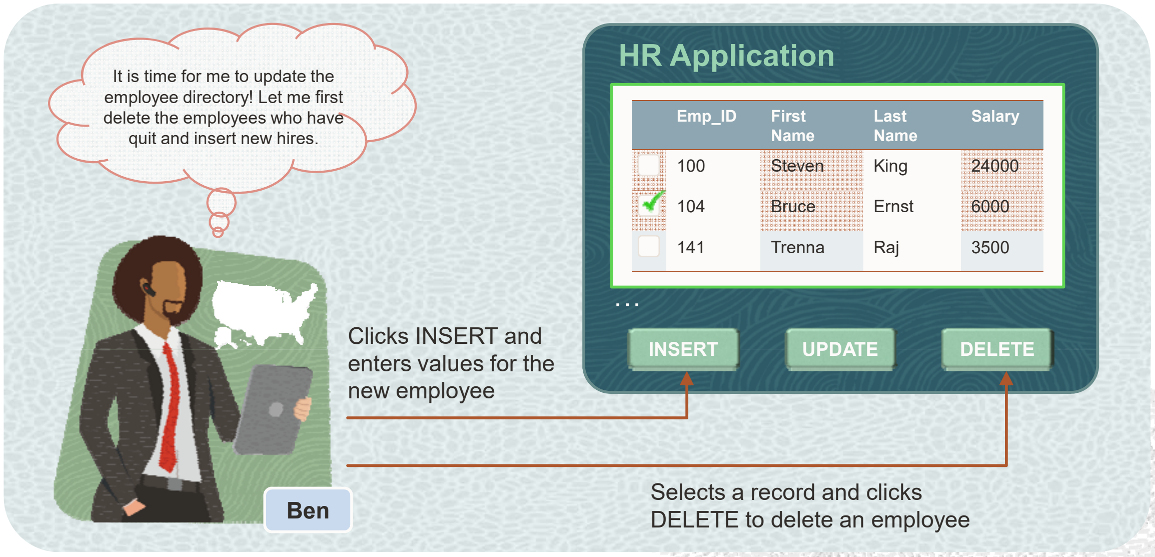
ORACLE

10

Managing Tables Using DML Statements in Oracle

HR Application Scenario


It is time for me to update the employee directory! Let me first delete the employees who have quit and insert new hires.
Ben
HR Application
INSERT
UPDATE
DELETE
Clicks INSERT and enters values for the new employee
Selects a record and clicks DELETE to delete an employee

	Emp_ID	First Name	Last Name	Salary
☐	100	Steven	King	24000
✓	104	Bruce	Ernst	6000
☐	141	Trenna	Raj	3500

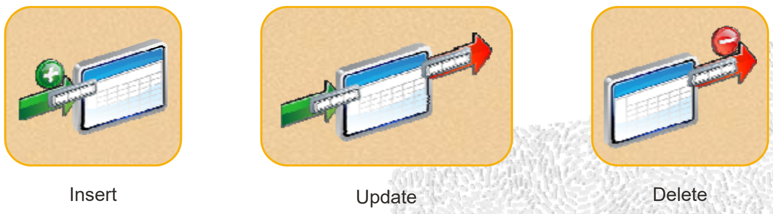
...

