

FIAP GRADUAÇÃO

# **Tecnologia em Análise e Desenvolvimento de Sistemas**

Application Development For Databases

**PROF. MILTON**

# **Manipulando Grandes Conjuntos de Datos**

## Objetivos

**Ao concluir esta lição, você será capaz de:**

- **Manipular dados usando subconsultas**
- **Descrever os recursos de inserções em várias tabelas**
- **Usar os seguintes tipos de inserções em várias tabelas:**
  - **INSERT Incondicional**
  - **INSERT de Criação de Pivô**
  - **ALL INSERT Condicional**
  - **FIRST INSERT Condicional**
- **Intercalar linhas em uma tabela**
- **Controlar as alterações de dados durante um período**

## Objetivos

Nesta lição, você aprenderá a manipular os dados do banco de dados Oracle usando subconsultas. Você também conhecerá as instruções de inserção em várias tabelas e a instrução MERGE, além de aprender a controlar as alterações feitas no banco de dados.

## Usando Subconsultas para Manipular Dados

**É possível usar subconsultas em instruções DML para:**

- **Copiar dados de uma tabela para outra**
- **Recuperar dados de uma view em linha**
- **Atualizar dados em uma tabela com base nos valores de outra tabela**
- **Deletar linhas de uma tabela com base nas linhas de outra tabela**

### Usando Subconsultas para Manipular Dados

As subconsultas podem ser usadas para recuperar dados a partir de uma tabela usada como entrada para fazer um `INSERT` em uma tabela diferente. Desse modo, você pode copiar facilmente grandes volumes de dados de uma tabela para outra com uma única instrução `SELECT`. Da mesma forma, você pode usar subconsultas para fazer atualizações e deleções em massa, incluindo-as na cláusula `WHERE` das instruções `UPDATE` e `DELETE`. Também é possível usar subconsultas na cláusula `FROM` de uma instrução `SELECT`. Esse processo se chama view em linha.

## Copiando Linhas de Outra Tabela

- Crie a instrução **INSERT** com uma subconsulta.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

4 rows created.

- Não use a cláusula **VALUES**.
- Estabeleça uma correspondência entre o número de colunas na cláusula **INSERT** e o número de colunas na subconsulta.

### Copiando Linhas de Outra Tabela

É possível usar a instrução **INSERT** para adicionar linhas a uma tabela cujos valores são provenientes de tabelas existentes. No lugar da cláusula **VALUES**, use uma subconsulta.

#### Sintaxe

```
INSERT INTO table [ column (, column) ] subquery;
```

Na sintaxe:

<i>table</i>	é o nome da tabela
<i>column</i>	é o nome da coluna da tabela a ser preenchida
<i>Subquery</i>	é a subconsulta que retorna linhas para a tabela

O número de colunas e os respectivos tipos de dados na lista de colunas da cláusula **INSERT** devem corresponder ao número de valores e aos respectivos tipos de dados na subconsulta. Para criar uma cópia das linhas de uma tabela, use **SELECT \*** na subconsulta.

```
INSERT INTO EMPL3
SELECT *
FROM employees;
```

Para obter mais informações, consulte o manual *Oracle Database 11g SQL Reference*.

## Inserção Usando uma Subconsulta como Destino

```
INSERT INTO
    (SELECT employee_id, last_name,
            email, hire_date, job_id, salary,
            department_id
     FROM   empl3
     WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000, 50);

1 row created.
```

### Inserção Usando uma Subconsulta como Destino

É possível usar uma subconsulta no lugar do nome da tabela na cláusula INTO da instrução INSERT.

A lista de seleção da subconsulta deve ter o mesmo número de colunas que a lista de colunas da cláusula VALUES. Para a execução bem-sucedida da instrução INSERT, todas as regras nas colunas da tabela base devem ser cumpridas. Por exemplo, não é possível especificar um ID de funcionário duplicado nem omitir um valor de uma coluna NOT NULL obrigatória.

Essa aplicação de subconsultas ajuda a evitar que seja necessário criar uma view apenas para executar uma inserção.

## Inserção Usando uma Subconsulta como Destino

Verifique os resultados.

```
SELECT employee_id, last_name, email, hire_date,
       job_id, salary, department_id
FROM   employees
WHERE  department_id = 50;
```

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
120	Weiss	MWEISS	18-JUL-96	ST_MAN	8000	50
121	Fripp	AFRIPP	10-APR-97	ST_MAN	8200	50
122	Kaufling	PKAUFLIN	01-MAY-95	ST_MAN	7900	50
...						
193	Everett	BEVERETT	03-MAR-97	SH_CLERK	3900	50
194	McCain	SMCCAIN	01-JUL-98	SH_CLERK	3200	50
195	Jones	VJONES	17-MAR-99	SH_CLERK	2800	50
196	Walsh	AWALSH	24-APR-98	SH_CLERK	3100	50
197	Feeney	KFEENEY	23-MAY-98	SH_CLERK	3000	50
198	OConnell	DOCONNEL	21-JUN-99	SH_CLERK	2600	50
199	Grant	DGRANT	13-JAN-00	SH_CLERK	2600	50
99999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50

46 rows selected.

### Inserção Usando uma Subconsulta como Destino (continuação)

O exemplo mostra os resultados da subconsulta usada para identificar a tabela para a instrução INSERT.



