# 14

# Relatórios por Agrupamento de Dados

# Objetivos deste Capítulo

- Ao concluir este capítulo, você poderá:
  - Usar a operação `ROLLUP` para produzir valores de subtotais
  - Usar a operação `CUBE` para produzir valores de tabelas de referência cruzada
  - Usar a função `GROUPING` para identificar os valores das linhas criadas por `ROLLUP` ou `CUBE`
  - Usar `GROUPING SETS` para produzir um único conjunto de resultados

14-2

**Objetivos deste Capítulo**

Neste capítulo, você aprenderá a:

- Agrupar dados para obter:

  Valores de subtotais usando o operador `ROLLUP`

  Valores de tabelas de referência cruzada usando o operador `CUBE`

- Usar a função `GROUPING` para identificar o nível de agregação no conjunto de resultados produzido por um operador `ROLLUP` ou `CUBE`.
- Usar `GROUPING SETS` para produzir um único conjunto de resultados que seja equivalente a um método `UNION ALL`

# Funções de Agrupamento

- As funções de agrupamento operam em conjuntos de linhas para fornecer um resultado por grupo.

```
SELECT        [coluna,] função_de_agrupamento(coluna). . .
FROM          tabela
[WHERE        condição]
[GROUP BY     expressão_de_agrupamento]
[ORDER BY     coluna];
```

```
SELECT AVG(salario), STDDEV(salario),
       COUNT(comissao),MAX(data_admissao)
FROM   funcionario
WHERE  cod_cargo LIKE 'VE%';
```

14-3

**Funções de Agrupamento**

Você pode usar a cláusula GROUP BY para dividir as linhas de uma tabela em grupos. Em seguida, você pode usar funções de agrupamento para retornar informações resumidas de cada grupo. As funções de agrupamento podem aparecer em listas de seleção e nas cláusulas ORDER BY e HAVING. O Oracle aplica as funções de agrupamento a cada grupo de linhas e retorna uma única linha de resultados para cada grupo.

Cada uma das funções AVG, SUM, MAX, MIN, COUNT, STDDEV e VARIANCE aceita apenas um argumento. As funções AVG, SUM, STDDEV e VARIANCE operam apenas em valores numéricos. MAX e MIN podem operar em valores numéricos, de caracteres e de dados de datas. COUNT retorna o número de linhas não nulas para determinada expressão. No exemplo do slide, são calculados o salário médio, o desvio padrão relativo ao salário, o número de funcionários que recebem comissão e a data de admissão máxima para os funcionários cujo COD_CARGO começa com VE.

Os tipos de dados para os argumentos podem ser CHAR, VARCHAR2, NUMBER ou DATE.

Todas as funções de agrupamento, exceto COUNT(*), ignoram valores nulos. Para substituir um valor por valores nulos, use a função NVL. COUNT retorna um número ou zero.

Quando você usa a cláusula GROUP BY, o Oracle ordena, implicitamente, o conjunto de resultados das colunas de agrupamento especificadas em ordem crescente. Para sobrepor essa ordenação default, é possível usar DESC em uma cláusula ORDER BY.

## Análise da Cláusula GROUP BY

- Sintaxe:

```
SELECT      [coluna,] função_de_agrupamento(coluna). . .
FROM        tabela
[WHERE      condição]
[GROUP BY   expressão_de_agrupamento]
[ORDER BY   coluna];
```

- Exemplo:

```
SELECT   cod_departamento, cod_cargo, SUM(salario),
         COUNT(cod_funcionario)
FROM     funcionario
GROUP BY cod_departamento, cod_cargo ;
```

14-4

### Análise da Cláusula GROUP BY

O exemplo ilustrado no slide é avaliado pelo Oracle da seguinte forma:

- A cláusula SELECT especifica que as seguintes colunas devem ser recuperadas:

  Colunas de código do departamento e código do cargo da tabela FUNCIONARIO

  A soma de todos os salários e o número de funcionários de cada grupo especificado na cláusula GROUP BY

- A cláusula GROUP BY especifica como as linhas devem ser agrupadas na tabela. O salário total e o número de funcionários são calculados para cada código de cargo em cada departamento. As linhas são agrupadas por código de departamento e, em seguida, são agrupadas por cargo, dentro de cada departamento.

## Análise da Cláusula HAVING

- Use a cláusula HAVING para especificar quais grupos devem ser exibidos.
- Você pode restringir ainda mais os grupos com base em uma condição limitante.

```
SELECT      [coluna,] função_de_agrupamento(coluna)...
FROM        tabela
[WHERE      condição]
[GROUP BY   expressão_de_agrupamento]
[HAVING     expressão_having]
[ORDER BY   coluna];
```

14-5

### Análise da Cláusula HAVING

Os grupos são formados e as funções de agrupamento são calculadas antes de a cláusula HAVING ser aplicada aos grupos. A cláusula HAVING pode anteceder a cláusula GROUP BY, mas, por motivos lógicos, é recomendável usar primeiro a cláusula GROUP BY.

Quando você usa a cláusula HAVING, o Oracle segue as seguintes etapas:

1. Agrupa linhas
2. Aplica as funções de agrupamento aos grupos e exibe os grupos que correspondem aos critérios da cláusula HAVING

## Operadores ROLLUP e CUBE

- Use o operador ROLLUP ou CUBE com GROUP BY para produzir linhas superagregadas por colunas de referência cruzada.
- O agrupamento de ROLLUP produz um conjunto de resultados com as linhas agrupadas normais e os valores dos subtotais.
- O agrupamento de CUBE produz um conjunto de resultados com as linhas de ROLLUP e as linhas de tabelas de referência cruzada.

14-6

### GROUP BY com os Operadores ROLLUP e CUBE

Especifique os operadores ROLLUP e CUBE na cláusula GROUP BY de uma consulta. O agrupamento de ROLLUP produz um conjunto de resultados com as linhas agrupadas normais e as linhas de subtotais. A operação CUBE na cláusula GROUP_BY agrupa as linhas selecionadas com base nos valores de todas as combinações possíveis de expressões na especificação e retorna uma única linha de informações resumidas para cada grupo. Você pode usar o operador CUBE para produzir linhas de tabelas de referência cruzada.

Quando estiver trabalhando com ROLLUP e CUBE, certifique-se de que as colunas após a cláusula GROUP BY tenham relacionamentos significativos e reais umas com as outras; caso contrário, os operadores retornarão informações irrelevantes.