Lesson Agenda

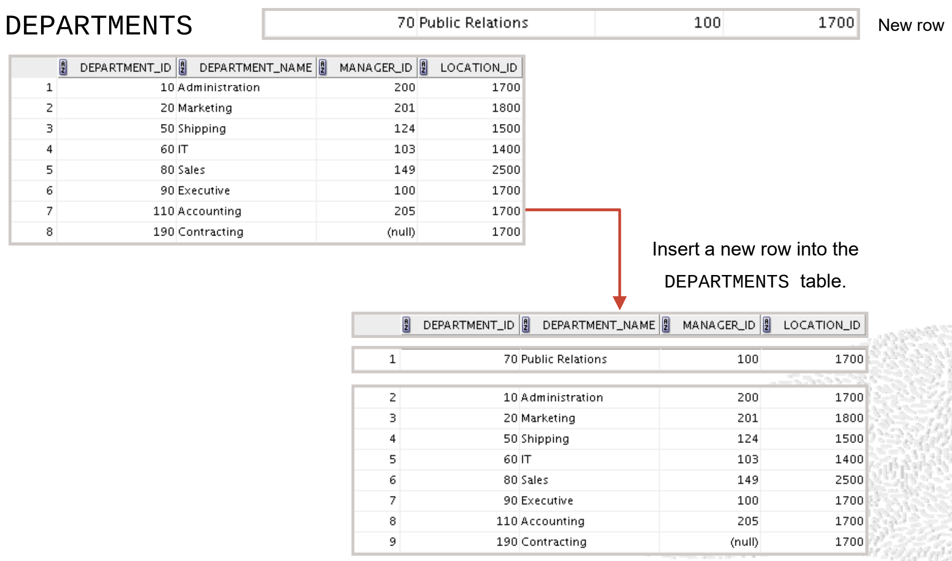
- Adding new rows in a table
- Changing data in a table
- Removing rows from a table
- Database transaction control using `COMMIT, ROLLBACK,` and `SAVEPOINT`
- Read consistency
- Manual Data Locking

Data Manipulation Language

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



Adding a New Row to a Table



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
70	Public Relations	100	1700

	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

	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

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 depts(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```

```
1 row inserted.
```

- Enclose character and date values within single quotation marks.

Inserting Rows with Null Values

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

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

```
1 row inserted.
```

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

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

```
1 row inserted.
```

Inserting Special Values

The CURRENT_DATE function records the current date and time in Oracle.

```
INSERT INTO emps (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',
                  CURRENT_DATE, 'AC_ACCOUNT', 6900,
                  NULL, 205, 110);
```

```
1 row inserted.
```

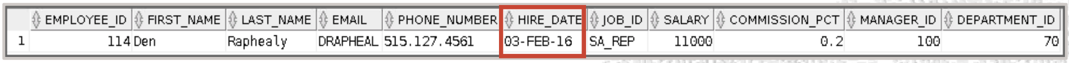
Inserting Specific Date and Time Values

- Add a new employee.

```
INSERT INTO emps
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 2016', 'MON DD, YYYY'),
             'SA_REP', 11000, 0.2, 100, 70);
```

```
1 row inserted.
```

- Verify your addition.

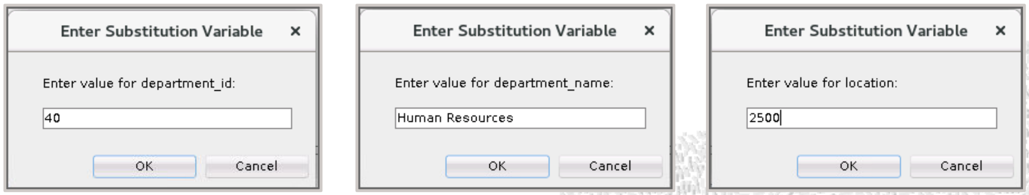


	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-16	SA_REP	11000	0.2	100	70

Creating a Script

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

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



Enter value for department_id: 40

Enter value for department_name: Human Resources

Enter value for location: 2500

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%';
```

```
5 rows inserted.
```

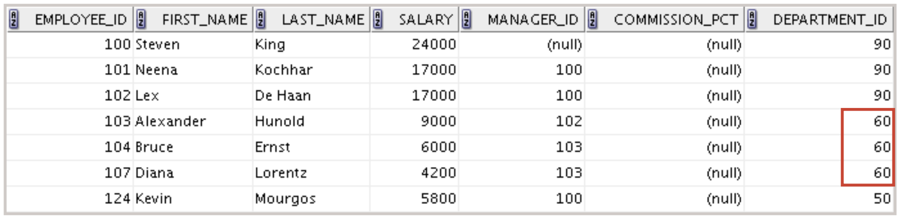
- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.
- Insert all the rows returned by the subquery in the table, `sales_reps`.

Lesson Agenda

- Adding new rows in a table
- Changing data in a table
- Removing rows from a table
- Database transaction control using `COMMIT, ROLLBACK,` and `SAVEPOINT`
- Read consistency
- Manual Data Locking

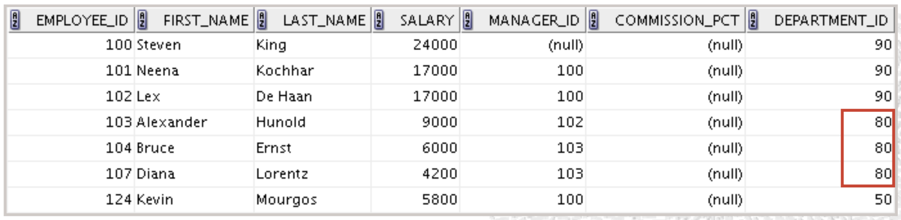
Changing Data in a Table

`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:



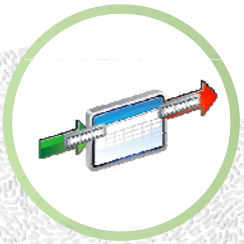
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)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

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 employees
SET    department_id = 50
WHERE  employee_id = 113;
```

```
1 row updated.
```

