8 Manipulating Data

ORACLE

Schedule:	Timing	Topic
	60 minutes	Lecture
	30 minutes	Practice
	90 minutes	Total

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**

ORACLE

Lesson Aim

In this lesson, you learn how 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

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.

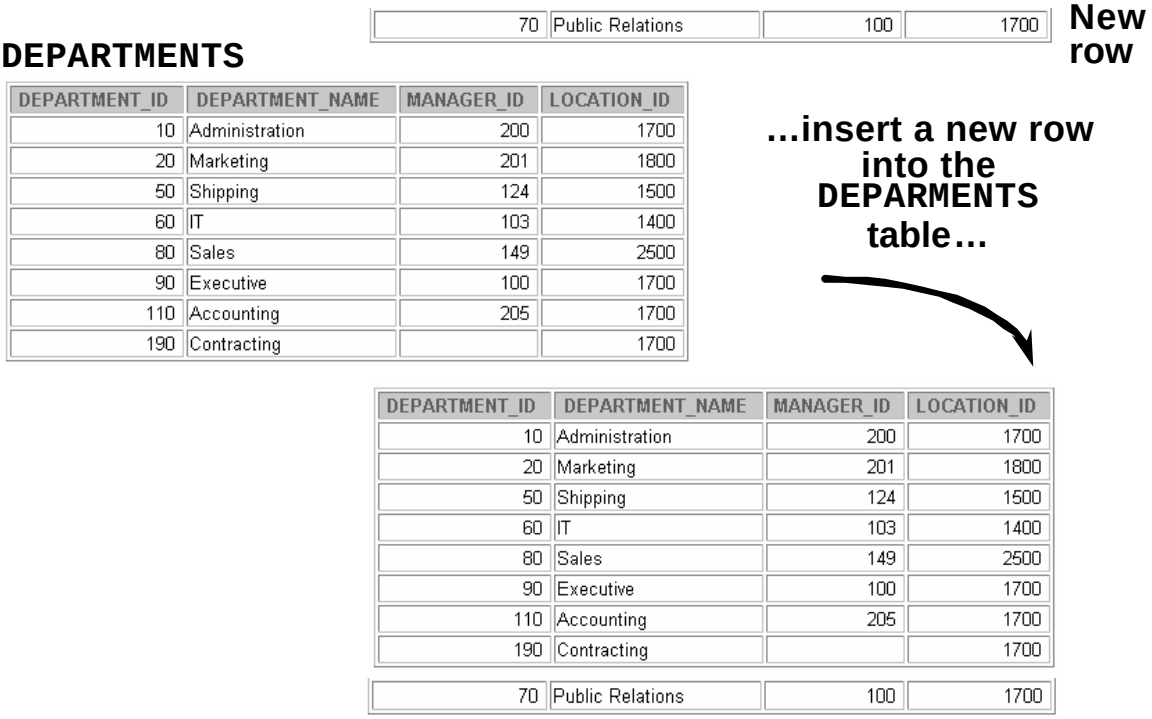
Instructor Note

DML statements can be issued directly in *i*SQL*Plus, performed automatically by tools such as Oracle Forms Services, or programmed with tools such as the 3GL precompilers.

Every table has `INSERT`, `UPDATE`, and `DELETE` privileges associated with it. These privileges are automatically granted to the creator of the table, but in general they must be explicitly granted to other users.

Starting with Oracle 7.2, you can place a subquery in the place of the table name in an `UPDATE` statement, essentially the same way you use a view.

Adding a New Row to a Table



ORACLE

Adding a New Row to a Table

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

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.**

ORACLE

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.**

ORACLE

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)

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.
```

ORACLE

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.
```

ORACLE

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
113	Popp	AC_ACCOUNT	27-SEP-01	

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_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

ORACLE

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 on the slide records information for employee Raphealy 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 on the slide, the year of the hire_date is interpreted as 2099.

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             '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.

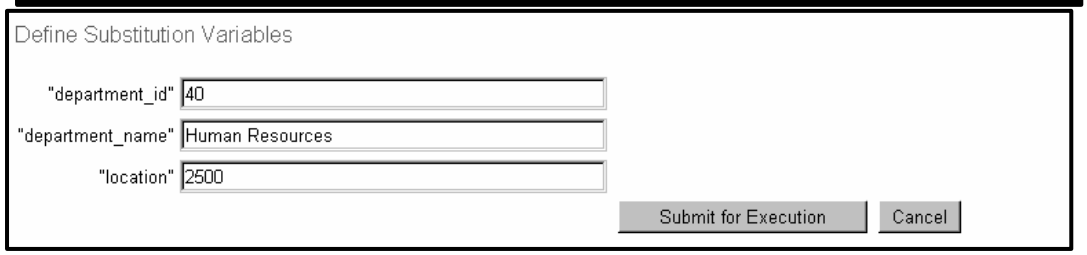
Instructor Note

The default date format in Oracle9*i* is DD-MON-RR. Prior to release 8.16, the default format was DD-MON-YY.

