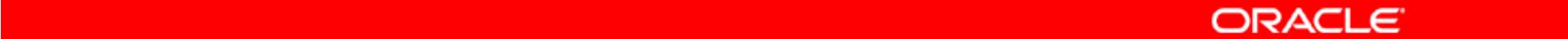




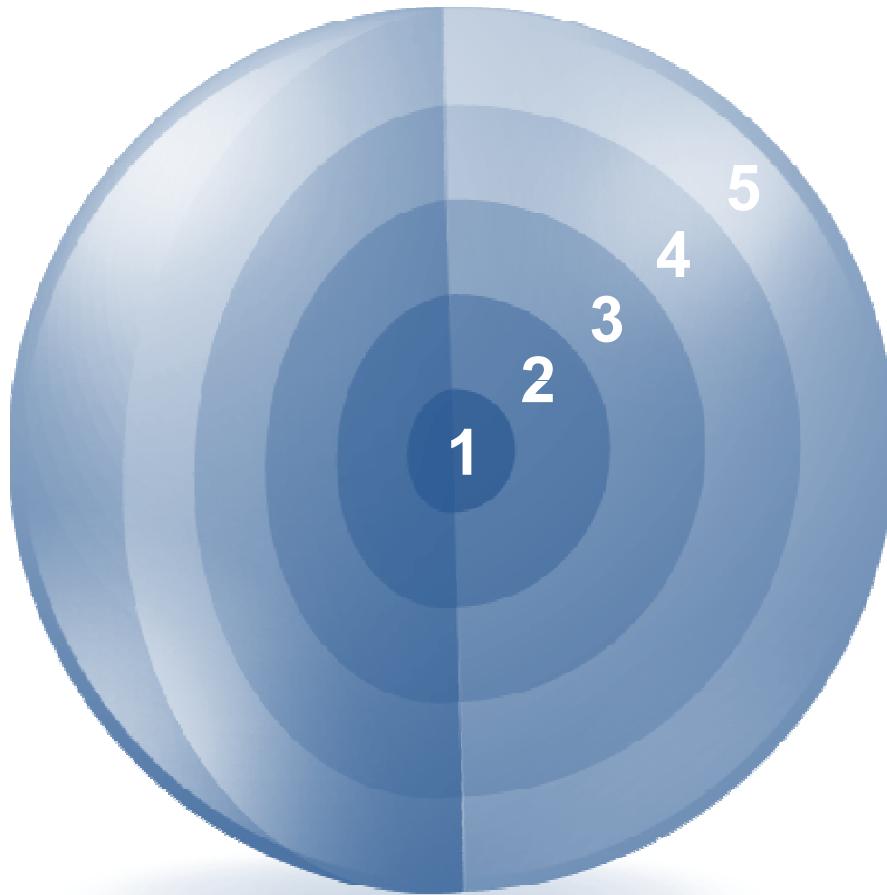
Lesson 9

Manipulating Data



ORACLE

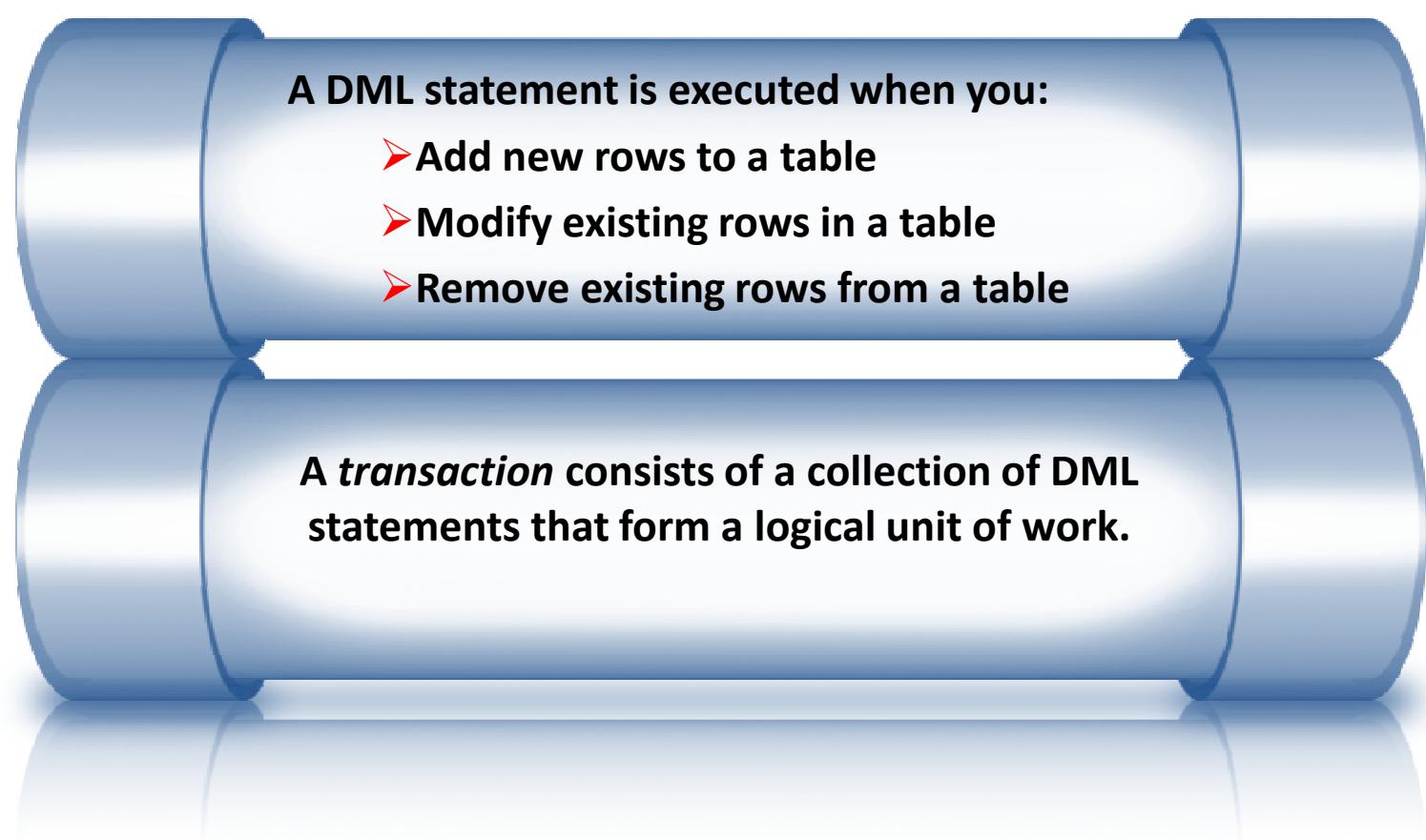
What You will Learn at the end of this Session?



1. **Describe each data manipulation language (DML) statement**
2. **Insert rows into a table**
3. **Update rows in a table**
4. **Delete rows from a table**
5. **Control transactions**

ORACLE

Data Manipulation Language



ORACLE

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

ORACLE

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.

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 order_items (order_id,  
line_item_id, product_id, unit_price, quantity)  
VALUES (2355, 1, 3108, 46, 200);  
1 rows inserted
```

Enclose character and date values within single quotation marks.

ORACLE

Inserting Rows with Null Values

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

```
INSERT INTO promotions (promo_id)
VALUES (3);
```

```
1 rows inserted
```

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

```
INSERT INTO promotions
VALUES (3, NULL);
```

```
1 rows inserted
```

ORACLE

Inserting Special Values

- The SYSDATE function records the current date and time.

```
INSERT INTO runreport
  (date_run,
   user_run,
   comments)
VALUES
  (SYSDATE,
   'OE',
   'Editing Report');
```

```
1 rows inserted
```

ORACLE

Inserting Specific Date and Time Values

- Add a new report.

```
INSERT INTO runreport
          (date_run, user_run, comments)
VALUES
      (to_date('24 FEB,1999'),
       'OE',
       'Editing Report');

