8

Data Manipulation

Objectives

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

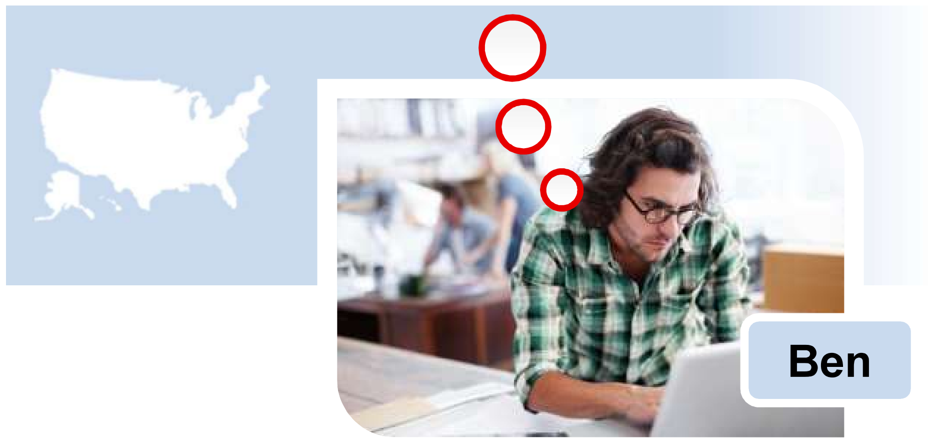
- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Control transactions

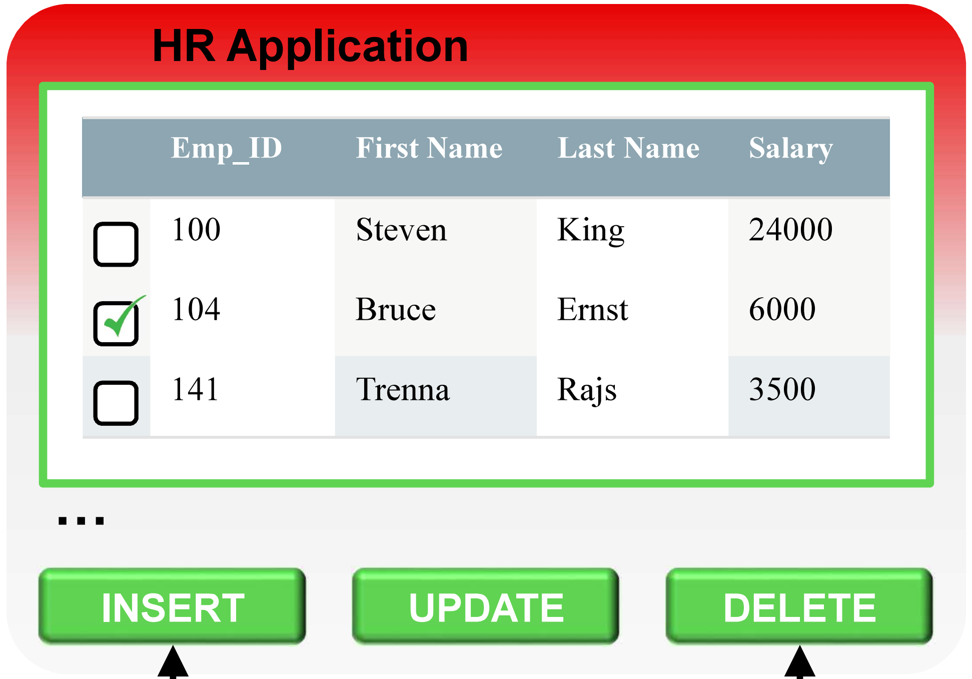
Course Roadmap

- Lesson 1: Introduction
- Unit 1: Retrieving, Restricting, and Sorting Data
- Unit 2: Joins, Subqueries, and Set Operators
- **Unit 3: DML and DDL**
  - **Lesson 10: Managing Tables Using DML Statements** — **You are here!**
  - Lesson 11: Introduction to Data Definition Language
  - Lesson 12: Other Schema Objects

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.





HR Application

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

...

Clicks INSERT and enters values for the new employee.

Selects a record and clicks DELETE to delete an employee.

