4

Manipulating Large Data Sets

Objectives

After completing this lesson, you should be able to do the following:

- Manipulate data by using subqueries
- Specify explicit default values in the `INSERT` and `UPDATE` statements
- Describe the features of multitable `INSERTs`
- Use the following types of multitable `INSERTs`:
  - Unconditional `INSERT`
  - Pivoting `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
- Merge rows in a table
- Track the changes to data over a period of time

Lesson Agenda

- Manipulating data by using subqueries
- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERTs`:
  - Unconditional `INSERT`
  - Pivoting `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
- Merging rows in a table
- Tracking the changes to data over a period of time

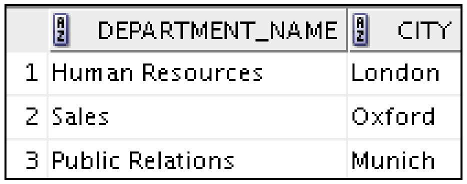
Using Subqueries to Manipulate Data

You can use subqueries in data manipulation language (DML) statements to:

- Retrieve data by using an inline view
- Copy data from one table to another
- Update data in one table based on the values of another table
- Delete rows from one table based on rows in another table

Retrieving Data by Using a Subquery as Source

```
SELECT department_name, city
FROM   departments
NATURAL JOIN (SELECT l.location_id, l.city, l.country_id
              FROM   loc l
              JOIN   countries c
              ON(l.country_id = c.country_id)
              JOIN regions USING(region_id)
              WHERE region_name = 'Europe');
```



	DEPARTMENT_NAME	CITY
1	Human Resources	London
2	Sales	Oxford
3	Public Relations	Munich

Inserting by Using a Subquery as a Target

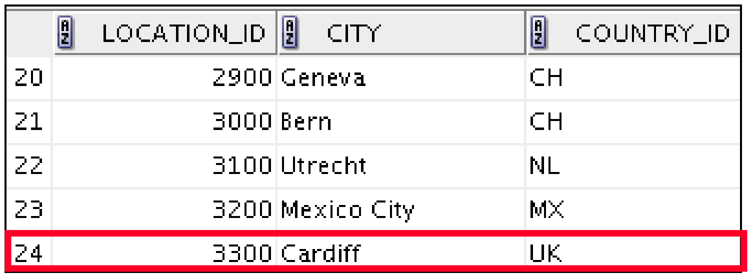
```
INSERT INTO (SELECT l.location_id, l.city, l.country_id
             FROM   locations l
             JOIN   countries c
             ON(l.country_id = c.country_id)
             JOIN regions USING(region_id)
             WHERE region_name = 'Europe')
VALUES (3300, 'Cardiff', 'UK');
```

```
1 rows inserted
```

Inserting by Using a Subquery as a Target

Verify the results.

```
SELECT location_id, city, country_id
FROM   loc
```

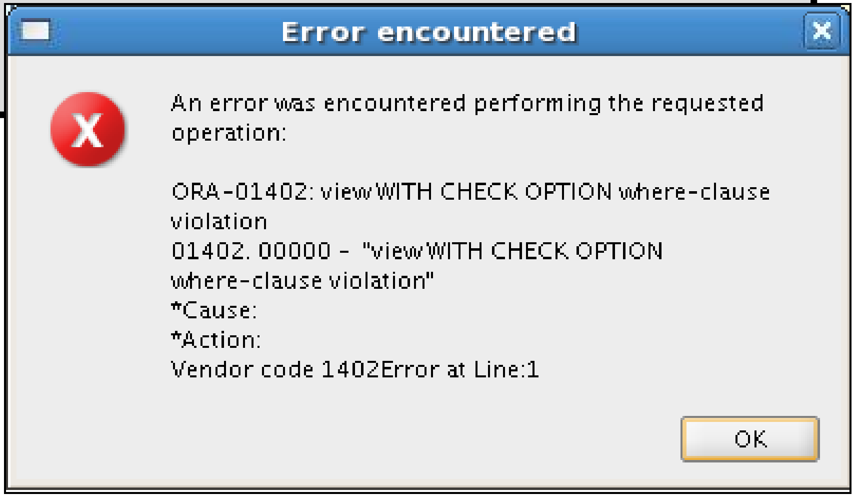


	LOCATION_ID	CITY	COUNTRY_ID
20	2900	Geneva	CH
21	3000	Bern	CH
22	3100	Utrecht	NL
23	3200	Mexico City	MX
24	3300	Cardiff	UK

