_date	invoice_total	balance_due
2022-07-24	NULL	503.20	503.20
2022-07-24	2022-08-19	3689.99	0.00
2022-07-24	2022-08-23	67.00	0.00
2022-07-24	2022-08-27	23517.58	0.00
2022-07-24	NULL	27777.77	503.20
2022-07-25	2022-08-22	1000.46	0.00
2022-07-25	NULL	1000.46	0.00
2022-07-28	NULL	90.36	90.36
2022-07-28	NULL	90.36	90.36
2022-07-30	2022-09-03	22.57	0.00
2022-07-30	NULL	22.57	0.00
2022-07-31	NULL	10976.06	10976.06
2022-07-31	NULL	10976.06	10976.06
NULL	NULL	39867.22	11569.62

A query that substitutes literals for nulls in summary rows

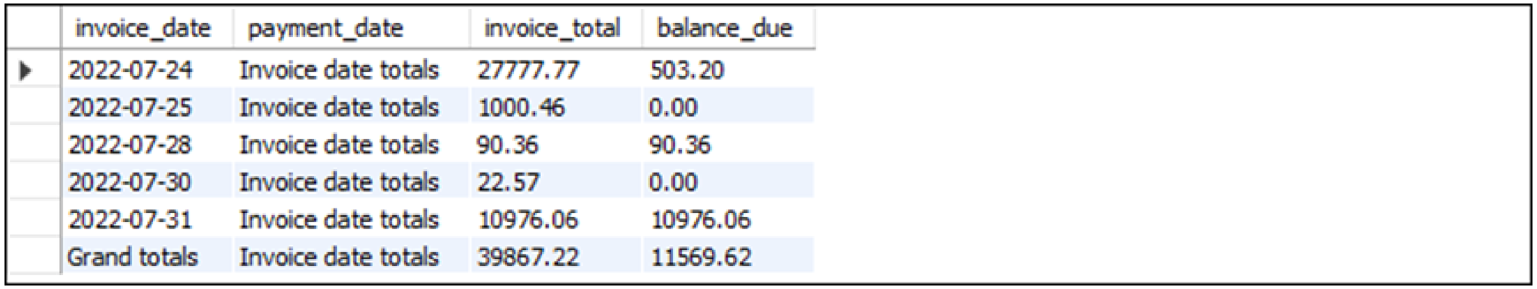
```
SELECT IF(GROUPING(invoice_date) = 1, 'Grand totals',
          invoice_date) AS invoice_date,
       IF(GROUPING(payment_date) = 1, 'Invoice date totals',
          payment_date) AS payment_date,
       SUM(invoice_total) AS invoice_total,
       SUM(invoice_total - credit_total - payment_total)
           AS balance_due
FROM invoices
WHERE invoice_date BETWEEN '2022-07-24' AND '2022-07-31'
GROUP BY invoice_date, payment_date WITH ROLLUP
```



invoice_date	payment_date	invoice_total	balance_due
2022-07-24	NULL	503.20	503.20
2022-07-24	2022-08-19	3689.99	0.00
2022-07-24	2022-08-23	67.00	0.00
2022-07-24	2022-08-27	23517.58	0.00
2022-07-24	Invoice date totals	27777.77	503.20
2022-07-25	2022-08-22	1000.46	0.00
2022-07-25	Invoice date totals	1000.46	0.00
2022-07-28	NULL	90.36	90.36
2022-07-28	Invoice date totals	90.36	90.36
2022-07-30	2022-09-03	22.57	0.00
2022-07-30	Invoice date totals	22.57	0.00
2022-07-31	NULL	10976.06	10976.06
2022-07-31	Invoice date totals	10976.06	10976.06
Grand totals	Invoice date totals	39867.22	11569.62

A query that displays only summary rows

```
SELECT IF(GROUPING(invoice_date) = 1, 'Grand totals', invoice_date)
           AS invoice_date,
       IF(GROUPING(payment_date) = 1, 'Invoice date totals',
          payment_date) AS payment_date,
       SUM(invoice_total) AS invoice_total,
       SUM(invoice_total - credit_total - payment_total)
           AS balance_due
FROM invoices
WHERE invoice_date BETWEEN '2022-07-24' AND '2022-07-31'
GROUP BY invoice_date, payment_date WITH ROLLUP
HAVING GROUPING(invoice_date) = 1 OR GROUPING(payment_date) = 1
```



