

8

Manipulating Data

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

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

- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Control transactions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objective

In this lesson, you learn how to use DML statements to insert rows into a table, update existing rows in a table, and delete existing rows from a table. You also learn how to control transactions with the COMMIT, SAVEPOINT, and ROLLBACK statements.

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.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Data Manipulation Language

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction*.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account, and record the transaction in the transaction journal. The Oracle server must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

Adding a New Row to a Table

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

70 Public Relations	100	1700
---------------------	-----	------

**New
row**

**Insert new row
into the
DEPARTMENTS table**

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700
9	190	Contracting	(null)	1700

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Adding a New Row to a Table

The graphic in the slide illustrates adding a new department to the DEPARTMENTS table.

INSERT Statement Syntax

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

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

- With this syntax, only one row is inserted at a time.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Adding a New Row to a Table (continued)

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

In the syntax:

<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to populate
<i>value</i>	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 rows inserted
```

- Enclose character and date values in single quotation marks.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Adding a New Row to a Table (continued)

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

DESCRIBE departments		
Name	Null	Type

DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)
4 rows selected		

For clarity, use the column list in the `INSERT` clause.

Enclose character and date values in single quotation marks; it is not recommended that you enclose numeric values in single quotation marks.

Number values should not be enclosed in single quotation marks, because implicit conversion may take place for numeric values that are assigned to `NUMBER` data type columns if single quotation marks 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 rows inserted
```

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

```
INSERT INTO departments
VALUES      (100, 'Finance', NULL, NULL);
1 rows inserted
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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 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 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Inserting Special Values by Using SQL Functions

You can use functions to enter special values in your table.

The slide example records information for employee Popp in the EMPLOYEES table. It supplies the current date and time in the HIRE_DATE column. It uses the SYSDATE function for current date and time.

You can also use the USER function when inserting rows in a table. The USER function records the current username.

Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
1	113	Popp	AC_ACCOUNT	10-NOV-08	(null)

Inserting Specific Date Values

- Add a new employee.

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

1 rows inserted

- Verify your addition.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
1	114	Den	Rapehealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Inserting Specific Date and Time Values

The DD-MON-YY format is usually used to insert a date value. With this format, recall that the century defaults to the current century. Because the date also contains time information, the default time is midnight (00:00:00).

If a date must be entered in a format other than the default format (for example, with another century or a specific time), you must use the TO_DATE function.

The example in the slide records information for employee Rapehealy in the EMPLOYEES table. It sets the HIRE_DATE column to be February 3, 1999. If you use the following statement instead of the one shown in the slide, the year of the hire date is interpreted as 2099.

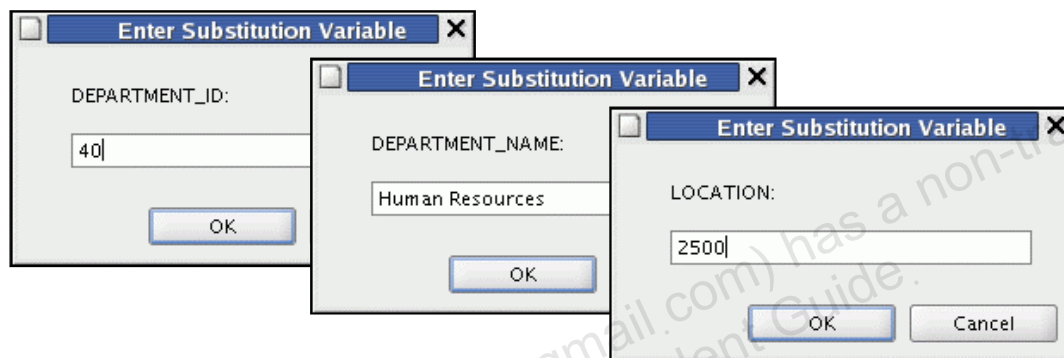
```
INSERT INTO employees
VALUES      (114,
             'Den', 'Rapehealy',
             'DRAPHEAL', '515.127.4561',
             '03-FEB-99',
             'AC_ACCOUNT', 11000, NULL, 100, 30);
```

If the RR format is used, the system provides the correct century automatically, even if it is not the current one.

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_id);
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Script to Manipulate Data

You can save commands with substitution variables to a file and execute the commands in the file. The slide example records information for a department in the DEPARTMENTS table.

Run the script file and you are prompted for input for each of the & substitution variables. After entering a value for the substitution variable, click the Continue button. The values that you input are then substituted into the statement. This enables you to run the same script file over and over but supply a different set of values each time you run it.

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 inserted

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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. In the slide example, for the INSERT INTO statement to work, you must have already created the sales_reps table using the CREATE TABLE statement. CREATE TABLE is discussed in the next lesson titled “Using DDL Statements to Create and Manage Tables.”

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

Update rows in the **EMPLOYEES** table:

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Changing Data in a Table

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

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).

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Updating Rows

You can modify existing rows by using the UPDATE statement.

In the syntax:

<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to populate
<i>value</i>	is the corresponding value or subquery for the column
<i>condition</i>	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 “UPDATE” in the *Oracle Database SQL Reference*.

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 rows updated
```