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.
```

ORACLE

Creating a Script to Manipulate Data

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

Run the script file and you are prompted for input for the & substitution variables. The values you input are then substituted into the statement. This allows you to run the same script file over and over, but supply a different set of values each time you run it.

Instructor Note

Be sure to mention the following points about the script:

- Do not prefix the *i*SQL*Plus substitution parameter with the ampersand in the DEFINE command.
- Use a dash to continue an *i*SQL*Plus command on the next line.

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.**

ORACLE

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;
```

For more information, see *Oracle9i SQL Reference*, "SELECT," subqueries section.

Instructor Note

Please run the script 8_cretabs.sql to create the COPY_EMP and SALES_REPS tables before demonstrating the code examples. Do not get into too many details on copying rows from another table.

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_P
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	COMMISSIO
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	

ORACLE

Changing Data in a Table

The slide graphic 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.
```

ORACLE

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;
```

LAST_NAME	DEPARTMENT_ID
King	110
Kochhar	110
De Haan	110
Hunold	110
Ernst	110
Lorentz	110

▪ ▪ ▪

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 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.
```

ORACLE

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, 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.
```

ORACLE

Updating Rows Based on Another Table

You can use subqueries in UPDATE statements to update rows in a table. The example on 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.

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

ORACLE

Integrity Constraint Error

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

In the example on the slide, department number 55 does not exist in the parent table, `DEPARTMENTS`, and so you receive the *parent key* violation `ORA-02291`.

Note: Integrity constraints ensure that the data adheres to a predefined set of rules. A subsequent lesson covers integrity constraints in greater depth.

Instructor Note

Explain integrity constraints, and review the concepts of primary key and foreign key.

Removing a Row from a Table

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

Delete a row from the DEPARTMENTS table.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

ORACLE

Removing a Row from a Table

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

Instructor Note

After all the rows have been eliminated with the DELETE statement, only the data structure of the table remains. 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. It is covered in a subsequent 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. Disable the constraint before issuing the TRUNCATE statement.

The DELETE Statement

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

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

ORACLE

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, a message "0 rows deleted." is returned:

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

Instructor Note

The DELETE statement does not ask for confirmation. However, the delete operation is not made permanent until the data transaction is committed. Therefore, you can undo the operation with the ROLLBACK statement if you make a mistake.

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.
```

ORACLE

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';

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.
```

ORACLE

Deleting Rows Based on Another Table

You can use subqueries to delete rows from a table based on values from another table. The example on 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.

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.

ORACLE

Integrity Constraint Error

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

The example on 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.
```

Instructor Note

If referential integrity constraints are in use, you may receive an Oracle server error message when you attempt to delete a row. However, if the referential integrity constraint contains the ON DELETE CASCADE option, then the selected row and its children are deleted from their respective tables.

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.
```

ORACLE

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 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.

Using a Subquery in an INSERT Statement

```
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.

Using a Subquery in an INSERT Statement

The example shows the results of the subquery that was used to identify 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
```

ORACLE

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 identifies 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 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.

ORACLE

Explicit Defaults

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;
```

ORACLE

Using Explicit Default Values

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. This is discussed in the "Creating and Managing Tables" lesson.

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**

ORACLE

MERGE Statements

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.

Since 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.

An alternative approach is to use PL/SQL loops and multiple DML statements. The MERGE statement, however, is easy to use and more simply expressed as a single SQL 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.

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 = col_val1,
    col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

ORACLE

Merging Rows

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

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

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);
```

ORACLE

Example of Merging Rows

```
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.email          = e.email,
    c.phone_number   = e.phone_number,
    c.hire_date      = e.hire_date,
    c.job_id         = e.job_id,
    c.salary         = e.salary,
    c.commission_pct = e.commission_pct,
    c.manager_id     = e.manager_id,
    c.department_id  = e.department_id
WHEN NOT MATCHED THEN
  INSERT VALUES(e.employee_id, e.first_name, e.last_name,
       e.email, e.phone_number, e.hire_date, e.job_id,
       e.salary, e.commission_pct, e.manager_id,
       e.department_id);
```

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.

