
FIAP
FIAP

FIAP

Advanced SQL

Manipulação de Grandes Conjuntos de Dados

- Uma subconsulta pode ser utilizada selecionar dados de uma tabela para serem inseridos em outra tabela.
- Ela também pode ser utilizada para na cláusula `WHERE` para realizar operações de atualizações ou de deleções em massa.
- A subconsulta pode ser utilizada na cláusula `FROM` de uma instrução de seleção.

Instruções DML – Data Manipulation Language – EXEMPLOS (AVANÇADO)

FIAP

INSERT

```
INSERT INTO EMPLOYEES_RETIRED
   (employee_id, first_name, last_name, email,
    phone_number, hire_date, retired_date, job_id,
    salary, commission_pct)
SELECT employee_id, first_name, last_name, email,
       phone_number, hire_date, sysdate, job_id,
       salary, commission_pct
FROM employees
WHERE employee_id=110;
```

Para evitar a criação de uma visão para executar uma inserção é possível utilizar uma subconsulta no lugar do nome da tabela.

```
INSERT INTO EMPLOYEES_RETIRED
  (employee_id, first_name, last_name, email,
   phone_number, hire_date, retired_date, job_id,
   salary, commission_pct)
SELECT employee_id, first_name, last_name, email,
      phone_number, hire_date, sysdate, job_id,
      salary, commission_pct
FROM employees
WHERE employee_id=110;
```

Instruções DML – Data Manipulation Language – EXEMPLOS (AVANÇADO)

FIAP

UPDATE

```
UPDATE   employees
SET      department_id   =   (SELECT department_id
                              FROM employees
                              WHERE employee_id = 130)
WHERE    job_id          =   (SELECT job_id
                              FROM employees
                              WHERE employee_id = 114);
```

Transfira para o mesmo departamento do funcionário 130 todos os funcionários que tem o mesmo cargo do funcionário de código 114

```
UPDATE  employees
SET     department_id  =  (SELECT department_id
                  FROM employees
                  WHERE employee_id = 130)
WHERE   job_id         =  (SELECT job_id
                  FROM employees
                  WHERE employee_id = 114);
```

Instruções DML – Data Manipulation Language – EXEMPLOS (AVANÇADO)

FIAP

DELETE

```
DELETE FROM employees
WHERE  department_id =
          (SELECT department_id
           FROM   departments
           WHERE  department_name  = 'Retail Sales');
```

Delete todos os funcionário do departamento chamado Retail Sales

```
DELETE FROM employees
WHERE  department_id =
          (SELECT department_id
           FROM   departments
           WHERE  department_name  = 'Retail Sales');
```

Inserção em múltiplas tabelas

- Uma inserção em múltiplas tabelas é o resultado das linhas retornadas por uma subconsulta executada em uma ou mais tabelas.
- Esse tipo de inserção é muito utilizado em processos de extração, transformação e carga - ETL (*Extraction, Transformation and Loading*)
- Os tipos de inserções em múltiplas tabelas são os seguintes:
  - INSERT Incondicional
  - ALL INSERT Condicional
  - FIRST INSERT Condicional
  - INSERT de Criação de Pivô

Inserção em múltiplas tabelas

- A instrução para a inserção em múltiplas tabelas apresenta a seguinte sintaxe geral a seguir:

```
INSERT [ALL] [clasula_insercao_condicional]
[clausula_insercao valores] (subconsulta);

clasula_insercao_condicional

[ALL] [FIRST]
[WHEN condicao THEN] [clausula_insercao valores]
[ELSE] [clausula_insercao valores]
```

INSERT Incondicional

- Para cada linha retornada pela subconsulta será executada a clasula_insercao_condicional para diversas tabelas.

```
INSERT  ALL
INTO sal_hist_dept VALUES(EMP_ID,HIREDATE,SALARY)
INTO mgr_hist_dept VALUES(EMP_ID,MANAGER,SALARY)
SELECT employee_id EMP_ID,
       hire_date   HIREDATE,
       salary,
       manager_id  MANAGER
FROM  employees
WHERE department_id = 90 ;
```

Nesse exemplo serão inseridos nas tabelas sal_hist_dept e mgr_hist_dept as informações dos funcionários que trabalham no departamento cujo código é igual a 90.

```
INSERT  ALL
INTO sal_hist_dept VALUES(EMP_ID,HIREDATE,SALARY)
INTO mgr_hist_dept VALUES(EMP_ID,MANAGER,SALARY)
SELECT employee_id EMP_ID,
    hire_date   HIREDATE,
    salary,
    manager_id  MANAGER
FROM  employees
WHERE department_id = 90 ;
```

INSERT Condicional: clasula_insercao_condicional

- Serão filtradas para cada `clasula_insercao_condicional` pela condição `WHEN` correspondente, que irá determinar se essa cláusula será executada.
- A instrução de inserção pode ser composta por até 127 cláusulas `WHEN`.
- INSERT Condicional:
  - Será avaliada cada cláusula `WHEN` independentemente dos resultados de qualquer outra cláusula `WHEN` avaliada.
  - Cada cláusula `WHEN` que retorna verdade irá executar as cláusulas `INTO` correspondentes.