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Updating Rows (continued)

The `UPDATE` statement modifies specific rows if the `WHERE` clause is specified. The slide example 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;
```

R	LAST_NAME	R	DEPARTMENT_ID
1	Whalen		110
2	Hartstein		110
3	Fay		110
4	Higgins		110
5	Gietz		110
6	Raphealy		110

...

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 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 rows updated
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Updating Two Columns with a Subquery

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, the 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 rows updated
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Updating Rows Based on Another Table

You can use subqueries in UPDATE statements to update rows in a table. The example in the slide updates the COPY_EMP table based on the values from the EMPLOYEES table. It changes the department number of all employees with employee 200's job ID to employee 100's current department number.

Removing a Row from a Table

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	30	Purchasing	(null)	(null)
2	40	Human Resources	(null)	2500
3	10	Administration	200	1700
4	20	Marketing	201	1800
5	50	Shipping	124	1500
6	60	IT	103	1400

Delete a row from the DEPARTMENTS table:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	40	Human Resources	(null)	2500
2	10	Administration	200	1700
3	20	Marketing	201	1800
4	50	Shipping	124	1500
5	60	IT	103	1400

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Removing a Row from a Table

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

DELETE Statement

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Deleting Rows

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, the message “0 rows deleted” is returned.

For more information, see “DELETE” in the *Oracle Database SQL Reference*.

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

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

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Deleting Rows (continued)

You can delete specific rows by specifying the WHERE clause in the DELETE statement. The slide example 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';
0 rows selected.
```

If you omit the WHERE clause, all rows in the table are deleted. The second example in 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 rows 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 rows deleted
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Deleting Rows Based on Another Table

You can use subqueries to delete rows from a table based on values from another table. The example in the slide deletes all the employees who are in a department where the department name contains the string `Public`. The subquery searches the `DEPARTMENTS` table to find the department number based on the department name containing the string `Public`. The subquery then feeds the department number to the main query, which deletes rows of data from the `EMPLOYEES` table based on this department number.

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

TRUNCATE Statement

A more efficient method of emptying a table is with the TRUNCATE statement.

You can use the TRUNCATE statement to quickly remove all rows from a table or cluster.

Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information. Rollback information is covered later in this lesson.
- Truncating a table does not fire the delete triggers of the table.
- If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement. Disabling constraints is covered in a subsequent lesson.

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 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using a Subquery in an INSERT Statement

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 if the INSERT statement is to work successfully. For example, you could not put in a duplicate employee ID or omit a value for a mandatory not-null column.