## Recuperando Dados com uma Subconsulta como Origem

```
SELECT  a.last_name, a.salary,  
        a.department_id, b.salavg  
FROM    employees a, (SELECT  department_id,  
                        AVG(salary) salavg  
                        FROM    employees  
                        GROUP BY department_id) b  
WHERE   a.department_id = b.department_id  
AND     a.salary > b.salavg;
```

LAST_NAME	SALARY	DEPARTMENT_ID	SALAVG
King	24000	90	19333.3333
Hunold	9000	60	5760
Ernst	6000	60	5760
Greenberg	12000	100	8600
Faviet	9000	100	8600
Raphaely	11000	30	4150
Weiss	8000	50	3475.55556
Fripp	8200	50	3475.55556

...

### Recuperando Dados Usando uma Subconsulta como Origem

Você pode usar uma subconsulta na cláusula FROM da instrução SELECT, que é muito semelhante à forma como as views são usadas. Uma subconsulta na cláusula FROM de uma instrução SELECT também é chamada de *view em linha*. Uma subconsulta na cláusula FROM de uma instrução SELECT define uma origem de dados apenas para essa instrução SELECT específica. O exemplo do slide exibe os sobrenomes dos funcionários, os salários, os números dos departamentos e os salários médios de todos os funcionários que recebem mais que o salário médio dos respectivos departamentos. A subconsulta na cláusula FROM é denominada b, e a consulta exterior faz referência à coluna SALAVG usando esse apelido.

## Atualizando Duas Colunas com uma Subconsulta

**Atualize o cargo e o salário do funcionário 114 para corresponder ao cargo e ao salário do funcionário 205.**

```
UPDATE emp13
SET   job_id = (SELECT job_id
                FROM   employees
                WHERE  employee_id = 205),
      salary = (SELECT salary
                FROM   employees
                WHERE  employee_id = 205)
WHERE employee_id = 114;
1 row updated.
```

10

### Atualizando Duas Colunas com uma Subconsulta

É possível atualizar diversas colunas na cláusula SET de uma instrução UPDATE criando várias subconsultas.

#### Sintaxe

```
UPDATE table
SET   column =
      (SELECT column
       FROM table
       WHERE condition)
[ ,
  column =
      (SELECT column
       FROM table
       WHERE condition) ]
[WHERE condition] ;
```

**Observação:** Se nenhuma linha for atualizada, a mensagem "0 rows updated." será exibida:

## Atualizando Linhas com Base em Outra Tabela

Use subconsultas nas instruções UPDATE para atualizar linhas de uma tabela com base em valores de outra tabela.

```
UPDATE empl3
SET    department_id = (SELECT department_id
                        FROM employees
                        WHERE employee_id = 100)
WHERE  job_id        = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);

1 row updated.
```

### Atualizando Linhas com Base em Outra Tabela

É possível usar subconsultas em instruções UPDATE para atualizar as linhas de uma tabela. O exemplo do slide atualiza a tabela EMPL3 com base nos valores da tabela EMPLOYEES. Ele altera o número do departamento de todos os funcionários com o ID de cargo do funcionário 200 para o número do departamento atual do funcionário 100.

## Deletando Linhas com Base em Outra Tabela

Use subconsultas em instruções DELETE para remover linhas de uma tabela com base nos valores de outra tabela.

```
DELETE FROM empl3
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name
       LIKE '%Public%');

1 row deleted.
```

12

### Deletando Linhas com Base em Outra Tabela

É possível usar subconsultas para deletar linhas de uma tabela com base nos valores de outra tabela. O exemplo do slide deleta todos os funcionários que trabalham em um departamento cujo nome contém a string "Public". A subconsulta pesquisa a tabela DEPARTMENTS para localizar o número do departamento com base no nome do departamento que contém a string "Public". Em seguida, a subconsulta informa o número do departamento para a consulta principal, que deleta as linhas de dados da tabela EMPLOYEES com base nesse número de departamento.

## Usando a Palavra-Chave WITH CHECK OPTION em Instruções DML

- Uma subconsulta é usada para identificar a tabela e as colunas da instrução DML.
- A palavra-chave WITH CHECK OPTION impede a alteração de linhas que não estão na subconsulta.

```
INSERT INTO (SELECT employee_id, last_name, email,
                hire_date, job_id, salary
            FROM emp13
            WHERE department_id = 50
            WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000);
INSERT INTO
*
```

**ERROR at line 1:**  
**ORA-01402: view WITH CHECK OPTION where-clause violation**

13

### A Palavra-Chave WITH CHECK OPTION

Especifique WITH CHECK OPTION para indicar que, se a subconsulta for usada no lugar de uma tabela em uma instrução INSERT, UPDATE ou DELETE, não serão permitidas alterações nessa tabela que produzam linhas não incluídas na subconsulta.

No exemplo mostrado, a palavra-chave WITH CHECK OPTION é usada. A subconsulta identifica linhas que estão no departamento 50, mas o ID do departamento não está na lista SELECT e não tem um valor especificado na lista VALUES. A inserção dessa linha resulta em um ID de departamento nulo, que não está na subconsulta.

## Visão Geral do Recurso de Default Explícito

- Com o recurso de default explícito, é possível usar a palavra-chave **DEFAULT** como um valor de coluna onde se deseja especificar o valor default de coluna.
- Esse recurso é incluído para manter a compatibilidade com o padrão **SQL:1999**.
- O recurso permite ao usuário controlar onde e quando o valor default deve ser aplicado aos dados.
- É possível usar defaults explícitos em instruções **INSERT** e **UPDATE**.