Using the `WITH CHECK OPTION` Keyword on DML Statements

The `WITH CHECK OPTION` keyword prohibits you from changing rows that are not in the subquery.

```
INSERT INTO ( SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
              (SELECT country_id
               FROM countries
               NATURAL JOIN regions
               WHERE region_name = 'Europe')
               WITH CHECK OPTION )
VALUES (3600, 'Washington', 'US');
```


Error encountered

An error was encountered performing the requested operation:

```
ORA-01402: view WITH CHECK OPTION where-clause
violation
01402. 00000 - "view WITH CHECK OPTION
where-clause violation"
*Cause:
*Action:
Vendor code 1402Error at Line:1
```

Lesson Agenda

- Manipulating data by using subqueries
- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERTs`:
  – Unconditional `INSERT`
  – Pivoting `INSERT`
  – Conditional `INSERT ALL`
  – Conditional `INSERT FIRST`
- Merging rows in a table
- Tracking the changes to data over a period of time

Overview of the Explicit Default Feature

- Use the `DEFAULT` keyword as a column value where the default column value is desired.
- This allows the user to control where and when the default value should be applied to data.
- Explicit defaults can be used in `INSERT` and `UPDATE` statements.

Using Explicit Default Values

- `DEFAULT` with `INSERT`:

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

- `DEFAULT` with `UPDATE`:

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

Copying Rows from Another Table

- Write your `INSERT` statement with a subquery.