1 rows inserted
```

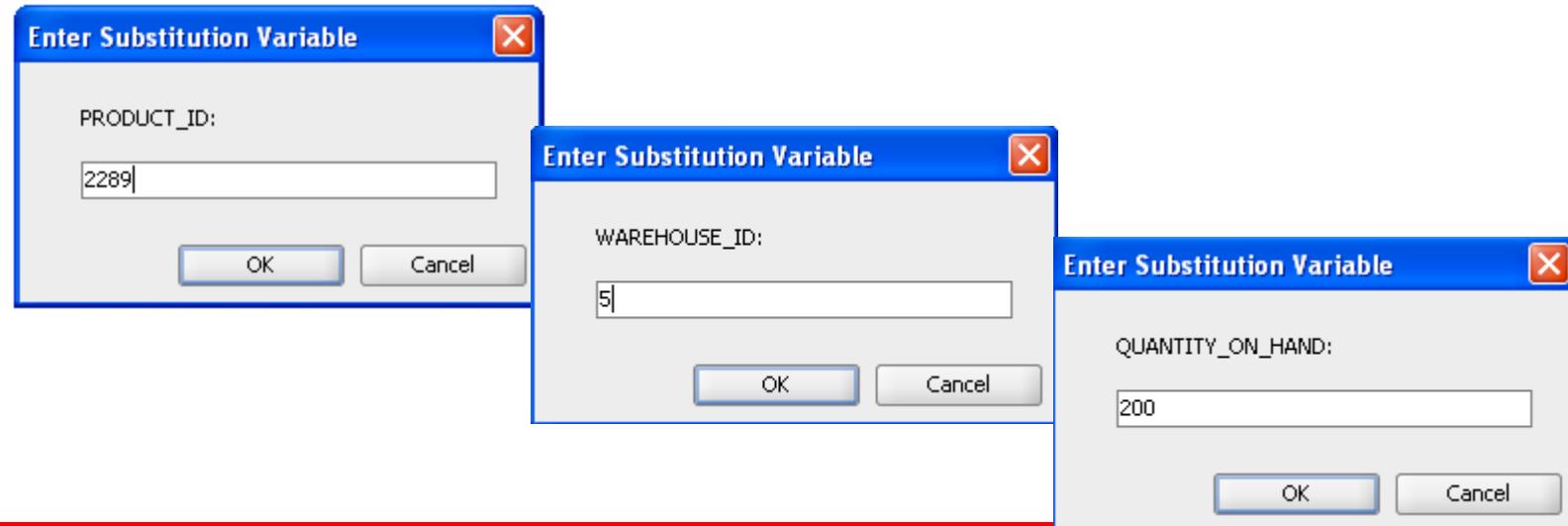
- Verify your addition.

	DATE_RUN	USER_RUN	COMMENTS
1	24-FEB-99	OE	Editing Report

Creating a Script

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

```
INSERT INTO inventories
( product_id, warehouse_id, quantity_on_hand )
VALUES ( &product_id, &warehouse_id, &quantity_on_hand ) ;
```



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.

Inserts all the rows returned by the subquery in the table, sales_reps.

ORACLE

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

Update rows in the EMPLOYEES table:



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

ORACLE

UPDATE Statement Syntax

- Modify existing values in a table with the UPDATE statement:

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

- Update more than one row at a time (if required).

Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the WHERE clause:

```
UPDATE inventories  
SET warehouse_id = 7  
WHERE product_id = 3108 ;
```

1 rows updated

- Values for all the rows in the table are modified if you omit the WHERE clause:

```
UPDATE inventories  
SET warehouse_id = 7 ;
```

- Specify SET *column_name*= NULL to update a column value to NULL.

ORACLE

Updating Two Columns with a Subquery

- Update employee 113's job and salary to match those of employee 205.

```
UPDATE orders
SET order_date = ( SELECT order_date
                   FROM orders
                   WHERE order_id = 2397 ) ,
    customer_id = ( SELECT customer_id
                     FROM orders
                     WHERE order_id = 2397 )
WHERE order_id = 2458;
```

1 rows updated

ORACLE

Updating Rows Based on Another Table

- Use the subqueries in the UPDATE statements to update row values 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
```



Removing a Row from 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

Delete a row from 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

ORACLE

DELETE Statement

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

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

Deleting Rows from a Table

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

```
DELETE FROM runreport  
WHERE comments = 'Editing Report';
```

```
1 rows deleted
```

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

```
DELETE FROM copy_emp;
```

```
22 rows deleted
```