### Defaults Explícitos

É possível usar a palavra-chave **DEFAULT** em instruções **INSERT** e **UPDATE** para identificar um valor de coluna default. Se não houver um valor default, será usado um valor nulo.

A opção **DEFAULT** dispensa a codificação do valor default nos programas e a consulta ao dicionário para encontrá-lo, como se fazia antes da introdução deste recurso. A codificação do valor default é um problema quando ele se altera porque o código conseqüentemente precisa ser alterado. O acesso ao dicionário geralmente não é feito em um programa de aplicação. Portanto, trata-se de um recurso muito importante.

## Usando Valores Default Explícitos

- **DEFAULT com INSERT:**

```
INSERT INTO deptm3  
  (department_id, department_name, manager_id)  
VALUES (300, 'Engineering', DEFAULT ;
```

- **DEFAULT com UPDATE:**

```
UPDATE deptm3  
SET manager_id = DEFAULT  
WHERE department_id = 10;
```

### Usando Valores Default Explícitos

Especifique **DEFAULT** para definir a coluna para o valor especificado anteriormente como o seu valor default. Se não tiver sido especificado um valor default para a coluna correspondente, o servidor Oracle definirá a coluna como nula.

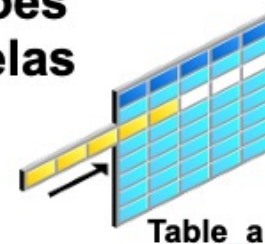
No primeiro exemplo do slide, a instrução **INSERT** usa um valor default para a coluna **MANAGER\_ID**. Se não houver um valor default definido para a coluna, um valor nulo será inserido.

O segundo exemplo usa a instrução **UPDATE** para definir a coluna **MANAGER\_ID** com um valor default para o departamento 10. Se nenhum valor default for definido para a coluna, o valor será alterado para nulo.

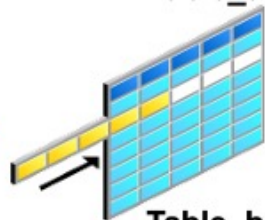
**Observação:** Ao criar uma tabela, você poderá especificar um valor default para uma coluna. Esse assunto será abordado na lição intitulada "Criando e Gerenciando Tabelas".

## Visão Geral de Instruções INSERT em Várias Tabelas

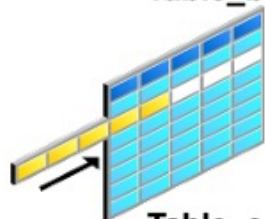
```
INSERT ALL  
  INTO table_a VALUES (...,...,...)  
  INTO table_b VALUES (...,...,...)  
  INTO table_c VALUES (...,...,...)  
  SELECT ...  
  FROM sourcetab  
  WHERE ...;
```



Table\_a



Table\_b



Table\_c

### Visão Geral de Instruções INSERT em Várias Tabelas

Em uma instrução INSERT em várias tabelas, são inseridas as linhas calculadas derivadas das linhas retornadas da avaliação de uma subconsulta em uma ou mais tabelas.

As instruções INSERT em várias tabelas podem desempenhar um papel muito útil em um cenário de data warehouse. Carregue o data warehouse regularmente para que ele atenda ao propósito de facilitar a análise de negócios. Para isso, é necessário extrair e copiar os dados de um ou mais sistemas operacionais para o warehouse. O processo de extração de dados do sistema de origem e de adição desses dados ao data warehouse é comumente chamado de ETL (Extraction, Transformation and Loading), que significa extração, transformação e carga.

Durante a extração, os dados desejados precisam ser identificados e extraídos de várias origens distintas, tais como aplicações e sistemas de banco de dados. Depois da extração, os dados precisam ser transportados fisicamente para o sistema de destino ou para um sistema intermediário para processamento adicional. Dependendo do meio de transporte escolhido, é possível realizar algumas transformações durante esse processo. Por exemplo, uma instrução SQL que acesse diretamente um destino remoto através de um gateway pode concatenar duas colunas como parte da instrução SELECT.

Após a carga dos dados no banco de dados Oracle, é possível executar transformações nos



dados usando operações SQL. Uma instrução `INSERT` em várias tabelas é uma das técnicas para implementar transformações de dados SQL.

## Visão Geral de Instruções INSERT em Várias Tabelas

- A instrução **INSERT...SELECT** pode ser usada para inserir linhas em várias tabelas como parte de uma única instrução DML.
- Várias instruções **INSERT** podem ser usadas em sistemas de data warehouse para transferir dados de uma ou mais origens operacionais para um conjunto de tabelas de destino.
- Elas permitem uma melhoria significativa no desempenho em relação a:
  - Uma única instrução DML x várias instruções **INSERT...SELECT**
  - Uma única instrução DML x um procedimento para executar várias inserções usando a sintaxe **IF...THEN**

### Visão Geral de Instruções **INSERT** em Várias Tabelas (Continuação)

As instruções **INSERT** em várias tabelas oferecem os benefícios da instrução **INSERT ... SELECT** quando há várias tabelas como destino. Antes do Banco de Dados Oracle9i, para usar essa funcionalidade, era necessário lidar com  $n$  instruções **INSERT ... SELECT** independentes, processando, assim, os mesmos dados-fonte  $n$  vezes e aumentando a carga de trabalho de transformação  $n$  vezes.