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

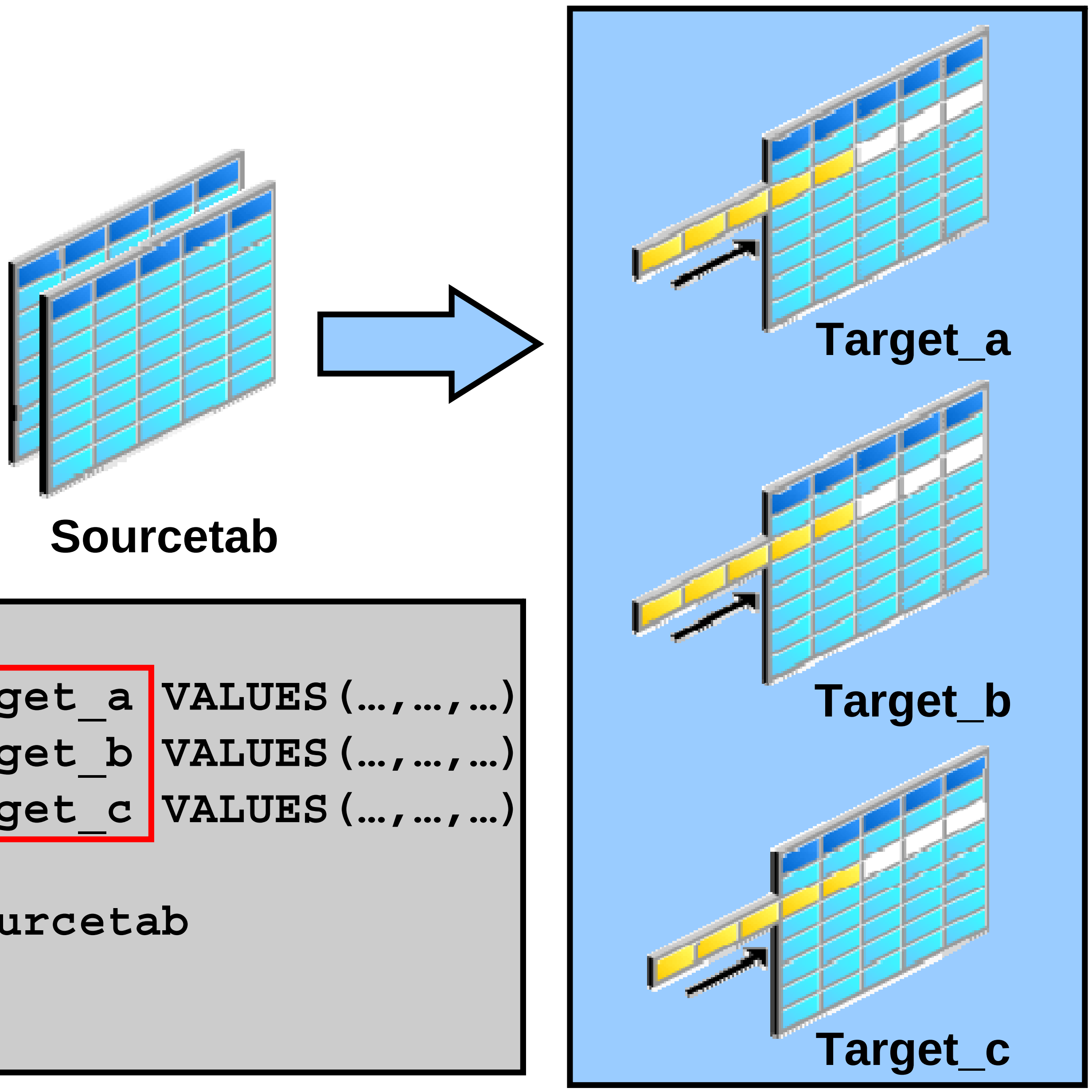
```
33 rows inserted
```

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause with that in the subquery.

Lesson Agenda

- Manipulating data by using subqueries
- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERTs`:
  - Unconditional `INSERT`
  - Pivoting `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
- Merging rows in a table
- Tracking the changes to data over a period of time

Overview of Multitable `INSERT` Statements



```
INSERT  ALL
   INTO  target_a  VALUES(…,…,…)
   INTO  target_b  VALUES(…,…,…)
   INTO  target_c  VALUES(…,…,…)
   SELECT …
   FROM   sourcetab
   WHERE …;
```

Subquery

Overview of Multitable `INSERT` Statements

- Use the `INSERT…SELECT` statement to insert rows into multiple tables as part of a single DML statement.
- Multitable `INSERT` statements are used in data warehousing systems to transfer data from one or more operational sources to a set of target tables.
- They provide significant performance improvement over:
  - Single DML versus multiple `INSERT…SELECT` statements
  - Single DML versus a procedure to perform multiple inserts by using the `IF...THEN` syntax

Types of Multitable `INSERT` Statements

The different types of multitable `INSERT` statements are:

- Unconditional `INSERT`
- Conditional `INSERT ALL`
- Pivoting `INSERT`
- Conditional `INSERT FIRST`

Multitable `INSERT` Statements

- Syntax for multitable `INSERT`:

```
INSERT [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]
```