ORACLE

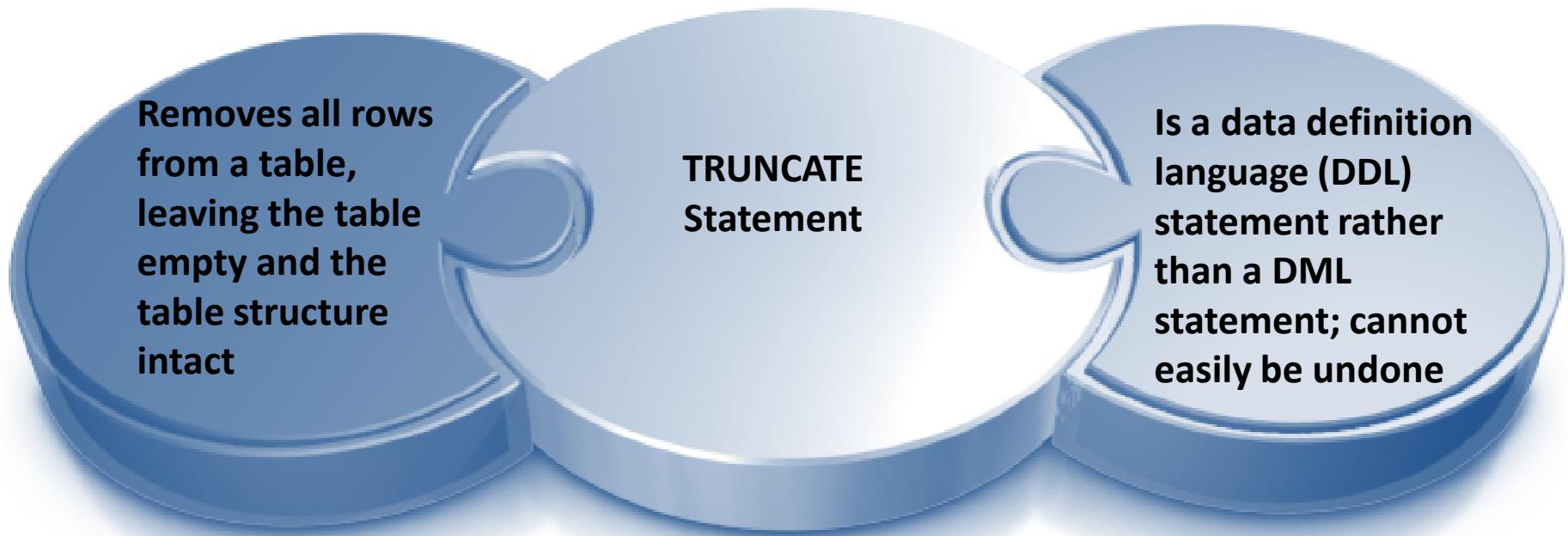
Deleting Rows Based on Another Table

- Use the subqueries in the 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
```

TRUNCATE Statement



Syntax

```
TRUNCATE TABLE table_name;
```

Example

```
TRUNCATE TABLE copy_emp;
```

ORACLE

- 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

Database Transactions: Start and End

Begin when the first DML SQL statement is executed.

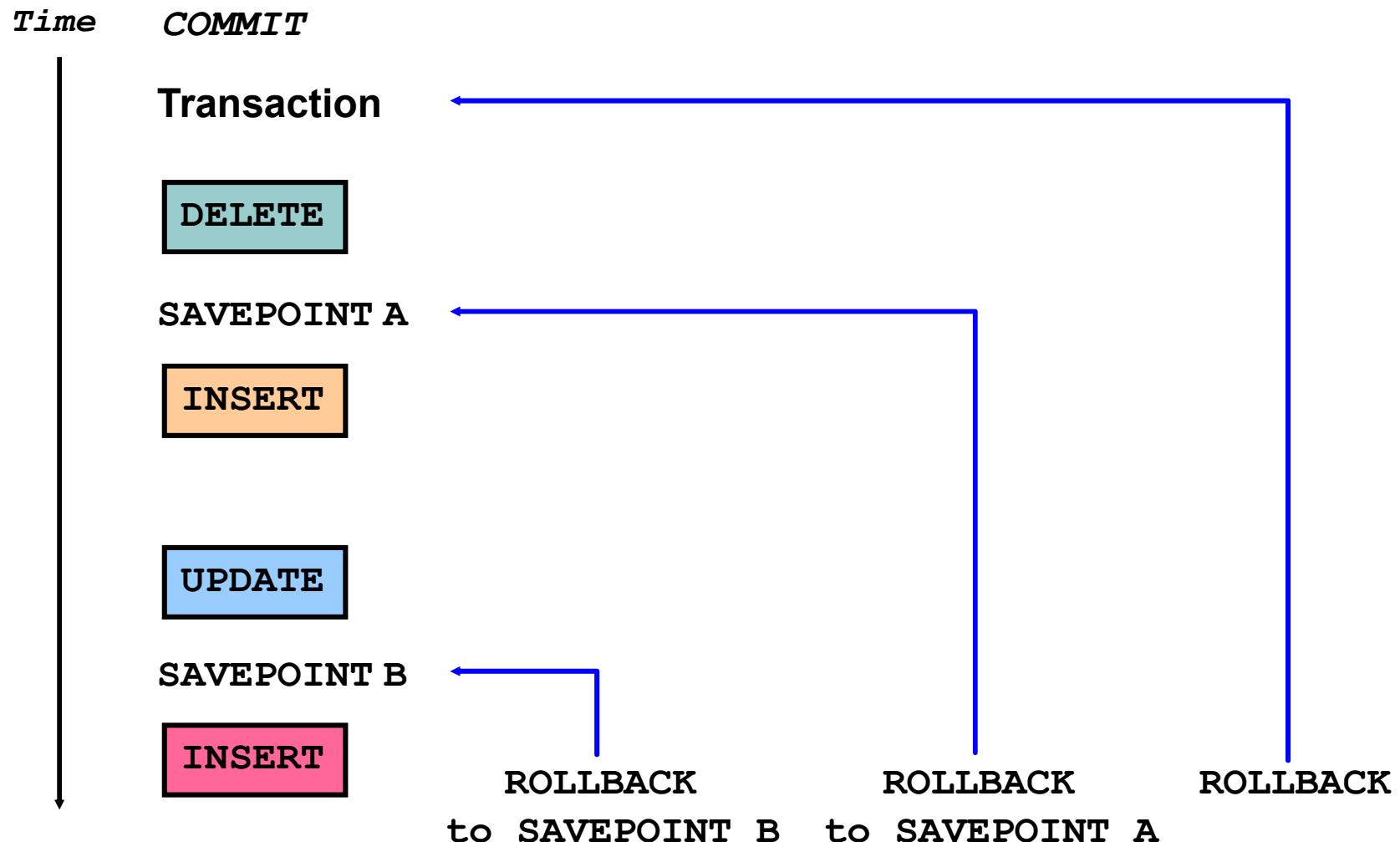
End with one of the following events:

- 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.



Explicit Transaction Control Statements



Rolling Back Changes to a Marker

- Create a marker in the 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 . . .
```