# Operador ROLLUP

- ROLLUP é uma extensão da cláusula GROUP BY.
- Use o operador ROLLUP para produzir agregados cumulativos, como subtotais.

```
SELECT      [coluna,] função_de_agrupamento(coluna). . .
FROM        tabela
[WHERE      condição]
[GROUP BY   [ROLLUP] expressão_de_agrupamento]
[HAVING     expressão_having];
[ORDER BY   coluna];
```

14-7

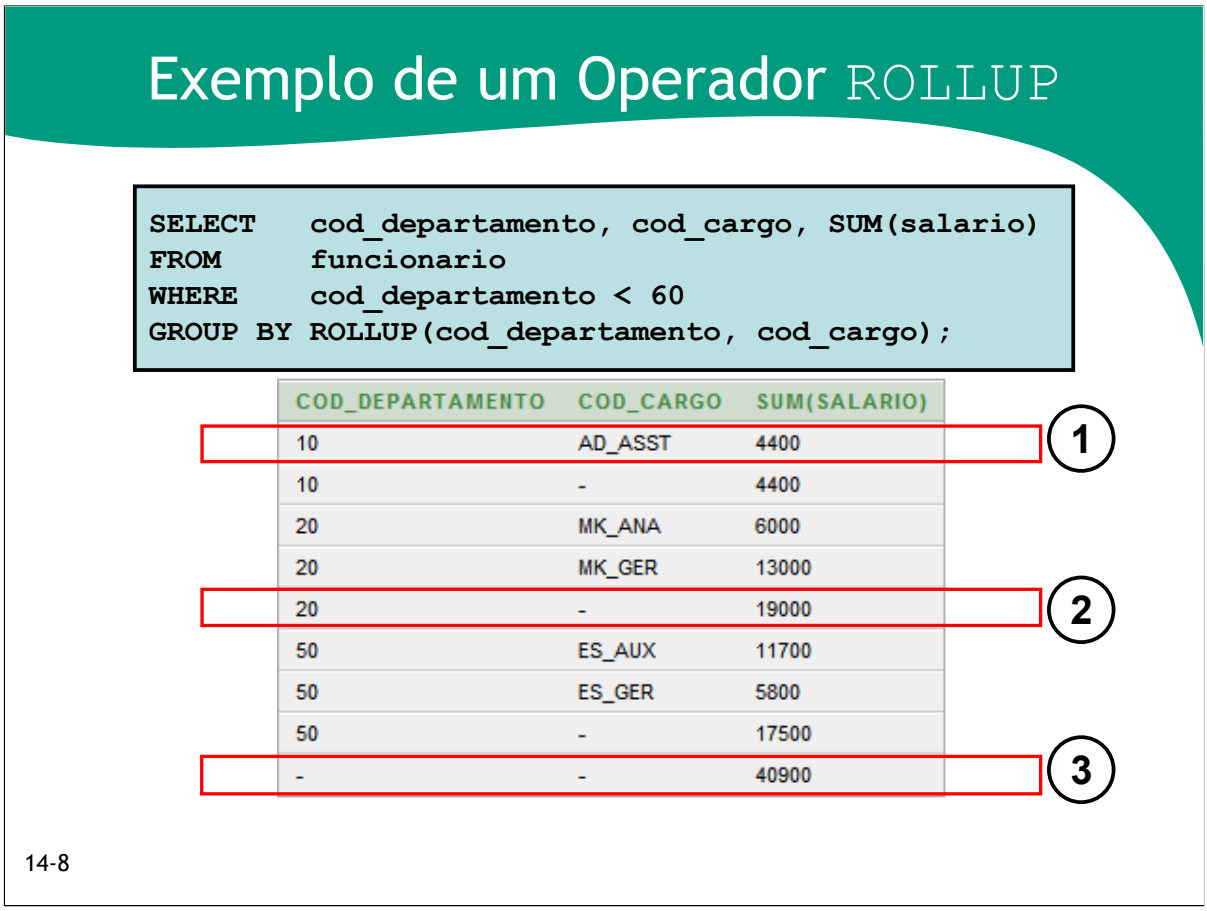
## Operador ROLLUP

O operador ROLLUP fornece agregados e superagregados para expressões contidas no comando GROUP BY. Os autores de relatórios podem usar o operador ROLLUP para extrair estatísticas e informações resumidas de conjuntos de resultados. É possível usar agregados cumulativos em relatórios, tabelas e gráficos.

O operador ROLLUP cria agrupamentos movendo-se em uma direção, da direita para a esquerda, pela lista de colunas especificada na cláusula GROUP BY. Depois, ele aplica a função agregada a esses agrupamentos.

Para produzir subtotais em *n* dimensões (isto é, *n* colunas na cláusula GROUP BY) sem usar o operador ROLLUP, é necessário unir *n*+1 instruções SELECT com UNION ALL. Isso torna a execução da consulta ineficiente, pois cada uma das instruções SELECT produz um acesso à tabela. O operador ROLLUP reúne os resultados com apenas um acesso à tabela. Esse operador será útil quando houver várias colunas envolvidas na produção de subtotais.

Os subtotais e totais são produzidos com ROLLUP. CUBE também produz totais e faz um rollup eficiente dos valores em todas as direções possíveis, produzindo dados de referência cruzada.




### Exemplo de um Operador ROLLUP

No exemplo do slide o total dos salários de todos os códigos de cargos relativos aos departamentos cujo código é menor que 60 são exibidos pela cláusula GROUP BY. O operador ROLLUP exibe o salário total de cada departamento cujo código é menor que 60 e também o salário total de todos os departamentos cujo código é menor que 60, independentemente dos códigos dos cargos.

Neste exemplo, 1 indica um grupo totalizado por COD_DEPARTAMENTO e COD_CARGO, 2 indica um grupo totalizado apenas por COD_DEPARTAMENTO e 3 indica o total geral.

O operador ROLLUP cria subtotais nos quais ocorre um rollup do nível mais detalhado para um total geral, seguindo a lista de agrupamento especificada na cláusula GROUP BY. Primeiro, ele calcula os valores agregados padrão para os grupos especificados na cláusula GROUP BY (no exemplo, a soma dos salários agrupada em cada cargo de um departamento). Em seguida, ele cria subtotais progressivamente mais altos, movendo-se da direita para a esquerda pela lista de colunas de agrupamento. (No exemplo, a soma dos salários de cada departamento é calculada, seguida pela soma dos salários de todos os departamentos.)

Fornecidas *n* expressões no operador ROLLUP da cláusula GROUP BY, a operação resulta em agrupamentos *n* + 1 (neste caso, 2 + 1 = 3).