Using a Subquery in an INSERT Statement

Verify the results:

```
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
1	99999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50
2	124	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50
3	141	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	50
4	142	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	50
5	143	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	50
6	144	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	50

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using a Subquery in an INSERT Statement (continued)

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

Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Database Transactions

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that make up one consistent change to the data. For example, a transfer of funds between two accounts should include the debit to one account and the credit to another account in the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only 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 SQL Developer or SQL*Plus.
 - The system crashes.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

When Does a Transaction Start and End?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A COMMIT or ROLLBACK statement is issued.
- A DDL statement, such as CREATE, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

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

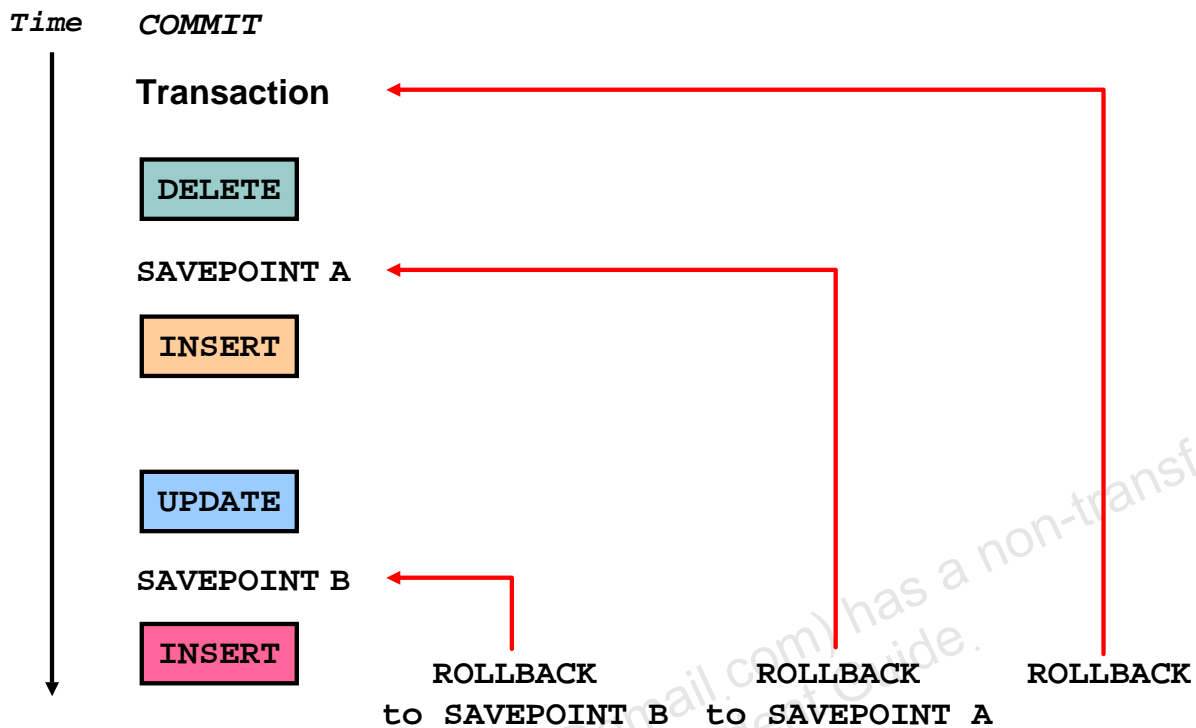
The Oracle logo, consisting of the word "ORACLE" in a white, sans-serif font, centered within a red rectangular bar.

Copyright © 2009, Oracle. All rights reserved.

Advantages of COMMIT and ROLLBACK

With the COMMIT and ROLLBACK statements, you have control over making changes to the data permanent.

Controlling Transactions



Copyright © 2009, Oracle. All rights reserved.

Explicit Transaction Control Statements

You can control the logic of transactions by using the **COMMIT**, **SAVEPOINT**, and **ROLLBACK** statements.

Statement	Description
COMMIT	Ends the current transaction by making all pending data changes permanent
SAVEPOINT <i>name</i>	Marks a savepoint within the current transaction
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes.
ROLLBACK TO <i>SAVEPOINT name</i>	ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding any changes and or savepoints that were created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

Note: **SAVEPOINT** is not ANSI standard SQL.

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.

```
UPDATE...  
SAVEPOINT update_done;  
SAVEPOINT update_done succeeded  
INSERT...  
ROLLBACK TO update_done;  
ROLLBACK TO succeeded
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Rolling Back Changes to a Marker

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

If you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

Implicit Transaction Processing

- An automatic commit occurs under the following circumstances:
 - DDL statement is issued
 - DCL statement is issued
 - Normal exit from SQL Developer or SQL*Plus, without explicitly issuing COMMIT or ROLLBACK statements
- An automatic rollback occurs under an abnormal termination of SQL Developer or SQL*Plus or a system failure.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Implicit Transaction Processing

Status	Circumstances
Automatic commit	DDL statement or DCL statement is issued. SQL Developer or SQL*Plus exited normally, without explicitly issuing COMMIT or ROLLBACK commands.
Automatic rollback	Abnormal termination of SQL Developer or SQL*Plus or system failure

Note: In SQL*Plus, the AUTOCOMMIT command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the COMMIT statement can still be issued explicitly. Also, the COMMIT statement is issued when a DDL statement is issued or when you exit SQL*Plus. The SET AUTOCOMMIT ON/OFF command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled. To enable Autocommit, perform the following:

- In the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.
- On the right pane, check the Autocommit in SQL Worksheet option. Click OK.