INSERT Condicional

```
INSERT
  WHEN SALARY < 20000 THEN
    INTO sal_hist_dept
VALUES(EMP_ID,HIREDATE,SALARY)
  WHEN MANAGER IS NULL   THEN
    INTO mgr_hist_dept
VALUES(EMP_ID,MANAGER,SALARY)
    SELECT employee_id EMP_ID,
           hire_date   HIREDATE,
           salary,
           manager_id   MANAGER
    FROM  employees
    WHERE department_id = 90 ;
```

Nesse exemplo serão inseridos:
Na tabela sal_hist_dept os funcionários que têm o salário menor que 20000.
Na tabela mgr_hist_dept os funcionários que se reportam para um funcionário superior na hierarquia.
Todos os funcionários selecionados trabalham no departamento cujo código é igual a 90.

```
INSERT
 WHEN SALARY < 20000 THEN
  INTO sal_hist_dept VALUES(EMP_ID,HIREDATE,SALARY)
 WHEN MANAGER IS NULL   THEN
  INTO mgr_hist_dept VALUES(EMP_ID,MANAGER,SALARY)
  SELECT employee_id EMP_ID,
       hire_date   HIREDATE,
       salary,
       manager_id   MANAGER
  FROM  employees
  WHERE department_id = 90 ;
```

INSERT Condicional: FIRST

- Será avaliada cada cláusula `WHEN` da instrução. Se a primeira cláusula retornar verdade será executará a cláusula `INTO` correspondente e as cláusulas `WHEN` subsequentes serão ignoradas.

INSERT Condicional: FIRST

```
drop table special_sal;
create table special_sal
(deptid number(4),
sal number(8,2));

drop table hiredate_history_00;
create table hiredate_history_00
(deptid number(4),
hiredate date);

drop table hiredate_history_99;
create table hiredate_history_99
(deptid number(4),
hiredate date);

drop table hiredate_history;
create table hiredate_history
(deptid number(4),
hiredate date);
```

INSERT Condicional: FIRST

```
INSERT FIRST
  WHEN SUM_SALARY  > 25000 THEN
    INTO highest_sal VALUES(DEPT_ID, SUM_SALARY)
  WHEN HIREDATE like ('%00%') THEN
    INTO hiredate_hist_00 VALUES(DEPT_ID, HIREDATE)
  WHEN HIREDATE like ('%99%') THEN
    INTO hiredate_hist_99 VALUES(DEPT_ID, HIREDATE)
  ELSE
    INTO hiredate_hist VALUES(DEPT_ID, HIREDATE)
  SELECT department_id  DEPT_ID,
         SUM(salary)    SUM_SALARY,
         MAX(hire_date) HIREDATE
  FROM   employees
  GROUP BY department_id;
```

Vejamos um exemplo onde a instrução de seleção irá retornar o maior salário e a data de contratação do último funcionário contratado de todos os departamentos.
Na tabela highest_sal serão inseridas as informações do departamento em que o total de salários é maior que 25000, independente da data de admissão.
Caso a primeira condição não tenha sido atendida as próximas condições serão avaliadas. As linhas selecionadas pela instrução de seleção serão inseridas na tabela hiredate_hist_00 para os departamentos caso a data de admissão mais recente tenha sido no ano 2000.
Serão inseridas na tabela hiredate_hist_99 os dados relativos aos departamentos cuja data de admissão mais recente tenha sido no ano 1999.
Para as linhas que não satisfizerem as condições de data de admissão mais recente elas serão inseridas na tabela hiredate_hist.

```
drop table special_sal;
create table special_sal
(deptid number(4),
sal number(8,2));
```

INSERT Condicional: Cláusula ELSE

- Se nenhuma cláusula `WHEN` retornar verdade então:
- Caso tenha sido especificada uma cláusula `ELSE`, será executará a lista de cláusulas `INTO` associada à cláusula `ELSE`.
- Caso não tenha sido especificada uma cláusula `ELSE`, não será executada nenhuma ação para essa linha.