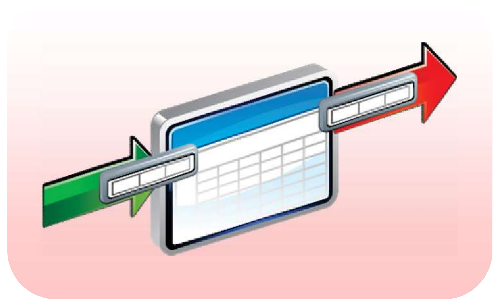
Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

Data Manipulation Language

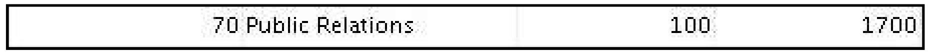
- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

Insert


Update

Delete

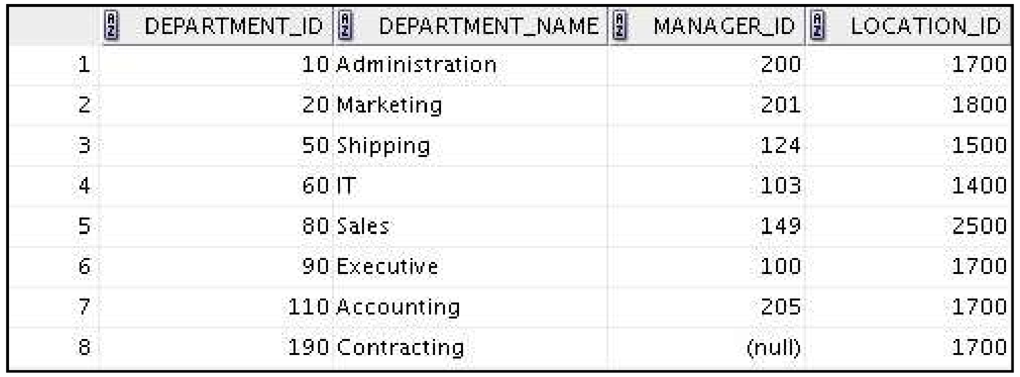
Adding a New Row to a Table



70	Public Relations	100	1700

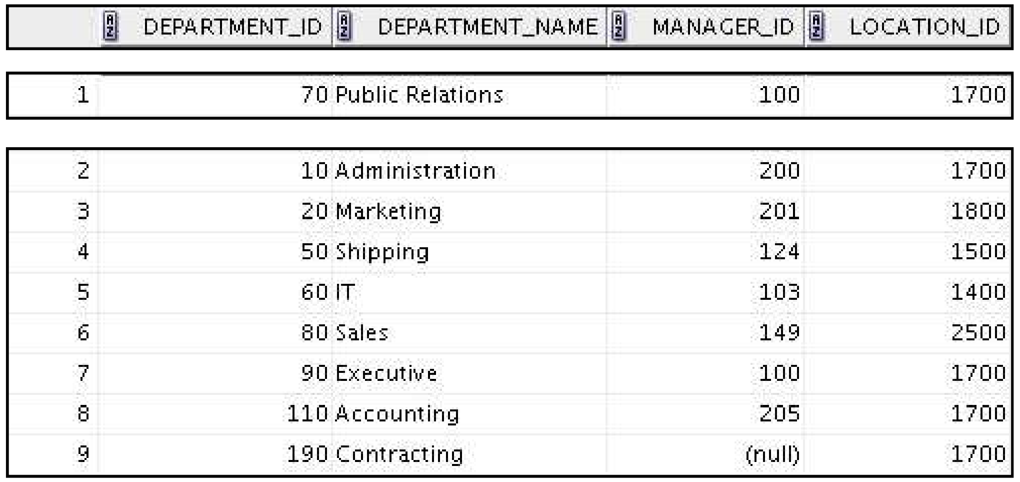
New row

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

Insert a 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

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 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.

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
```

Inserting Special Values

- The SYSDATE function records the current date and time.

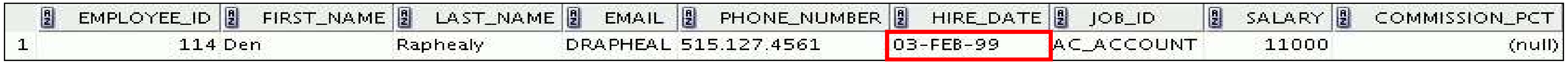
```
INSERT INTO employees (employee_id,
                  first_name, last_name,
                  email, phone_number,
                  hire_date, job_id, salary,
                  commission_pct, manager_id,
                  department_id)
VALUES           (113,
                  'Louis', 'Popp',
                  'LPOPP', '515.124.4567',
                  SYSDATE, 'AC_ACCOUNT', 6900,
                  NULL, 205, 100);
1 rows inserted
```

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 rows inserted
```

- Verify your addition.