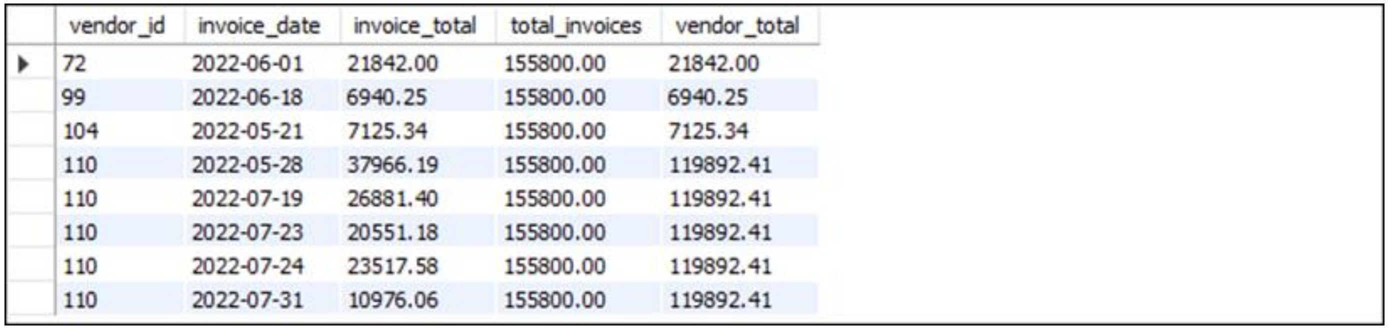
invoice_date	payment_date	invoice_total	balance_due
2022-07-24	Invoice date totals	27777.77	503.20
2022-07-25	Invoice date totals	1000.46	0.00
2022-07-28	Invoice date totals	90.36	90.36
2022-07-30	Invoice date totals	22.57	0.00
2022-07-31	Invoice date totals	10976.06	10976.06
Grand totals	Invoice date totals	39867.22	11569.62

The basic syntax of the OVER clause

```
OVER([PARTITION BY expression1 [, expression2]...
     [ORDER BY expression1 [ASC|DESC]
            [, expression2 [ASC|DESC]]...)
```

A SELECT statement with two aggregate window functions

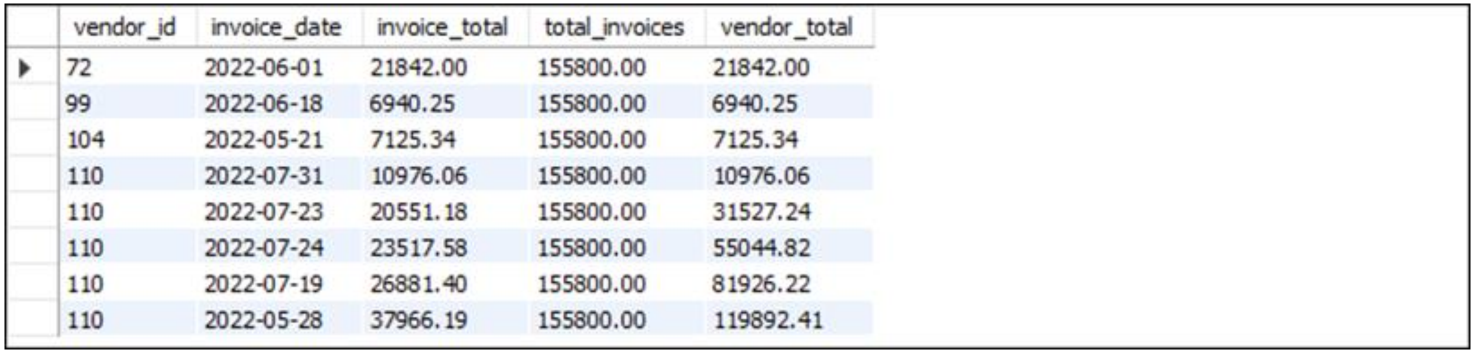
```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total) OVER() AS total_invoices,
       SUM(invoice_total) OVER(PARTITION BY vendor_id)
           AS vendor_total
FROM invoices
WHERE invoice_total > 5000
```



vendor_id	invoice_date	invoice_total	total_invoices	vendor_total
72	2022-06-01	21842.00	155800.00	21842.00
99	2022-06-18	6940.25	155800.00	6940.25
104	2022-05-21	7125.34	155800.00	7125.34
110	2022-05-28	37966.19	155800.00	119892.41
110	2022-07-19	26881.40	155800.00	119892.41
110	2022-07-23	20551.18	155800.00	119892.41
110	2022-07-24	23517.58	155800.00	119892.41
110	2022-07-31	10976.06	155800.00	119892.41

