DATABASE MANAGEMENT SYSTEMS

Subject Teacher: Zartasha Baloch

MANIPULATING DATA

Lecture # 23 & 24

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

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

70	Public Relations	100	1700

New row

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

…insert a new row into the DEPARMENTS table…

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

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.

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.

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.
```

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.
```

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	

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);
```

Define Substitution Variables

"department_id"	40
"department_name"	Human Resources
"location"	2500

Submit for Execution   Cancel

```
1 row created.
```

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.

CHANGING DATA IN A TABLE

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_F
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	

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 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.
```

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.
```

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.
```

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

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

THE 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 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 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.
```

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.

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.
```

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 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
```

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.

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;
```

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

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);
```

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);
```

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.
```

DATABASE TRANSACTIONS

A database transaction consists of one of the
following:

- DML statements which constitute one consistent change to the data

- One DDL statement
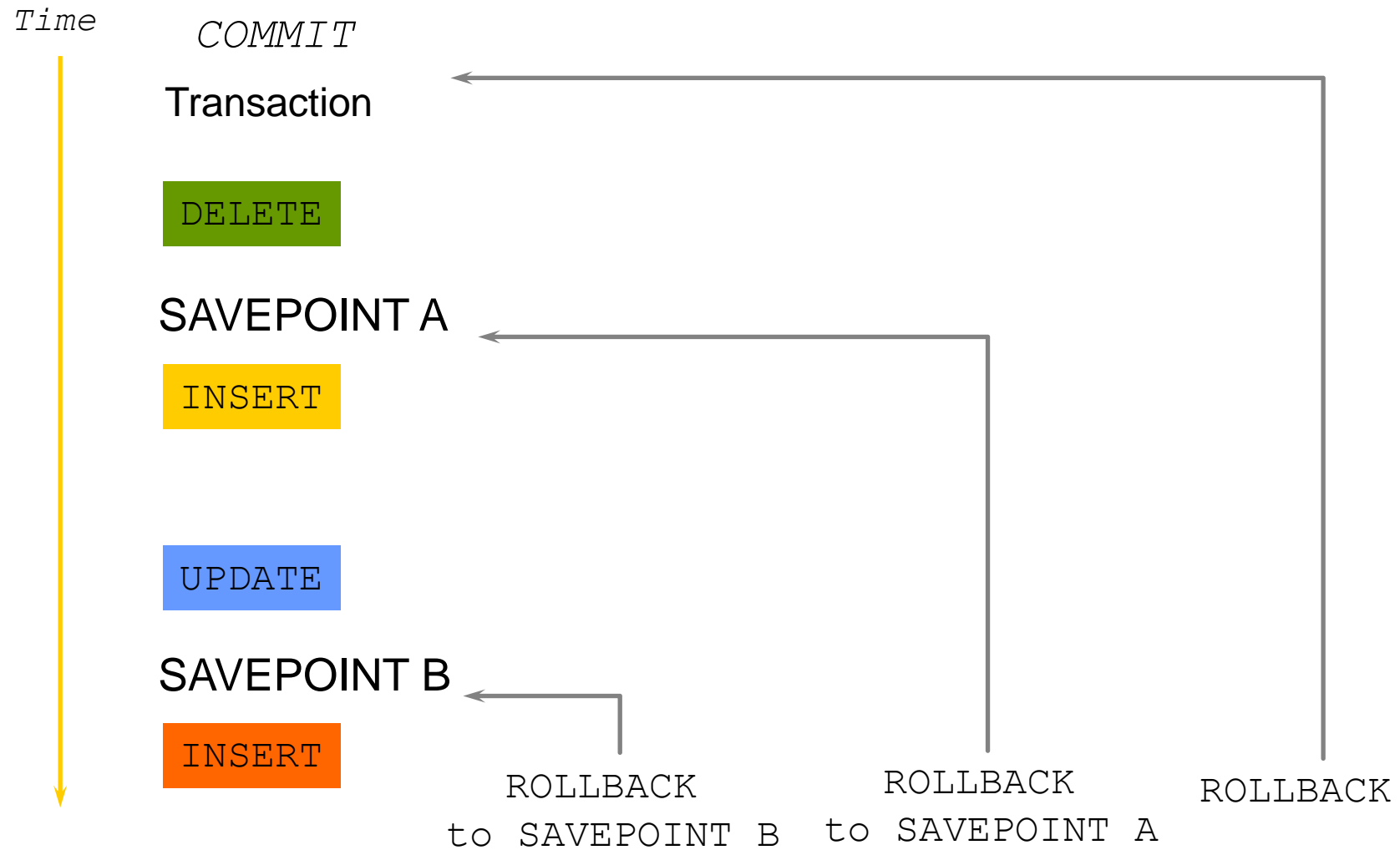
- One DCL statement

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

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

CONTROLLING TRANSACTIONS

Time

COMMIT

Transaction

DELETE

SAVEPOINT A

INSERT

UPDATE

SAVEPOINT B

INSERT

ROLLBACK
to SAVEPOINT B

ROLLBACK
to SAVEPOINT A

ROLLBACK

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.
```

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.

