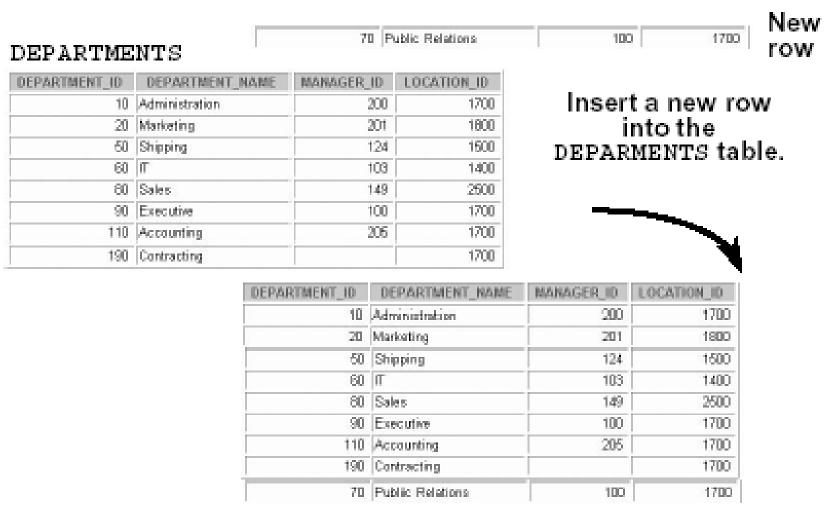
MANIPULATING DATA

Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A transaction consists of a collection of DML statements that form a logical unit of work.

Adding a New Row to a Table


New row
DEPARTMENTS
Insert a new row into the DEPARMENTS table.

70	Public Relations	100	1700

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

The `INSERT` Statement Syntax

- Add new rows to a table by using the `INSERT` statement.

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

- Only one row is inserted at a time with this syntax.

Adding a New Row to a Table (continued)

You can add new rows to a table by issuing the `INSERT` statement.

In the syntax:

table	is the name of the table
column	is the name of the column in the table to populate
value	is the corresponding value for the column

Note: This statement with the `VALUES` clause adds only one row at a time to a table.

Inserting New Rows

- **Insert a new row containing values for each column.**
- **List values in the default order of the columns in the table.**
- **Optionally, list the columns in the `INSERT` clause.**

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Public Relations', 100, 1700);
1 row created.
```

- **Enclose character and date values within single quotation marks.**

For clarity, use the column list in the `INSERT` clause.
Enclose character and date values within single quotation marks; it is not recommended to enclose numeric values within single quotation marks.

Number values should not be enclosed in single quotes, because implicit conversion may take place for numeric values assigned to `NUMBER` data type columns if single quotes are included.

Inserting Rows with Null Values

- **Implicit method: Omit the column from the column list.**

```
INSERT INTO   departments (department_id,
                           department_name    )
VALUES        (30, 'Purchasing');
1 row created.
```

- **Explicit method: Specify the `NULL` keyword in the `VALUES` clause.**

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 row created.
```

Methods for Inserting Null Values

Method	**Description**
Implicit	Omit the column from the column list.
Explicit	Specify the `NULL` keyword in the `VALUES` list, specify the empty string (`''`) in the `VALUES` list for character strings and dates.

Be sure that you can use null values in the targeted column by verifying the `Null?` status with the *i*SQL*Plus `DESCRIBE` command.

The Oracle Server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input:

- Mandatory value missing for a `NOT NULL` column
- Duplicate value violates uniqueness constraint
- Foreign key constraint violated
- `CHECK` constraint violated
- Data type mismatch
- Value too wide to fit in column

Inserting Special Values

The `SYSDATE` function records the current date and time.

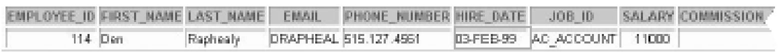
```
INSERT INTO employees (employee_id,
                 first_name, last_name,
                 email, phone_number,
                 hire_date, job_id, salary,
                 commission_pct, manager_id,
                 department_id)
VALUES           (113,
                 'Louis', 'Popp',
                 'LPOPP', '515.124.4567',
                 SYSDATE, 'AC_ACCOUNT', 6900,
                 NULL, 205, 100);
1 row created.
```

Inserting Specific Date Values

- **Add a new employee.**

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
            TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- **Verify your addition.**