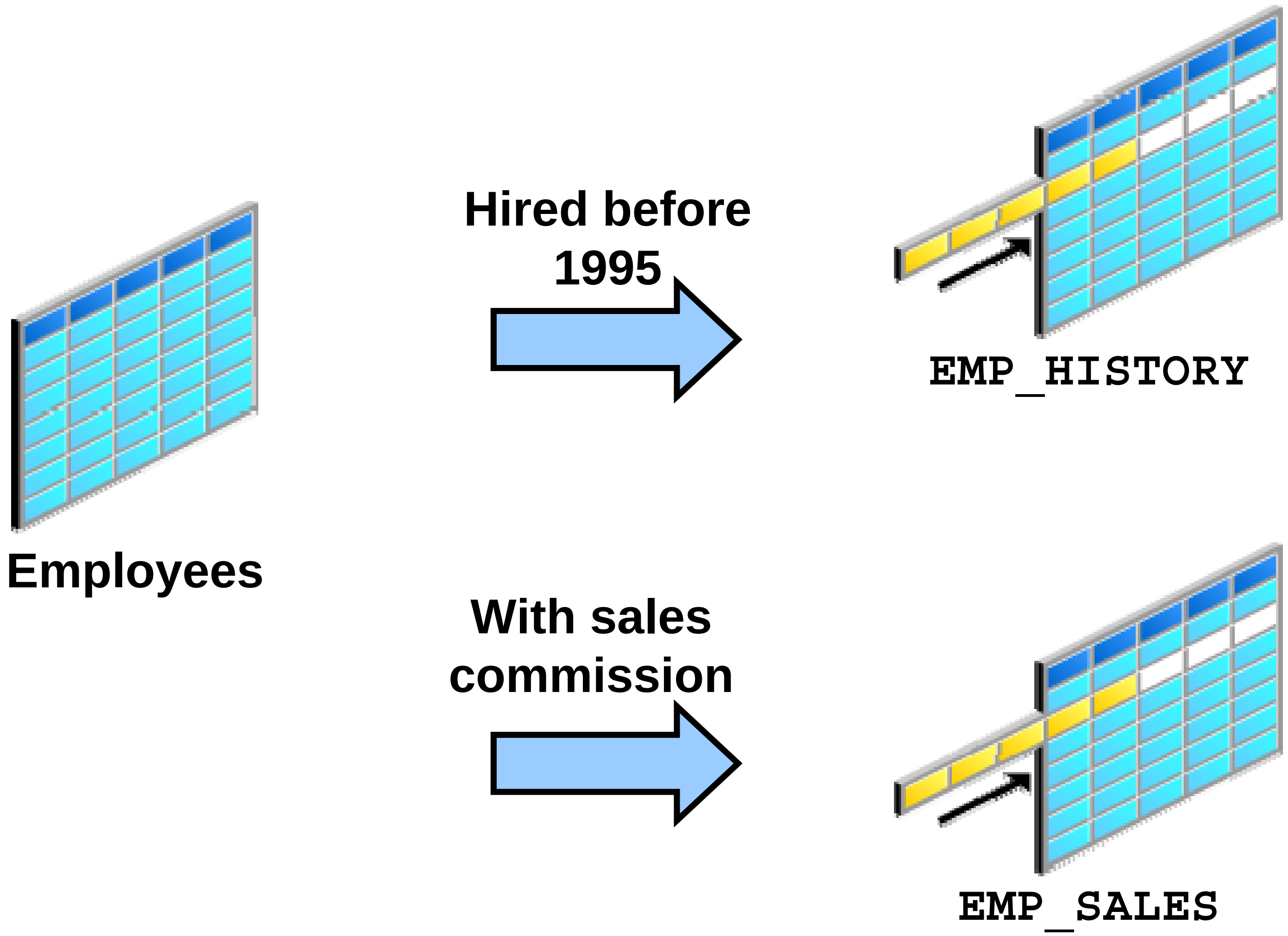
Unconditional INSERT ALL

- Select the EMPLOYEE_ID, HIRE_DATE, SALARY, and MANAGER_ID values from the EMPLOYEES table for those employees whose EMPLOYEE_ID is greater than 200.
- Insert these values into the SAL_HISTORY and MGR_HISTORY tables by using a multitable INSERT.

```
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;
```

```
12 rows inserted
```