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.
```

ORACLE

Example of Merging Rows

The condition `c.employee_id = e.employee_id` is evaluated. Because the `COPY_EMP` table is empty, the condition returns false: there are no matches. The logic falls into the `WHEN NOT MATCHED` clause, and the `MERGE` command inserts the rows of the `EMPLOYEES` table into the `COPY_EMP` table.

If rows existed in the `COPY_EMP` table and employee IDs matched in both tables (the `COPY_EMP` and `EMPLOYEES` tables), the existing rows in the `COPY_EMP` table would be updated to match the `EMPLOYEES` table.

Instructor Note

In a data warehousing environment, you may have a large fact table and a smaller dimension table with rows that need to be inserted into the large fact table conditionally. The `MERGE` statement is useful in this situation.

You may want to have a break here.

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**

ORACLE

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 *i*SQL*Plus**
  - **The system crashes**

ORACLE

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 *i*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.

Instructor Note

Please run the script `8_cretest.sql` to create the test table and insert data into the table.

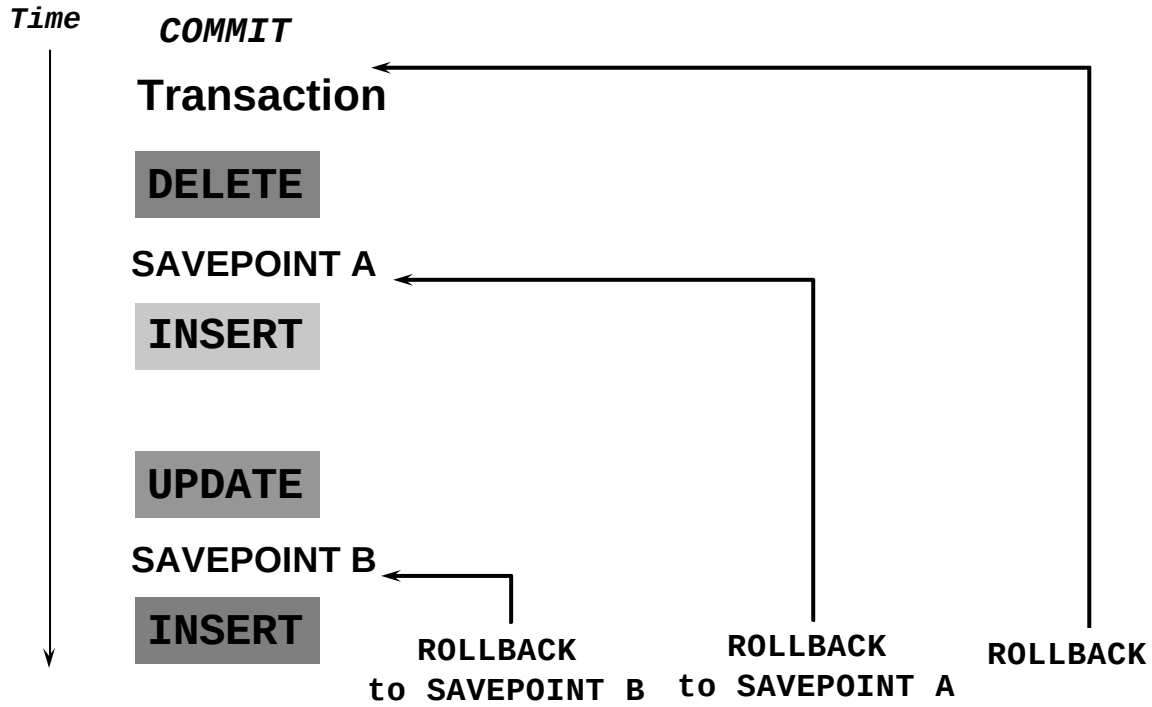
Advantages of COMMIT and ROLLBACK Statements

With COMMIT and ROLLBACK statements, you can:

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

ORACLE

Controlling Transactions



ORACLE

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` *name*	Marks a savepoint within the current transaction
`ROLLBACK`	`ROLLBACK` ends the current transaction by discarding all pending data changes
`ROLLBACK TO` *SAVEPOINT name*	`ROLLBACK TO SAVEPOINT` rolls back the current transaction to the specified savepoint, thereby discarding any changes and or savepoints 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. As savepoints are logical, there is no way to list the savepoints you have created.

Note: `SAVEPOINT` is not ANSI standard SQL.

Instructor Note

Savepoints are not schema objects and cannot be referenced in the data dictionary.

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 created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```

ORACLE

Rolling Back Changes to a Savepoint

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.

Instructor Note

Savepoints are especially useful in PL/SQL and 3GL programs in which recent changes can be undone conditionally based on run-time conditions.

Implicit Transaction Processing

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

ORACLE

Implicit Transaction Processing

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

Note: A third command is available in *i*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 from *i*SQL*Plus.

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.

From *i*SQL*Plus, a normal exit from the session is accomplished by clicking the Exit button. With SQL*Plus, a normal exit is accomplished by typing the command `EXIT` 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 within the affected rows.

ORACLE

Committing Changes

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

State of the data before COMMIT or ROLLBACK statements are issued:

- 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.

Instructor Note

With the Oracle server, data changes can actually be written to the database files before transactions are committed, but they are still only temporary.

If a number of users are making changes simultaneously to the same table, then each user sees only his or her changes until other users commit their changes.

By default, the Oracle server has *row-level locking*. It is possible to alter the default locking mechanism.

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

Committing Changes (continued)

Make all pending changes permanent by using the COMMIT statement. Following a COMMIT statement:

- Data changes are written to the database.
- The previous state of the data is permanently lost.
- 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 row deleted.

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

- **Commit the changes.**