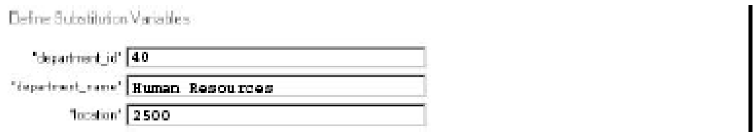


EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

Creating a Script

- **Use `&` substitution in a SQL statement to prompt for values.**
- **`&` is a placeholder for the variable value.**

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```



Define Substitution Variables

"department_id"	40
"department_name"	Human Resources
"location"	2500

```
1 row created.
```

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%';
4 rows created.
```

- **Do not use the `VALUES` clause.**
- **Match the number of columns in the `INSERT` clause to those in the subquery.**

Copying Rows from Another Table

You can use the INSERT statement to add rows to a table where the values are derived from existing tables. In place of the VALUES clause, you use a subquery.

Syntax

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

In the syntax:

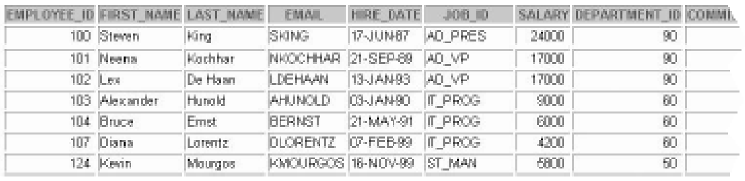
table	is the table name
column	is the name of the column in the table to populate
subquery	is the subquery that returns rows into the table

The number of columns and their data types in the column list of the INSERT clause must match the number of values and their data types in the subquery. To create a copy of the rows of a table, use SELECT * in the subquery.

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

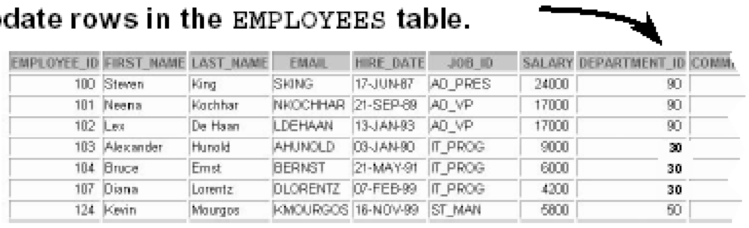
Changing Data in a Table

EMPLOYEES



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMM
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the `EMPLOYEES` table.



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMM
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	**30**	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	**30**	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	**30**	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

The graphic in the slide illustrates changing the department number for employees in department 60 to department 30.

The `UPDATE` Statement Syntax

- **Modify existing rows with the `UPDATE` statement.**

```
UPDATE        table
SET           column = value [, column = value, ...]
[WHERE        condition];
```

- **Update more than one row at a time, if required.**

Updating Rows

You can modify existing rows by using the `UPDATE` statement.

In the syntax:

table	is the name of the table
column	is the name of the column in the table to populate
value	is the corresponding value or subquery for the column
condition	identifies the rows to be updated and is composed of column names expressions, constants, subqueries, and comparison operators

Confirm the update operation by querying the table to display the updated rows.

For more information, see *Oracle9i SQL Reference*, "UPDATE."

Note: In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the `EMPLOYEES` table by name is dangerous, because more than one employee may have the same name.

Updating Rows in a Table

- **Specific row or rows are modified if you specify the `WHERE` clause.**

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

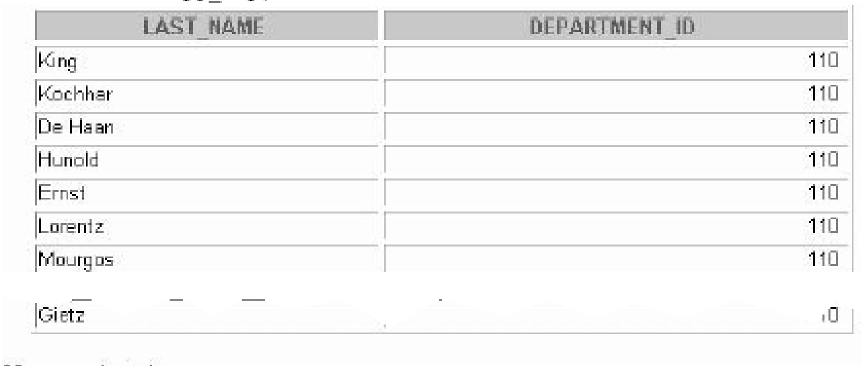
- **All rows in the table are modified if you omit the `WHERE` clause.**

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```

The `UPDATE` statement modifies specific rows if the `WHERE` clause is specified. The example in the slide transfers employee 113 (Popp) to department 70.

If you omit the `WHERE` clause, all the rows in the table are modified.