Conditional `INSERT ALL`: Example



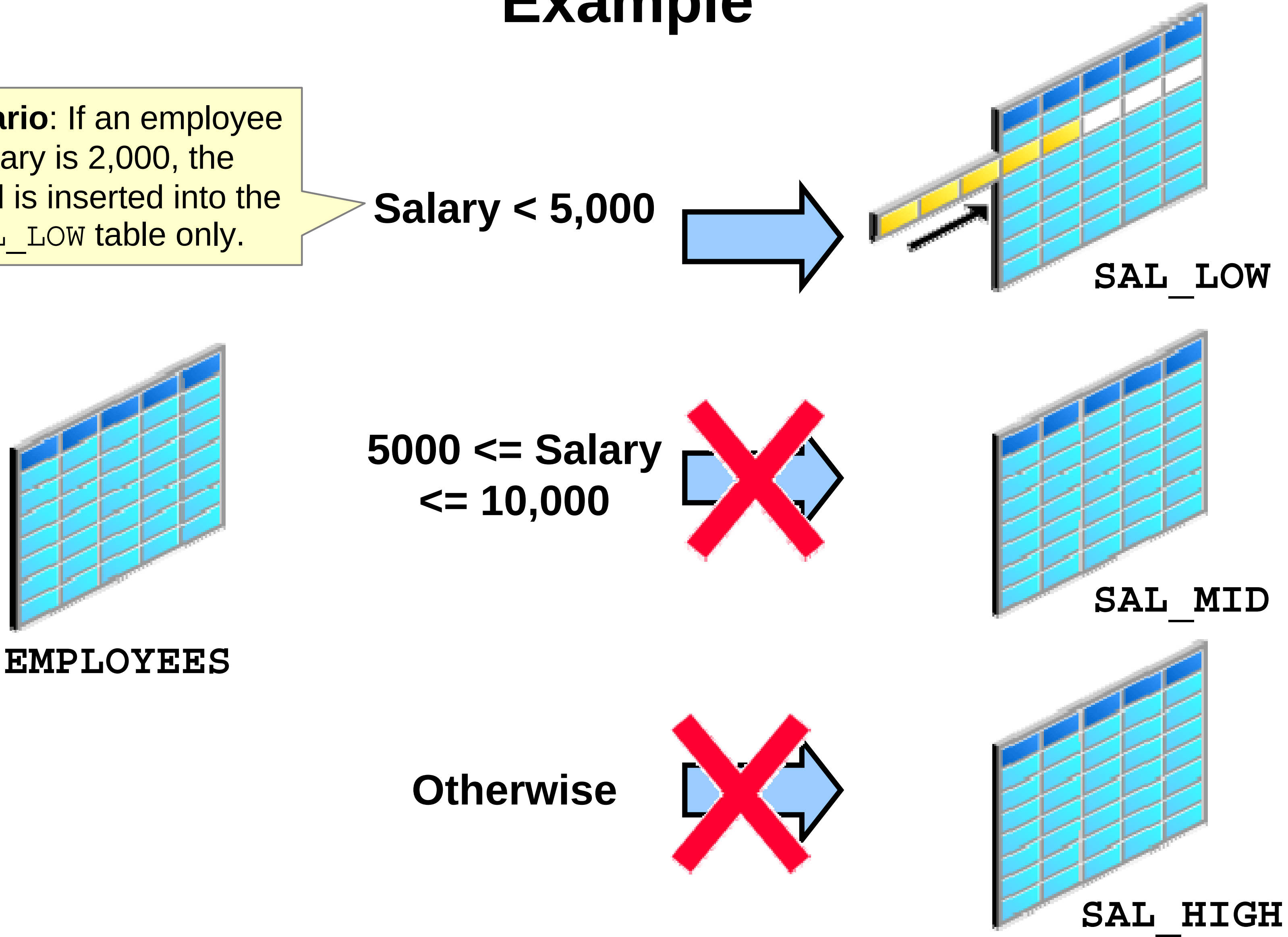
Conditional `INSERT ALL`

```
INSERT  ALL

 WHEN HIREDATE < '01-JAN-95' THEN

   INTO emp_history VALUES(EMPID,HIREDATE,SAL)

 WHEN COMM IS NOT NULL THEN

   INTO emp_sales VALUES(EMPID,COMM,SAL)

   SELECT employee_id EMPID, hire_date HIREDATE,

          salary SAL, commission_pct COMM

   FROM  employees
```

```
48 rows inserted
```

Conditional `INSERT FIRST`: Example

Scenario: If an employee salary is 2,000, the record is inserted into the `SAL_LOW` table only.