A SELECT statement with a cumulative total

```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total) OVER() AS total_invoices,
       SUM(invoice_total) OVER(PARTITION BY vendor_id
           ORDER BY invoice_total) AS vendor_total
FROM invoices
WHERE invoice_total > 5000
```



vendor_id	invoice_date	invoice_total	total_invoices	vendor_total
72	2022-06-01	21842.00	155800.00	21842.00
99	2022-06-18	6940.25	155800.00	6940.25
104	2022-05-21	7125.34	155800.00	7125.34
110	2022-07-31	10976.06	155800.00	10976.06
110	2022-07-23	20551.18	155800.00	31527.24
110	2022-07-24	23517.58	155800.00	55044.82
110	2022-07-19	26881.40	155800.00	81926.22
110	2022-05-28	37966.19	155800.00	119892.41

The syntax for defining a frame

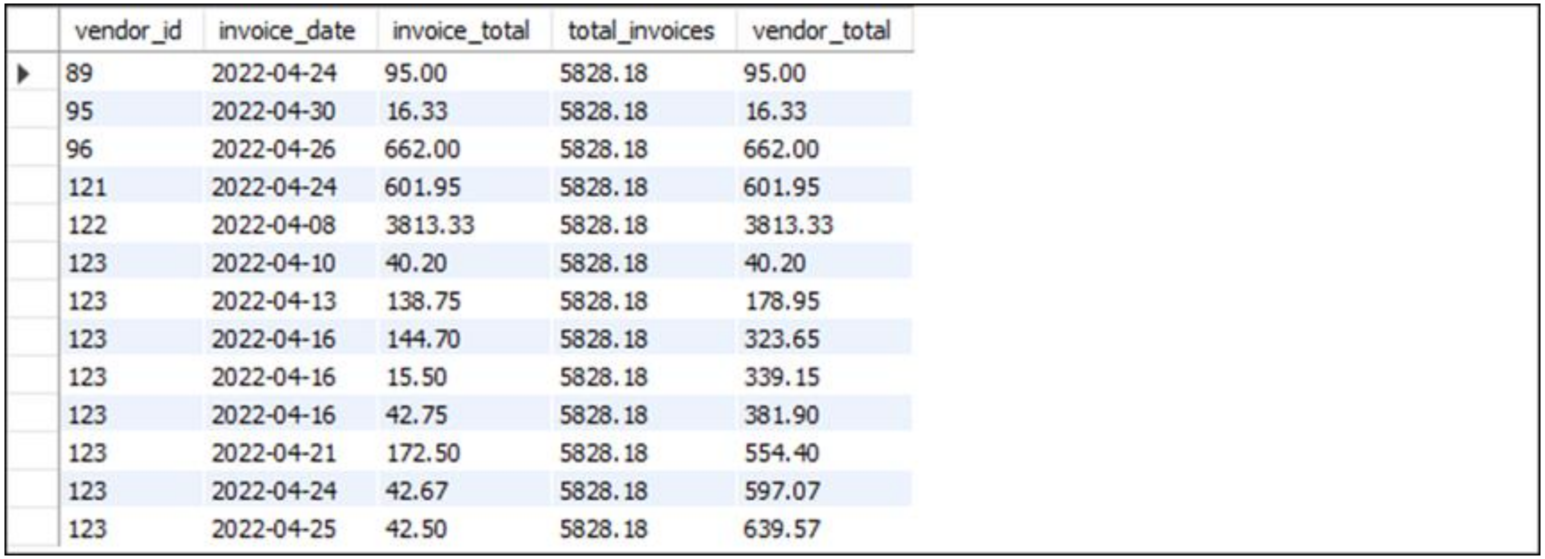
```
{ROWS | RANGE} {frame_start |
                BETWEEN frame_start AND frame_end}
```

Possible values for frame_start and frame_end

```
CURRENT ROW
UNBOUNDED PRECEDING
UNBOUNDED FOLLOWING
expr PRECEDING
expr FOLLOWING
```