Implicit Transaction Processing (continued)

System Failures

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to their state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL*Plus, a normal exit is accomplished by entering the EXIT command at the prompt.

Closing the window is interpreted as an abnormal exit.

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 in the affected rows.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Committing Changes

Every data change made during the transaction is temporary until the transaction is committed.

The state of the data before COMMIT or ROLLBACK statements are issued can be described as follows:

- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit.
- The affected rows are locked; other users cannot change the data in 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.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Committing Changes (continued)

Make all pending changes permanent by using the COMMIT statement. Here is what happens after a COMMIT statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All users can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
- All savepoints are erased.

Committing Data

- Make the changes:

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 rows deleted

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

- Commit the changes:

```
COMMIT;
Commit complete
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Committing Changes (continued)

The slide example deletes a row from the EMPLOYEES table and inserts a new row into the DEPARTMENTS table. It then makes the change permanent by issuing the COMMIT statement.

Example

Remove departments 290 and 300 in the DEPARTMENTS table, and update a row in the EMPLOYEES table. Make the data change permanent.

```
DELETE FROM departments
WHERE  department_id IN (290, 300);
1 rows deleted.
```

```
UPDATE  employees
SET     department_id = 80
WHERE  employee_id = 206;
1 rows updated.
```

```
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.

```
DELETE FROM copy_emp;  
20 rows deleted  
ROLLBACK ;  
Rollback complete
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Rolling Back Changes

Discard all pending changes by using the ROLLBACK statement, which results in the following:

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

State of the Data After ROLLBACK

```
DELETE FROM test;  
25,000 rows deleted  
  
ROLLBACK;  
Rollback complete  
  
DELETE FROM test WHERE id = 100;  
1 rows deleted  
  
SELECT * FROM test WHERE id = 100;  
No rows selected  
  
COMMIT;  
Commit complete
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Example

While attempting to remove a record from the TEST table, you can accidentally empty the table. You can correct the mistake, reissue the proper statement, and make the data change permanent.

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.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Statement-Level Rollback

Part of a transaction can be discarded by an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the transaction are not discarded. They can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Read Consistency

Database users access the database in two ways:

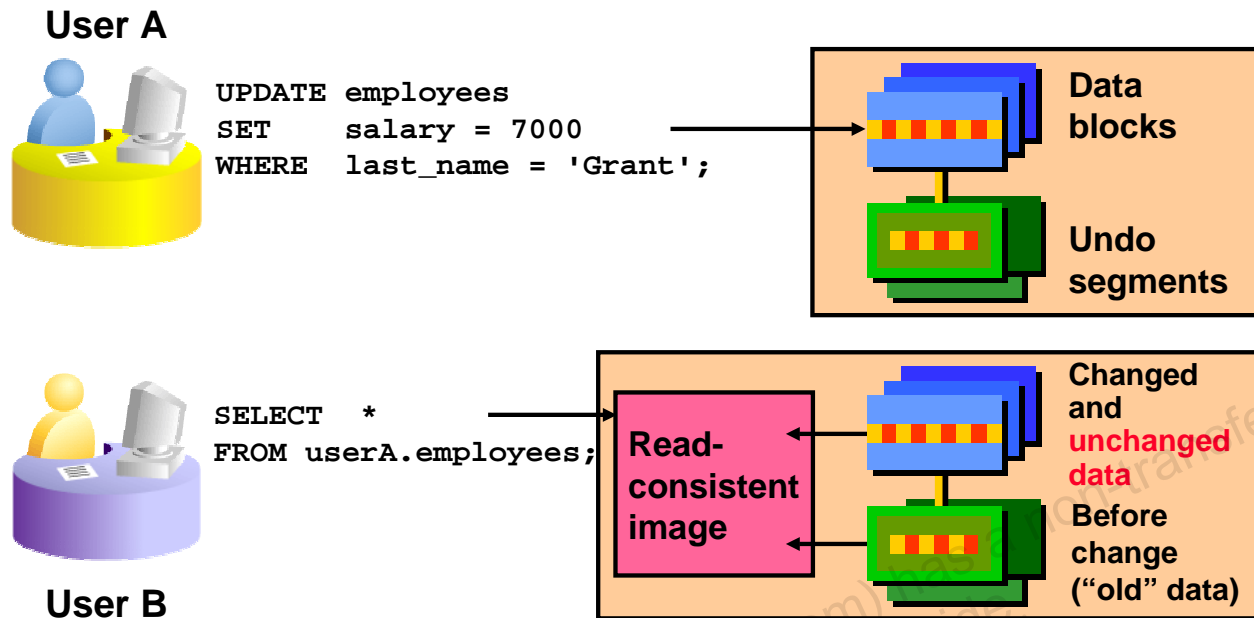
- Read operations (SELECT statement)
- Write operations (INSERT, UPDATE, DELETE statements)