Conditional `INSERT FIRST`

```
INSERT FIRST
WHEN salary < 5000 THEN
  INTO sal_low VALUES (employee_id, last_name, salary)
WHEN salary between 5000 and 10000 THEN
  INTO sal_mid VALUES (employee_id, last_name, salary)
ELSE
  INTO sal_high VALUES (employee_id, last_name, salary)
SELECT employee_id, last_name, salary
FROM employees
```

```
107 rows inserted
```

Pivoting `INSERT`

Convert the set of sales records from the nonrelational database table to relational format.

Emp_ID	Week_ID	MON	TUES	WED	THUR	FRI
176	6	2000	3000	4000	5000	6000

Employee_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000

Pivoting `INSERT`

```
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 inserted
```

Lesson Agenda

- Manipulating data by using subqueries
- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Pivoting `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
- Merging rows in a table
- Tracking the changes to data over a period of time

`MERGE` Statement

- Provides the ability to conditionally update, insert, or delete data into a database table
- Performs an `UPDATE` if the row exists, and an `INSERT` if it is a new row:
  - Avoids separate updates
  - Increases performance and ease of use
  - Is useful in data warehousing applications

MERGE Statement Syntax

You can conditionally insert, update, or delete rows in a table by using the MERGE statement.

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
    col1 = col1_val,
    col2 = col2_val
 WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

Merging Rows: Example

Insert or update rows in the COPY_EMP3 table to match the EMPLOYEES table.

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM 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,
...
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
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);
```

Merging Rows: Example

```
TRUNCATE TABLE copy_emp3;
SELECT * FROM copy_emp3;
0 rows selected
```

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM 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,
...
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, ...
```

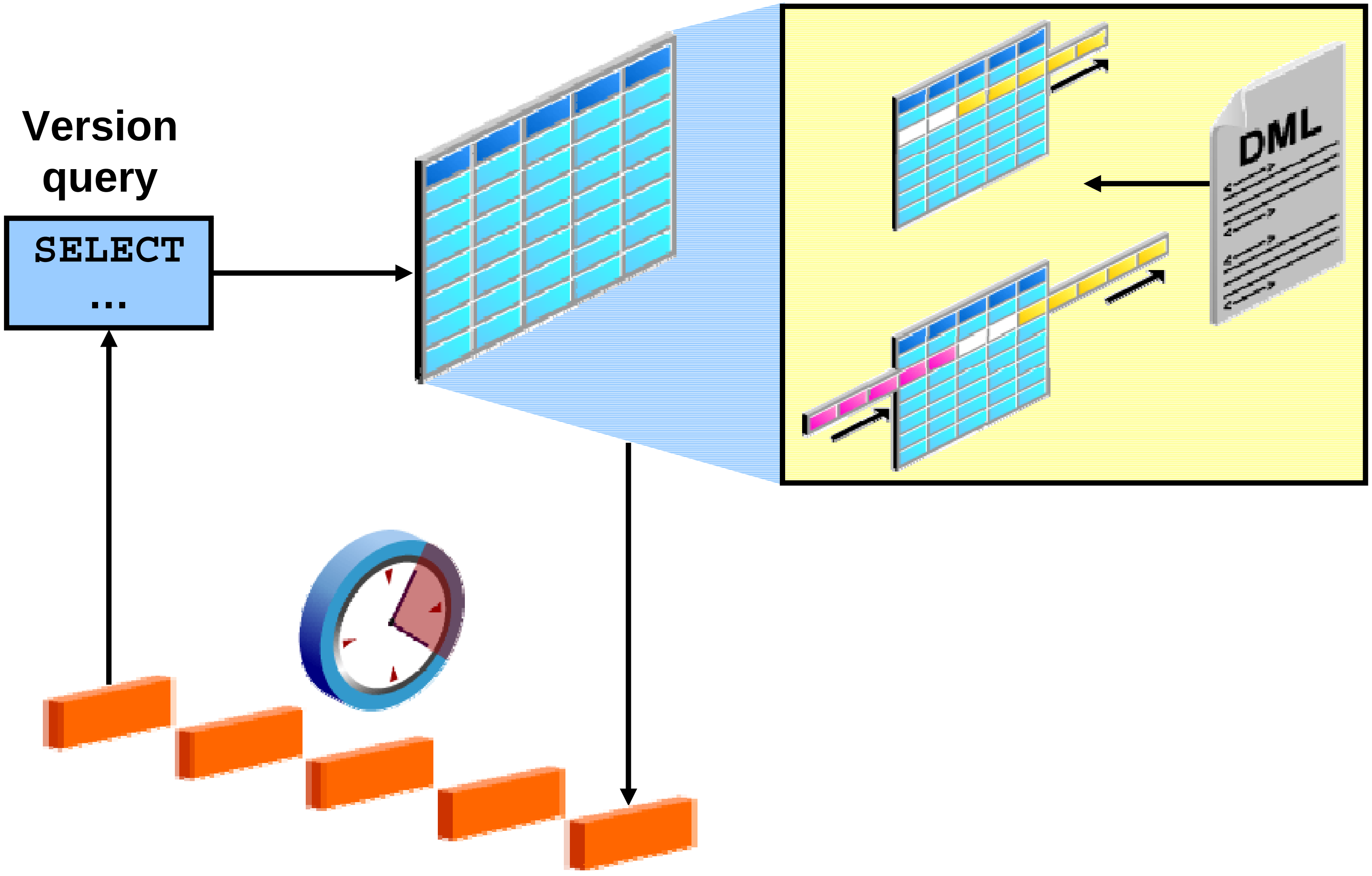
```
SELECT * FROM copy_emp3;
107 rows selected.
```