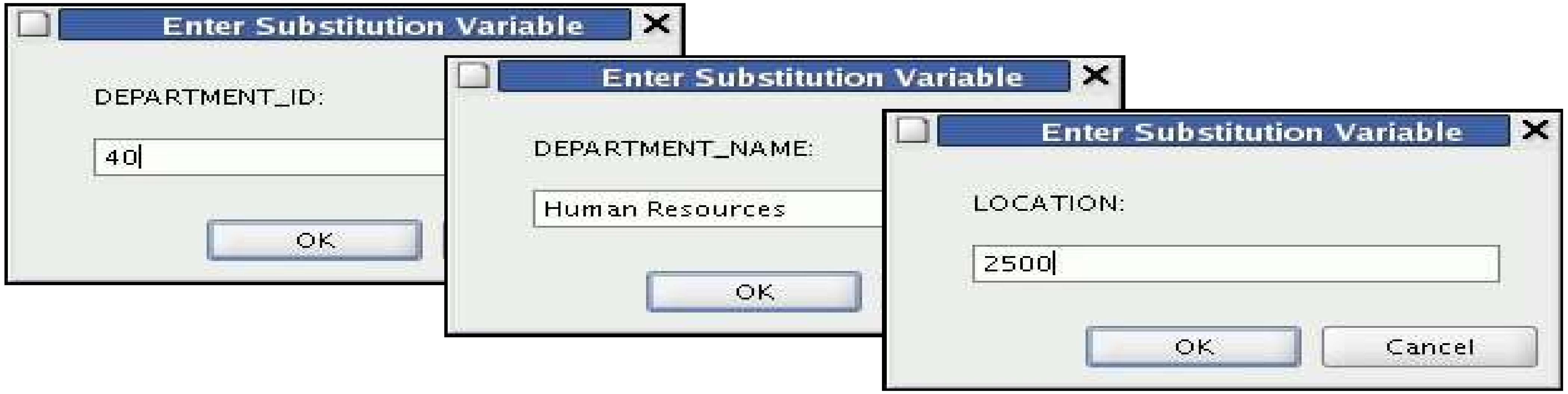


	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	(null)

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);
```



Enter Substitution Variable

DEPARTMENT_ID: 40

Enter Substitution Variable

DEPARTMENT_NAME: Human Resources

Enter Substitution Variable

LOCATION: 2500

Copying Rows

- 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
```

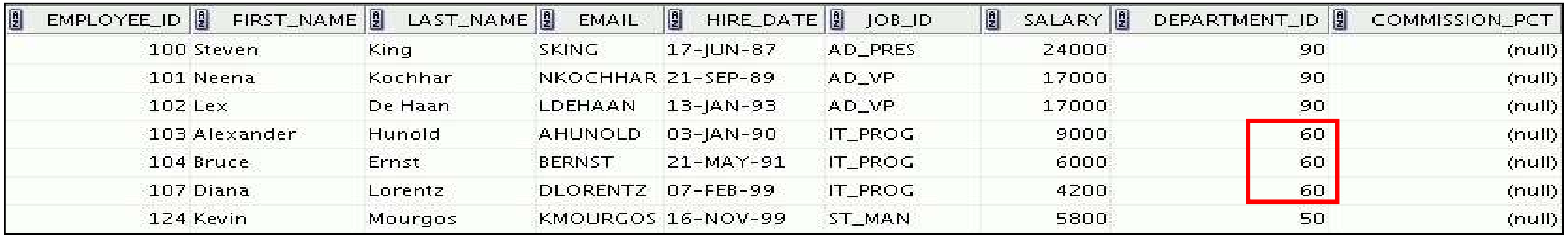
- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