INSERT de Criação de Pivô

- A criação de pivô é uma operação onde será realizada uma transformação de forma que cada linha de uma tabela será convertia em várias outras linhas.

INSERT de Criação de Pivô – Exemplo

```
INSERT ALL
  INTO sales_informations
  VALUES (emp_id, last_name, week_id, MONDAY)
  INTO sales_informations
  VALUES (emp_id, last_name, week_id, TUESDAY)
  INTO sales_informations
  VALUES (emp_id, last_name, week_id, WEDNESDAY)
  INTO sales_informations
  VALUES (emp_id, last_name, week_id, THURSDAY)
  INTO sales_informations
  VALUES (emp_id, last_name, week_id, FRIDAY)
  SELECT employee_id emp_id,
         last_name name,
         week_id,MONDAY,TUESDAY,WEDNESDAY, THURSDAY,FRIDAY
    FROM sales_data;
```

```
drop table sales_informations;
create  table sales_informations
(emp_id number(5),
last_name varchar(50) not null,
week_id number(2),
week_day varchar(10));

drop TABLE SALES_DATA;
CREATE TABLE SALES_DATA
(employee_id  NUMBER(6),
last_name  VARCHAR(50),
WEEK_ID        NUMBER(2),
MONDAY         NUMBER(8,2),
TUESDAY        NUMBER(8,2),
WEDNESDAY   NUMBER(8,2),
THURSDAY      NUMBER(8,2),
FRIDAY NUMBER(8,2));
```

Instrução MERGE

- Essa instrução permite atualizar ou inserir dados em uma tabela de acordo com uma condição.
- Caso a linha exista será executada uma atualização senão será executada uma inserção caso a linha seja nova.

Instrução MERGE - Sintaxe

```
MERGE INTO nome_tabela apelido_tabela
  USING (tabela|visão|sub_consulta) apelido
  ON (condicao_juncao)
  WHEN MATCHED THEN
    UPDATE SET
    coluna1 = coluna_valor1,
    coluna2 = coluna_valor2
  WHEN NOT MATCHED THEN
    INSERT (list_colunas)
 VALUES (valores);
```

`INTO`	Define a tabela de destino onde os dados será atualizado ou a linha será inserida.
`USING`	Define a origem dos dados para a atualização ou para a inserção. Pode ser uma tabela, ou visão ou subconsulta.
`ON`	Condição que irá definir se a operação será de inserção ou de atualização.
`WHEN MATHED | WHEN NOT MATCHED`	Define como serão realizadas as condições de junções.

Instrução MERGE - Exemplo

```
MERGE
INTO catalog1 s1
USING catalog2 s2
ON (s1.id = s2.id)
WHEN MATCHED THEN
   UPDATE SET s1.price = s2.price
WHEN NOT MATCHED THEN
   INSERT (id, item, price)
   values (s2.id, s2.item, s2.price);
```

Caso a linha exista será executada uma atualização senão será executada uma inserção caso a linha seja nova.

```
create table catalog1
(id number(3),
item varchar2 (20),
price number(6));

insert into catalog1 values(1, 'laptop', 800);
insert into catalog1 values(2, 'iphone', 500);
insert into catalog1 values(3, 'camera', 700);
commit;
select * from catalog1;

create table catalog2
(id number(3),
item varchar2 (20),
price number(6));

insert into catalog2 values(1, 'laptop', 899);
insert into catalog2 values(2, 'iphone', 599);
insert into catalog2 values(5, 'video camera', 799);
```

```
commit;
select * from catalog2;
```

AGRUPAMENTO com os operadores ROLLUP e CUBE

- A criação de pivô é uma operação onde será realizada uma transformação de forma que cada linha de uma tabela será convertia em várias outras linhas.

AGRUPAMENTO ROLLUP

- O agrupamento `ROLLUP` resulta em um conjunto com as linhas agrupadas e os valores de subtotais.
- Além dos resultados regulares de agregação retornados pela cláusula `GROUP BY`, o operador `ROLLUP` produz subtotais de grupo .

AGRUPAMENTO ROLLUP-Exemplo

```
SELECT   department_id dept, job_id job,
         SUM(salary) sum_salary
FROM     employees
WHERE    department_id >= 50
GROUP BY ROLLUP(department_id, job_id);
```

No exemplo serão exibidos os totais de salários agrupados por cargos dentro dos respectivos departamentos, os subtotais dos salários por departamento e o total geral para todos os departamentos que têm o código maior ou igual que 50.

```
SELECT   department_id dept, job_id job,
       SUM(salary) sum_salary
FROM     employees
WHERE    department_id >= 50
GROUP BY ROLLUP(department_id, job_id);
```

AGRUPAMENTO CUBE

- O agrupamento `CUBE` resulta em um conjunto com a linhas de `ROLLUP` e as linhas de tabelas de referência cruzada. Além dos subtotais gerados pela oeprador `ROLLUP`, o operador `CUBE` gera subtotais para todas as combinações das dimensões especificadas.

