

# 9

## Manipulating Data

# Objectives

**At the end of this lesson, you should be able to:**

- **Describe each DML statement**
- **Insert rows into a table**
- **Update rows in a table**
- **Delete rows from 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

50	DEVELOPMENT	DETROIT
----	-------------	---------

New row

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

“...insert a new row  
into DEPT table...”



DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	DETROIT

# The INSERT Statement

- 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.
- Optionally list the columns in the INSERT clause.

```
SQL> INSERT INTO      dept (deptno, dname, loc)
      2  VALUES      (50, 'DEVELOPMENT', 'DETROIT') ;
1 row created.
```

- List values in the default order of the columns in the table.
- Enclose character and date values within single quotation marks.

# Inserting Rows with Null Values

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

```
SQL> INSERT INTO      dept (deptno, dname )  
      2 VALUES        (60, 'MIS') ;  
1 row created.
```

- **Explicit method: Specify the NULL keyword.**

```
SQL> INSERT INTO      dept  
      2 VALUES        (70, 'FINANCE', NULL) ;  
1 row created.
```

# Inserting Special Values

**The SYSDATE function records the current date and time.**

```
SQL> INSERT INTO      emp (empno, ename, job,  
2                      mgr, hiredate, sal, comm,  
3                      deptno)  
4  VALUES             (7196, 'GREEN', 'SALESMAN',  
5                      7782, SYSDATE, 2000, NULL,  
6                      10);  
  
1 row created.
```



# Inserting Specific Date Values

- Add a new employee.

```
SQL> INSERT INTO emp
  2  VALUES      (2296, 'AROMANO', 'SALESMAN', 7782,
  3                TO_DATE('FEB 3, 97', 'MON DD, YY'),
  4                1300, NULL, 10);
1 row created.
```

- Verify your addition.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
-----	-----	-----	-----	-----	-----	-----	-----
2296	AROMANO	SALESMAN	7782	03-FEB-97	1300		10

# Inserting Values by Using Substitution Variables

Create an interactive script by using SQL\*Plus substitution parameters.

```
SQL> INSERT INTO      dept (deptno, dname, loc)
      2  VALUES      (&department_id,
      3                '&department_name', '&location');
```

```
Enter value for department_id: 80
Enter value for department_name: EDUCATION
Enter value for location: ATLANTA

1 row created.
```

# Creating a Script with Customized Prompts

- **ACCEPT** stores the value into a variable.

- **PROMPT** displays your customized text.

```
ACCEPT department_id PROMPT 'Please enter the -  
department number: '  
  
ACCEPT department_name PROMPT 'Please enter -  
the department name: '  
  
ACCEPT location PROMPT 'Please enter the -  
location: '  
  
INSERT INTO dept (deptno, dname, loc)  
VALUES (&department_id, '&department_name',  
&location);
```

# Copying Rows from Another Table

- Write your INSERT statement with a subquery.

```
SQL> INSERT INTO managers(id, name, salary, hiredate)
2      SELECT empno, ename, sal, hiredate
3      FROM    emp
4      WHERE   job = 'MANAGER';
3 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

**EMP**

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

“...update a row  
in EMP table...”



**EMP**

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		20
7566	JONES	MANAGER		20
...				

# The UPDATE Statement

- **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 when you specify the WHERE clause.

```
SQL> UPDATE    emp
      2  SET      deptno = 20
      3  WHERE    empno = 7782 ;
1 row updated.
```

- All rows in the table are modified if you omit the WHERE clause.

```
SQL> UPDATE    employee
      2  SET      deptno = 20 ;
14 rows updated.
```

# Updating with Multiple-Column Subquery

**Update employee 7698's job and department to match that of employee 7499.**

```
SQL> UPDATE emp
  2 SET      (job, deptno) =
  3          (SELECT job, deptno
  4             FROM   emp
  5             WHERE  empno = 7499)
  6 WHERE    empno = 7698;
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.

```
SQL> UPDATE    employee
  2  SET        deptno = (SELECT    deptno
  3                                FROM      emp
  4                                WHERE     empno = 7788)
  5  WHERE      job      = (SELECT    job
  6                                FROM      emp
  7                                WHERE     empno = 7788) ;
```

**2 rows updated.**

# Updating Rows: Integrity Constraint Error

```
SQL> UPDATE emp  
2 SET deptno = 55  
3 WHERE deptno = 10;
```

```
UPDATE emp  
*
```

```
ERROR at line 1
```

```
ORA-02291: integrity constraint (USR.EMP_DEPTNO_FK)  
violated - parent key not found
```

Department number 55 does not exist


# Removing a Row from a Table

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	DETROIT
60	MIS	
...		