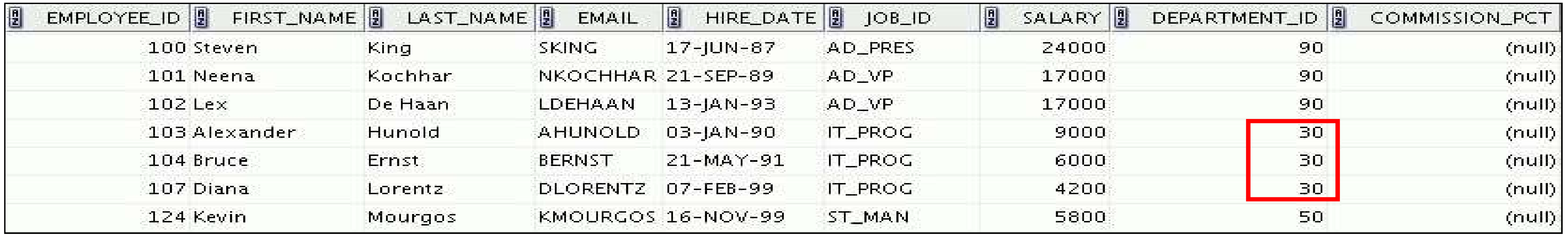
Changing Data in a Table

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:



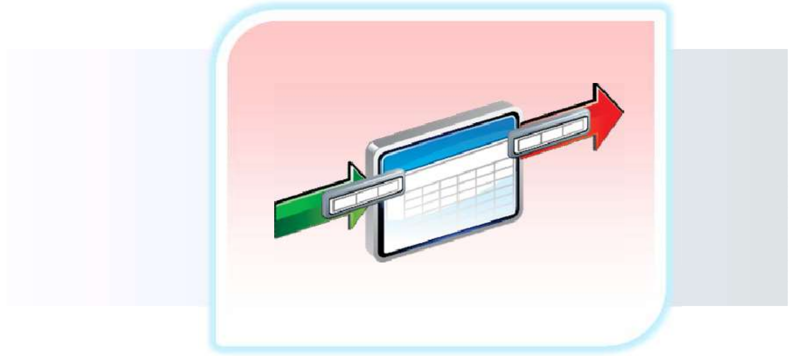
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_PCT
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	(null)
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	(null)
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	(null)
103	Alexander	Hunold	AHUNOLD	03-JAN-90	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)

UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement:

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

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



Updating Rows in a Table

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

```
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
```