AGRUPAMENTO CUBE – Exemplo

```
SELECT   department_id, job_id job,
         SUM(salary) sum_salary
FROM     employees
WHERE    department_id >= 50
GROUP BY CUBE(department_id, job_id);
```

No exemplo serão exibidos os totais de salários agrupados por cargos dentro dos respectivos departamentos, os subtotais dos salários por departamento, o total por cargo independente dos departamentos, o total por departamentos independente dos cargos e o total geral para todos os departamentos que tem o código igual ou maior que 50.

Função GROUPING

- A função `GROUPING` é utilizada com os operadores `CUBE` ou `ROLLUP` para indicar qual o grupo que foram um subtotal de uma linha.
- Essa função retorna 1 para indicar que a linha pertence a um determinado grupo e 0 para indicar que a linha não pertence do grupo.
- Vejamos um exemplo a seguir de utilização da função `GROUPING`

Função GROUPING - Exemplo

```
SELECT   department_id dept, job_id job,
         SUM(salary) sum_salary,
         GROUPING(department_id) group_department,
         GROUPING(job_id) group_job
FROM     employees
WHERE    department_id >= 50
GROUP BY CUBE(department_id, job_id);
```

Essa função retorna:
1 para indicar que a linha pertence a um determinado grupo e
0 para indicar que a linha não pertence do grupo.

Consultas Hierárquicas

- As consultas hierárquicas são utilizadas para recuperar dados baseando-se em um relacionamento hierárquico.
- Uma consulta hierárquica utiliza as cláusulas `CONNECT BY` e `START WITH`

Consultas Hierárquicas - Sintaxe

```
SELECT [LEVEL], coluna, expressao...
FROM   tabela
[WHERE condicao]
[START WITH condicao]
[CONNECT BY PRIOR condicao]
```

`SELECT`	É a cláusula de seleção básica.
`LEVEL`	Para cada linha retornada pela consulta a pseudocoluna LEVEL irá retornar 1 para uma linha-raiz, 2 para uma linha filha da raiz, e assim sucessivamente.
`FROM tabela`	Tabela que será consultada
`WHERE`	Determinar as linhas que serão selecionadas conforme uma determinada condição
`condicao`	É uma comparação com expressões
`START WITH`	Determina as linhas-raiz da hierarquia (onde iniciar). Essa cláusula é obrigatória para uma consulta hierárquica.
`CONNECT BY PRIOR`	Define as colunas que determinam o relacionamento entre linhas mães e filhas. Esta cláusula é obrigatória para uma consulta hierárquica.

Consultas Hierárquicas - Sintaxe

- A posição inicial da árvore é determinada pela cláusula START WITH da seguinte forma:
  - `START WITH coluna1 = valor`
- Para percorrer a árvore de cima para baixo deve ser utilizada a cláusula `CONNECT BY PRIOR` da seguinte forma:
  - `CONNECT BY PRIOR coluna1 = coluna2`

`SELECT`	É a cláusula de seleção básica.
`LEVEL`	Para cada linha retornada pela consulta a pseudocoluna LEVEL irá retornar 1 para uma linha-raiz, 2 para uma linha filha da raiz, e assim sucessivamente.
`FROM tabela`	Tabela que será consultada
`WHERE`	Determinar as linhas que serão selecionadas conforme uma determinada condição
`condicao`	É uma comparação com expressões
`START WITH`	Determina as linhas-raiz da hierarquia (onde iniciar). Essa cláusula é obrigatória para uma consulta hierárquica.
`CONNECT BY PRIOR`	Define as colunas que determinam o relacionamento entre linhas mães e filhas. Esta cláusula é obrigatória para uma consulta hierárquica.