“...delete a row  
from DEPT table...”

DEPT



DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
60	MIS	
...		

# 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 row or rows are deleted when you specify the WHERE clause.

```
SQL> DELETE FROM      department
      2  WHERE          dname = 'DEVELOPMENT';
1 row deleted.
```

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

```
SQL> DELETE FROM      department;
4 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.

```
SQL> DELETE FROM      employee
      2  WHERE          deptno =
      3                  (SELECT      deptno
      4                      FROM        dept
      5                      WHERE       dname = 'SALES' ) ;
6 rows deleted.
```

# Deleting Rows: Integrity Constraint Error

```
SQL> DELETE FROM dept  
2 WHERE deptno = 10;
```

```
DELETE FROM dept  
*
```

```
ERROR at line 1:
```

```
ORA-02292: integrity constraint (USR.EMP_DEPTNO_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.**

# Database Transactions

**Contain one of the following statements:**

- **DML statements that make up one consistent change to the data**
- **One DDL statement**
- **One DCL statement**



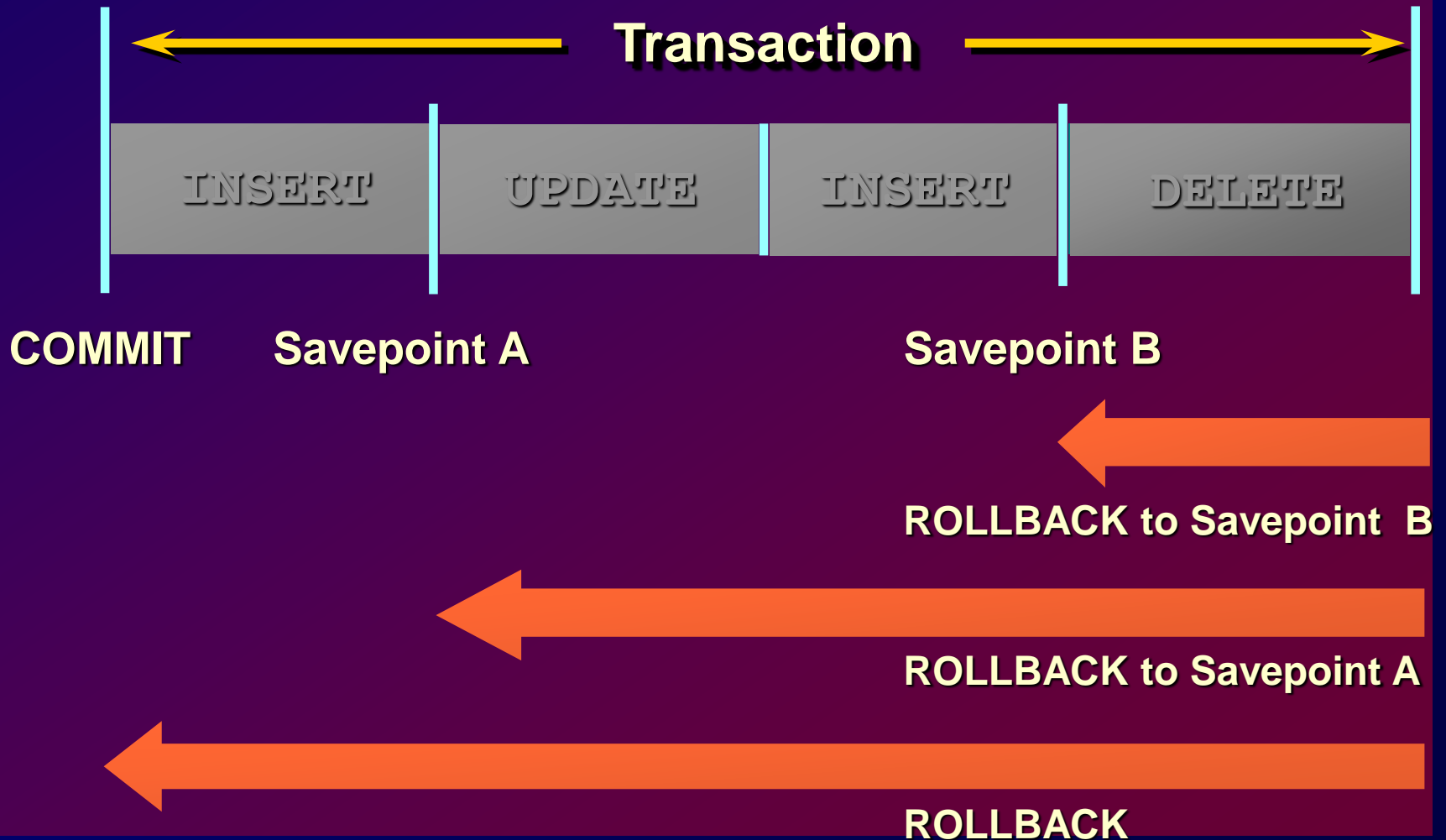
# Database Transactions

- **Begin when the first executable SQL statement is executed**
- **End with one of the following events:**
  - **COMMIT or ROLLBACK**
  - **DDL or DCL statement executes (automatic commit)**
  - **Certain errors, exit, or system crash**

# **Advantages of COMMIT and ROLLBACK**

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

# Controlling Transactions



# Implicit Transaction Processing

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

# State of the Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered because the database buffer is affected.
- 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.

```
SQL> UPDATE    emp
      2  SET      deptno = 10
      3  WHERE    empno = 7782;