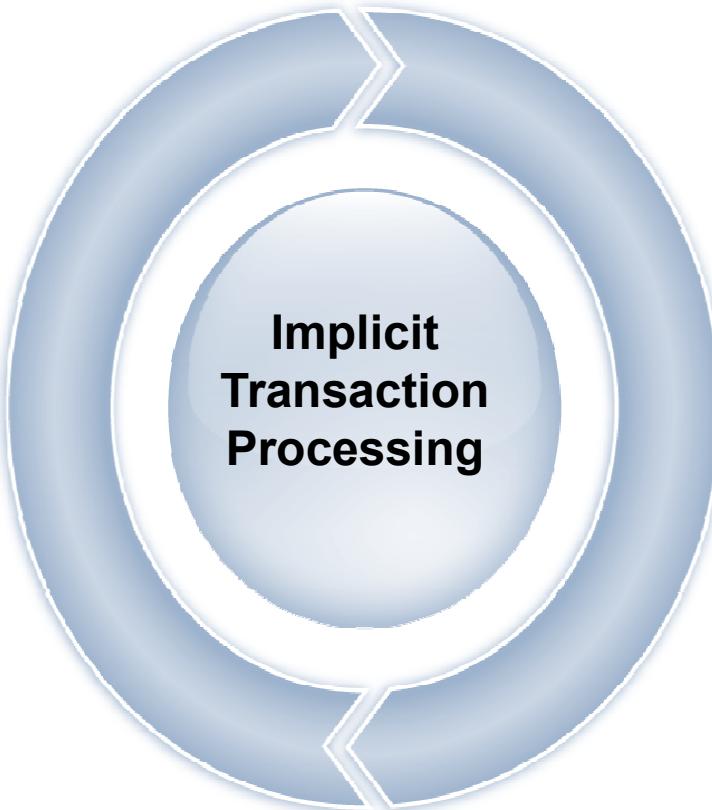
```
ROLLBACK TO update_done;
```

```
ROLLBACK TO succeeded.
```

ORACLE

Implicit Transaction Processing

An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus or a system failure



An automatic commit occurs in the following circumstances:

- A DDL statement issued
- A DCL statement issued
- Normal exit from SQL Developer or SQL*Plus, without explicitly issuing COMMIT or ROLLBACK statements

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 issued by the current user.

The affected rows are *locked*; other users cannot change the data in the affected rows.

ORACLE

State of the Data After COMMIT

Data changes are saved in the database.

All save points are erased.



The previous state of the data is overwritten

All users can view the results.