# Operador CUBE

- CUBE é uma extensão da cláusula GROUP BY.
- Você pode usar o operador CUBE para produzir valores de tabelas de referência cruzada com uma único comando SELECT.

```
SELECT      [coluna,] função_de_agrupamento(coluna)...
FROM        tabela
[WHERE      condição]
[GROUP BY   [CUBE] expressão_de_agrupamento]
[HAVING     expressão_having]
[ORDER BY   coluna];
```

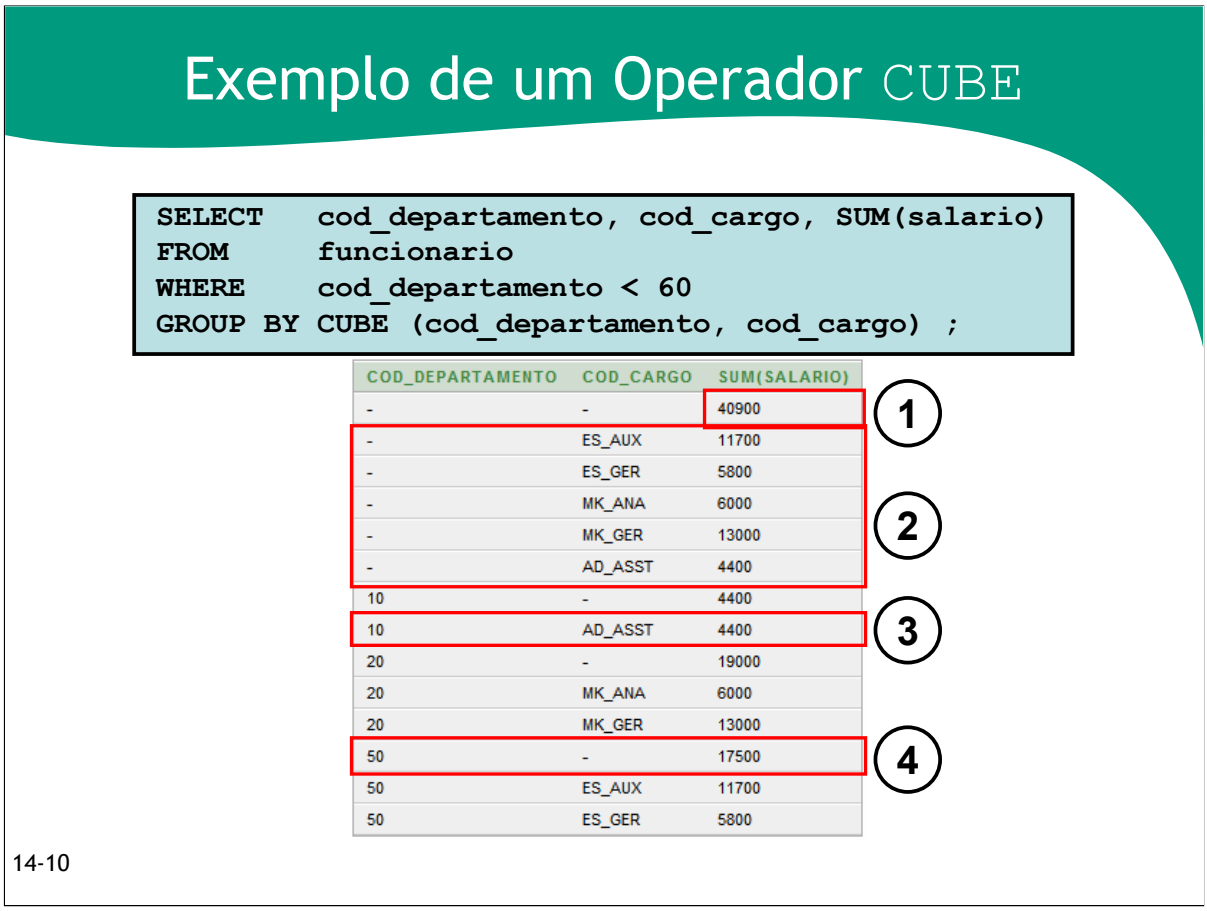
14-9

**Operador CUBE**

O operador CUBE é mais uma alternativa na cláusula GROUP BY de um comando SELECT. É possível aplicar esse operador a todas as funções agregadas, inclusive AVG, SUM, MAX, MIN e COUNT. Ele é usado para produzir conjuntos de resultados usados geralmente em relatórios de tabelas de referência cruzada. Enquanto ROLLUP produz somente uma fração das combinações possíveis de subtotais, CUBE produz subtotais para todas as combinações possíveis de agrupamentos especificados na cláusula GROUP BY e um total geral.

O operador CUBE é usado com uma função agregada para gerar outras linhas em um conjunto de resultados. É feita referência cruzada às colunas incluídas na cláusula GROUP BY para produzir um superconjunto de agrupamentos. A função agregada especificada na lista de seleção é aplicada a esses grupos para produzir valores resumidos para as linhas superagregadas adicionais. O número de agrupamentos extras no conjunto de resultados é determinado pelo número de colunas incluídas na cláusula GROUP BY.

Na verdade, todas as combinações possíveis das colunas ou expressões da cláusula GROUP BY são usadas para produzir superagregados. Se você tiver *n* colunas ou expressões na cláusula GROUP BY, haverá $2^n$ combinações superagregadas possíveis. Matematicamente, essas combinações formam um cubo *n* dimensional, que é a origem do nome do operador.




### Exemplo de um Operador CUBE

É possível interpretar a saída do comando SELECT no exemplo da seguinte maneira:

- O salário total de todos os cargos de um departamento (para os departamentos com código menor que 60) é exibido na cláusula GROUP BY.
- O salário total dos departamentos cujo código é menor que 60.
- O salário total de todos os cargos independentemente do departamento.
- O salário total dos departamentos cujo código é menor que 60, independentemente dos cargos.

Neste exemplo, 1 indica o total geral. 2 indica as linhas totalizadas apenas por COD_CARGO. 3 indica algumas linhas totalizadas por COD_DEPARTAMENTO e COD_CARGO. 4 indica algumas linhas totalizadas apenas por COD_DEPARTAMENTO.

O operador CUBE também executou a operação ROLLUP para exibir os subtotais, para os departamentos cujo código é menor que 60, e o salário total dos departamentos cujo código é menor que 60, independentemente dos cargos. Além disso, o operador CUBE exibe o salário total de todos os cargos independentemente do departamento.