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
```

Updating Rows Based

- Use subqueries in `UPDATE` statements to update rows in a table based on values from another table:

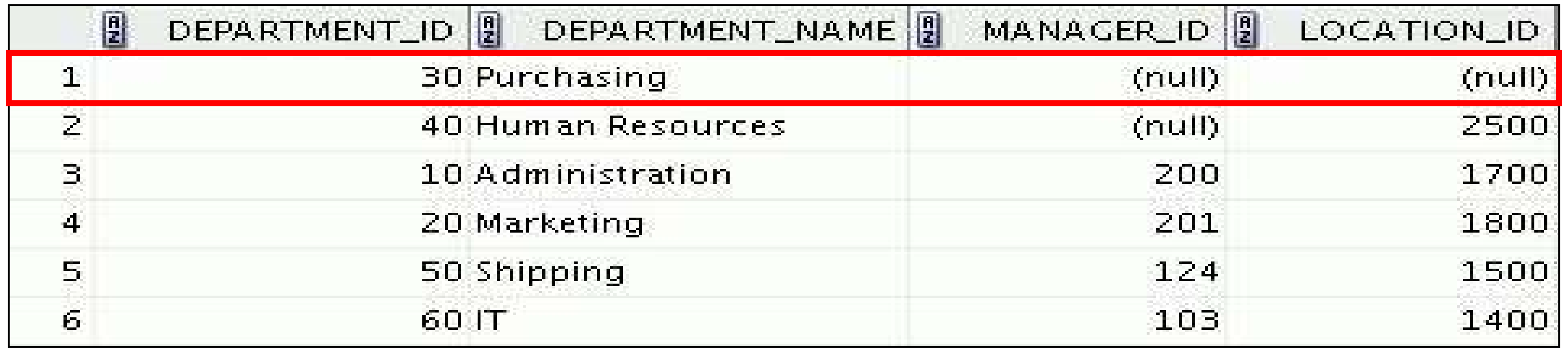
```
UPDATE   copy_emp
SET      department_id   =   (SELECT department_id
                               FROM employees
                               WHERE employee_id = 100)
WHERE    job_id          =   (SELECT job_id
                               FROM employees
                               WHERE employee_id = 200);
1 rows updated
```

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

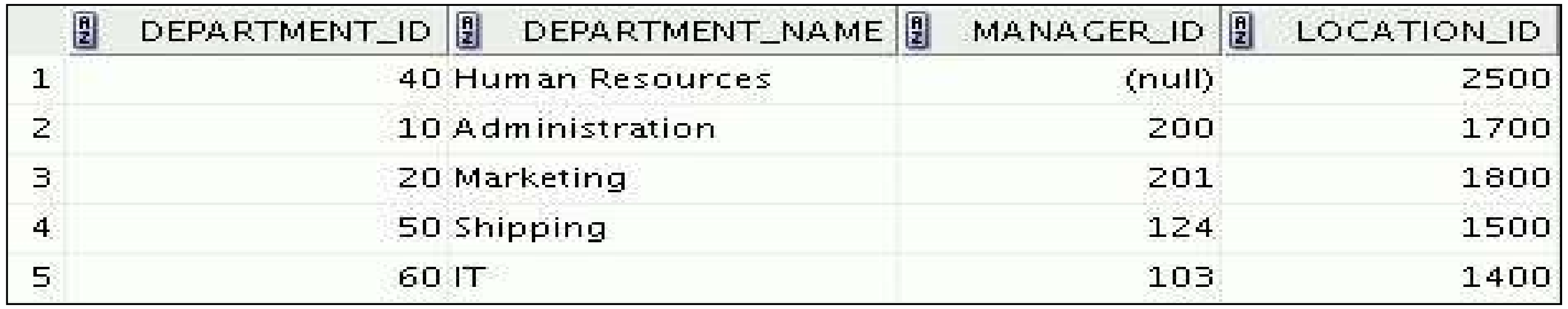
Removing a Row from a Table

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:

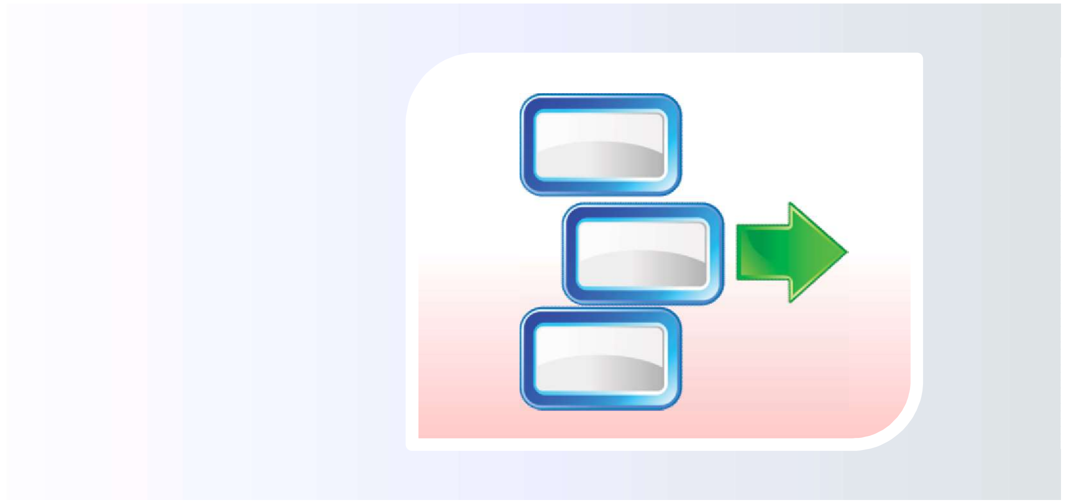


	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	40	Human Resources	(null)	2500
2	10	Administration	200	1700
3	20	Marketing	201	1800
4	50	Shipping	124	1500
5	60	IT	103	1400

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 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
```

Deleting Rows Based

- 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
```

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;
```

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` 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

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.