A SELECT statement that defines a frame

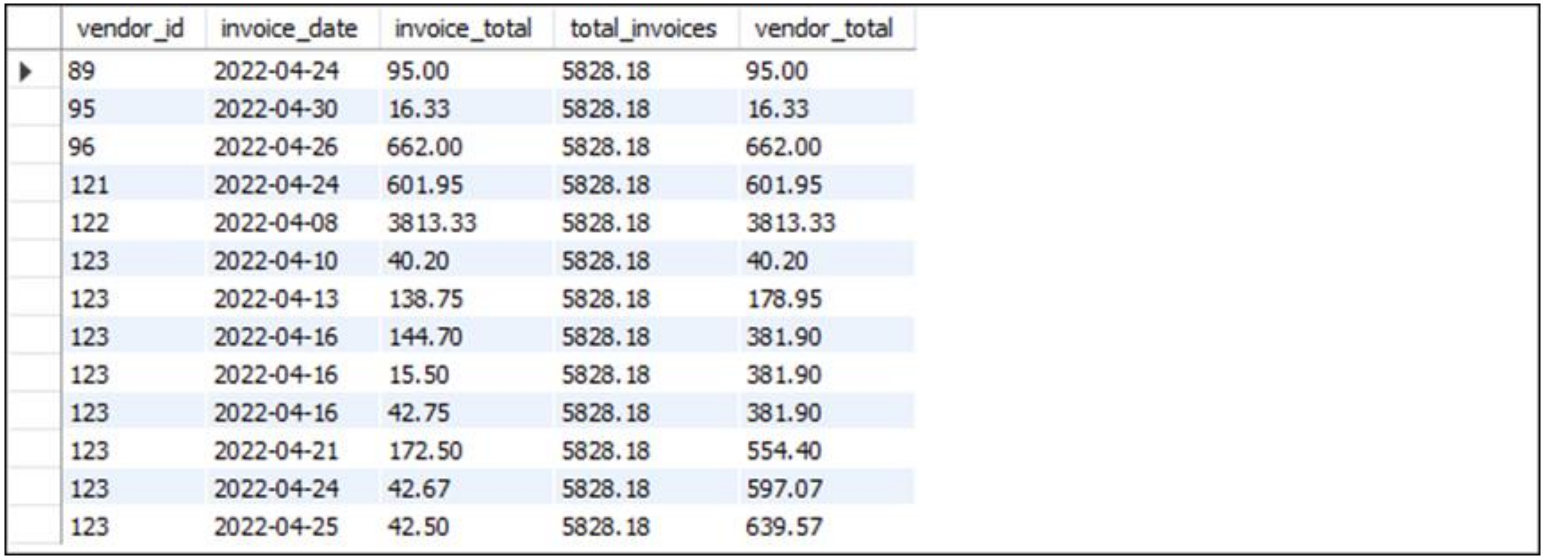
```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total) OVER() AS total_invoices,
       SUM(invoice_total) OVER(PARTITION BY vendor_id
         ORDER BY invoice_date
         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
         AS vendor_total
FROM invoices
WHERE invoice_date BETWEEN '2022-04-01' AND '2022-04-30'
```



vendor_id	invoice_date	invoice_total	total_invoices	vendor_total
89	2022-04-24	95.00	5828.18	95.00
95	2022-04-30	16.33	5828.18	16.33
96	2022-04-26	662.00	5828.18	662.00
121	2022-04-24	601.95	5828.18	601.95
122	2022-04-08	3813.33	5828.18	3813.33
123	2022-04-10	40.20	5828.18	40.20
123	2022-04-13	138.75	5828.18	178.95
123	2022-04-16	144.70	5828.18	323.65
123	2022-04-16	15.50	5828.18	339.15
123	2022-04-16	42.75	5828.18	381.90
123	2022-04-21	172.50	5828.18	554.40
123	2022-04-24	42.67	5828.18	597.07
123	2022-04-25	42.50	5828.18	639.57

A SELECT statement that creates peer groups

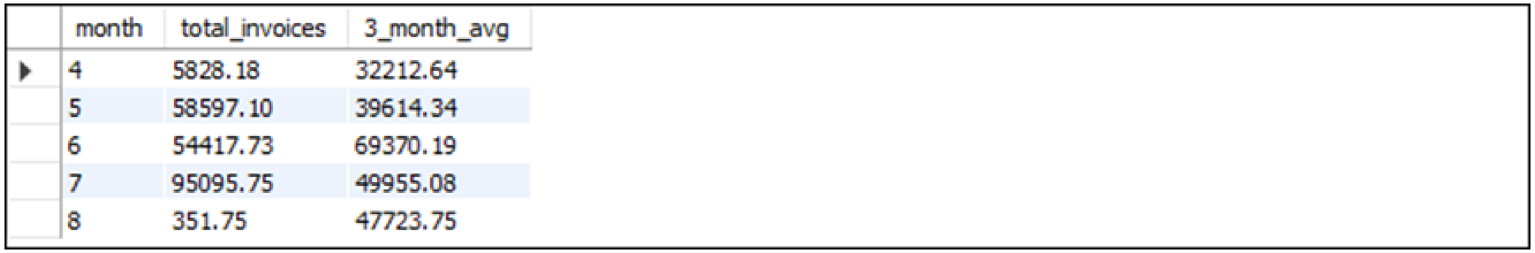
```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total) OVER() AS total_invoices,
       SUM(invoice_total) OVER(PARTITION BY vendor_id
         ORDER BY invoice_date
         RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
         AS vendor_total
FROM invoices
WHERE invoice_date BETWEEN '2022-04-01' AND '2022-04-30'
```