## Função GROUPING

- É usada com o operador CUBE ou ROLLUP
- É usada para localizar os grupos que formam o subtotal em uma linha
- É usada para diferenciar valores NULL armazenados de valores NULL criados por ROLLUP ou CUBE
- Retorna 0 ou 1

```
SELECT    [coluna,] função_de_agrupamento(coluna) ... ,
          GROUPING(expressão)
FROM      tabela
[WHERE    condição]
[GROUP BY [ROLLUP][CUBE] expressão_de_agrupamento]
[HAVING   expressão_having]
[ORDER BY coluna];
```

14-11

### A Função GROUPING

É possível usar a função GROUPING com o operador CUBE ou ROLLUP para compreender como um valor resumido foi obtido.

A função GROUPING usa uma única coluna como argumento. O termo expressão na função GROUPING deve corresponder a uma das expressões na cláusula GROUP BY. A função retorna o valor 0 ou 1.

Os valores retornados pela função GROUPING são úteis para:

- Determinar o nível de agregação de um subtotal fornecido; isto é, o grupo (ou grupos) no qual o subtotal é baseado
- Identificar se um valor NULL na coluna da expressão de uma linha do conjunto de resultados indica:
  - Um valor NULL da tabela-base (valor NULL armazenado)
  - Um valor NULL criado por ROLLUP ou CUBE (como resultado da função de agrupamento nessa expressão)

O valor 0 retornado pela função GROUPING com base em uma expressão indica uma das seguintes situações:

- A expressão foi usada para calcular o valor agregado.
- O valor NULL na coluna da expressão é um valor NULL armazenado.

O valor 1 retornado pela função GROUPING com base em uma expressão indica uma das seguintes situações:

- A expressão não foi usada para calcular o valor agregado.
- O valor NULL na coluna da expressão é criado por ROLLUP/CUBE como resultado de agrupamento.

## Exemplo de Função GROUPING

```
SELECT   cod_departamento, cod_cargo,
SUM(salario),
         GROUPING(cod_departamento) GRP_DEPT,
         GROUPING(cod_cargo) GRP_CARGO
FROM     funcionario
WHERE    cod_departamento < 50
GROUP BY ROLLUP(cod_departamento, cod_cargo);
```

| | COD_DEPARTAMENTO | COD_CARGO | SUM(SALARIO) | GRP_DEPT | GRP_CARGO | |
|---|---|---|---|---|---|---|
| 1 → | 10 | AD_ASST | 4400 | 0 | 0 | |
| | 10 | - | 4400 | 0 | 1 | → 2 |
| | 20 | MK_ANA | 6000 | 0 | 0 | |
| | 20 | MK_GER | 13000 | 0 | 0 | |
| | 20 | - | 19000 | 0 | 1 | |
| | - | - | 23400 | 1 | 1 | → 3 |

14-12

### Exemplo de uma Função GROUPING

No exemplo do slide, considere o valor resumido 4400 na primeira linha (indicado por 1). Esse valor resumido é o salário total de cada código do cargo AD_ASST no departamento 10. Para calcular esse valor resumido, as colunas COD_DEPARTAMENTO e COD_CARGO foram consideradas. Assim, o valor 0 é retornado para as expressões GROUPING(cod_departamento) e GROUPING(cod_cargo).

Considere o valor resumido 4400 na segunda linha (indicado por 2). Esse valor é o salário total do departamento 10 e foi calculado considerando a coluna COD_DEPARTAMENTO. Portanto, o valor 0 foi retornado por GROUPING(cod_departamento). Como a coluna COD_CARGO não foi considerada no cálculo desse valor, o valor 1 foi retornado para GROUPING(cod_cargo). Você pode observar uma saída semelhante na quinta linha.

Na última linha, considere o valor resumido 23400 (indicado por 3). Este é o salário total de todos os cargos dos departamentos cujo código é menor que 50. Para calcular esse valor resumido, não foram consideradas as colunas COD_DEPARTAMENTO e COD_CARGO. Assim, é retornado o valor 1 para as expressões GROUPING(cod_departamento) e GROUPING(cod_cargo).

## GROUPING SETS

- A sintaxe de GROUPING SETS é usada para definir vários agrupamentos na mesma consulta.
- Todos os agrupamentos especificados na cláusula GROUPING SETS são computados, e os resultados de agrupamentos individuais são combinados com uma operação UNION ALL.
- Eficiência de conjuntos de agrupamentos:
  - Só é preciso uma passagem pela tabela-base
  - Não há necessidade de criar instruções UNION complexas.
  - Quanto mais elementos GROUPING SETS houver, mais vantagens de desempenho serão obtidas.

14-13

**GROUPING SETS**

GROUPING SETS é uma extensão da cláusula GROUP BY que você pode usar para especificar vários agrupamentos de dados. Essa especificação facilita a agregação eficiente, portanto facilita também a análise de dados em várias dimensões.

Agora, é possível criar uma único comando SELECT usando GROUPING SETS para especificar vários agrupamentos (que também podem incluir os operadores ROLLUP ou CUBE), em vez de diversas instruções SELECT combinadas por operadores UNION ALL. Por exemplo:

```
SELECT cod_departamento, cod_cargo, cod_gerente, AVG(salario)
 FROM funcionario
 GROUP BY GROUPING SETS
 ((cod_departamento, cod_cargo, cod_gerente),
 (cod_departamento, cod_gerente),(cod_cargo, cod_gerente));
```

Esso comando calcula agregados em três agrupamentos:

```
 (cod_departamento, cod_cargo, cod_gerente),
(cod_departamento,cod_gerente) e (cod_cargo, cod_gerente)
```

Sem esse recurso, são necessárias várias consultas combinadas com UNION ALL para obter a saída do comando SELECT precedente. Uma abordagem de várias consultas não é eficaz, pois ela requer várias varreduras nos mesmos dados.

## GROUPING SETS (continuação)

Compare o exemplo anterior com a seguinte alternativa:

```
   SELECT cod_departamento, cod_cargo, cod_gerente,
AVG(salario)
   FROM funcionario
   GROUP BY CUBE(cod_departamento, cod_cargo, cod_gerente);
```

Esso comando engloba os 8 agrupamentos (2 *2 *2), apesar de apenas os grupos `(cod_departamento, cod_cargo, cod_gerente)`, `(cod_departamento, cod_gerente)` e `(cod_cargo, cod_gerente)` serem relevantes para você.

