Objectives

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

• Describe each DML statement
• Insert rows into a table
• Update rows in a table
• Delete rows from a table
• Merge rows in a table
• Control transactions
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

**DEPARTMENTS**

<table>
<thead>
<tr>
<th>DEPARTMENT_ID</th>
<th>DEPARTMENT_NAME</th>
<th>MANAGER_ID</th>
<th>LOCATION_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Administration</td>
<td>200</td>
<td>1700</td>
</tr>
<tr>
<td>20</td>
<td>Marketing</td>
<td>201</td>
<td>1800</td>
</tr>
<tr>
<td>50</td>
<td>Shipping</td>
<td>124</td>
<td>1500</td>
</tr>
<tr>
<td>60</td>
<td>IT</td>
<td>103</td>
<td>1400</td>
</tr>
<tr>
<td>80</td>
<td>Sales</td>
<td>149</td>
<td>2500</td>
</tr>
<tr>
<td>90</td>
<td>Executive</td>
<td>100</td>
<td>1700</td>
</tr>
<tr>
<td>110</td>
<td>Accounting</td>
<td>205</td>
<td>1700</td>
</tr>
<tr>
<td>190</td>
<td>Contracting</td>
<td></td>
<td>1700</td>
</tr>
</tbody>
</table>

New row

...insert a new row into the **DEPARTMENTS** table...

<table>
<thead>
<tr>
<th>DEPARTMENT_ID</th>
<th>DEPARTMENT_NAME</th>
<th>MANAGER_ID</th>
<th>LOCATION_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>Public Relations</td>
<td>100</td>
<td>1700</td>
</tr>
</tbody>
</table>
The INSERT Statement Syntax

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

\[
\text{INSERT INTO} \quad \text{table} \quad ((\text{column} \quad [, \quad \text{column} \ldots])) \\
\text{VALUES} \quad (\text{value} \quad [, \quad \text{value} \ldots]);
\]

- Only one row is inserted at a time with this syntax.
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.
Inserting Rows with Null Values

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

```sql
INSERT INTO departments (department_id, department_name) VALUES (30, 'Purchasing');
```

1 row created.

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

```sql
INSERT INTO departments VALUES (100, 'Finance', NULL, NULL);
```

1 row created.
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.

```sql
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.

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
<th>PHONE_NUMBER</th>
<th>HIRE_DATE</th>
<th>JOB_ID</th>
<th>SALARY</th>
<th>COMMISSION_P</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>Den</td>
<td>Raphealy</td>
<td>DRAPEAL</td>
<td>515.127.4561</td>
<td>03-FEB-99</td>
<td>AC_ACCOUNT</td>
<td>11000</td>
<td></td>
</tr>
</tbody>
</table>
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);
```

1 row created.

Define Substitution Variables

```
|...department_id... | 40          |
|...department_name...| Human Resources |
|...location...       | 2500        |
```

Submit for Execution  Cancel

1 row created.
Copying Rows from Another Table

- Write your INSERT statement with a subquery.

```sql
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.
## Changing Data in a Table

**EMPLOYEES**

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

Update rows in the **EMPLOYEES** table.
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 in a Table

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

```sql
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.

```sql
UPDATE copy_emp
SET  department_id = 110;
22 rows updated.
```
Updating Two Columns with a Subquery

Update employee 114’s job and salary 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.
```
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;
```

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

Department number 55 does not exist
Removing a Row from a Table

**DEPARTMENTS**

<table>
<thead>
<tr>
<th>DEPARTMENT_ID</th>
<th>DEPARTMENT_NAME</th>
<th>MANAGER_ID</th>
<th>LOCATION_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Administration</td>
<td>200</td>
<td>1700</td>
</tr>
<tr>
<td>20</td>
<td>Marketing</td>
<td>201</td>
<td>1800</td>
</tr>
<tr>
<td>30</td>
<td>Purchasing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Finance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Shipping</td>
<td>124</td>
<td>1500</td>
</tr>
<tr>
<td>60</td>
<td>IT</td>
<td>103</td>
<td>1400</td>
</tr>
</tbody>
</table>

Delete a row from the **DEPARTMENTS** table.

<table>
<thead>
<tr>
<th>DEPARTMENT_ID</th>
<th>DEPARTMENT_NAME</th>
<th>MANAGER_ID</th>
<th>LOCATION_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Administration</td>
<td>200</td>
<td>1700</td>
</tr>
<tr>
<td>20</td>
<td>Marketing</td>
<td>201</td>
<td>1800</td>
</tr>
<tr>
<td>30</td>
<td>Purchasing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Shipping</td>
<td>124</td>
<td>1500</td>
</tr>
<tr>
<td>60</td>
<td>IT</td>
<td>103</td>
<td>1400</td>
</tr>
</tbody>
</table>
The **DELETE** Statement

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

```
DELETE [FROM] table [WHERE condition];
```
Deleting Rows from a Table

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

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

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

```sql
DELETE FROM copy_emp;
22 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

DELETES FROM departments
WHERE department_id = 60;

DELETES 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.
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.
Using a Subquery in an `INSERT` Statement

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

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
<th>HIRE_DATE</th>
<th>JOB_ID</th>
<th>SALARY</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>Mourgos</td>
<td>KMOURGOS</td>
<td>16-NOV-99</td>
<td>ST_MAN</td>
<td>5800</td>
<td>50</td>
</tr>
<tr>
<td>141</td>
<td>Rajs</td>
<td>TRAJS</td>
<td>17-OCT-95</td>
<td>ST_CLERK</td>
<td>3500</td>
<td>50</td>
</tr>
<tr>
<td>142</td>
<td>Davies</td>
<td>CDAVIES</td>
<td>29-JAN-97</td>
<td>ST_CLERK</td>
<td>3100</td>
<td>50</td>
</tr>
<tr>
<td>143</td>
<td>Matos</td>
<td>RMATOS</td>
<td>15-MAR-98</td>
<td>ST_CLERK</td>
<td>2600</td>
<td>50</td>
</tr>
<tr>
<td>144</td>
<td>Vargas</td>
<td>PVARGAS</td>
<td>09-JUL-98</td>
<td>ST_CLERK</td>
<td>2500</td>
<td>50</td>
</tr>
<tr>
<td>99999</td>
<td>Taylor</td>
<td>DTAYLOR</td>
<td>07-JUN-99</td>
<td>ST_CLERK</td>
<td>5000</td>
<td>50</td>
</tr>
</tbody>
</table>

6 rows selected.
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.