vendor_id	invoice_date	invoice_total	total_invoices	vendor_total
89	2022-04-24	95.00	5828.18	95.00
95	2022-04-30	16.33	5828.18	16.33
96	2022-04-26	662.00	5828.18	662.00
121	2022-04-24	601.95	5828.18	601.95
122	2022-04-08	3813.33	5828.18	3813.33
123	2022-04-10	40.20	5828.18	40.20
123	2022-04-13	138.75	5828.18	178.95
123	2022-04-16	144.70	5828.18	381.90
123	2022-04-16	15.50	5828.18	381.90
123	2022-04-16	42.75	5828.18	381.90
123	2022-04-21	172.50	5828.18	554.40
123	2022-04-24	42.67	5828.18	597.07
123	2022-04-25	42.50	5828.18	639.57

A SELECT statement that calculates moving averages

```
SELECT MONTH(invoice_date) AS month,
       SUM(invoice_total) AS total_invoices,
       ROUND(AVG(SUM(invoice_total))
         OVER(ORDER BY MONTH(invoice_date)
         RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING), 2)
         AS 3_month_avg
FROM invoices
GROUP BY MONTH(invoice_date)
```



month	total_invoices	3_month_avg
4	5828.18	32212.64
5	58597.10	39614.34
6	54417.73	69370.19
7	95095.75	49955.08
8	351.75	47723.75

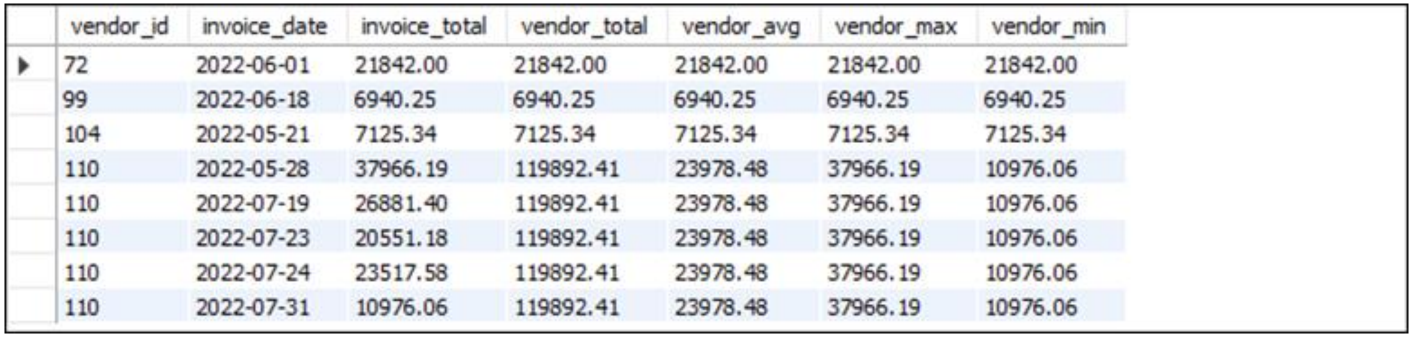
The syntax for naming a window

```
WINDOW window_name AS
    ([partition_clause] [order_clause] [frame_clause])
```

A SELECT statement with four functions that use the same window

```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total) OVER(PARTITION BY vendor_id)
           AS vendor_total,
       ROUND(AVG(invoice_total) OVER(PARTITION BY vendor_id), 2)
           AS vendor_avg,
       MAX(invoice_total) OVER(PARTITION BY vendor_id)
           AS vendor_max,
       MIN(invoice_total) OVER(PARTITION BY vendor_id)
           AS vendor_min
FROM invoices
WHERE invoice_total > 5000
```