Outra alternativa seria a seguinte comando:

```
   SELECT cod_departamento, cod_cargo, cod_gerente,
AVG(salario)
   FROM funcionario
   GROUP BY cod_departamento, cod_cargo, cod_gerente
   UNION ALL
   SELECT cod_departamento, NULL, cod_gerente, AVG(salario)
   FROM funcionario
   GROUP BY cod_departamento, cod_gerente
   UNION ALL
   SELECT NULL, cod_cargo, cod_gerente, AVG(salario)
   FROM funcionario
   GROUP BY cod_cargo, cod_gerente;
```

Esso comando requer três varreduras da tabela-base, o que a torna ineficiente.

Os operadores `CUBE` e `ROLLUP` podem ser considerados conjuntos de agrupamento com semântica muito específica. As seguintes equivalências confirmam esse fato:

| | |
|---|---|
| `CUBE(a, b, c)` é equivalente a | `GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())` |
| `ROLLUP(a, b,c)` é equivalente a | `GROUPING SETS ((a, b, c), (a, b),(a), ())` |

## GROUPING SETS: Exemplo

```
SELECT    cod_departamento, cod_cargo,
          cod_gerente,avg(salario)
FROM      funcionario
GROUP BY GROUPING SETS
((cod_departamento,cod_cargo), (cod_cargo,cod_gerente));
```

| COD_DEPARTAMENTO | COD_CARGO | COD_GERENTE | AVG(SALARIO) |
|---|---|---|---|
| - | ES_AUX | 124 | 2733,33333333333333333333333333333333 |
| - | VE_GER | 100 | 10500 |
| - | MK_ANA | 201 | 6000 |
| - | AD_PRES | - | 24000 |
| - | VE_REP | 149 | 8866,66666666666666666666666666666667 |
| - | AD_ASST | 101 | 4400 |

1

| COD_DEPARTAMENTO | COD_CARGO | COD_GERENTE | AVG(SALARIO) |
|---|---|---|---|
| 20 | MK_GER | - | 13000 |
| 20 | MK_ANA | - | 6000 |
| 110 | CTPUB_GER | - | 8300 |
| 90 | AD_PRES | - | 24000 |
| 60 | IT_PROG | - | 6400 |
| 50 | ES_AUX | - | 2733,33333333333333333333333333333333 |
| 80 | VE_REP | - | 9800 |

2

14-15

### GROUPING SETS: Exemplo

A consulta do slide calcula agregados em dois agrupamentos. A tabela é dividida nos seguintes grupos:

- código do Cargo, código do Gerente
- código do Departamento, código do Cargo

Os salários médios de cada um desses grupos são calculados. O conjunto de resultados exibe o salário médio de cada um dos dois grupos.

Na saída, é possível interpretar o grupo marcado por 1 como:

- O salário médio de todos os funcionários com o código de cargo `VE_GER` abaixo do gerente 100 é R$ 10.500.
- O salário médio de todos os funcionários com o código de cargo `MK_ANA` abaixo do gerente 201 é R$ 6.000 e assim por diante.

O grupo marcado com 2 na saída é interpretado como:

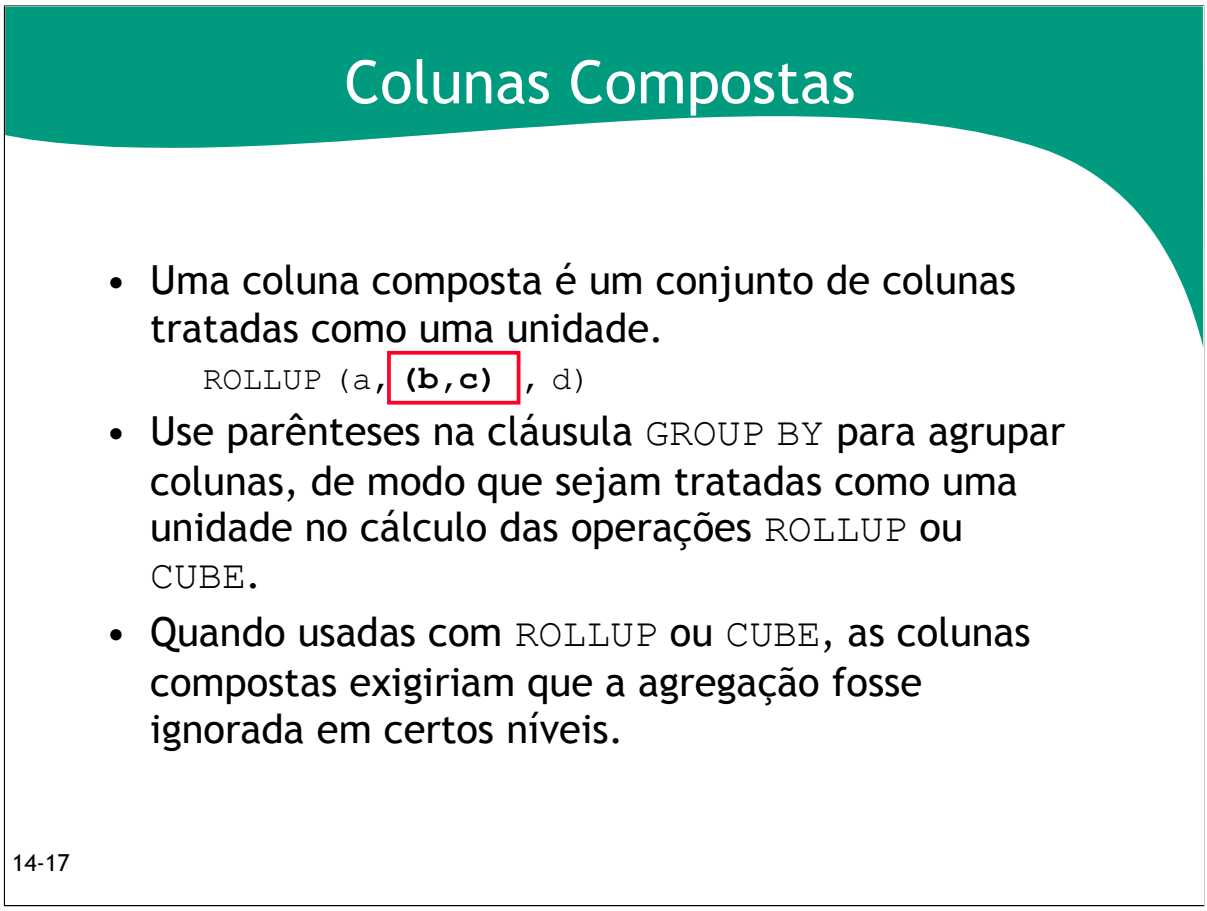
- O salário médio de todos os funcionários com o código de cargo `IT_PROG` no departamento 60 é R$ 6.400.
- O salário médio de todos os funcionários com o código de cargo `MK_GER` no departamento 20 é R$ 13.000 e assim por diante.