```sql
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);
```

**ERROR at line 1:**
ORA-01402: view WITH CHECK OPTION where-clause violation
Overview of the Explicit 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.
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;
```
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
The **MERGE** Statement Syntax

You can conditionally insert or update 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_val1,
        col2 = col2_val2
    WHEN NOT MATCHED THEN
        INSERT (column_list)
        VALUES (column_values);
```
Merging Rows

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

MERGE INTO copy_emp 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);
Merging Rows

```
SELECT *
FROM COPY_EMP;

no rows selected

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

SELECT *
FROM COPY_EMP;

20 rows selected.
```
Database Transactions

A database transaction consists of one of the following:

- DML statements which constitute one consistent change to the data
- One DDL statement
- One DCL statement
Database Transactions

• Begin when the first DML SQL statement is executed
• End with one of the following events:
  – A COMMIT or ROLLBACK statement is issued
  – A DDL or DCL statement executes (automatic commit)
  – The user exits iSQL*Plus
  – The system crashes
Advantages of **COMMIT** and **ROLLBACK** Statements

With **COMMIT** and **ROLLBACK** statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations
Controlling Transactions

Time

Transaction

SAVEPOINT A

DELETE

INSERT

UPDATE

SAVEPOINT B

INSERT

ROLLBACK to SAVEPOINT B

ROLLBACK to SAVEPOINT A

ROLLBACK
Rolling Back Changes to a Marker

- Create a marker in a current transaction by using the `SAVEPOINT` statement.
- Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```sql
UPDATE...
SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```
Implicit Transaction Processing

• An automatic commit occurs under the following circumstances:
  – DDL statement is issued
  – DCL statement is issued
  – Normal exit from iSQL*Plus, without explicitly issuing COMMIT or ROLLBACK statements

• An automatic rollback occurs under an abnormal termination of iSQL*Plus or a system failure.
State of the Data
Before COMMIT or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the SELECT statement.
- Other users cannot view the results of the DML statements by the current user.
- The affected rows are locked; other users cannot change the data within the affected rows.
State of the Data after COMMIT

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.
Committing Data

• Make the changes.

```
DELETE FROM employees
WHERE employee_id = 99999;
1 row deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

• Commit the changes.

```
COMMIT;
Commit complete.
```
State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```sql
DELETE FROM copy_emp;
22 rows deleted.
ROLLBACK;
Rollback complete.
```
Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.
Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with changes made by another user.
- Read consistency ensures that on the same data:
  - Readers do not wait for writers.
  - Writers do not wait for readers.
Implementation of Read Consistency

User A

```
UPDATE employees
SET    salary = 7000
WHERE  last_name = 'Goyal';
```

User B

```
SELECT  *
FROM userA.employees;
```
Locking

In an Oracle database, locks:

• Prevent destructive interaction between concurrent transactions
• Require no user action
• Automatically use the lowest level of restrictiveness
• Are held for the duration of the transaction
• Are of two types: explicit locking and implicit locking
Implicit Locking

- **Two lock modes:**
  - Exclusive: Locks out other users
  - Share: Allows other users to access

- **High level of data concurrency:**
  - DML: Table share, row exclusive
  - Queries: No locks required
  - DDL: Protects object definitions

- **Locks held until commit or rollback**
Summary

In this lesson, you should have learned how to use DML statements and control transactions.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>Adds a new row to the table</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Modifies existing rows in the table</td>
</tr>
<tr>
<td>DELETE</td>
<td>Removes existing rows from the table</td>
</tr>
<tr>
<td>MERGE</td>
<td>Conditionally inserts or updates data in a table</td>
</tr>
<tr>
<td>COMMIT</td>
<td>Makes all pending changes permanent</td>
</tr>
<tr>
<td>SAVEPOINT</td>
<td>Is used to rollback to the savepoint marker</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>Discards all pending data changes</td>
</tr>
</tbody>
</table>
Practice 8 Overview

This practice covers the following topics:

- Inserting rows into the tables
- Updating and deleting rows in the table
- Controlling transactions
## Read Consistency Example

<table>
<thead>
<tr>
<th>Output</th>
<th>Time</th>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td>(t_1)</td>
<td>SELECT salary FROM employees WHERE last_name='King';</td>
<td>UPDATE employees SET salary=salary+10000 WHERE last_name='King';</td>
</tr>
<tr>
<td>24000</td>
<td>(t_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24000</td>
<td>(t_3)</td>
<td>SELECT salary FROM employees WHERE last_name='King';</td>
<td></td>
</tr>
<tr>
<td>34000</td>
<td>(t_4)</td>
<td>SELECT salary FROM employees WHERE last_name='King';</td>
<td>COMMIT;</td>
</tr>
<tr>
<td>34000</td>
<td>(t_5)</td>
<td>SELECT salary FROM employees WHERE last_name='King';</td>
<td></td>
</tr>
</tbody>
</table>