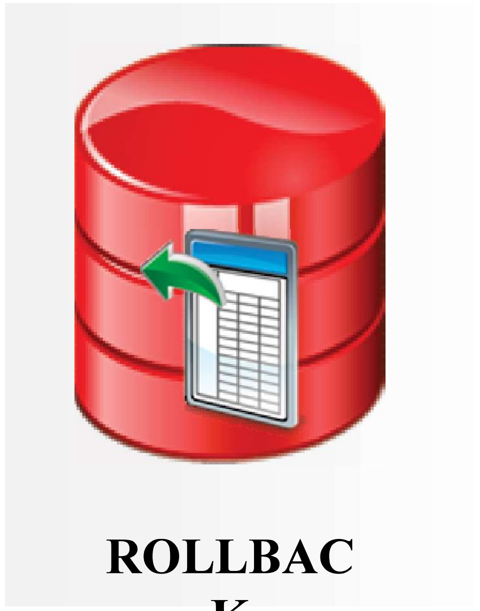
Database transaction on table/s →

Advantages of COMMIT and ROLLBACK Statements

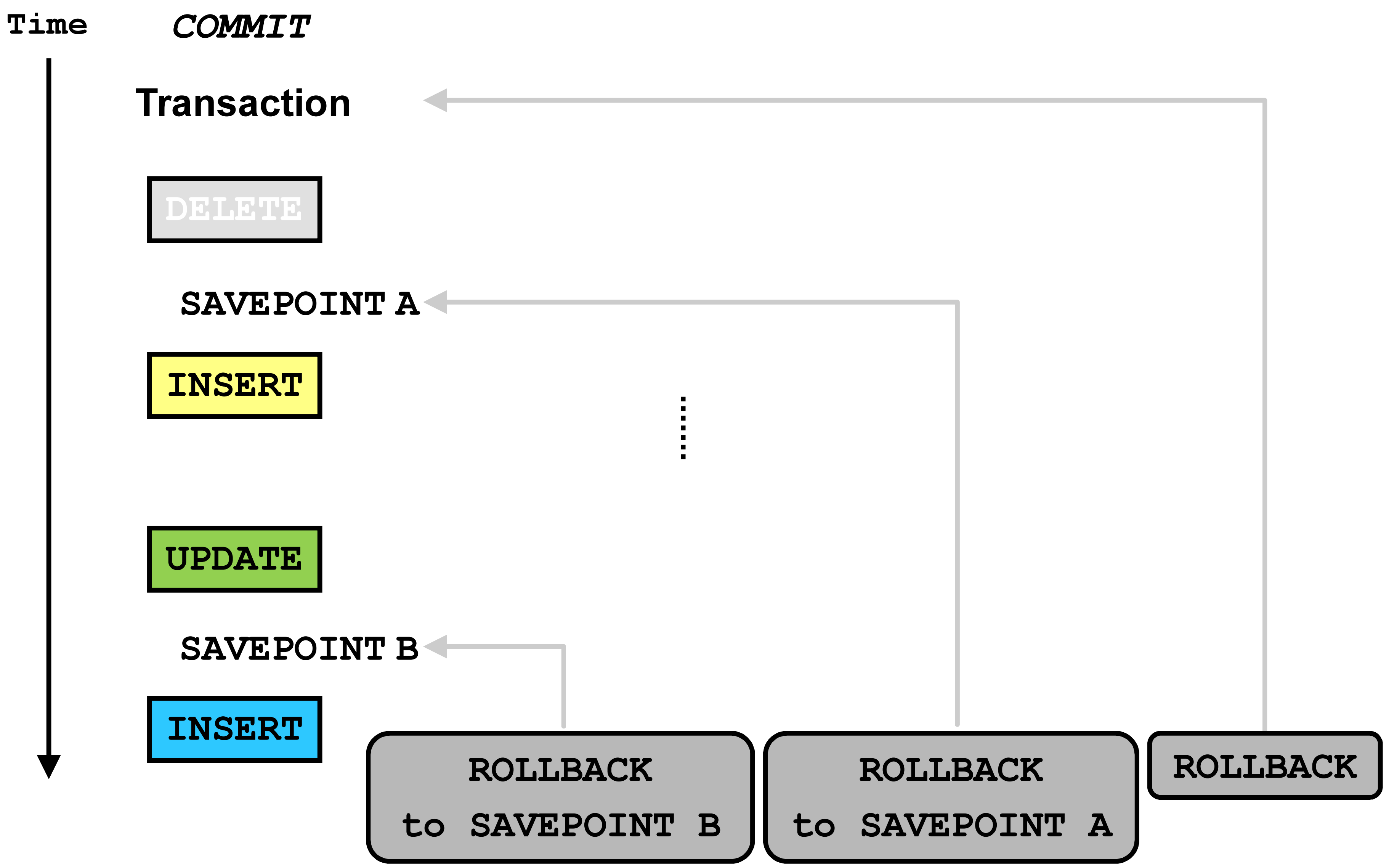
Using COMMIT and ROLLBACK statements, you can:

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

COMMIT

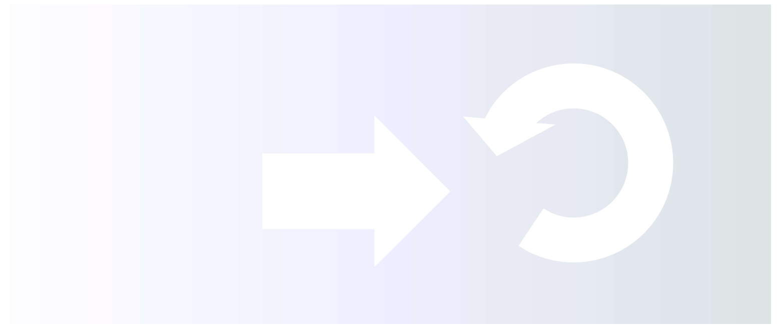

ROLLBAC
K

Explicit Transaction Control Statements



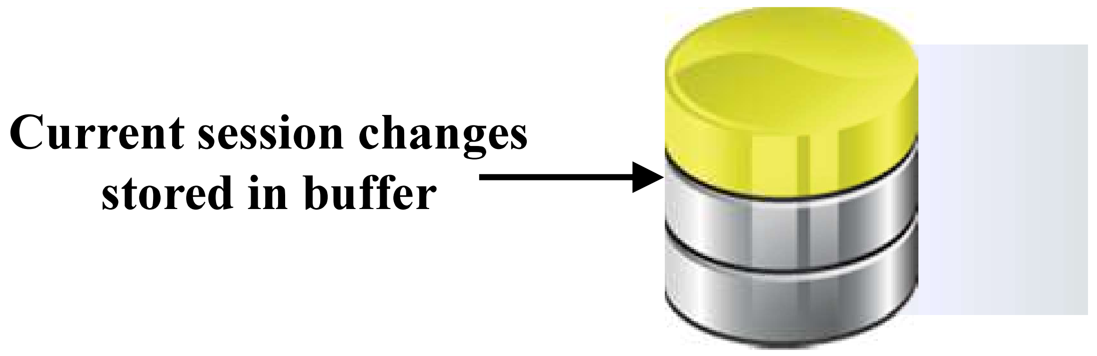
Implicit Transaction Processing

- An automatic commit occurs in the following circumstances:
  - A DDL statement is issued
  - A DCL statement is issued
  - 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.



State of Data Before `COMMIT` or `ROLLBACK`

- You can recover the data of the previous state.
- You can review the results of the DML operations by using the `SELECT` statement in the current session.
- Other sessions *cannot* view the results of the DML statements issued by the current session.
- The affected rows are *locked*; other sessions cannot change the data in the affected rows.



State of Data After COMMIT

- Data changes are saved in the database.
- The previous state of the data is overwritten.
- All sessions can view the results.
- Locks on the affected rows are released; those rows are available for other sessions to manipulate.
- All savepoints are erased.

COMMIT

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
```

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 ;
```