Consultas Hierárquicas-Exemplos

- **Arvore de hierarquia percorrida de baixo para cima:**

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  last_name = 'Kochhar'
CONNECT BY PRIOR manager_id = employee_id ;
```

```
--Arvore de hierarquia  percorrida de baixo para cima:
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  last_name = 'Kochhar'
CONNECT BY PRIOR manager_id = employee_id ;

--Árvore  percorrida de cima para baixo:
SELECT  last_name||' Responde para '||
PRIOR   last_name "Árvore de Cima para Baixo",
LEVEL   "Nível"
FROM    employees
START   WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id;
```

Consultas Hierárquicas-Exemplos

- **Árvore percorrida de cima para baixo:**

```
SELECT  last_name||' Responde para '||
PRIOR   last_name "Árvore Hiede Cima para Baixo",
LEVEL   "Nível"
FROM    employees
START   WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id;
```

```
--Arvore de hierarquia  percorrida de baixo para cima:
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  last_name = 'Kochhar'
CONNECT BY PRIOR manager_id = employee_id ;

--Árvore  percorrida de cima para baixo:
SELECT  last_name||' Responde para '||
PRIOR   last_name "Árvore de Cima para Baixo",
LEVEL   "Nível"
FROM    employees
START   WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id;
```

CLÁUSULA WITH

- A cláusula `WITH`, apresenta a seguinte sintaxe a seguir

```
WITH nome_da_consulta AS (subconsulta)
        [, nome_da_consulta AS (subconsulta) ]...
```

CLÁUSULA WITH

- Vejamos um exemplo onde a consulta a seguir irá selecionar o nome do departamento e o total de salários dos departamentos quando o somatório de salários dos departamentos é maior que a média dos salários dos departamentos:

CLÁUSULA WITH – Exemplo

- A seleção a seguir será transformada em uma instrução com com a cláusula `WITH`

```
SELECT    d.department_name, SUM(e.salary) dept_total
FROM      employees e INNER JOIN departments d
ON        e.department_id = d.department_id
GROUP BY  d.department_name
HAVING    SUM(e.salary)>(SELECT AVG(dept_total)
                     FROM (SELECT d.department_name,
                                  SUM(e.salary) dept_total
                           FROM employees e JOIN departments d
                           ON e.department_id = d.department_id
                           GROUP BY d.department_name ))
       ORDER BY department_name;
```

A cláusula WITH tem por característica facilitar a leitura da consulta, apenas uma cláusula por vez é avaliada mesmo se ela aparecer inúmeras vezes na consulta.

Nesse exemplo a consulta irá selecionar o nome do departamento e o total de salários dos departamentos quando o somatório de salários dos departamentos é maior que a média do somatório dos salários dos departamentos

CLÁUSULA WITH – Exemplo

- Após reescrever a consulta anterior com a cláusula `WTH` e ficará da seguinte forma:

```
WITH
    dept_costs AS( SELECT d.department_name,
                          SUM(e.salary) dept_total
                   FROM  employees e JOIN departments d
                   ON e.department_id = d.department_id
                   GROUP BY  d.department_name),
    avg_cost AS (SELECT AVG(dept_total)AS dept_avg
                 FROM dept_costs)
SELECT  *
FROM   dept_costs
WHERE  dept_total >( SELECT dept_avg
                     FROM   avg_cost )
ORDER BY department_name;
```

Nesse exemplo será utilizada a cláusula WITH para produzir o mesmo resultado do comando SELECT anterior

Referências

- Oracle. **Oracle Database Sample Schemas 19c**. Disponível em: https://docs.oracle.com/en/database/oracle/oracle-database/19/comsc/index.html. Acesso em: 06 de jul. de 2022.
- Oracle. **SQL Language Reference 12*c* Release 2 (12.2)**, 2022. Disponível em: <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/index.html>. Acesso em: 06 de jul. de 2022.
- Oracle Education. **Oracle Database 12c: SQL Workshop I/II**. 2013
- Oracle Education. **Oracle Database 12c Administration Workshop Ed 2**, 2014
- PRICE, Jason. **Oracle Database 12c SQL**. Oracle Press, 2013.

Exercício - Manipulação de Grandes Conjuntos de Dadps

1. Execute o script a seguir para criar a tabela SAL_HISTORY.

```
DROP TABLE SAL_HISTORY;
CREATE table SAL_HISTORY
(EMPLOYEE_ID  NUMBER(6),
HIRE_DATE     DATE,
SALARY   NUMBER(8,2));
```

2. Exiba a estrutura da tabela SAL_HISTORY.

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
HIRE_DATE		DATE
SALARY		NUMBER(8,2)

3. Execute o script a seguir para criar a tabela MGR_HISTORY.

```
DROP TABLE MGR_HISTORY;
CREATE table MGR_HISTORY
(EMPLOYEE_ID  NUMBER(6),
MANAGER_ID    NUMBER(6),
SALARY   NUMBER(8,2));
```

4. Exiba a estrutura da tabela MGR_HISTORY.

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
MANAGER_ID		NUMBER(6)
SALARY		NUMBER(8,2)

5. Execute o script a seguir para criar a tabela SPECIAL_SAL.