Como ocorre com a instrução **INSERT ... SELECT** existente, a nova instrução pode ser paralelizada e usada com o mecanismo de carga direta para garantir um melhor desempenho.

Agora os registros de qualquer fluxo de entrada, como, por exemplo, uma tabela de banco de dados não relacional, podem ser convertidos em vários registros para um ambiente de tabela de banco de dados mais relacional. Para implementar essa funcionalidade opcionalmente, foi solicitado que você criasse várias instruções **INSERT**.

## **Tipos de Instruções INSERT em Várias Tabelas**

**Os diversos tipos de instruções INSERT em várias tabelas são:**

- **INSERT Incondicional**
- **ALL INSERT Condicional**
- **FIRST INSERT Condicional**
- **INSERT de Criação de Pivô**

### **Tipos de Instruções INSERT em Várias Tabelas**

Os tipos de instruções INSERT em várias tabelas são:

- INSERT Incondicional
- ALL INSERT Condicional
- FIRST INSERT Condicional
- INSERT de Criação de Pivô

Use cláusulas distintas para indicar o tipo de instrução INSERT a ser executada.

## Instruções INSERT em Várias Tabelas

- **Sintaxe**

```
INSERT [ALL] [conditional_insert_clause]  
[insert_into_clause values_clause] (subquery)
```

- **conditional\_insert\_clause**

```
[ALL] [FIRST]  
[WHEN condition THEN] [insert_into_clause values_clause]  
[ELSE] [insert_into_clause values_clause]
```

### Instruções INSERT em Várias Tabelas

O slide exibe o formato genérico para as instruções INSERT em várias tabelas.

#### **INSERT Incondicional: ALL into\_clause**

Especifique ALL seguido por diversas insert\_into\_clauses para executar uma inserção incondicional em várias tabelas. O servidor Oracle executa cada insert\_into\_clause uma vez por linha retornada pela subconsulta.

#### **INSERT Condicional: conditional\_insert\_clause**

Especifique conditional\_insert\_clause para executar uma instrução INSERT condicional em várias tabelas. O servidor Oracle filtra cada insert\_into\_clause pela condição WHEN correspondente, que determina se essa insert\_into\_clause será executada. Uma única instrução INSERT em várias tabelas pode conter até 127 cláusulas WHEN.

#### **INSERT Condicional: ALL**

Se você especificar ALL, o servidor Oracle avaliará cada cláusula WHEN independentemente dos resultados da avaliação de qualquer outra cláusula WHEN. Para cada cláusula WHEN cuja condição é avaliada como verdadeira, o servidor Oracle executa a lista de cláusulas INTO correspondente.

## INSERT ALL Incondicional

- **Selecione os valores de EMPLOYEE\_ID, HIRE\_DATE, SALARY e MANAGER\_ID na tabela EMPLOYEES para os funcionários cujo EMPLOYEE\_ID é maior que 200.**
- **Insira esses valores nas tabelas SAL\_HISTORY e MGR\_HISTORY usando uma instrução INSERT em várias tabelas.**

```
INSERT ALL  
  INTO sal_history VALUES (EMPID, HIREDATE, SAL)  
  INTO mgr_history VALUES (EMPID, MGR, SAL)  
  SELECT employee_id EMPID, hire_date HIREDATE,  
         salary SAL, manager_id MGR  
  FROM employees  
  WHERE employee_id > 200;  
8 rows created.
```

21

### INSERT ALL Incondicional

O exemplo do slide insere linhas nas tabelas SAL\_HISTORY e MGR\_HISTORY.

A instrução SELECT recupera, na tabela EMPLOYEES, os detalhes sobre o ID do funcionário, a data de admissão, o salário e o ID do gerente dos funcionários cujo ID é maior que 200. Os detalhes sobre o ID do funcionário, a data de admissão e o salário são inseridos na tabela SAL\_HISTORY. Os detalhes sobre o ID do funcionário, o ID do gerente e o salário são inseridos na tabela MGR\_HISTORY.

Essa instrução INSERT é denominada INSERT incondicional, pois não são aplicadas outras restrições às linhas recuperadas pela instrução SELECT. Todas as linhas recuperadas pela instrução SELECT são inseridas nas duas tabelas, SAL\_HISTORY e MGR\_HISTORY. A cláusula VALUES nas instruções INSERT especifica as colunas da instrução SELECT que precisam ser inseridas em cada uma das tabelas. Cada linha retornada pela instrução SELECT resulta em duas inserções, uma na tabela SAL\_HISTORY e outra na tabela MGR\_HISTORY.

É possível interpretar as 8 linhas criadas e retornadas como um total de oito inserções executadas nas tabelas-base, SAL\_HISTORY e MGR\_HISTORY.

## INSERT ALL Condicional

- **Selecione os valores de EMPLOYEE\_ID, HIRE\_DATE, SALARY e MANAGER\_ID na tabela EMPLOYEES para os funcionários cujo EMPLOYEE\_ID é maior que 200.**
- **Se o valor de SALARY for maior que \$10.000, insira esse valor na tabela SAL\_HISTORY usando uma instrução INSERT condicional em várias tabelas.**
- **Se o valor de MANAGER\_ID for maior que 200, insira esse valor na tabela MGR\_HISTORY usando uma instrução INSERT condicional em várias tabelas.**

### INSERT ALL Condicional

As orientações para criar uma instrução INSERT ALL condicional estão especificadas no slide. A solução para esse problema está indicada na próxima página.