```
COMMIT;
Commit complete.
```

ORACLE

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 COPY_EMP table. Make the data change permanent.

```
DELETE FROM departments
WHERE  department_id IN (290, 300);

2 rows deleted.

UPDATE  copy_emp
  SET   department_id = 80
  WHERE employee_id = 206;

1 row updated.

COMMIT;

Commit Complete.
```

Instructor Note

Use this example to explain how COMMIT ensures that two related operations occur together or not at all. In this case, COMMIT prevents empty departments from being created.

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;
22 rows deleted.
ROLLBACK;
Rollback complete.
```

ORACLE

Rolling Back Changes

Discard all pending changes by using the ROLLBACK statement. Following a ROLLBACK statement:

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

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.

```
DELETE FROM test;
25,000 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM test
WHERE  id = 100;
1 row deleted.

SELECT *
FROM   test
WHERE  id = 100;
No rows selected.

COMMIT;
Commit 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.**

ORACLE

Statement-Level Rollbacks

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.

Oracle issues an implicit commit before and after any data definition language (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.

Instructor Note

The Oracle server implements locks on data to provide data concurrency in the database. Those locks are released when certain events occur (such as a system failure) or when the transaction is completed. Implicit locks on the database are obtained when a DML statement is successfully executed. The Oracle Server locks data at the lowest level possible by default .
Manually acquire locks on the database tables by executing a `LOCK TABLE` statement or the `SELECT` statement with the `FOR UPDATE` clause.

Starting with Oracle9*i*, the DBA has the choice of managing undo segments or having Oracle automatically manage undo data in an undo tablespace.

Please read the Instructor Note on page 8-52.

For more information on locking, refer to *Oracle9i Concepts*, "Data Concurrency and Consistency."

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

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:

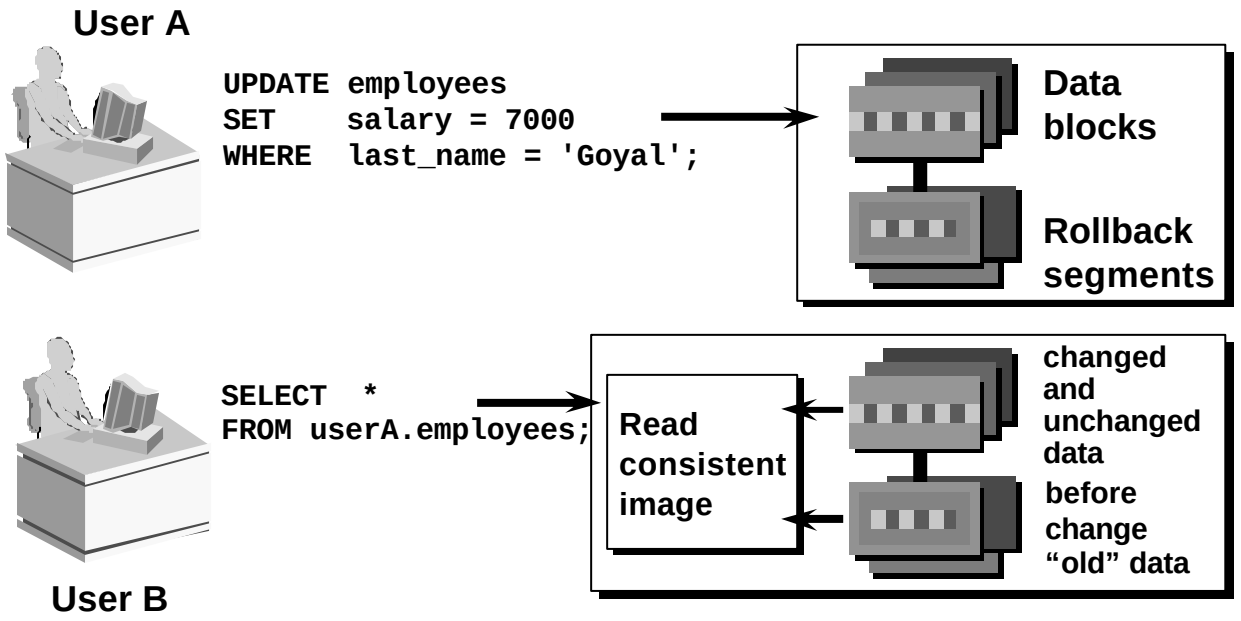
- The database reader and writer are ensured a consistent view of the data.
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent way.
- Changes made by one writer do not disrupt or conflict with changes another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

Instructor Note

Please read the Instructor note on page 8-53.

Implementation of Read Consistency



ORACLE

Implementation of Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in undo segments.

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 a *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 rollback 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 executing a SELECT statement. 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.

Instructor Note

When you commit a transaction, the Oracle server releases the rollback information but does not immediately destroy it. The information remains in the undo segment to create read-consistent views of pertinent data for queries that started before the transaction committed.