```
drop TABLE SPECIAL_SAL;
CREATE table SPECIAL_SAL
(EMPLOYEE_ID  NUMBER(6),
SALARY   NUMBER(8,2));
```

6. Exiba a estrutura da tabela SPECIAL_SAL.

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
SALARY		NUMBER(8,2)

Exercício (continuação)

7. a. Crie uma consulta que faça o seguinte:

   Recupere na tabela `EMPLOYEES` os detalhes de ID do funcionário, data de admissão, salário e o ID do gerente desses funcionários cujo ID é inferior a 125.

   Se o salário for superior a $20.000, insira os detalhes sobre o ID do funcionário e o salário na tabela `SPECIAL_SAL`.

   Insira o ID do funcionário, a data de admissão e o salário na tabela `SAL_HISTORY`.

   Insira os detalhes sobre o ID do funcionário, o ID do gerente e o salário na tabela `MGR_HISTORY`.

Exercício (continuação)

b. Exiba os registros da tabela `SPECIAL_SAL`.

EMPLOYEE_ID	SALARY
100	24000

c. Exiba os registros da tabela `SAL_HISTORY`.

EMPLOYEE_ID	HIRE_DATE	SALARY
101	21-SEP-89	17000
102	13-JAN-93	17000
103	03-JAN-90	9000
104	21-MAY-91	6000
105	25-JUN-97	4800
106	05-FEB-98	4800
107	07-FEB-99	4200
108	17-AUG-94	12000
109	16-AUG-94	9000
110	28-SEP-97	8200
111	30-SEP-97	7700
112	07-MAR-98	7800
113	07-DEC-99	6900
114	07-DEC-94	11000
115	18-MAY-95	3100
116	24-DEC-97	2900
117	24-JUL-97	2800
118	15-NOV-98	2600
119	10-AUG-99	2500
120	18-JUL-96	8000
121	10-APR-97	8200
122	01-MAY-95	7900
123	10-OCT-97	6500
124	16-NOV-99	5800

24 rows selected.

Exercício (continuação)

d. Exiba os registros da tabela `MGR_HISTORY`.

EMPLOYEE_ID	MANAGER_ID	SALARY
101	100	17000
102	100	17000
103	102	9000
104	103	6000
105	103	4800
106	103	4800
107	103	4200
108	101	12000
109	108	9000
110	108	8200
111	108	7700
112	108	7800
113	108	6900
114	100	11000
115	114	3100
116	114	2900
117	114	2800
118	114	2600
119	114	2500
120	100	8000
121	100	8200
122	100	7900
123	100	6500
124	100	5800

24 rows selected.

Exercício (continuação)

8. a. Execute o script a seguir para criar a tabela SALES_SOURCE_DATA.

```
drop TABLE SALES_SOURCE_DATA;
CREATE TABLE SALES_SOURCE_DATA
(employee_id   NUMBER(6),
WEEK_ID   NUMBER(2),
SALES_MON NUMBER(8,2),
SALES_TUE NUMBER(8,2),
SALES_WED NUMBER(8,2),
SALES_THUR     NUMBER(8,2),
SALES_FRI NUMBER(8,2));
```

b. Execute o script a seguir para inserir registros na tabela SALES_SOURCE_DATA.

```
INSERT INTO SALES_SOURCE_DATA VALUES
(178, 6, 1750,2200,1500,1500,3000);
commit;
```

c. Exiba a estrutura da tabela SALES_SOURCE_DATA.

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
WEEK_ID		NUMBER(2)
SALES_MON		NUMBER(8,2)
SALES_TUE		NUMBER(8,2)
SALES_WED		NUMBER(8,2)
SALES_THUR		NUMBER(8,2)
SALES_FRI		NUMBER(8,2)

d. Exiba os registros da tabela SALES_SOURCE_DATA.

EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
178	6	1750	2200	1500	1500	3000

Exercício (continuação)

e. Execute o script a seguir para criar a tabela SALES_INFO.