ROLLBAC
K

State of Data After ROLLBACK: Example

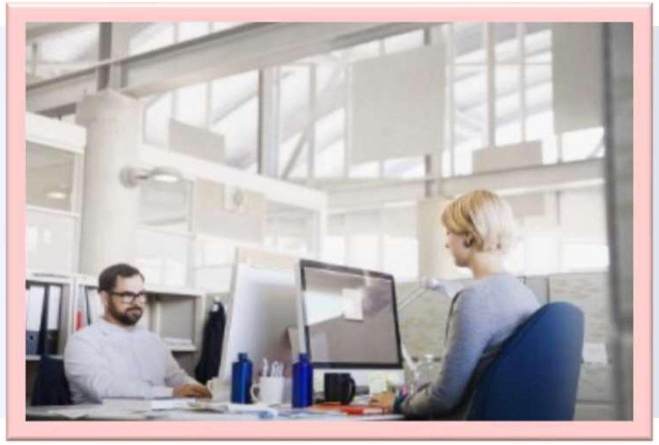
```
DELETE FROM test;
4 rows deleted.

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

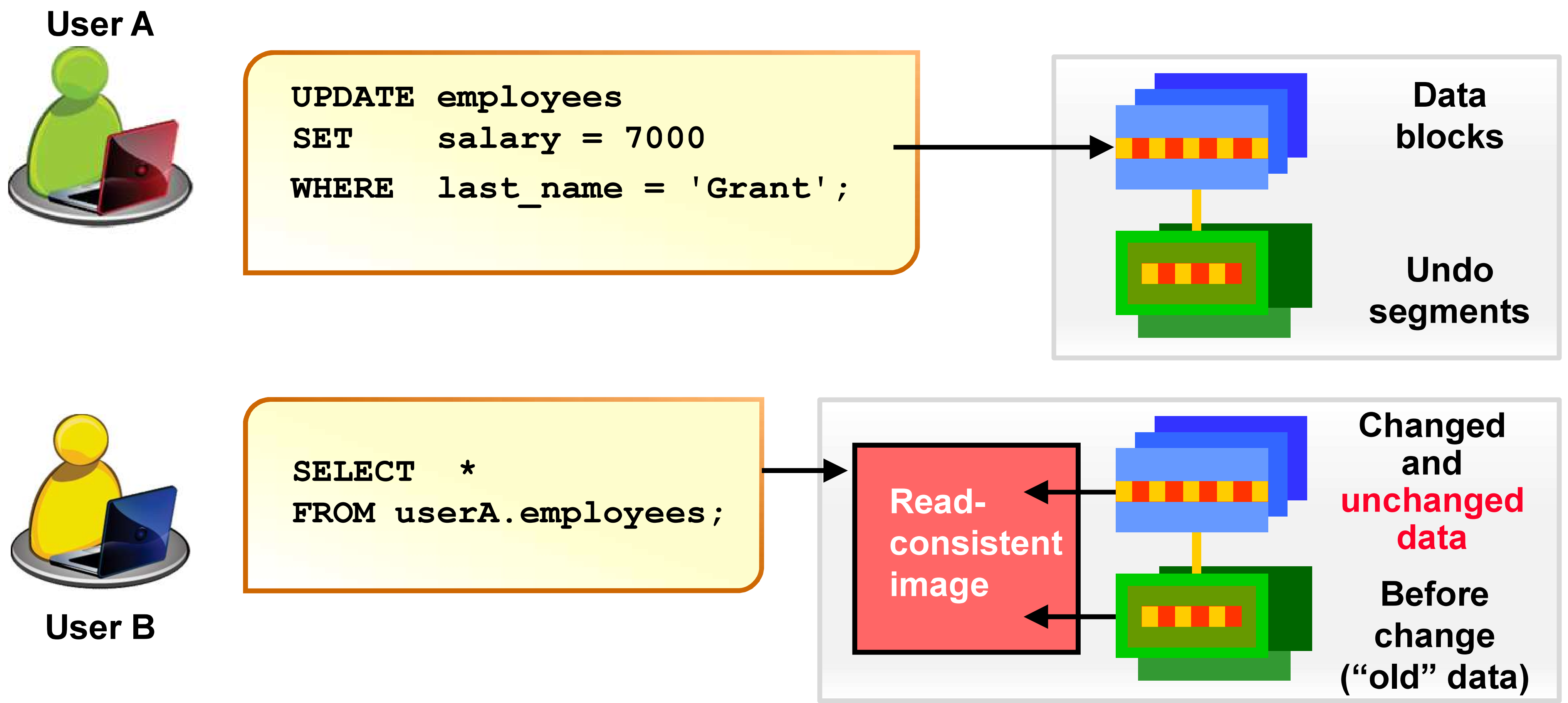
- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

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.
- Read consistency ensures that, on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers
  - Writers wait for writers



Implementing Read Consistency



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

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

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

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.

Quiz

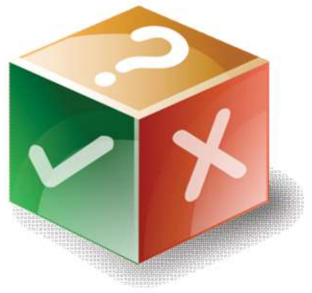
The following statements produce the same results:

```
DELETE FROM copy_emp;
```

```
TRUNCATE TABLE copy_emp;
```

a. True

b. False



Summary

In this lesson, you should have learned how to:

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 8: Overview

This practice covers the following topics:

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