9 Manipulating Data

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 transactions control using COMMIT,ROLLBACK

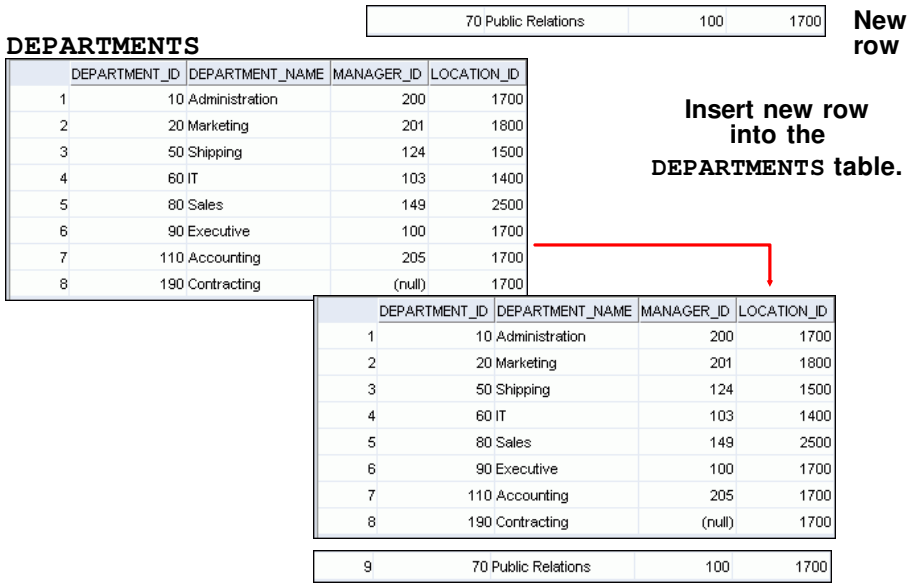
SQL Statements

SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

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



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.

In the syntax:	
table	is the name of the table
column	is the name of the column in the table to populate
value	is the corresponding value for the column

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 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        (31,'Purchasing');
1 rows inserted
```

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

```
INSERT  INTO   departments
VALUES         (111,'Finance',NULL,NULL);
```

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,  110);
```


Confirming Additions to the Table

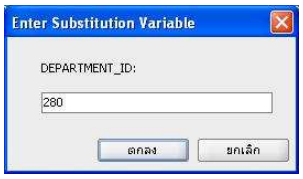
Display data

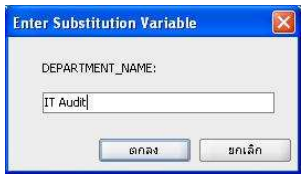
	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
1	113	Popp	FI_ACCOUNT	07-DEC-99	(null)

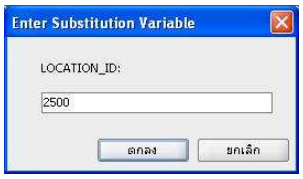
```
SELECT  employee_id,last_name,job_id,hire_date,
        commission_pct
FROM    employees
WHERE   employee_id = 113;
```


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_id);
```

```
SELECT   *
FROM     departments;
```


Practice

Consider the CONTACTS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Copying Rows from Another Table

- Write your INSERT statement with a subquery:
- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.
- Example: Inserts all the rows returned by the subquery in the table, department where manager not null.

```
INSERT INTO department
   SELECT department_id, department_name
   FROM   departments
   WHERE  manager_id IS NOT NULL;
```

```
SELECT * FROM  department;
```


Practice

Copy the complete CONTACTS table into the CONTACTS_BKP table

Practice

Display all data from CONTACTS table where id with salary more than 4500 (use subquery to display data CONTACTS_BKP table)

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```


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).

In the syntax:	
table	is the name of the table
column	is the name of the column in the table to populate
value	is the corresponding value or subquery for the column
condition	Identifies the rows to be updated and is composed of column names,expressions,constants,subqueries,and comparison operators

Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the WHERE clause.
- The example in the slide shows the transfer of employee 113 (Popp) to department 50

Old data:

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	DEPARTMENT_ID
1	113	Popp	FI_ACCOUNT	07-DEC-99	100

New data:

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	DEPARTMENT_ID
1	113	Popp	FI_ACCOUNT	07-DEC-99	50

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

1 rows updated

Updating Two Columns with a Subquery

Update employee 113's phone number and salary to match those of employee 205.

	EMPLOYEE_ID	PHONE_NUMBER	SALARY
1	113	515.124.4567	6900

	EMPLOYEE_ID	PHONE_NUMBER	SALARY
1	205	515.123.8080	12000

Updating Two Columns with a Subquery

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

Practice

Updates SALARY by 0.25 times in the CONTACTS table for all the customers whose AGE is greater than or equal to 27 (use subquery to update data from CONTACTS_BKP table).

Practice

Updates SALARY by 0.25 times in the CONTACTS table for all the customers whose AGE is greater than or equal to 27 (use subquery to update data).

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];
```