Locks on the affected rows are released; those rows are available for other users to manipulate

ORACLE

Committing Data

- Make the changes:

```
DELETE FROM inventories  
WHERE product_id = 2458 ;
```

1 rows deleted

```
INSERT INTO Inventories  
VALUES (2670, 6, 159);
```

1 rows inserted

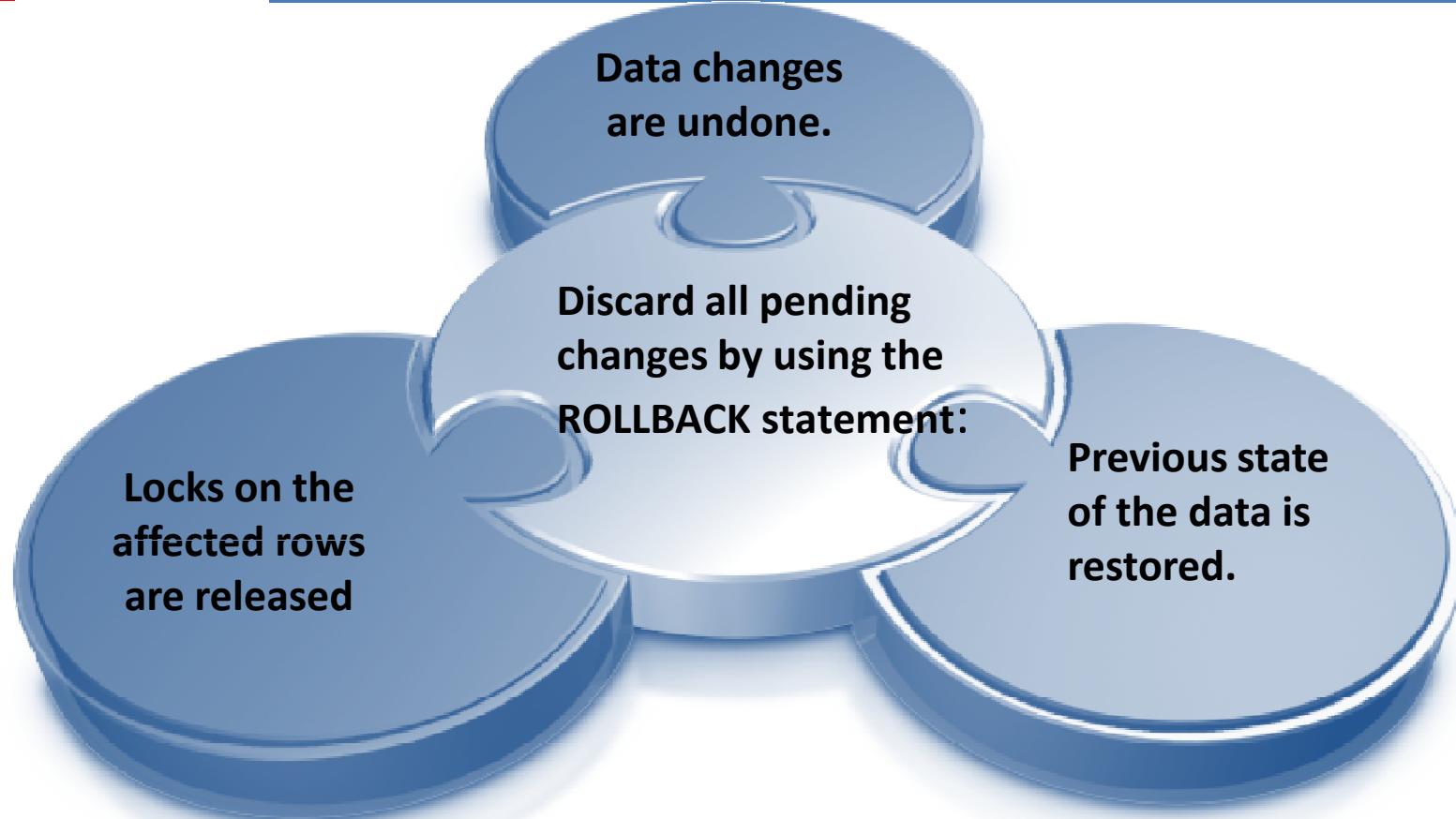
- Commit the changes:

```
COMMIT;
```

COMMIT succeeded.