Starting with Oracle9*i*, the DBA has the choice of managing undo segments or having Oracle automatically manage undo data in an undo tablespace. This is discussed in the DBA courses.

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**

ORACLE

What Are Locks?

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource, either a user object (such as tables or rows) or a system object not visible to users (such as shared data structures and data dictionary rows).

How the Oracle Database Locks Data

Oracle locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Implicit locking occurs for all SQL statements except `SELECT`.

The users can also lock data manually, which is called explicit locking.

Instructor Note

See the Instructor Note on page 8-52.

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

ORACLE

DML Locking

When performing data manipulation language (DML) operations, the Oracle server provides data concurrency through DML locking. DML locks occur at two levels:

- A share lock is automatically obtained at the table level during DML operations. With share lock mode, several transactions can acquire share locks on the same resource.
- An exclusive lock is acquired automatically for each row modified by a DML statement. Exclusive locks prevent the row from being changed by other transactions until the transaction is committed or rolled back. This lock ensures that no other user can modify the same row at the same time and overwrite changes not yet committed by another user.
- DDL locks occur when you modify a database object such as a table.

Instructor Note

A `SELECT...FOR UPDATE` statement also implements a lock. This is covered in the Oracle 9*i* PL/SQL course.

Summary

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

Statement	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
MERGE	Conditionally inserts or updates data in a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to rollback to the savepoint marker
ROLLBACK	Discards all pending data changes

ORACLE

Summary

In this lesson, you should have learned how to manipulate data in the Oracle database by using the `INSERT`, `UPDATE`, and `DELETE` statements. Control data changes by using the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

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

Locking can be implicit or explicit.

Practice 8 Overview

This practice covers the following topics:

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

ORACLE

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.

Insert data into the MY_EMPLOYEE table.

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

Practice 3 (continued)

6. Write an insert statement in a text 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.
7. Populate the table with the next two rows of sample data by running the insert statement in the script that you created.
8. Confirm your additions to the table.

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

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 1000 for all employees with a salary less than 900.
12. Verify your changes to the table.

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

13.
14. Confirm your changes to the table.

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

15. Commit all pending changes.

Control data transaction to the `MY_EMPLOYEE` table.

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

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000