```
SELECT last_name, department_id
FROM   copy_emp;
```



LAST_NAME	DEPARTMENT_ID
King	110
Kochhar	110
De Haan	110
Hunold	110
Ernst	110
Lorentz	110
Mourgos	110
Gietz	ı0

22 rows selected.

Note: The `COPY_EMP` table has the same data as the `EMPLOYEES` table.

Updating Two Columns with a Subquery

Update employee 114's job and department to match that of employee 205.

```
UPDATE   employees
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.
```

You can update multiple columns in the SET clause of an UPDATE statement by writing multiple subqueries.

Syntax

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

Note: If no rows are updated, a message "0 rows updated." is returned.

Updating Rows Based on Another Table

Use subqueries in UPDATE statements to update rows in a table based on values from another table.

```
UPDATE  copy_emp
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.
```

Updating Rows: Integrity Constraint Error

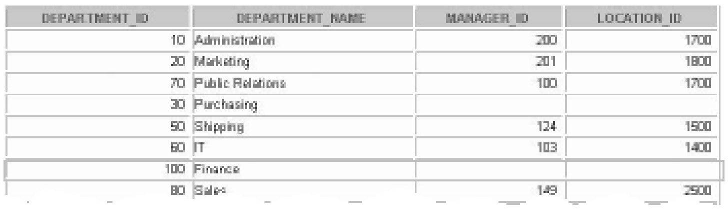
```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

Department number 55 does not exist

```
UPDATE employees
       *
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

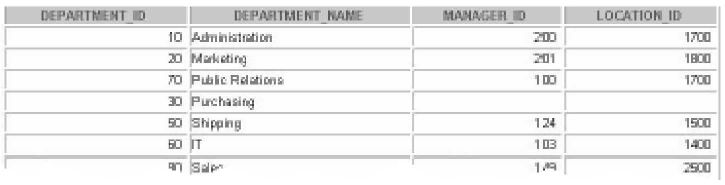
Removing a Row from a Table

DEPARTMENTS



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
100	Finance		
80	Sales	149	2500

Delete a row from the `DEPARTMENTS` table.



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

The graphic in the slide removes the Finance department from the `DEPARTMENTS` table (assuming that there are no constraints defined on the `DEPARTMENTS` table).

The `DELETE` Statement

You can remove existing rows from a table by using the `DELETE` statement.

```
DELETE [FROM]   table
[WHERE          condition];
```

You can remove existing rows by using the `DELETE` statement.

In the syntax:

table	is the table name
condition	identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries, and comparison operators

Note: If no rows are deleted, a message "`0 rows deleted.`" is returned:

Deleting Rows from a Table

- **Specific rows are deleted if you specify the `WHERE` clause.**

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

- **All rows in the table are deleted if you omit the `WHERE` clause.**

```
DELETE FROM  copy_emp;
22 rows deleted.
```

You can delete specific rows by specifying the WHERE clause in the DELETE statement. The example in the slide deletes the Finance department from the DEPARTMENTS table. You can confirm the delete operation by displaying the deleted rows using the SELECT statement.

```
SELECT  *
FROM    departments
WHERE   department_name = 'Finance';

no rows selected.
```

If you omit the WHERE clause, all rows in the table are deleted. The second example on the slide deletes all the rows from the COPY_EMP table, because no WHERE clause has been specified.

Example

Remove rows identified in the WHERE clause.

```
DELETE FROM  employees
WHERE        employee_id = 114;

1 row deleted.

DELETE FROM  departments
WHERE        department_id IN (30, 40);

2 rows deleted.
```

Deleting Rows Based on Another Table

Use subqueries in DELETE statements to remove rows from a table based on values from another table.