ORACLE

State of the Data After ROLLBACK



```
DELETE FROM copy_emp;  
ROLLBACK ;
```

ORACLE

State of the Data After ROLLBACK: Example

```
DELETE FROM order_items;
603 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM order_items WHERE product_id = 2348;
1 row deleted.

SELECT * FROM order_id WHERE product_id = 2348;
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.

Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with the 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
 - Writers wait for writers



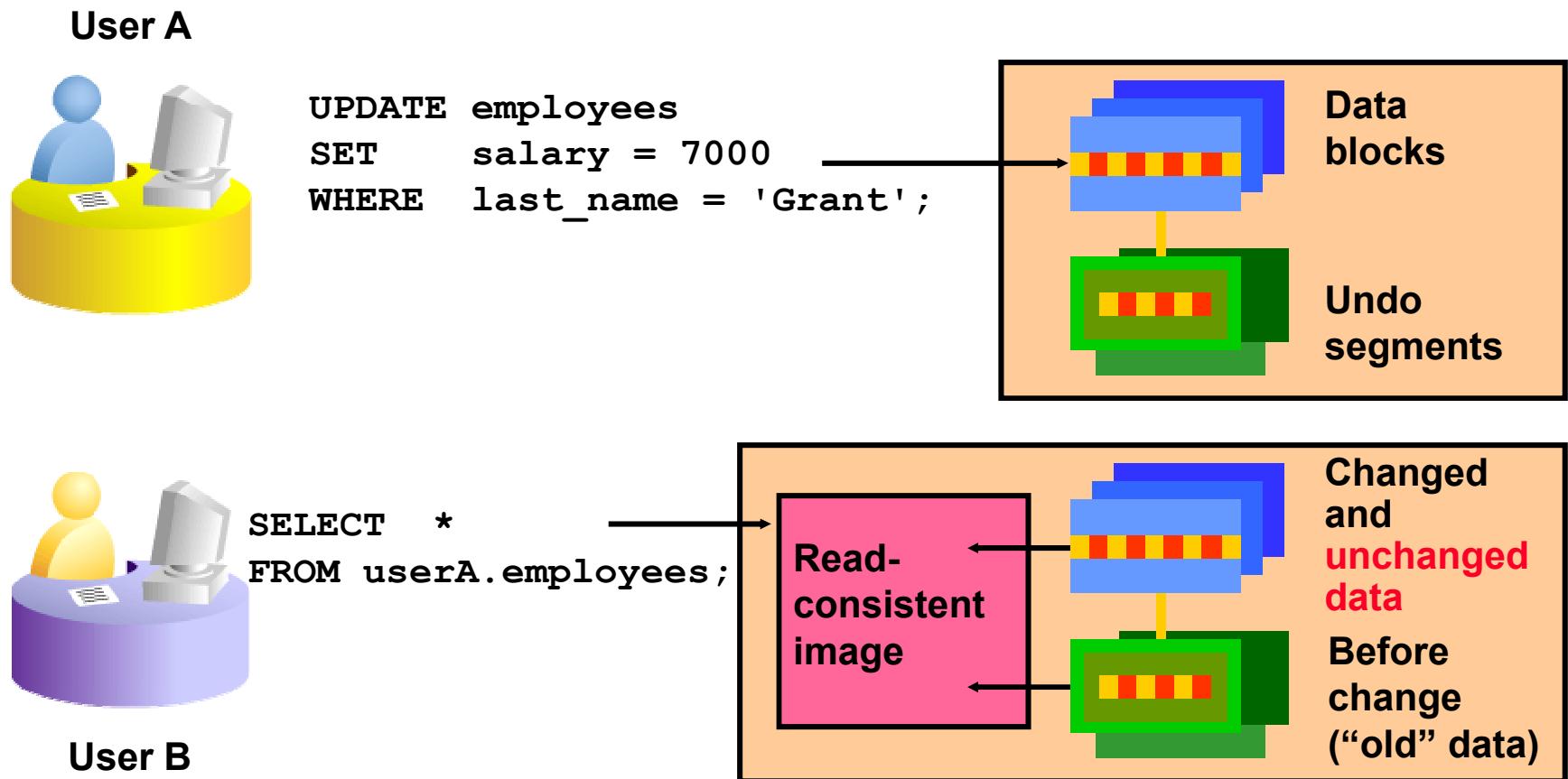
FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the ORDERS table where ORDER_id is 2348.