## `GROUPING SETS`: Exemplo (continuação)

O exemplo do slide também pode ser criado assim:

```
SELECT cod_departamento, cod_cargo, NULL as cod_gerente,
       AVG(salario) as SALMED
FROM funcionario
GROUP BY cod_departamento, cod_cargo
UNION ALL
SELECT NULL, cod_cargo, cod_gerente, avg(salario) as AVGSAL
FROM funcionario
GROUP BY cod_cargo, cod_gerente;
```

A consulta precedente precisaria de duas varreduras da tabela-base (`FUNCIONARIO`). Isso pode ser muito ineficiente. Portando, recomenda-se o uso do comando `GROUPING SETS`.




**Colunas Compostas**

Uma coluna composta é um conjunto de colunas tratadas como uma unidade durante o cálculo de agrupamentos. Especifique as colunas entre parênteses como na seguinte comando:

```
ROLLUP (a, (b, c), d)
```

Em que `(b,c)` forma uma coluna composta e é tratado como uma unidade. Em geral, as colunas compostas são úteis em `ROLLUP`, `CUBE` e `GROUPING SETS`. Por exemplo, em `CUBE` ou `ROLLUP`, as colunas compostas exigiriam que a agregação fosse ignorada em certos níveis.

Isto é, `GROUP BY ROLLUP(a, (b, c))` equivale a

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Em que `(b, c)` é tratado como uma unidade e não é aplicado `ROLLUP` em `(b, c)`. É como se você tivesse um apelido, por exemplo, `z`, para `(b, c)` e a expressão `GROUP BY` fosse reduzida a `GROUP BY ROLLUP(a, z)`.

Normalmente, `GROUP BY( )` é um comando `SELECT` com valores `NULL` para as colunas `a` e `b`, e apenas a função agregada. Quase sempre ela é usada para gerar totais gerais.

```
SELECT   NULL, NULL, coluna_agregada
FROM     <nome_tabela>
GROUP BY ( );
```

**Colunas Compostas (continuação)**

Compare com o ROLLUP normal, como em:

```
GROUP BY ROLLUP(a, b, c)
```

que seria

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Da mesma forma,

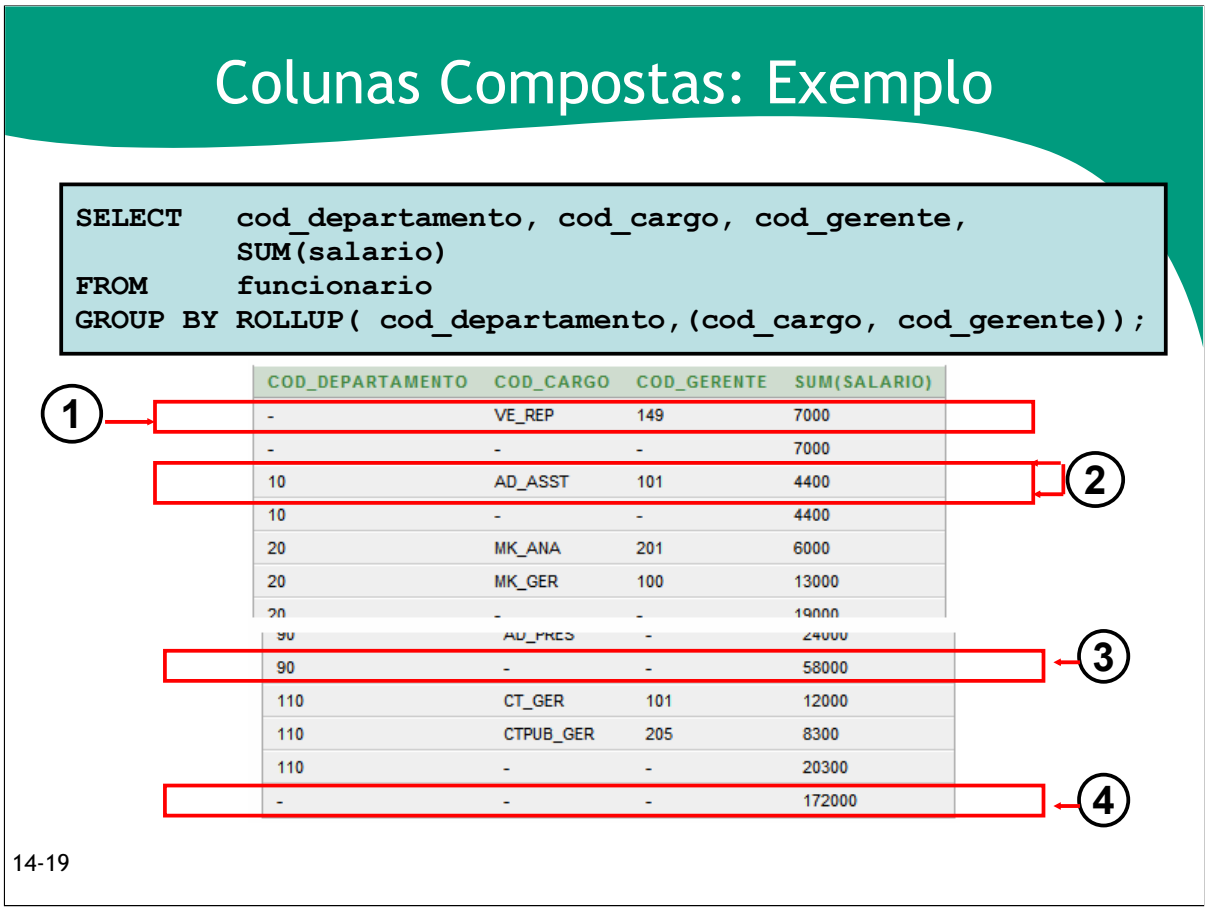
```
GROUP BY CUBE((a, b), c)
```

seria equivalente a

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP By ()
```

A tabela a seguir mostra a especificação de conjuntos de agrupamento e a especificação da cláusula GROUP BY equivalente.

| **Instruções GROUPING SETS** | **Instruções GROUP BY Equivalentes** |
|---|---|
| `GROUP BY GROUPING SETS(a, b, c)` | `GROUP BY a UNION ALL`<br>`GROUP BY b UNION ALL`<br>`GROUP BY c` |
| `GROUP BY GROUPING SETS(a, b,(b, c))` (A expressão `GROUPING SETS` tem uma coluna composta.) | `GROUP BY a UNION ALL`<br>`GROUP BY b UNION ALL`<br>`GROUP BY b, c` |
| `GROUP BY GROUPING SETS((a, b, c))` | `GROUP BY a, b, c` |
| `GROUP BY GROUPING SETS(a, (b), ())` | `GROUP BY a UNION ALL`<br>`GROUP BY b UNION ALL`<br>`GROUP BY ()` |
| `GROUP BY GROUPING SETS (a,ROLLUP(b, c))`<br>(A expressão `GROUPING SETS` tem uma coluna composta.) | `GROUP BY a UNION ALL`<br>`GROUP BY ROLLUP(b, c)` |




**Colunas Compostas: Exemplo**

Considere o exemplo:

```
SELECT cod_departamento, cod_cargo,cod_gerente, SUM(salario)
 FROM   funcionario
 GROUP BY ROLLUP( cod_departamento,cod_cargo, cod_gerente);
```

Esta consulta resulta no cálculo do Oracle dos seguintes agrupamentos:

1. (cod_departamento , cod_cargo, cod_gerente)
2. (cod_departamento, cod_cargo)
3. (cod_departamento)
4. Total geral

Se você estiver interessado apenas em grupos específicos, não poderá limitar o cálculo aos agrupamentos sem usar colunas compostas. Com colunas compostas, isso é possível tratando as colunas `COD_CARGO` e `COD_GERENTE` como uma unidade durante o rollup. As colunas entre parênteses são tratadas como uma unidade no cálculo de `ROLLUP` e `CUBE`. Essa situação é ilustrada no exemplo do slide. Colocando as colunas `COD_CARGO` e `COD_GERENTE` entre parênteses, você informa ao Oracle para tratar `COD_CARGO` e `COD_GERENTE` como uma unidade, ou seja, como uma coluna composta.

**Colunas Compostas: Exemplo (continuação)**

O exemplo do slide calcula os seguintes agrupamentos:

- `(cod_departamento, cod_cargo, cod_gerente)`
- `(cod_departamento)`
- `( )`

O exemplo do slide exibe o seguinte:

- O salário total de cada cargo e gerente (indicado por 1)
- O salário total de cada departamento, cargo e gerente (indicado por 2)
- O salário total de cada departamento (indicado por 3)
- Total geral (indicado por 4)

O exemplo do slide também pode ser criado assim:

```
SELECT  cod_departamento, cod_cargo, cod_gerente, SUM(salario)
FROM    funcionario
GROUP   BY cod_departamento,cod_cargo, cod_gerente
UNION   ALL
SELECT  cod_departamento, TO_CHAR(NULL),TO_NUMBER(NULL),
SUM(salario)
FROM    funcionario
GROUP BY cod_departamento
UNION ALL
SELECT TO_NUMBER(NULL), TO_CHAR(NULL),TO_NUMBER(NULL),
SUM(salario)
FROM    funcionario
GROUP BY ();
```

A consulta precedente precisaria de três varreduras da tabela-base (`FUNCIONARIO`). Isso pode ser muito ineficiente. Portanto, recomenda-se o uso de colunas compostas.

# Agrupamentos Concatenados

- Os agrupamentos concatenados oferecem uma maneira concisa de gerar combinações úteis de agrupamentos.
- Para especificar conjuntos de agrupamentos concatenados, separe vários conjuntos de agrupamentos, bem como operações `ROLLUP` e `CUBE`, por vírgulas para que o Oracle os combine em uma única cláusula `GROUP BY`.
- O resultado é um produto híbrido de agrupamentos de cada conjunto de agrupamentos.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

14-21

**Agrupamentos Concatenados**

Os agrupamentos concatenados oferecem uma maneira concisa de gerar combinações úteis de agrupamentos. Os agrupamentos concatenados são especificados pela listagem de vários conjuntos de agrupamentos, cubos e rollups, separados por vírgulas. Este é um exemplo de conjuntos de agrupamentos concatenados:

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

Este comando SQL define os seguintes agrupamentos:

```
(a, c), (a, d), (b, c), (b, d)
```

A concatenação de conjuntos de agrupamentos é muito útil pelo seguinte:

- **Facilidade de desenvolvimento de consultas:** não é necessário enumerar todos os agrupamentos manualmente.
- **Uso por aplicações:** o comando SQL gerado por aplicações OLAP envolve, com freqüência, a concatenação de conjuntos de agrupamentos; cada conjunto define os agrupamentos necessários para uma dimensão

# Agrupamentos Concatenados: Exemplo