```
DELETE FROM employees
WHERE  department_id =
                       (SELECT department_id
                        FROM   departments
                        WHERE  department_name LIKE '%Public%');
1 row deleted.
```

Deleting Rows: Integrity Constraint Error

```
DELETE FROM departments
WHERE       department_id = 60;
```

```
DELETE FROM departments
            *
ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)
violated - child record found
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example in the slide tries to delete department number 60 from the DEPARTMENTS table, but it results in an error because department number is used as a foreign key in the EMPLOYEES table. If the parent record that you attempt to delete has child records, then you receive the child record found violation ORA-02292.

The following statement works because there are no employees in department 70:

```
DELETE FROM  departments
WHERE        department_id = 70;

1 row deleted.
```

Using a Subquery in an `INSERT` Statement

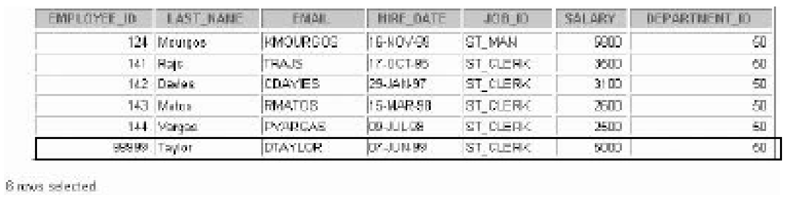
```
INSERT INTO
        (SELECT employee_id, last_name,
                email, hire_date, job_id, salary,
                department_id
         FROM   employees
         WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000,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

1 row created.
```

You can use a subquery in place of the table name in the `INTO` clause of the `INSERT` statement.

The select list of this subquery must have the same number of columns as the column list of the `VALUES` clause. Any rules on the columns of the base table must be followed in order for the `INSERT` statement to work successfully. For example, you could not put in a duplicate employee ID, nor leave out a value for a mandatory not null column.

```
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
124	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50
141	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	50
142	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	50
143	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	50
144	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	50
99999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50

6 rows selected.

The example shows the results of the subquery that was used to i dentify the table for the `INSERT` statement.

Using the `WITH CHECK OPTION` Keyword on DML Statements

- **A subquery is used to identify the table and columns of the DML statement.**
- **The `WITH CHECK OPTION` keyword prohibits you from changing rows that are not in the subquery.**

```
INSERT INTO  (SELECT employee_id, last_name, email,
                     hire_date, job_id, salary
              FROM   employees
              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
```

The `WITH CHECK OPTION` Keyword

Specify `WITH CHECK OPTION` to indicate that, if the subquery is used in place of a table in an `INSERT`, `UPDATE`, or `DELETE` statement, no changes that would produce rows that are not included in the subquery are permitted to that table.

In the example shown, the `WITH CHECK OPTION` keyword is used. The subquery identifes rows that are in department 50, but the department ID is not in the `SELECT` list, and a value is not provided for it in the `VALUES` list. Inserting this row would result in a department ID of null, which is not in the subquery.

Overview of the Explict Default Feature

- **With the explicit default feature, you can use the `DEFAULT` keyword as a column value where the column default is desired.**
- **The addition of this feature is for compliance with the SQL: 1999 Standard.**
- **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.**

The `DEFAULT` keyword can be used in `INSERT` and `UPDATE` statements to identify a default column value. If no default value exists, a null value is used.

Using Explicit Default Values

- **`DEFAULT` with `INSERT`:**

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

- **`DEFAULT` with `UPDATE`:**

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

Specify `DEFAULT` to set the column to the value previously specified as the default value for the column. If no default value for the corresponding column has been specified, Oracle sets the column to null.

In the first example shown, the `INSERT` statement uses a default value for the `MANAGER_ID` column. If there is no default value defined for the column, a null value is inserted instead.

The second example uses the `UPDATE` statement to set the `MANAGER_ID` column to a default value for department 10. If no default value is defined for the column, it changes the value to null.

Note: When creating a table, you can specify a default value for a column.

The `MERGE` Statement

- **Provides the ability to conditionally update or insert 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**

SQL has been extended to include the `MERGE` statement. Using this statement, you can update or insert a row conditionally into a table, thus avoiding multiple `UPDATE` statements. The decision whether to update or insert into the target table is based on a condition in the `ON` clause.

Because the `MERGE` command combines the `INSERT` and `UPDATE` commands, you need both `INSERT` and `UPDATE` privileges on the target table and the `SELECT` privilege on the source table.

The `MERGE` statement is deterministic. You cannot update the same row of the target table multiple times in the same `MERGE` statement.

The `MERGE` statement is suitable in a number of data warehousing applications. For example, in a data warehousing application, you may need to work with data coming from multiple sources, some of which may be duplicates. With the `MERGE` statement, you can conditionally add or modify rows.

MERGE Statement Syntax

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