## INSERT ALL Condicional

```
INSERT ALL
  WHEN SAL > 10000 THEN
    INTO sal_history VALUES (EMPID, HIREDATE, SAL)
  WHEN MGR > 200 THEN
    INTO mgr_history VALUES (EMPID, MGR, SAL)
  SELECT employee_id EMPID, hire_date HIREDATE,
         salary SAL, manager_id MGR
  FROM   employees
  WHERE  employee_id > 200;
4 rows created.
```

23

### INSERT ALL Condicional (continuação)

O exemplo do slide é semelhante ao exemplo do slide anterior, pois ele insere linhas nas tabelas SAL\_HISTORY e MGR\_HISTORY. A instrução SELECT recupera, na tabela EMPLOYEES, os detalhes sobre o ID do funcionário, a data de admissão, o salário e o ID do gerente dos funcionários cujo ID é maior que 200. Os detalhes sobre o ID do funcionário, a data de admissão e o salário são inseridos na tabela SAL\_HISTORY. Os detalhes sobre o ID do funcionário, o ID do gerente e o salário são inseridos na tabela MGR\_HISTORY.

Essa instrução INSERT é denominada ALL INSERT condicional, pois são aplicadas outras restrições às linhas recuperadas pela instrução SELECT. Das linhas recuperadas pela instrução SELECT, apenas aquelas cujo valor na coluna SAL é maior que 10.000 são inseridas na tabela SAL\_HISTORY. Da mesma forma, apenas as linhas cujo valor na coluna MGR é maior que 200 são inseridas na tabela MGR\_HISTORY.

Observe que, diferentemente do exemplo anterior, no qual oito linhas foram inseridas nas tabelas, neste exemplo, apenas quatro linhas são inseridas.

É possível interpretar as 4 linhas criadas e retornadas como um total de quatro operações INSERT executadas nas tabelas-base, SAL\_HISTORY e MGR\_HISTORY.

## FIRST INSERT Condicional

- **Selecione DEPARTMENT\_ID, SUM(SALARY) e MAX(HIRE\_DATE) na tabela EMPLOYEES.**
- **Se o valor de SUM(SALARY) for maior que \$25.000, insira esse valor em SPECIAL\_SAL usando uma instrução FIRST INSERT condicional em várias tabelas.**
- **Se a primeira cláusula WHEN for avaliada como verdadeira, as cláusulas WHEN subsequentes relativas a essa linha deverão ser ignoradas.**
- **Insira as linhas que não atenderem à primeira condição WHEN na tabela HIREDATE\_HISTORY\_00, HIREDATE\_HISTORY\_99 ou HIREDATE\_HISTORY, com base no valor da coluna HIRE\_DATE usando uma instrução INSERT condicional em várias tabelas.**

### FIRST INSERT Condicional

As orientações para criar uma instrução FIRST INSERT condicional estão especificadas no slide. A solução para esse problema está indicada na próxima página.



## INSERT FIRST Condicional

```
INSERT FIRST
  WHEN SAL > 25000 THEN
    INTO special_sal VALUES (DEPTID, SAL)
  WHEN HIREDATE like ('%00%') THEN
    INTO hiredate_history_00 VALUES (DEPTID, HIREDATE)
  WHEN HIREDATE like ('%99%') THEN
    INTO hiredate_history_99 VALUES (DEPTID, HIREDATE)
  ELSE
    INTO hiredate_history VALUES (DEPTID, HIREDATE)
  SELECT department_id DEPTID, SUM(salary) SAL,
    MAX(hire_date) HIREDATE
  FROM employees
  GROUP BY department_id;
8 rows created.
```

25

### INSERT FIRST Condicional (continuação)

O exemplo do slide insere linhas em mais de uma tabela, usando uma única instrução INSERT. A instrução SELECT recupera os detalhes sobre o ID, o salário total e a data de admissão máxima relativos a todos os departamentos da tabela EMPLOYEES.

Essa instrução INSERT é denominada FIRST INSERT condicional, pois é feita uma exceção para os departamentos cujo salário total é maior que \$25.000. A condição WHEN ALL > \$25.000 é avaliada primeiro. Se o salário total de um departamento for maior que \$25.000, o registro será inserido na tabela SPECIAL\_SAL independentemente da data de admissão. Se a primeira cláusula WHEN for avaliada como verdadeira, o servidor Oracle executará a cláusula INTO correspondente e ignorará as cláusulas WHEN subsequentes relativas a essa linha.

Quando as linhas não atendem à primeira condição WHEN (WHEN SAL > 25.000), as outras condições são avaliadas exatamente como a instrução INSERT condicional, e os registros recuperados pela instrução SELECT são inseridos na tabela HIREDATE\_HISTORY\_00, HIREDATE\_HISTORY\_99 ou HIREDATE\_HISTORY, com base no valor da coluna HIREDATE.

É possível interpretar as 8 linhas criadas e retornadas como um total de oito operações INSERT executadas nas tabelas-base, SPECIAL\_SAL,

HIREDATE\_HISTORY\_00, HIREDATE\_HISTORY\_99 e HIREDATE\_HISTORY.