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.

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.

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.
```

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.
```
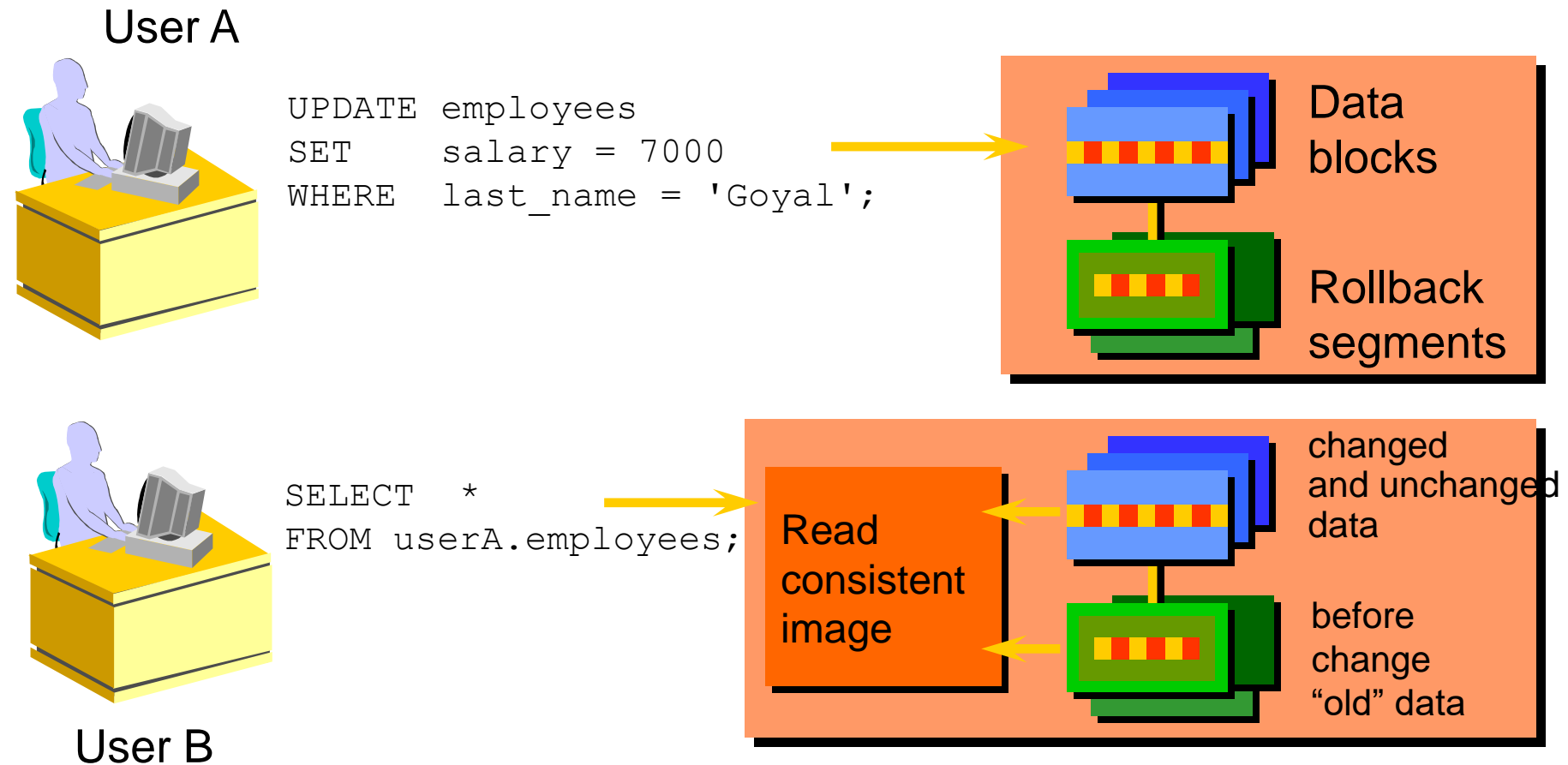
STATEMENT-LEVEL ROLLBACK

- 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 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.

IMPLEMENTATION OF READ CONSISTENCY

User A

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

Data blocks

Rollback segments

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

Read consistent image

changed and unchanged data

before change "old" data

User B

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

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

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