You need read consistency so that the following occur:

- The database reader and writer are ensured a consistent view of the data.
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent way.
- Changes made by one writer do not disrupt or conflict with changes that another writer is making.

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

Implementation of Read Consistency



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Implementation of Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in undo segments. The read-consistent image is constructed from committed data from the table and old data being changed and not yet committed from the undo segment.

When an insert, update, or delete operation is made to the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, still see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

Before changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a select statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:

- The original, older version of the data in the undo segment is written back to the table.
- All users see the database as it existed before the transaction began.

Summary

In this lesson, you should have learned how to use the following statements:

Function	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned how to manipulate data in the Oracle Database by using the INSERT, UPDATE, and DELETE statements, as well as how to control data changes by using the COMMIT, SAVEPOINT, and ROLLBACK statements.

The Oracle server guarantees a consistent view of data at all times.

Practice 8: Overview

This practice covers the following topics:

- Inserting rows into tables
- Updating and deleting rows in a table
- Controlling transactions

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

Practice 8: Overview

In this practice, you add rows to the MY_EMPLOYEE table, update and delete data from the table, and control your transactions.

Practice 8

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, you use the MY_EMPLOYEE table before giving the statements to the HR department.

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab_08_01.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)
5 rows selected		

3. Create an INSERT statement to 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. *Do not enter all rows yet.*

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	1	Patel	Ralph	rpatel	895
2	2	Dancs	Betty	bdancs	860

6. Write an INSERT statement in a dynamic reusable script file named loademp.sql to load rows into the MY_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID. Save this script to a file named lab_08_06.sql.
7. Populate the table with the next two rows of sample data listed in step 3 by running the INSERT statement in the script that you created.

Practice 8 (continued)

8. Confirm your additions to the table.

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

9. Make the data additions permanent.

Update and delete data in the MY_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.
 11. Change the salary to \$1,000 for all employees who have a salary less than \$900.
 12. Verify your changes to the table.

		LAST_NAME		SALARY
1		Patel		1000
2		Dancs		1000
3		Drexler		1100
4		Newman		1000

13. Delete Betty Dancs from the MY_EMPLOYEE table.
 14. Confirm your changes to the table.

		ID		LAST_NAME		FIRST_NAME		USERID		SALARY
1		1		Patel		Ralph		rpatel		1000
2		3		Drexler		Ben		bbiri		1100
3		4		Newman		Chad		cnewman		1000

15. Commit all pending changes.

Practice 8 (continued)**Control data transaction to the MY_EMPLOYEE table.**

16. Populate the table with the last row of sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.
17. Confirm your addition to the table.

	AZ	ID	AZ	LAST_NAME	AZ	FIRST_NAME	AZ	USERID	AZ	SALARY
1		1		Patel		Ralph		rpatel		1000
2		3		Drexler		Ben		bbiri		1100
3		4		Newman		Chad		cnewman		1000
4		5		Ropeburn		Audrey		aropebur		1550

18. Mark an intermediate point in the processing of the transaction.
19. Empty the entire table.
20. Confirm that the table is empty.
21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.
22. Confirm that the new row is still intact.

	AZ	ID	AZ	LAST_NAME	AZ	FIRST_NAME	AZ	USERID	AZ	SALARY
1		1		Patel		Ralph		rpatel		1000
2		3		Drexler		Ben		bbiri		1100
3		4		Newman		Chad		cnewman		1000
4		5		Ropeburn		Audrey		aropebur		1550

23. Make the data addition permanent.