```
drop TABLE SALES_INFO;
CREATE TABLE SALES_INFO
(employee_id   NUMBER(6),
WEEK NUMBER(2),
SALES     NUMBER(8,2));
```

f. Exiba a estrutura da tabela SALES_INFO.

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
WEEK		NUMBER(2)
SALES		NUMBER(8,2)

Exercício 3 (continuação)

g. Crie uma consulta que faça o seguinte:
Recupere da tabela `SALES_SOURCE_DATA` os detalhes sobre o ID do funcionário, o ID da semana, vendas na segunda-feira, vendas na terça-feira, vendas na quarta-feira, vendas na quinta-feira e vendas na sexta-feira.
Crie uma transformação de modo que cada registro recuperado da tabela `SALES_SOURCE_DATA` seja convertido em vários registros para a tabela `SALES_INFO`.

Dica: Use uma instrução `INSERT` de criação de pivô.

h. Exiba os registros da tabela `SALES_INFO`.

EMPLOYEE_ID	WEEK	SALES
178	6	1750
178	6	2200
178	6	1500
178	6	1500
178	6	3000

Exercício – Relatórios por Agrupamento de Dados Relacionados

1. Crie uma consulta para exibir as seguintes informações sobre os funcionários cujo ID de gerente é menor que 120:
   - ID do gerente
   - ID do cargo e salário total para cada ID de cargo para funcionários que estão subordinados ao mesmo gerente
   - Salário total desses gerentes
   - Salário total desses gerentes, independentemente dos IDs dos cargos

MANAGER_ID	JOB_ID	SUM(SALARY)
100	AD_VP	34000
100	MK_MAN	13000
100	PU_MAN	11000
100	SA_MAN	61000
100	ST_MAN	36400
100		155400
101	AC_MGR	12000
101	FI_MGR	12000
101	HR_REP	6500
101	PR_REP	10000
101	AD_ASST	4400
...		
101	AD_ASST	4400
101		44900
102	IT_PROG	9000
102		9000
103	IT_PROG	19800
103		19800
108	FI_ACCOUNT	39600
108		39600
114	PU_CLERK	13900
114		13900
		282600

21 rows selected.

Exercício (continuação)

2. Observe a resposta da questão 1. Crie uma consulta usando a function `GROUPING` para determinar se os valores `NULL` nas colunas correspondentes às expressões `GROUP BY` são causados pela operação `ROLLUP`.

MGR	JOB	SUM(SALARY)	GROUPING(MANAGER_ID)	GROUPING(JOB_ID)
100	AD_VP	34000	0	0
100	MK_MAN	13000	0	0
100	PU_MAN	11000	0	0
100	SA_MAN	61000	0	0
100	ST_MAN	36400	0	0
100		155400	0	1
...				
102	IT_PROG	9000	0	0
102		9000	0	1
103	IT_PROG	19800	0	0
103		19800	0	1
108	FI_ACCOUNT	39600	0	0
108		39600	0	1
114	PU_CLERK	13900	0	0
114		13900	0	1
		282600	1	1

21 rows selected.

Exercício (continuação)

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

MANAGER_ID	JOB_ID	SUM(SALARY)
		282600
	AD_VP	34000
	AC_MGR	12000
	FI_MGR	12000
	HR_REP	6500
	MK_MAN	13000

...

MANAGER_ID	JOB_ID	SUM(SALARY)
101	PR_REP	10000
101	AD_ASST	4400
102		9000
102	IT_PROG	9000
103		19800
103	IT_PROG	19800
108		39600
108	FI_ACCOUNT	39600
114		13900
114	PU_CLERK	13900

34 rows selected.

Exercício (continuação)

4. Observe a resposta da questão 3. Crie uma consulta usando a function `GROUPING` para determinar se os valores `NULL` nas colunas correspondentes às expressões `GROUP BY` são causados pela operação `CUBE`.

MGR	JOB	SUM(SALARY)	GROUPING(MANAGER_ID)	GROUPING(JOB_ID)
		282600	1	1
	AD_VP	34000	1	0
	AC_MGR	12000	1	0
	FI_MGR	12000	1	0
	HR_REP	6500	1	0
	MK_MAN	13000	1	0
	PR_REP	10000	1	0
	PU_MAN	11000	1	0
	SA_MAN	61000	1	0
	ST_MAN	36400	1	0
	AD_ASST	4400	1	0
	IT_PROG	28800	1	0
	PU_CLERK	13900	1	0
	FI_ACCOUNT	39600	1	0
100		155400	0	1

. . .

MGR	JOB	SUM(SALARY)	GROUPING(MANAGER_ID)	GROUPING(JOB_ID)
101	PR_REP	10000	0	0
101	AD_ASST	4400	0	0
102		9000	0	1
102	IT_PROG	9000	0	0
103		19800	0	1
103	IT_PROG	19800	0	0
108		39600	0	1
108	FI_ACCOUNT	39600	0	0
114		13900	0	1
114	PU_CLERK	13900	0	0

34 rows selected.

Exercício (continuação)

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