```
SELECT   cod_departamento, cod_cargo, cod_gerente,
         SUM(salario)
FROM     funcionario
GROUP BY cod_departamento,
         ROLLUP(cod_cargo),
         CUBE(cod_gerente);
```

| | COD_DEPARTAMENTO | COD_CARGO | COD_GERENTE | SUM(SALARIO) |
|---|---|---|---|---|
| 1 | - | VE_REP | 149 | 7000 |
| | 10 | AD_ASST | 101 | 4400 |
| | 20 | MK_ANA | 201 | 6000 |
| 2 | 20 | MK_GER | 100 | 13000 |
| | 90 | AD_VP | 100 | 34000 |
| | 90 | AD_PRES | - | 24000 |
| 3 | - | VE_REP | - | 7000 |
| | 10 | AD_ASST | - | 4400 |
| 4 | 110 | - | 101 | 12000 |
| | 110 | - | 205 | 8300 |
| 5 | 110 | - | - | 20300 |

49 linhas retornadas em 0,01 segundos

14-22

**Agrupamentos Concatenados: Exemplo**

O exemplo do slide resulta nos seguintes agrupamentos:

- `(cod_cargo, cod_gerente)` (1)
- `(cod_departamento,cod_cargo, cod_gerente)` (2)
- `(cod_cargo)` (3)
- `(cod_departamento,cod_gerente)` (4)
- `(cod_departamento)` (5)

O salário total de cada um desses grupos é calculado.

# Exercício 14



14-23

**Exercício 14**

1. Crie uma consulta para exibir as seguintes informações sobre os funcionários cujo código de gerente é menor que 120:
   - Código do gerente
   - Código do cargo e salário total para cada código de cargo para funcionários que estão subordinados ao mesmo gerente
   - Soma do salário dos funcionários de cada gerente
   - Soma do salário dos funcionários de todos esses gerentes, independentemente dos códigos dos cargos

   Salve a consulta no arquivo `cap_14_01.sql` e a execute.

| COD_GERENTE | COD_CARGO | SUM(SALARIO) |
|---|---|---|
| 100 | AD_VP | 34000 |
| 100 | ES_GER | 5800 |
| 100 | MK_GER | 13000 |
| 100 | VE_GER | 10500 |
| 100 | - | 63300 |
| 101 | CT_GER | 12000 |
| 101 | AD_ASST | 4400 |
| 101 | - | 16400 |
| 102 | IT_PROG | 9000 |
| 102 | - | 9000 |
| 103 | IT_PROG | 10200 |
| 103 | - | 10200 |
| - | - | 98900 |

13 linhas retornadas em 0,00 segundos

**Exercício 14 (continuação)**

2. Altere o arquivo `cap_14_01.sql` e use a função `GROUPING` para determinar se os valores `NULL` nas colunas correspondentes às expressões `GROUP BY` são causados pela operação `ROLLUP`.

| COD_GERENTE | COD_CARGO | SUM(SALARIO) | GER | CARGO |
|---|---|---|---|---|
| 100 | AD_VP | 34000 | 0 | 0 |
| 100 | ES_GER | 5800 | 0 | 0 |
| 100 | MK_GER | 13000 | 0 | 0 |
| 100 | VE_GER | 10500 | 0 | 0 |
| 100 | - | 63300 | 0 | 1 |
| 101 | CT_GER | 12000 | 0 | 0 |
| 101 | AD_ASST | 4400 | 0 | 0 |
| 101 | - | 16400 | 0 | 1 |
| 102 | IT_PROG | 9000 | 0 | 0 |
| 102 | - | 9000 | 0 | 1 |
| 103 | IT_PROG | 10200 | 0 | 0 |
| 103 | - | 10200 | 0 | 1 |
| - | - | 98900 | 1 | 1 |

13 linhas retornadas em 0,00 segundos

**Exercício 14 (continuação)**

3. Crie uma consulta para exibir as seguintes informações sobre os funcionários cujo código de gerente é menor que 120:
   - Código do gerente
   - Cargo e soma dos salários de cada cargo para funcionários que estão subordinados ao mesmo gerente
   - Soma dos salários dos funcionários desses gerentes
   - Valores de tabelas de referência para exibir a soma do salário para cada cargo, independentemente do gerente
   - Soma dos salário, independentemente dos cargos e gerentes

   Salve a consulta no arquivo `cap_14_03.sql` e a execute.

| COD_GERENTE | COD_CARGO | SUM(SALARIO) |
|---|---|---|
| - | - | 98900 |
| - | AD_VP | 34000 |
| - | CT_GER | 12000 |
| - | ES_GER | 5800 |
| - | MK_GER | 13000 |
| - | VE_GER | 10500 |
| - | AD_ASST | 4400 |
| - | IT_PROG | 19200 |
| 100 | - | 63300 |
| 100 | AD_VP | 34000 |
| 100 | ES_GER | 5800 |
| 100 | MK_GER | 13000 |
| 100 | VE_GER | 10500 |
| 101 | - | 16400 |
| 101 | CT_GER | 12000 |
| 101 | AD_ASST | 4400 |
| 102 | - | 9000 |
| 102 | IT_PROG | 9000 |
| 103 | - | 10200 |
| 103 | IT_PROG | 10200 |

20 linhas retornadas em 0,01 segundos

**Exercício 14 (continuação)**

4. Altere o arquivo `cap_14_03.sql` e use a função `GROUPING` para determinar se os valores `NULL` nas colunas correspondentes às expressões `GROUP BY` são causados pela operação `CUBE`.

| COD_GERENTE | COD_CARGO | SUM(SALARIO) | GER | CARGO |
|---|---|---|---|---|
| - | - | 98900 | 1 | 1 |
| - | AD_VP | 34000 | 1 | 0 |
| - | CT_GER | 12000 | 1 | 0 |
| - | ES_GER | 5800 | 1 | 0 |
| - | MK_GER | 13000 | 1 | 0 |
| - | VE_GER | 10500 | 1 | 0 |
| - | AD_ASST | 4400 | 1 | 0 |
| - | IT_PROG | 19200 | 1 | 0 |
| 100 | - | 63300 | 0 | 1 |
| 100 | AD_VP | 34000 | 0 | 0 |
| 100 | ES_GER | 5800 | 0 | 0 |
| 100 | MK_GER | 13000 | 0 | 0 |
| 100 | VE_GER | 10500 | 0 | 0 |
| 101 | - | 16400 | 0 | 1 |
| 101 | CT_GER | 12000 | 0 | 0 |
| 101 | AD_ASST | 4400 | 0 | 0 |
| 102 | - | 9000 | 0 | 1 |
| 102 | IT_PROG | 9000 | 0 | 0 |
| 103 | - | 10200 | 0 | 1 |
| 103 | IT_PROG | 10200 | 0 | 0 |

20 linhas retornadas em 0,00 segundos

**Exercício 14 (continuação)**

5. Usando `GROUPING SETS`, crie uma consulta para exibir os seguintes agrupamentos:

```
cod_departamento, cod_gerente, cod_cargo
cod_departamento, cod_cargo
cod_gerente, cod_cargo
```

A consulta deve calcular a soma dos salários para cada um desses grupos.

| COD_DEPARTAMENTO | COD_GERENTE | COD_CARGO | SUM(SALARIO) |
|---|---|---|---|
| - | 149 | VE_REP | 7000 |
| 10 | 101 | AD_ASST | 4400 |
| 20 | 201 | MK_ANA | 6000 |
| 20 | 100 | MK_GER | 13000 |
| 50 | 124 | ES_AUX | 11700 |
| 50 | 100 | ES_GER | 5800 |
| 60 | 102 | IT_PROG | 9000 |
| 60 | 103 | IT_PROG | 10200 |
| 80 | 100 | VE_GER | 10500 |
| 80 | 149 | VE_REP | 19600 |
| 90 | 100 | AD_VP | 34000 |
| 90 | - | AD_PRES | 24000 |
| 110 | 101 | CT_GER | 12000 |
| 110 | 205 | CTPUB_GER | 8300 |
| - | 205 | CTPUB_GER | 8300 |
| - | 149 | VE_REP | 26600 |
| - | 201 | MK_ANA | 6000 |
| - | 124 | ES_AUX | 11700 |
| - | 103 | IT_PROG | 10200 |
| - | - | AD_PRES | 24000 |
| - | 101 | AD_ASST | 4400 |

40 linhas retornadas em 0,01 segundos