```
SELECT order_id, order_date, order_mode, customer_id  
FROM orders  
WHERE order_id = '2348'  
FOR UPDATE  
ORDER BY order_id;
```

- If the SELECT statement attempts to lock a row that is locked by another user, the database waits until the row is available, and then returns the results of the SELECT statement.

Implementing Read Consistency



FOR UPDATE Clause: Examples

- You can use the FOR UPDATE clause in a SELECT statement against multiple tables.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

- Rows from both the EMPLOYEES and DEPARTMENTS tables are locked.
- Use FOR UPDATE OF *column_name* to qualify the column you intend to change, then only the rows from that specific table are locked.

- The following statements produce the same results:

```
DELETE FROM copy_emp;
```

```
TRUNCATE TABLE copy_emp;
```

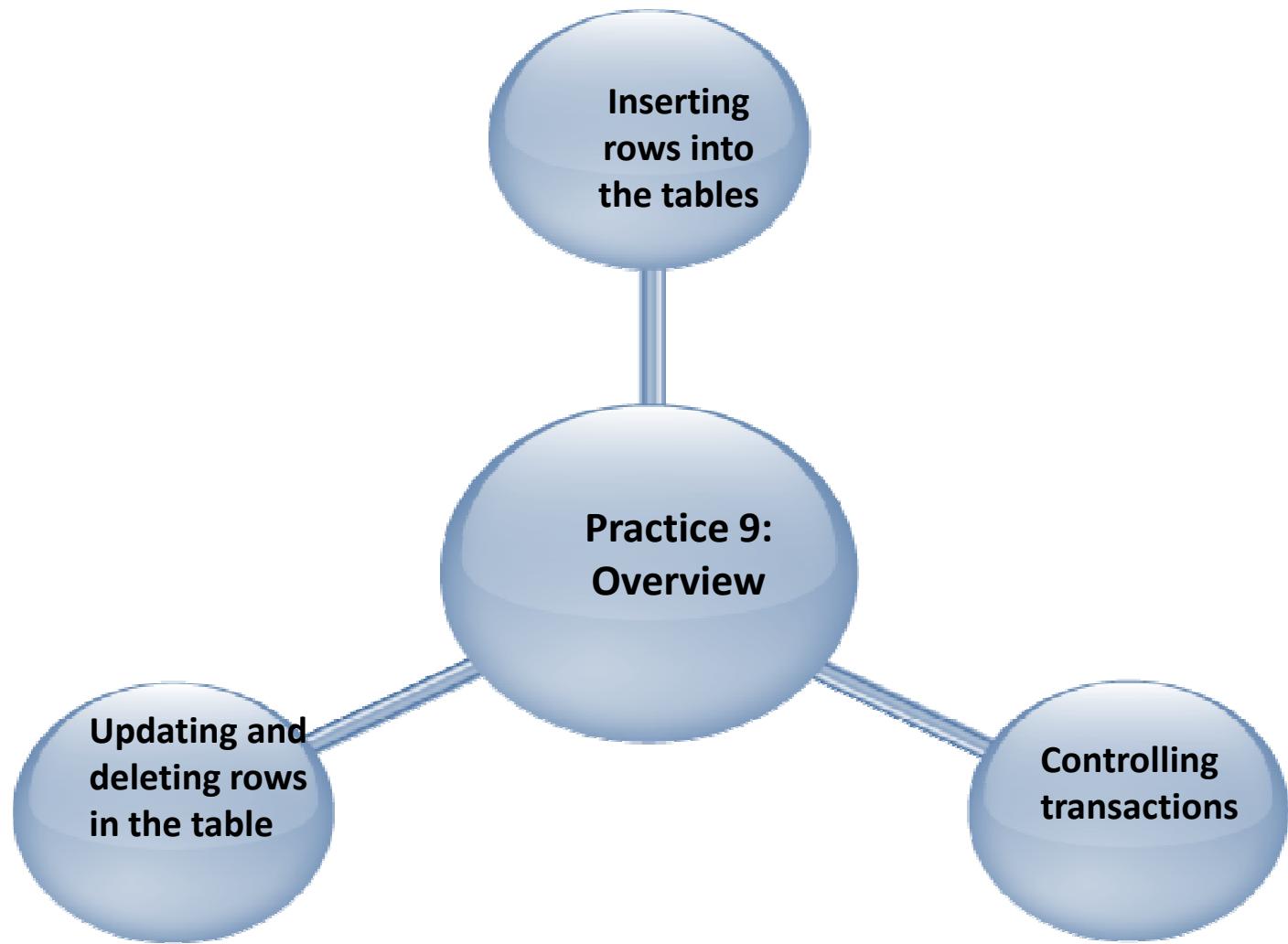
- 1.True
- 2.False

- 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
TRUNCATE	Removes all rows from a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes
FOR UPDATE clause in SELECT	Locks rows identified by the SELECT query

Practice 9: Overview

This practice covers the following topics:



ORACLE