```
department_id, manager_id, job_id
department_id, job_id
manager_id, job_id
```

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

DEPARTMENT_ID	MANAGER_ID	JOB_ID	SUM(SALARY)
90		AD_PRES	24000
90	100	AD_VP	34000
20	100	MK_MAN	13000
30	100	PU_MAN	11000
80	100	SA_MAN	61000
50	100	ST_MAN	36400
110	101	AC_MGR	12000
100	101	FI_MGR	12000
		AD_PRES	24000
...			
	100	AD_VP	34000
	100	MK_MAN	13000
	100	PU_MAN	11000
...			
		SA_REP	7000
10		AD_ASST	4400
20		MK_MAN	13000
20		MK_REP	6000
...			
50		ST_MAN	36400
50		SH_CLERK	64300
50		ST_CLERK	55700
60		IT_PROG	28800
70		PR_REP	10000
80		SA_MAN	61000
80		SA_REP	243500
90		AD_VP	34000
90		AD_PRES	24000
100		FI_MGR	12000
100		FI_ACCOUNT	39600
110		AC_MGR	12000
110		AC_ACCOUNT	8300

85 rows selected.

Exercício CONSULTAS HIERÁQUICAS

1. Observe os exemplos de saída a seguir. Essas saídas são o resultado de uma consulta hierárquica? Explique por que sim e por que não.
   Exemplo 1:

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY	DEPARTMENT_ID
100	King		24000	90
101	Kochhar	100	17000	90
102	De Haan	100	17000	90
201	Hartstein	100	13000	20
205	Higgins	101	12000	110
174	Abel	149	11000	80
149	Zlotkey	100	10500	80
103	Hunold	102	9000	60
...				
200	Whalen	101	4400	10
107	Lorentz	103	4200	60
141	Rajs	124	3500	50
142	Davies	124	3100	50
143	Matos	124	2600	50
144	Vargas	124	2500	50

20 rows selected.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
205	Higgins	110	Accounting
206	Gietz	110	Accounting
100	King	90	Executive
101	Kochhar	90	Executive
102	De Haan	90	Executive
149	Zlotkey	80	Sales
174	Abel	80	Sales
176	Taylor	80	Sales
103	Hunold	60	IT
104	Ernst	60	IT
107	Lorentz	60	IT

11 rows selected.

Exercício (continuação)

Exemplo 3:

RANK	LAST_NAME
1	King
2	Kochhar
2	De Haan
3	Hunold
4	Ernst

2. Gere um relatório que mostre um organograma do departamento de Mourgos. Imprima os sobrenomes, os salários e os IDs dos departamentos.

LAST_NAME	SALARY	DEPARTMENT_ID
Mourgos	5800	50
Rajs	3500	50
Davies	3100	50
Matos	2600	50
Vargas	2500	50
Walsh	3100	50
Feeney	3000	50
OConnell	2600	50
Grant	2600	50

9 rows selected.

3. Crie um relatório que mostre a hierarquia de gerentes para o funcionário Lorentz. Exiba primeiramente o seu gerente imediato.

LAST_NAME
Hunold
De Haan
King

Exercício (continuação)

4. Crie um relatório recuado mostrando a hierarquia de gerenciamento, começando pelo funcionário cujo `LAST_NAME` é Kochhar. Imprima o sobrenome, o ID do gerente e o ID do departamento do funcionário. Defina apelidos para as colunas, conforme indicado no exemplo de saída.

NAME	MGR	DEPTNO
Kochhar	100	90
__Greenberg	101	100
____Faviet	108	100
____Chen	108	100
____Sciarra	108	100
____Urman	108	100
____Popp	108	100
__Whalen	101	10
__Mavris	101	40
__Baer	101	70
__Higgins	101	110
____Gietz	205	110

12 rows selected.

5. Produza um organograma que mostre a hierarquia de gerenciamento da empresa. Comece pela pessoa que está no nível mais alto e exclua todas as outras que tenham um ID do cargo igual a `IT_PROG`. Exclua também De Haan e respectivos subordinados.

LAST_NAME	EMPLOYEE_ID	MANAGER_ID
King	100	
Kochhar	101	100
Greenberg	108	101
Faviet	109	108
Chen	110	108
Sciarra	111	108

. . .

LAST_NAME	EMPLOYEE_ID	MANAGER_ID
Livingston	177	149
Grant	178	149
Johnson	179	149
Hartstein	201	100
Fay	202	201

101 rows selected.

OBRIGADO

https://www.linkedin.com/in/alexandrebarcelos


FIAP