ORACLE

Lesson Agenda

- Manipulating data by using subqueries
- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERTs`:
  - Unconditional `INSERT`
  - Pivoting `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
- Merging rows in a table
- Tracking the changes to data over a period of time

Tracking Changes in Data



Versions of retrieved rows

Example of the Flashback Version Query

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

1

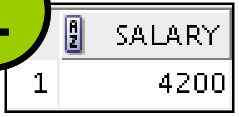
```
UPDATE employees3 SET salary = salary * 1.30
WHERE  employee_id = 107;

COMMIT;
```

2

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

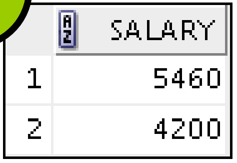
3

1



	SALARY
1	4200

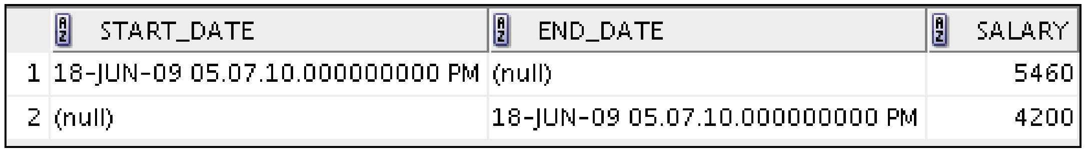
3



	SALARY
1	5460
2	4200

VERSIONS BETWEEN Clause

```
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
1	18-JUN-09 05.07.10.000000000 PM	(null)	5460
2	(null)	18-JUN-09 05.07.10.000000000 PM	4200

Quiz

When you use the `INSERT` or `UPDATE` command, the `DEFAULT` keyword saves you from hard-coding the default value in your programs or querying the dictionary to find it.

1. True
2. False

Summary

In this lesson, you should have learned how to:

- Use DML statements and control transactions
- Describe the features of multitable `INSERTs`
- Use the following types of multitable `INSERTs`:
  - Unconditional `INSERT`
  - Pivoting `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
- Merge rows in a table
- Manipulate data by using subqueries
- Track the changes to data over a period of time

Practice 4: Overview

This practice covers the following topics:

- Performing multitable `INSERT`s
- Performing `MERGE` operations
- Tracking row versions