## INSERT de Criação de Pivô

- Suponha que você receba um conjunto de registros de vendas de uma tabela de banco de dados não relacional, `SALES_SOURCE_DATA`, no seguinte formato:  
`EMPLOYEE_ID, WEEK_ID, SALES_MON, SALES_TUE, SALES_WED, SALES_THUR, SALES_FRI`
- Você quer armazenar esses registros na tabela `SALES_INFO` em um formato relacional mais usado:  
`EMPLOYEE_ID, WEEK, SALES`
- Com uma instrução `INSERT` de criação de pivô, converta o conjunto de registros de vendas da tabela de banco de dados não relacional em um formato relacional.

26

### INSERT de Criação de Pivô

A criação de pivô é uma operação na qual você precisa criar uma transformação de forma que cada registro de qualquer fluxo de entrada, como uma tabela de banco de dados não relacional, seja convertido em vários registros para um ambiente de tabela de banco de dados mais relacional.

Para solucionar o problema mencionado no slide, é preciso criar uma transformação para que cada registro da tabela de banco de dados não relacional original, `SALES_SOURCE_DATA`, seja convertido em cinco registros para a tabela `SALES_INFO` de data warehouse. Essa operação é geralmente chamada de *criação de pivô*.

As orientações para desenvolver uma instrução `INSERT` de criação de pivô estão especificadas no slide. A solução para esse problema está indicada na próxima página.

## INSERT de Criação de Pivô

```

INSERT ALL
  INTO sales_info VALUES (employee_id, week_id, sales_MON)
  INTO sales_info VALUES (employee_id, week_id, sales_TUE)
  INTO sales_info VALUES (employee_id, week_id, sales_WED)
  INTO sales_info VALUES (employee_id, week_id, sales_THUR)
  INTO sales_info VALUES (employee_id, week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
       sales_WED, sales_THUR, sales_FRI
FROM sales_source_data;
5 rows created.

```

27

### INSERT de Criação de Pivô (continuação)

No exemplo do slide, os dados de vendas, relativos aos detalhes das vendas realizadas por um representante de vendas em cada dia de uma semana com um ID de semana específico, são recebidos da tabela de banco de dados não relacional SALES\_SOURCE\_DATA.

```
DESC 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)

## A Instrução MERGE

- **Permite atualizar ou inserir dados de forma condicional em uma tabela de banco de dados**
- **Executa uma operação UPDATE se a linha existir e uma operação INSERT se a linha for nova**
  - **Evita atualizações separadas**
  - **Melhora o desempenho e facilita o uso**
  - **É útil nas aplicações de data warehouse**

### Instruções MERGE

O servidor Oracle suporta a instrução MERGE para as operações INSERT, UPDATE e DELETE. Ao usar essa instrução, você pode atualizar, inserir ou deletar uma linha de forma condicional em uma tabela, evitando, assim, várias instruções DML. A decisão de efetuar uma atualização, inserção ou deleção na tabela de destino baseia-se na condição na cláusula ON.

Você precisa ter privilégios de objeto INSERT e UPDATE na tabela de destino e o privilégio de objeto SELECT na tabela de origem. Para especificar a cláusula DELETE de merge\_update\_clause, é preciso ter o privilégio de objeto DELETE na tabela de destino.

A instrução MERGE é determinante. Não é possível atualizar a mesma linha da tabela de destino várias vezes na mesma instrução MERGE.

Uma abordagem alternativa é usar loops PL/SQL e várias instruções DML. No entanto, a instrução MERGE é fácil de usar e é expressa de forma mais simples como uma única instrução SQL.

A instrução MERGE é apropriada para várias aplicações de data warehouse. Por exemplo, em uma aplicação de data warehouse, talvez seja necessário trabalhar com dados

provenientes de várias origens, alguns dos quais podem ser duplicados. Com a instrução `MERGE`, é possível adicionar ou modificar linhas de forma condicional.

## A Sintaxe da Instrução MERGE

**É possível inserir ou atualizar as linhas de uma tabela de forma condicional usando a instrução MERGE.**

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
      col1 = col_val1,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

30

### Intercalando Linhas

É possível atualizar linhas existentes e inserir novas linhas de forma condicional usando a instrução MERGE.

Na sintaxe:

Cláusula INTO	especifica a tabela de destino para atualização ou inserção
Cláusula USING	identifica a origem dos dados a serem atualizados ou inseridos; pode ser uma tabela, uma view ou uma subconsulta
Cláusula ON	a condição com base na qual a operação MERGE efetua a atualização ou inserção
WHEN MATCHED	instrui o servidor sobre como responder aos resultados da condição de join
WHEN NOT MATCHED	

Para obter mais informações, consulte o item "MERGE" do manual *Oracle Database 11g SQL Reference*.

## Intercalando Linhas

**Insira ou atualize linhas da tabela EMPL3 para corresponder à tabela EMPLOYEES.**

```

MERGE INTO empl3 c
  USING employees e
  ON (c.employee_id = e.employee_id)
  WHEN MATCHED THEN
    UPDATE SET
      c.first_name      = e.first_name,
      c.last_name       = e.last_name,
      ...
      c.department_id   = e.department_id
  WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,
                  e.email, e.phone_number, e.hire_date, e.job_id,
                  e.salary, e.commission_pct, e.manager_id,
                  e.department_id);

```

31

### Exemplo de Intercalação de Linhas