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

```
UPDATE    copy_emp
SET       department_id = 110;
```

```
22 rows updated
```

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

Updating Two Columns with a Subquery

Update employee 103’s job and salary to match those of employee 205.

```
UPDATE   employees
SET      (job_id,salary)  = (SELECT  job_id,salary
                            FROM    employees
                              WHERE   employee_id = 205)
WHERE    employee_id    =  103;
```

```
1 row updated.
```

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  employees
SET     department_id   =  (SELECT department_id
                            FROM employees
                            WHERE employee_id = 100)
WHERE   job_id          =  (SELECT job_id
                            FROM employees
                            WHERE employee_id = 200);
```

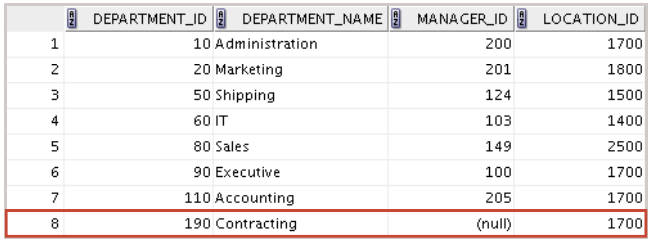
```
1 row updated.
```

Lesson Agenda

- Adding new rows in a table
- Changing data in a table
- Removing rows from a table
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking

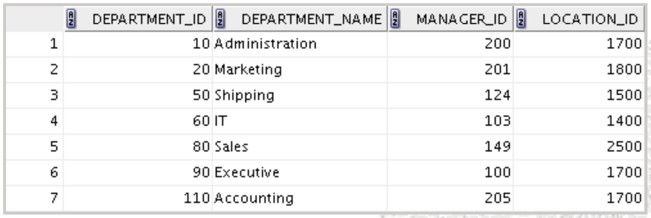
Removing a Row from a Table

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:

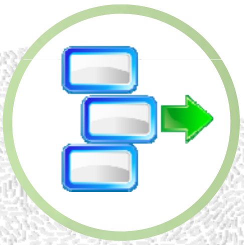


	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700

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 depts
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
```

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 emps
WHERE  department_id IN
                (SELECT department_id
                 FROM   departments
                 WHERE  department_name
                        LIKE '%Public%');
```