In the syntax:

`table`	is the name of the table
`condition`	identifies the rows to be deleted, and is composed of column names, expressions, constants, subqueries, and comparison operators

Deleting Rows from a Table

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

```
DELETE  FROM  departments
WHERE   department_name =  'Finance';
```

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

```
DELETE  FROM   sales_reps;
```


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:
- The example in the slide deletes all employees in a department where the department name contains the string "Public"

```
DELETE  FROM  employees
WHERE   department_id =
```

```
1 rows deleted
```


Practice

Deletes the records from the CONTACTS table for all the customers whose AGE is greater than or equal to 27. (use subquery to delete data from CONTACTS_BKP table)

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
```


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  CUSTOMERS_BKP;
```

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

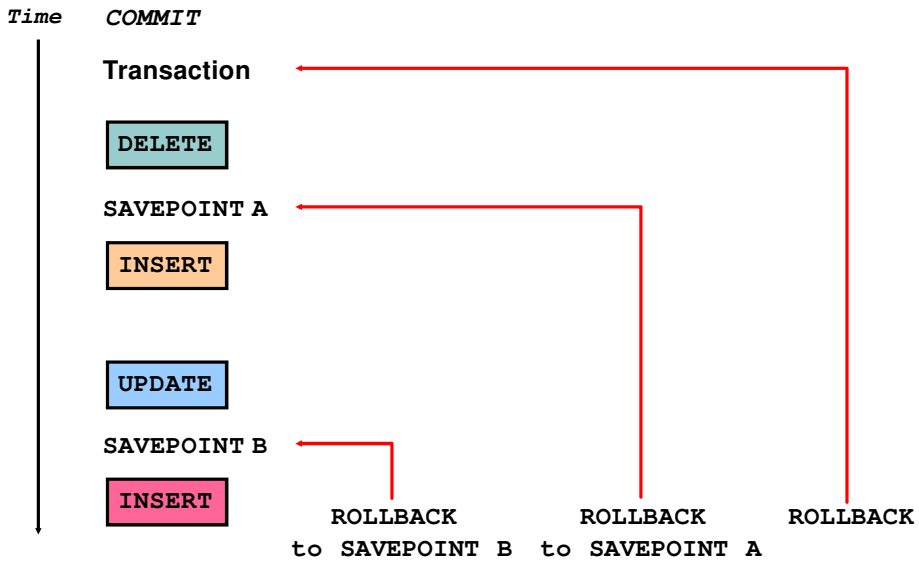
- Begin when the first DML SQL statement is executed.
- End with one of the following events:
  - A COMMIT or ROLLBACK statement is issued.
  - A DDL or DCL statement executes (automatic commit).
  - The user exits SQL Developer or SQL*Plus.
  - The system crashes.

Advantages of COMMIT and ROLLBACK

With COMMIT and ROLLBACK statements, you can:

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

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.
```


Implicit Transaction Processing

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

State of the Data After COMMIT

- Data changes are saved in the database.
- 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.
- All savepoints are erased.

Committing Data

In the example in the slide, a row is deleted from the EMPLOYEES table and a new row is inserted into the DEPARTMENTS table. The changes are saved by issuing the COMMIT statement.

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 succeeded.

Committing Data

Example:

Remove departments 290 and 300 in the DEPARTMENTS table and update a row in the EMPLOYEES table

```
DELETE FROM departments
WHERE  department_id IN (290,300);

UPDATE employees
SET department_id=80
WHERE employee_id=206;

COMMIT;
```

COMMIT succeeded.

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.
- Example:

```
DELETE FROM dept80;

SELECT * FROM dept80;

ROLLBACK;

SELECT * FROM dept80;
```


State of the Data After ROLLBACK

Example:

While attempting to remove a record from the TEST table, you may accidentally empty the table. However, you can correct the mistake, reissue a proper statement, and make the data change permanent.

```
DELETE FROM dept80;
SELECT * FROM dept80;
ROLLBACK;
SELECT * FROM dept80;

DELETE FROM dept80;

COMMIT;

SELECT * FROM dept80;
```

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 all rows from a 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

Quiz

The following statements produce the same results:

```
DELETE FROM copy_emp;
```

```
TRUNCATE FROM copy_emp;
```

- True
- False