Chapter 5
How to code summary queries

Objectives

Applied
• Code summary queries that use aggregate functions, including the use of the ROLLUP and CUBE operators.

Knowledge
• Describe summary queries.
• Describe the differences between the HAVING clause and the WHERE clause.
• Describe the use of the ROLLUP and CUBE operators. Note: We won’t cover these specialized operators.

The syntax of the 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

The result set

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

SELECT 'After 1/1/2008' 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 > '01-JAN-2014'

The result set

A summary query with MIN and MAX functions

SELECT 'After 1/1/2008' 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 > '01-JAN-2014'

The result set

• Note the string literal 'After 1/1/2008' used as an expression in the select list
• The numeric SQL function ROUND(lost number, #decimal places) is not SQL standard, but is supported in Oracle, mysql, and all the other products listed in SQL dialects info.
• invoice, date is a DATE, so the WHERE clause is comparing dates, not strings.
A summary query for non-numeric columns

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

The result set

Here MIN and MAX are finding the first and last strings in string order—note A... first, Z... last

The syntax with GROUP BY and HAVING clauses

```sql
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 with the DISTINCT keyword

```sql
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 > '01-JAN-2008'
```

The result set

A summary query that calculates average invoice amount by vendor

```sql
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
```

The result set

A summary query that counts the number of invoices by vendor

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

The result set

A summary query with a join

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

The result set

Here is an example of two columns in the GROUP BY, grouping by the pairs (vendor_state, vendor_city)
A summary query that limits the groups to those with two or more invoices

```sql
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
```

The result set

```
1  CA  1234  75
2  CA  134  12
3  CA  1234  75
4  CA  1234  75
5  CA  1234  75
6  CA  1234  75
```

(12 rows selected)

Here count(*) in HAVING is measuring the size of the group, which is also reported as invoice_qty for the qualifying groups.

A summary query with a search condition in the WHERE clause

```sql
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
WHERE invoice_total > 500
ORDER BY invoice_qty DESC
```

The result set

```
1  Oracle Naval Weapon
2  Boeing Aircraft
3  Airbus Lifting Helicopter
4  IBM
```

(19 rows selected)

A summary query with a search condition in the HAVING clause

```sql
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
```

The result set

```
1  Oracle Naval Weapon
2  Boeing Aircraft
3  Airbus Lifting Helicopter
4  IBM
```

(19 rows selected)

A summary query with a compound condition in the HAVING clause

```sql
SELECT invoice_date,
       COUNT(*) AS invoice_qty,
       SUM(invoice_total) AS invoice_sum
FROM invoices
GROUP BY invoice_date
HAVING invoice_date
      BETWEEN '01-MAY-2014' AND '31-MAY-2014'
      AND COUNT(*) > 1
      AND SUM(invoice_total) > 100
ORDER BY invoice_date DESC
```

The result set

```
1  01-MAY-2014  3
2  02-MAY-2014  3
3  03-MAY-2014  3
4  04-MAY-2014  3
5  05-MAY-2014  3
```

(15 rows selected)

It's valid to use "invoice_date" in the HAVING clause, but here it would be faster to do this test in the earlier-applied WHERE clause, see next slide.

A summary query with a final summary row

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

The result set

```
<table>
<thead>
<tr>
<th>vendor_id</th>
<th>invoice_count</th>
<th>invoice_total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2757.33</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1234.75</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1234.75</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1234.75</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1234.75</td>
</tr>
</tbody>
</table>
```

(35 rows selected)

We aren't covering ROLLUP and CUBE, but they are useful to calculate values needed in "pivot tables" beloved by financial people.
A summary query with a summary row for each grouping level

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

The result set

<table>
<thead>
<tr>
<th>Vendor State</th>
<th>Vendor City</th>
<th>Qty Vendors</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA</td>
<td>Minneapolis</td>
<td>1</td>
</tr>
<tr>
<td>IA</td>
<td>Chicago</td>
<td>1</td>
</tr>
<tr>
<td>NJ</td>
<td>New York</td>
<td>1</td>
</tr>
<tr>
<td>NJ</td>
<td>Miami</td>
<td>2</td>
</tr>
<tr>
<td>TX</td>
<td>Austin</td>
<td>1</td>
</tr>
<tr>
<td>GA</td>
<td>Atlanta</td>
<td>1</td>
</tr>
</tbody>
</table>

A summary query with a summary row at the start of the result set

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

The result set

<table>
<thead>
<tr>
<th>Vendor ID</th>
<th>Invoice Count</th>
<th>Invoice Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(35 rows selected)</td>
<td>(35 rows selected)</td>
</tr>
</tbody>
</table>