```
1 row deleted.
```

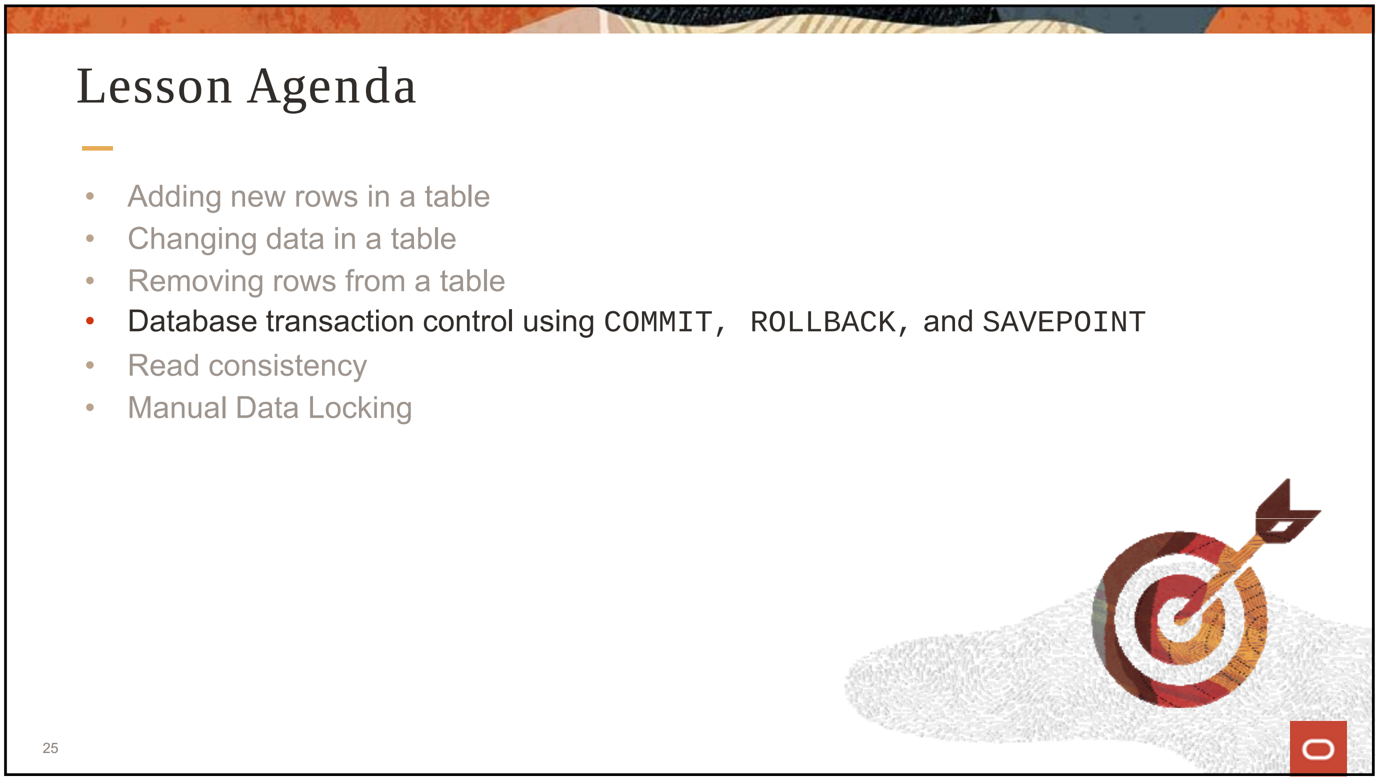
TRUNCATE Statement

- 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 be undone
-