```

MERGE INTO empl3 c
  USING employees e
  ON (c.employee_id = e.employee_id)
  WHEN MATCHED THEN
    UPDATE SET
      c.first_name      = e.first_name,
      c.last_name       = e.last_name,
      c.email           = e.email,
      c.phone_number    = e.phone_number,
      c.hire_date       = e.hire_date,
      c.job_id          = e.job_id,
      c.salary          = e.salary,
      c.commission_pct  = e.commission_pct,
      c.manager_id      = e.manager_id,
      c.department_id   = e.department_id
  WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,
                  e.email, e.phone_number, e.hire_date, e.job_id,
                  e.salary, e.commission_pct, e.manager_id,

```



```
e.department_id);
```

## Intercalando Linhas

```
TRUNCATE TABLE empl3;
```

```
SELECT *  
FROM empl3;  
no rows selected
```

```
MERGE INTO empl3 c  
  USING employees e  
  ON (c.employee_id = e.employee_id)  
WHEN MATCHED THEN  
  UPDATE SET  
    ...  
WHEN NOT MATCHED THEN  
  INSERT VALUES...;
```

```
SELECT *  
FROM empl3;  
20 rows selected.
```

32

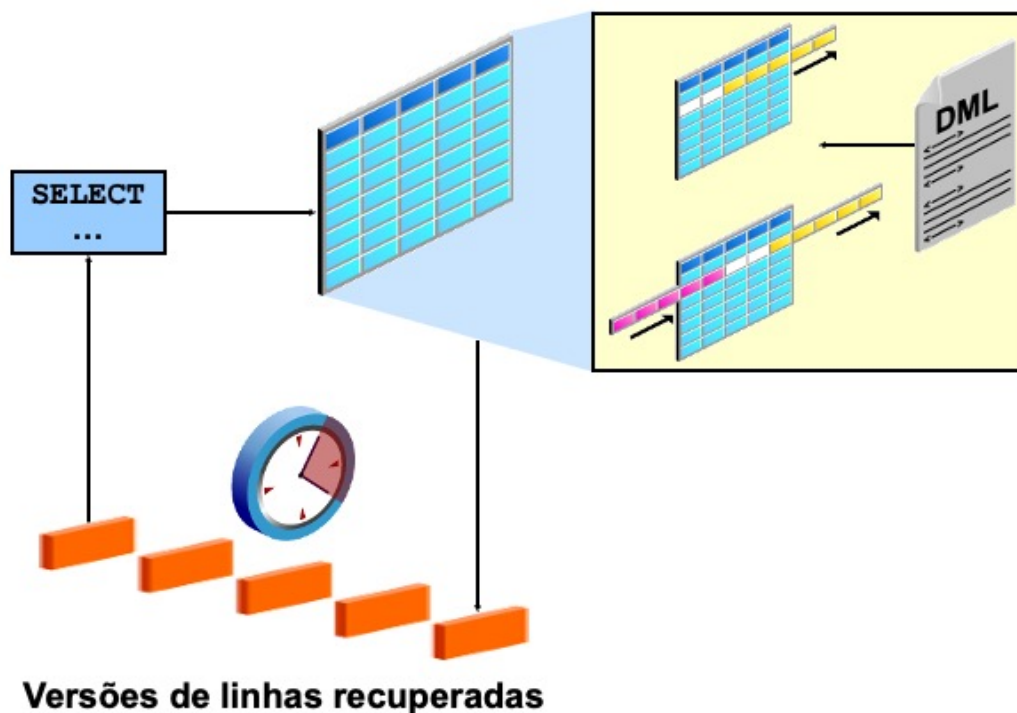
### Exemplo de Intercalação de Linhas (continuação)

O exemplo do slide estabelece a correspondência entre `EMPLOYEE_ID` da tabela `EMPL3` e `EMPLOYEE_ID` da tabela `EMPLOYEES`. Caso seja encontrada uma correspondência, a linha da tabela `EMPL3` será atualizada para corresponder à linha da tabela `EMPLOYEES`. Se não for encontrada, a linha será inserida na tabela `EMPL3`.

A condição `c.employee_id = e.employee_id` será avaliada. Como a tabela `EMPL3` está vazia, a condição retorna `FALSE`, indicando que não há correspondências. A lógica corresponde à cláusula `WHEN NOT MATCHED`, e o comando `MERGE` insere as linhas da tabela `EMPLOYEES` na tabela `EMPL3`.

Se houver linhas na tabela `EMPL3`, e os IDs dos funcionários corresponderem nas duas tabelas (`EMPL3` e `EMPLOYEES`), as linhas existentes na tabela `EMPL3` serão atualizadas para corresponderem à tabela `EMPLOYEES`.

## Controlando Alterações nos Dados



## Controlando Alterações nos Dados

Você poderá perceber que, de alguma maneira, os dados de uma tabela foram alterados de forma inadequada. Para pesquisar isso, é possível usar várias consultas de flashback para exibir dados das linhas em momentos específicos. Com mais eficiência, é possível usar o recurso Flashback de Consulta de Versão para exibir todas as alterações feitas em uma linha durante um período. Esse recurso permite que você anexe a cláusula `VERSIONS` a uma instrução `SELECT` que especifique um SCN ou uma faixa de timestamp dentro da qual deseja exibir as alterações feitas nos valores das linhas. A consulta também pode retornar metadados associados, tais como a transação responsável pela alteração.

Além disso, após identificar uma transação errada, você poderá usar o recurso Flashback de Consulta de Transação para identificar outras alterações feitas por essa transação. Em seguida, você poderá usar o recurso Flashback de Tabela para restaurar a tabela até um estado anterior às alterações.

É possível consultar uma tabela com a cláusula `VERSIONS` para produzir todas as versões de todas as linhas que existem ou que já existiram entre o momento da consulta e o momento da execução do parâmetro `undo retention`, segundos antes do momento atual. `undo retention` é um parâmetro de inicialização auto-ajustável. A consulta que inclui uma cláusula `VERSIONS` denomina-se consulta de versão. Os resultados de uma consulta de versão se comportam como se a cláusula `WHERE` fosse aplicada às versões das linhas. A consulta de versão retorna versões das linhas apenas durante as transações.

**SCN (número de alteração do sistema):** O servidor Oracle atribui um SCN (System Change Number) para identificar os registros de redo para cada transação submetida a commit.

## Exemplo de Flashback de Consulta de Versão