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.
23.

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

Demo: 8_select.sql

Purpose: To illustrate the concept that a reader does not lock another reader

Login to *i*SQL*Plus using the teach/oracle account.

Login to *i*SQL*Plus using an unused oraxx/oracle account.

Run the 8_select.sql script from the teach/oracle account. (This script selects all records from the DEPARTMENTS table).

Run the 8_select.sql script in the oraxx/oracle account. (This script selects all records from the DEPARTMENTS table).

In both the logins, the script executes successfully. This demonstrates the concept: *a reader does not lock another reader.*

Demo: 8_grant.sql, 8_update.sql, 8_select.sql

Purpose: To illustrate that a writer does not lock a reader

Run the 8_grant.sql script in the teach/oracle account. (This script grants SELECT and UPDATE privileges on the DEPARTMENTS table to the oraxx account).

Run the 8_update.sql script in the teach/oracle account. (This script updates the DEPARTMENTS table, changing location of the department ID 20 to location 1500. The update places a lock on the DEPARTMENTS table).

Run the 8_select.sql script in the teach/oracle account. (This script selects all records from the DEPARTMENTS table. Observe that the location of department ID 20 is changed to location ID 1500).

Run the 8_select.sql script in the oraxx/oracle account. (This script selects all records from the DEPARTMENTS table).

Observe that the script executes successfully in the oraxx/oracle account, but the location for department ID 20 still has the location ID of 1800. This demonstrates the concept: *a writer does not lock a reader.*

Demo: 8_update.sql, 8_rollback.sql, 8_select.sql

Purpose: To illustrate that a writer locks another writer

Run the 8_update.sql script in the oraxx/oracle account. (The script does not execute because the DEPARTMENTS table is locked by the teach/oracle account.)

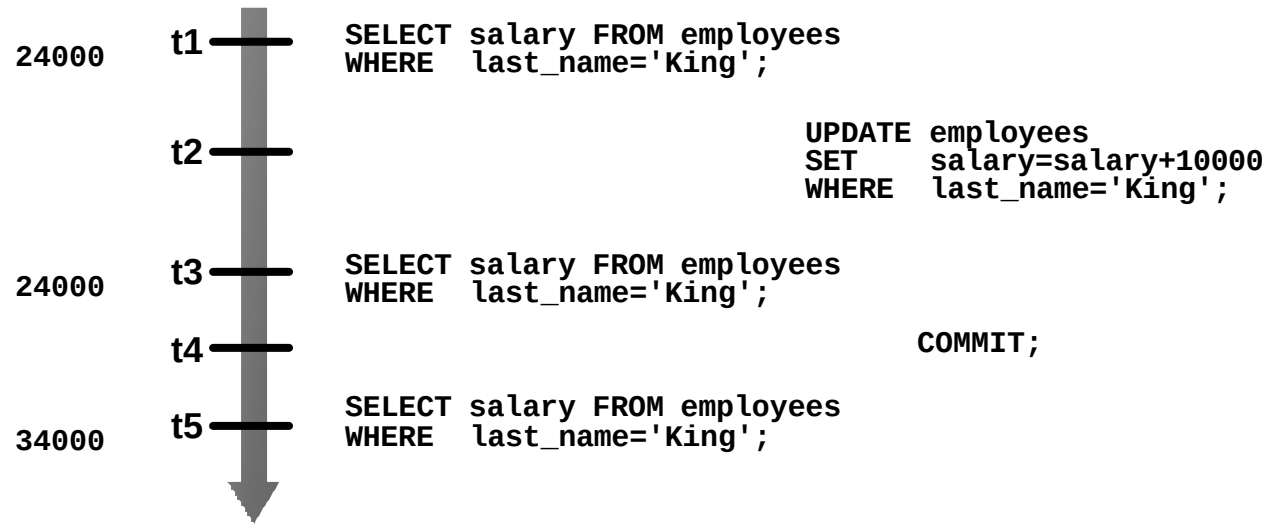
Switch to the teach/oracle account and run the 8_rollback.sql script. (This script rolls back the transaction, thereby releasing the lock on the DEPARTMENTS table.)

Switch to the oraxx/oracle account. You see that the 8_update.sql script has executed successfully because the lock on the DEPARTMENTS table has been released

Run the 8_select.sql script in the oraxx/oracle account. (This script selects all records from the DEPARTMENTS table. Observe that the location of department ID 20 is changed to location ID 1500.)

This demonstrates the concept: *a writer locks a writer.*

Read Consistency Example



ORACLE

Instructor Note (for page 8-43)

Read Consistency Example

For the duration of a SQL statement, read consistency guarantees that the selected data is consistent to the time point when the processing of the statement started.

Oracle server keeps noncommitted data in data blocks of undo segments (before images). As long as the changes are not committed, all users *see* the original data. The Oracle server uses data of both table segments and undo segments to generate a read-consistent view on the data.

In the example in the slide, the update of session 2 is not visible to process 1 until session 2 has committed the update (from t4 on). For the select statement at time point t3, the salary of King must be read from a data block of a undo segment that belongs to the transaction of session 2.