```
TRUNCATE TABLE table_name;
```

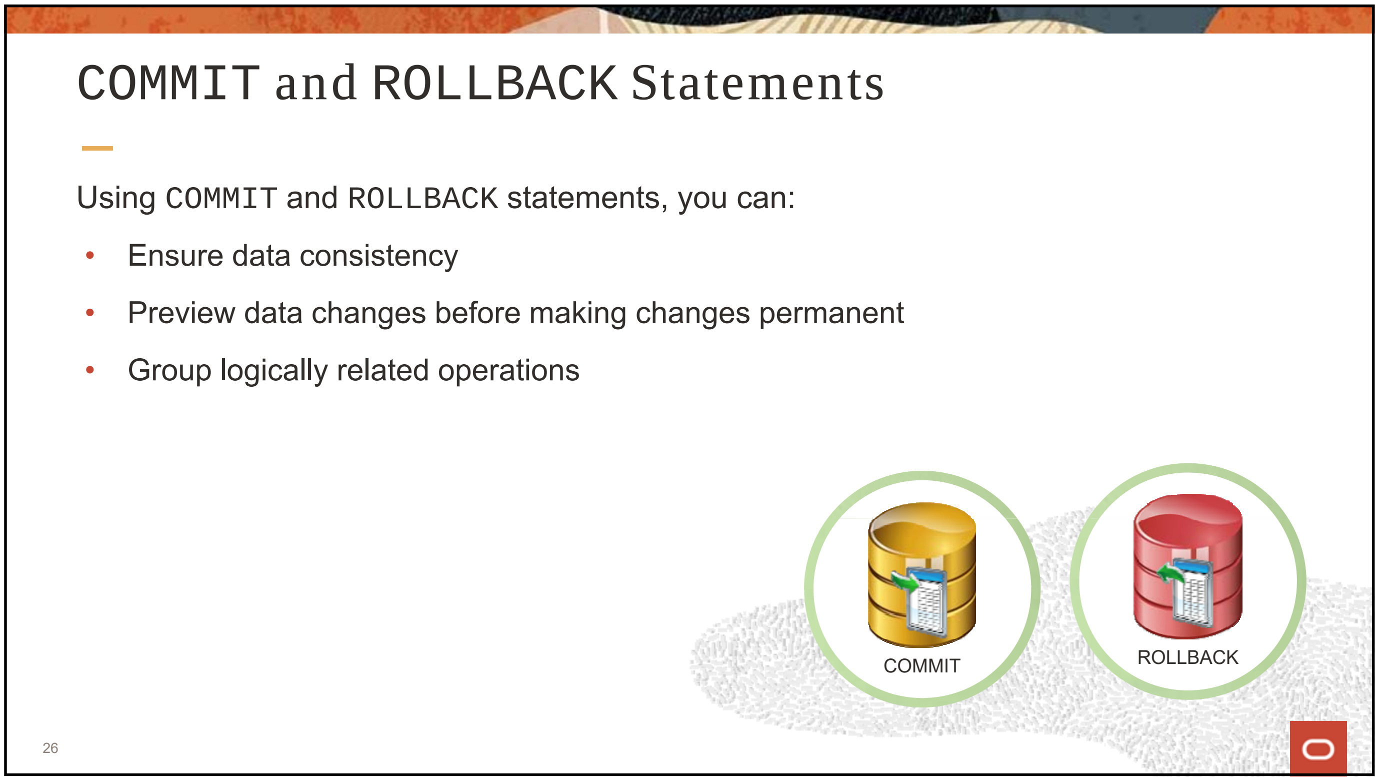
- Example:

```
TRUNCATE TABLE copy_emp;
```



Lesson Agenda

- Adding new rows in a table
- Changing data in a table
- Removing rows from a table
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking



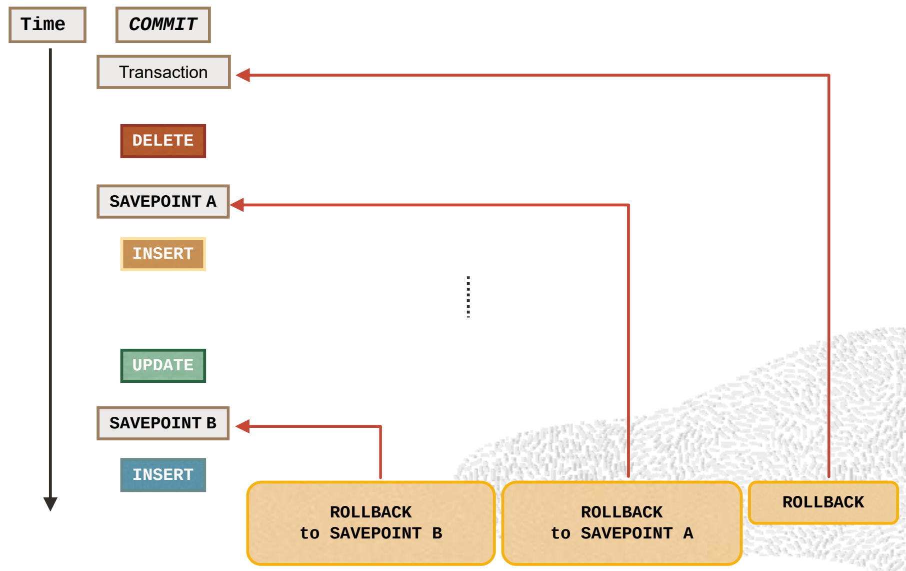
`COMMIT` and `ROLLBACK` Statements

Using `COMMIT` and `ROLLBACK` statements, you can:

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

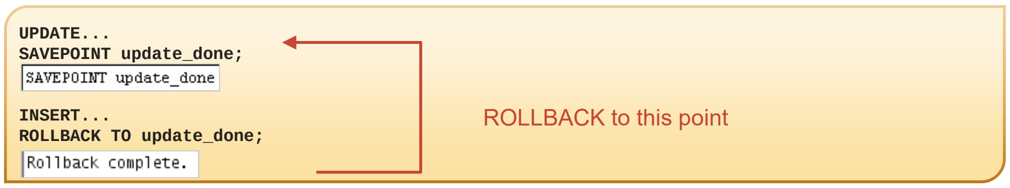
COMMIT
ROLLBACK

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

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

Implicit Transaction Processing

- An automatic commit occurs when:
  - A DDL statement is issued
  - A DCL statement is issued
  - There is a normal exit from SQL Developer or SQL*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements
- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus, or a system failure.

Committing Data

- Make the changes:

```
DELETE FROM employees
WHERE employee_id = 113;
1 row deleted.
INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

- Commit the changes:

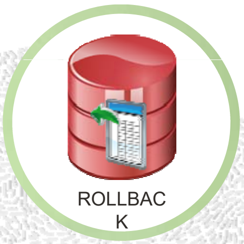
```
COMMIT;
Commit complete.
```

State of 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;
ROLLBACK ;
```



State of Data After ROLLBACK: Example

```
DELETE FROM test;
4 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.
```

Lesson Agenda

- Adding new rows in a table
- Changing data in a table
- Removing rows from a table
- Database transaction control using `COMMIT, ROLLBACK,` and `SAVEPOINT`
- Read consistency
- Manual Data Locking

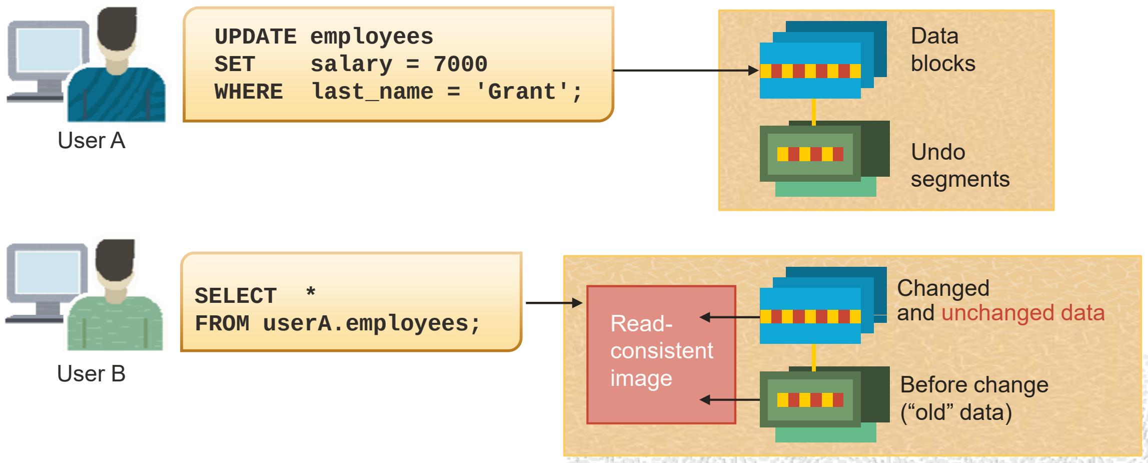
33

Read Consistency

- Read consistency guarantees a consistent view of data at all times.
- Changes made by one user do not conflict with the changes made by another user.

34

Implementing Read Consistency

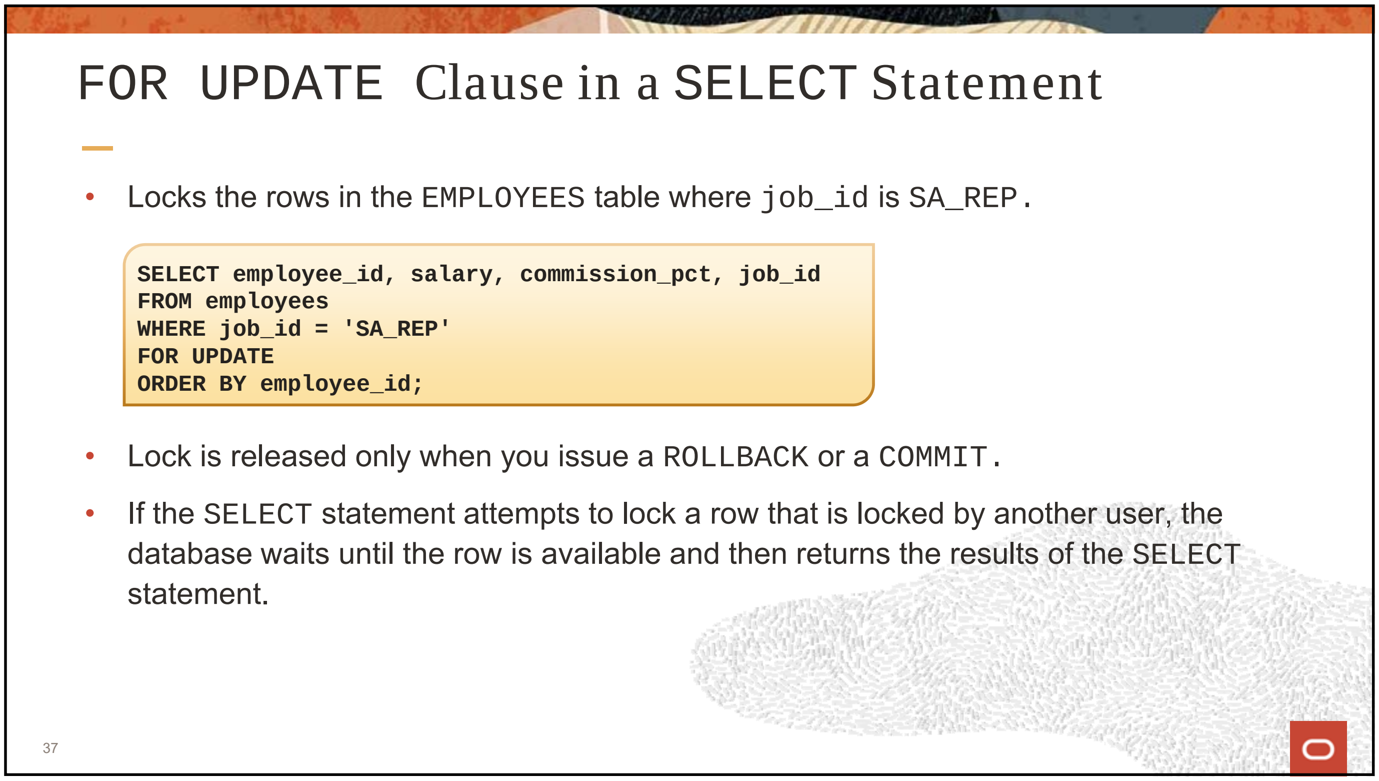