```
SELECT salary FROM employees3
WHERE employee_id = 107;
```

1

SALARY
4200

```
UPDATE employees3 SET salary = salary * 1.30
WHERE employee_id = 107;
```

```
COMMIT;
```

2

```
SELECT salary FROM employees3
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 107;
```

3

SALARY
5460
4200

34

### Exemplo de Flashback de Consulta de Versão

No exemplo do slide, o salário do funcionário 107 é recuperado (1). O salário do funcionário 107 é aumentado em 30%, e essa alteração é submetida a commit (2). São exibidas as diferentes versões de salário (3).

A cláusula `VERSIONS` não altera o plano da consulta. Por exemplo, se você executar uma consulta para uma tabela que usa o método de acesso por índice, a mesma consulta na mesma tabela com uma cláusula `VERSIONS` continuará usando o método de acesso por índice. As versões de linhas retornadas pela consulta são as versões das linhas durante as transações. A cláusula `VERSIONS` não tem efeito sobre o comportamento transacional de uma consulta. Isso significa que uma consulta a uma tabela com a cláusula `VERSIONS` também herda o ambiente de consulta da transação em andamento.

A cláusula `VERSIONS` default pode ser especificada como `VERSIONS BETWEEN {SCN|TIMESTAMP} MINVALUE AND MAXVALUE`.

A cláusula `VERSIONS` é uma extensão SQL apenas para consultas. É possível ter operações DML e DDL que usam uma cláusula `VERSIONS` dentro de subconsultas. A consulta de versão da linha recupera todas as versões submetidas a commit das linhas selecionadas. As alterações feitas pela transação ativa atual não são retornadas. A consulta de versão recupera todas as versões de linhas. Isso significa, essencialmente, que as versões retornadas incluem

versões de linhas deletadas e subsequentemente reinseridas.

## A Cláusula VERSIONS BETWEEN

```
SELECT versions_starttime "START_DATE",
       versions_endtime   "END_DATE",
       salary
FROM   employees
       VERSIONS BETWEEN SCN MINVALUE
       AND MAXVALUE
WHERE  last_name = 'Lorentz';
```

START_DATE	END_DATE	SALARY
13-FEB-04 11.16.41 AM		5460
	13-FEB-04 11.16.41 AM	4200

### A Cláusula VERSIONS BETWEEN

Você pode usar a cláusula `VERSIONS BETWEEN` para recuperar todas as versões das linhas que existem ou que já existiram entre o momento da consulta e um momento passado.

Se o tempo de retenção de undo for menor que o limite inferior de tempo/SCN da cláusula `BETWEEN`, a consulta recuperará apenas as versões até o período de retenção de undo. O intervalo de tempo da cláusula `BETWEEN` pode ser especificado como um intervalo SCN ou como uma faixa de horários. Esse intervalo de tempo é definido pelos limites inferior e superior.

No exemplo, as alterações do salário de Lorentz são recuperadas. O valor nulo para `END_DATE` na primeira versão indica que esta era a versão existente no momento da consulta. O valor nulo para `START_DATE` na última versão indica que essa versão foi criada em um momento anterior ao tempo de retenção de undo.

## Sumário

**Nesta lição, você aprendeu a:**

- **Usar instruções DML e controlar transações**
- **Descrever os recursos de inserções em várias tabelas**
- **Usar os seguintes tipos de inserções em várias tabelas:**
  - **INSERT Incondicional**
  - **INSERT de Criação de Pivô**
  - **ALL INSERT Condicional**
  - **FIRST INSERT Condicional**
- **Intercalar linhas em uma tabela**
- **Manipular dados usando subconsultas**
- **Controlar as alterações de dados durante um período**

**37**

## Sumário

Nesta lição, você aprendeu a manipular os dados do banco de dados Oracle usando subconsultas. Você também conheceu as instruções `INSERT` em várias tabelas e a instrução `MERGE`, além de aprender a controlar as alterações feitas no banco de dados.



## Exercício 15: Visão Geral

**Este exercício aborda os seguintes tópicos:**

- Executando INSERTs em várias tabelas
- Executando operações MERGE
- Controlando versões de linhas

### Exercício 15: Visão Geral

Neste exercício, você adiciona linhas à tabela `emp_data`, atualiza e deleta dados da tabela e controla suas transações.