```
MERGE INTO table_name AS table_alias
  USING (table|view|sub_query) AS 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);
```

You can update existing rows and insert new rows conditionally by using the MERGE statement.

In the syntax:

INTO clause	specifies the target table you are updating or inserting into
USING clause	identifies the source of the data to be updated or inserted; can be a table, view, or subquery
ON clause	the condition upon which the MERGE operation either updates or inserts
WHEN MATCHED | WHEN NOT MATCHED	instructs the server how to respond to the results of the join condition

Merging Rows

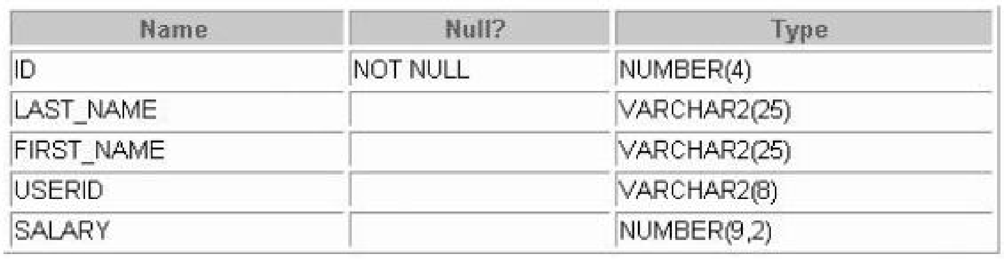
Insert or update rows in the COPY_EMP table to match the EMPLOYEES table.

```
MERGE INTO copy_emp AS 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);
```

The example shown matches the EMPLOYEE_ID in the COPY_EMP table to the EMPLOYEE_ID in the EMPLOYEES table. If a match is found, the row in the COPY_EMP table is updated to match the row in the EMPLOYEES table. If the row is not found, it is inserted into the COPY_EMP table.

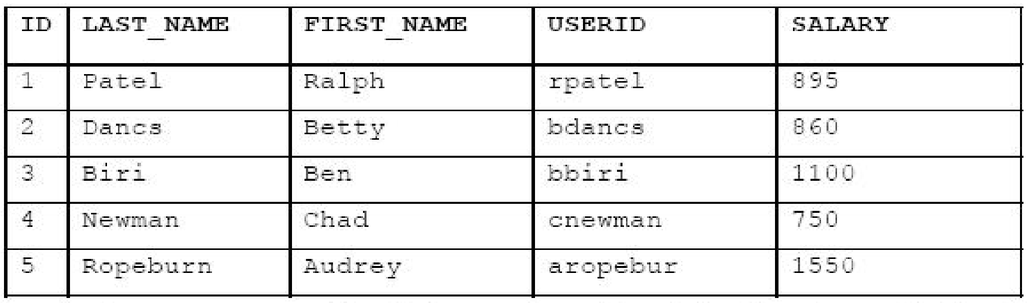
<u>ASSIGNMENTS</u>

1. Run the statement in the `lab8_1.sql` script to build the `MY_EMPLOYEE` table to be used for the lab.
2. Describe the structure of the `MY_EMPLOYEE` table to identify the column names.



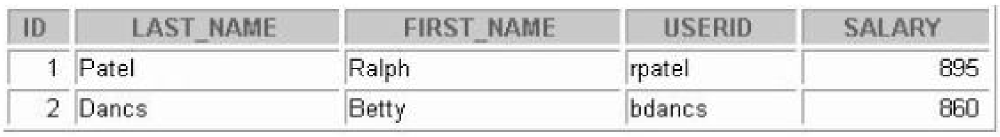
Name	Null?	Type
ID	NOT NULL	NUMBER(4)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
SALARY		NUMBER(9,2)

3. Add the first row of data to the `MY_EMPLOYEE` table from the following sample data. Do not list the columns in the `INSERT` clause.



ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

4. Populate the `MY_EMPLOYEE` table with the second row of sample data from the preceding list. This time, list the columns explicitly in the `INSERT` clause.
5. Confirm your addition to the table.



ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860