```
UPDATE employees
SET    salary = 7000
WHERE  last_name = 'Grant';
```

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

Lesson Agenda

- Adding new rows in a table
- Changing data in a table
- Removing rows from a table
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking

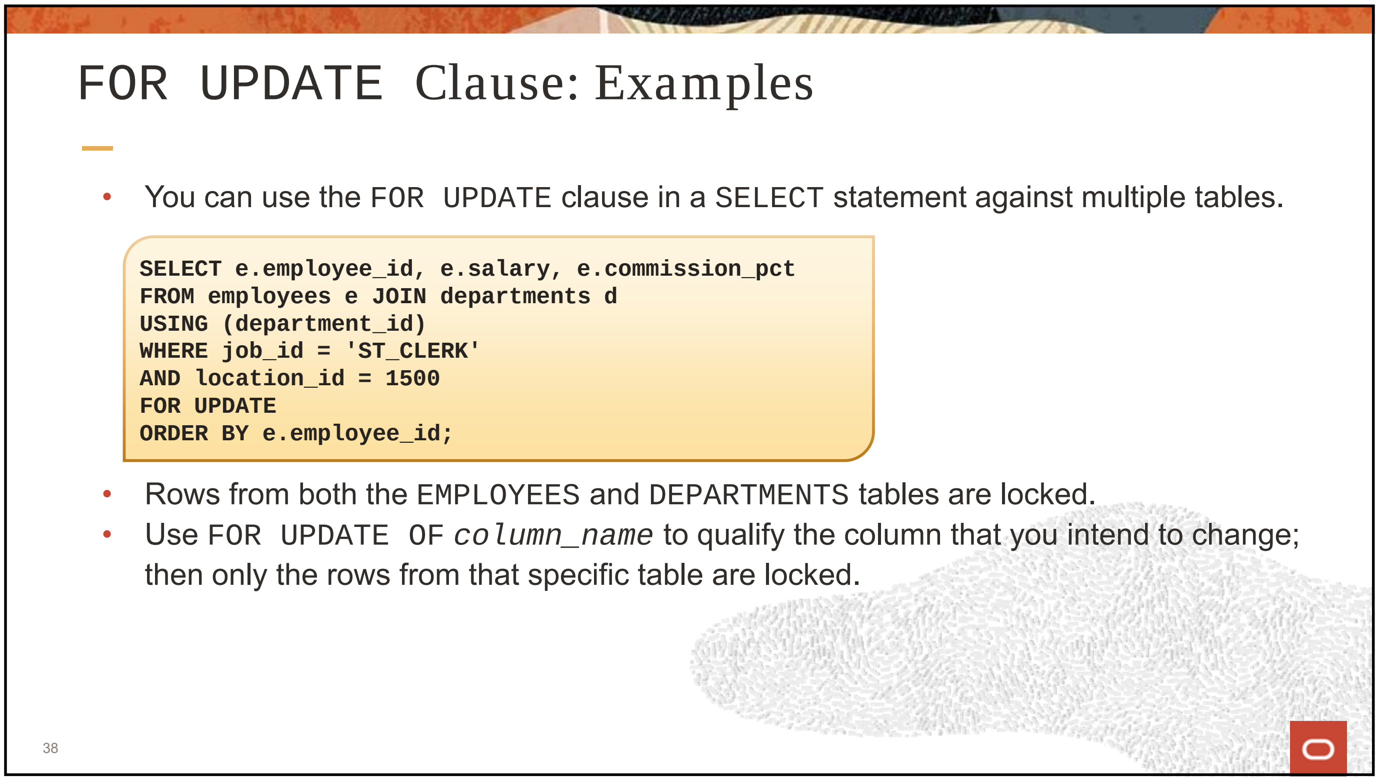


FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the EMPLOYEES table where job_id is SA_REP.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

- Lock is released only when you issue a ROLLBACK or a COMMIT.
- 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.



FOR UPDATE Clause: Examples

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

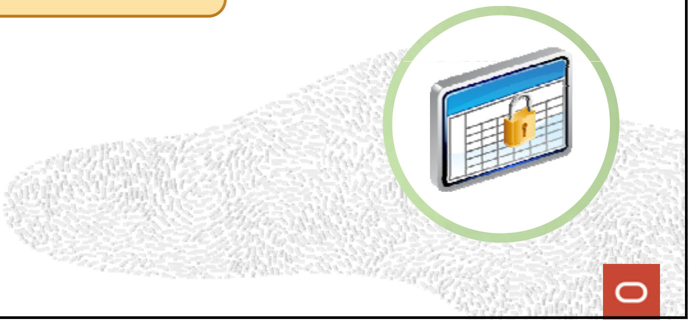
```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

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

LOCK TABLE Statement

- Use the `LOCK TABLE` statement to lock one or more tables in a specified mode.
- This manually overrides automatic locking.
- Tables are locked until you `COMMIT` or `ROLLBACK`.

```
LOCK TABLE table_name
IN [ROW SHARE/ROW EXCLUSIVE/SHARE UPDATE/SHARE/
    SHARE ROW EXCLUSIVE/ EXCLUSIVE] MODE
[NOWAIT];
```



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
`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