1 row updated.
```

- Commit the changes.

```
SQL> 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.

```
SQL> DELETE FROM      employee;
```

```
14 rows deleted.
```

```
SQL> ROLLBACK;
```

```
Rollback complete.
```



# Rolling Back Changes to a Marker

- Create a marker within a current transaction by using the **SAVEPOINT** statement.
- Roll back to that marker by using the **ROLLBACK TO SAVEPOINT** statement.

```
SQL> UPDATE ...  
SQL> SAVEPOINT update_done;  
Savepoint created.  
SQL> INSERT ...  
SQL> ROLLBACK TO update_done;  
Rollback complete.
```

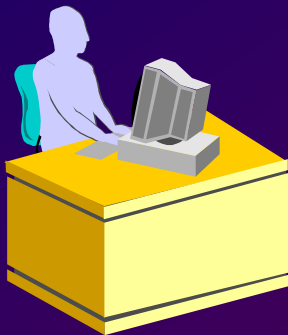
# Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- Oracle8 implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.

# Read Consistency

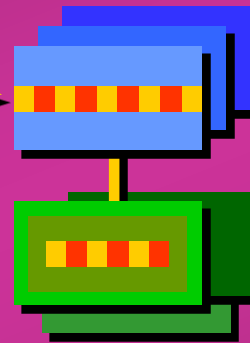
- **Read consistency guarantees a consistent view of the data at all times.**
- **Changes made by one user do not conflict with changes made by another user.**
- **Ensures that on the same data:**
  - **Readers do not wait for writers**
  - **Writers do not wait for readers**

# Implementation of Read Consistency



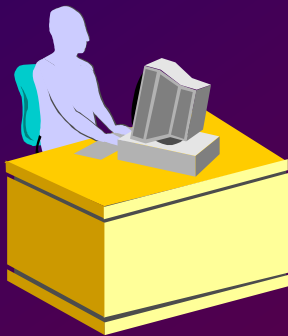
User A

update emp  
set sal = 2000  
where ename =  
'SCOTT'



Data  
blocks

Rollback  
segments

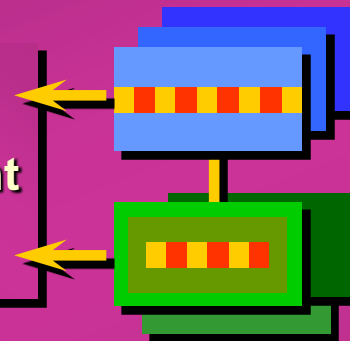


User B

select \*  
from emp



Read  
consistent  
image



changed  
and  
unchanged  
data

before  
change  
'old' data

# Locking

## Oracle8 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
- Have two basic modes:
  - Exclusive
  - Shared

# Summary

Statement	Description
<b>INSERT</b>	Adds a new row to the table
<b>UPDATE</b>	Modifies existing rows in the table
<b>DELETE</b>	Removes existing rows from the table
<b>COMMIT</b>	Makes all pending changes permanent
<b>SAVEPOINT</b>	Allows a rollback to the savepoint marker
<b>ROLLBACK</b>	Discards all pending data changes

# Practice Overview

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