The result set

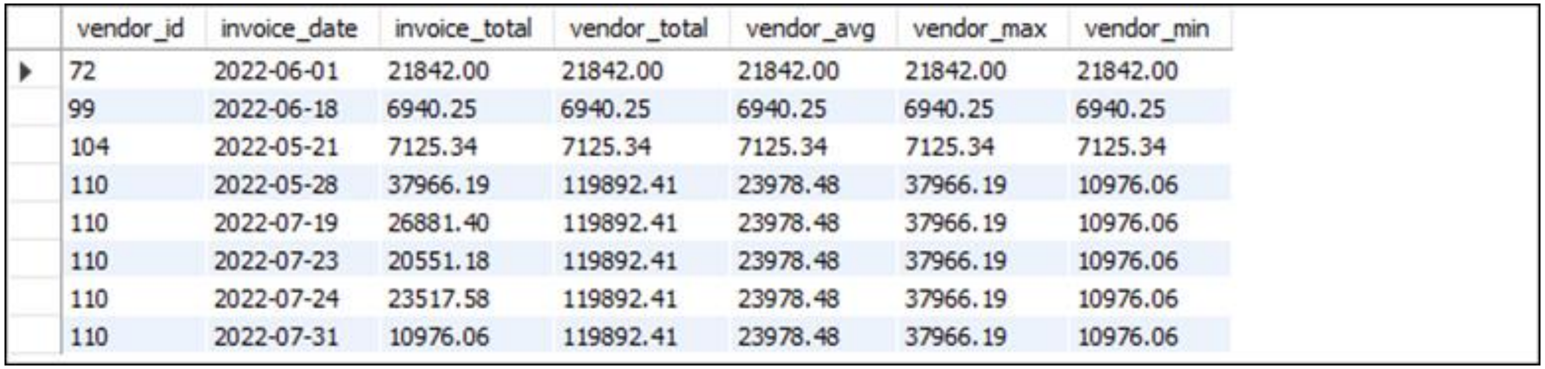


vendor_id	invoice_date	invoice_total	vendor_total	vendor_avg	vendor_max	vendor_min
72	2022-06-01	21842.00	21842.00	21842.00	21842.00	21842.00
99	2022-06-18	6940.25	6940.25	6940.25	6940.25	6940.25
104	2022-05-21	7125.34	7125.34	7125.34	7125.34	7125.34
110	2022-05-28	37966.19	119892.41	23978.48	37966.19	10976.06
110	2022-07-19	26881.40	119892.41	23978.48	37966.19	10976.06
110	2022-07-23	20551.18	119892.41	23978.48	37966.19	10976.06
110	2022-07-24	23517.58	119892.41	23978.48	37966.19	10976.06
110	2022-07-31	10976.06	119892.41	23978.48	37966.19	10976.06

A SELECT statement with a named window

```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total) OVER vendor_window
           AS vendor_total,
       ROUND(AVG(invoice_total) OVER vendor_window, 2)
           AS vendor_avg,
       MAX(invoice_total) OVER vendor_window AS vendor_max,
       MIN(invoice_total) OVER vendor_window AS vendor_min
FROM invoices
WHERE invoice_total > 5000
WINDOW vendor_window AS (PARTITION BY vendor_id)
```

The same result set



vendor_id	invoice_date	invoice_total	vendor_total	vendor_avg	vendor_max	vendor_min
72	2022-06-01	21842.00	21842.00	21842.00	21842.00	21842.00
99	2022-06-18	6940.25	6940.25	6940.25	6940.25	6940.25
104	2022-05-21	7125.34	7125.34	7125.34	7125.34	7125.34
110	2022-05-28	37966.19	119892.41	23978.48	37966.19	10976.06
110	2022-07-19	26881.40	119892.41	23978.48	37966.19	10976.06
110	2022-07-23	20551.18	119892.41	23978.48	37966.19	10976.06
110	2022-07-24	23517.58	119892.41	23978.48	37966.19	10976.06
110	2022-07-31	10976.06	119892.41	23978.48	37966.19	10976.06

A SELECT statement that adds to the specification for a named window

```
SELECT vendor_id, invoice_date, invoice_total,
       SUM(invoice_total)
           OVER (vendor_window ORDER BY invoice_date ASC)
           AS invoice_date_asc,
       SUM(invoice_total)
           OVER (vendor_window ORDER BY invoice_date DESC)
           AS invoice_date_desc
FROM invoices
WHERE invoice_total > 5000
WINDOW vendor_window AS (PARTITION BY vendor_id)
```