**Manipulating Data**

**Data Manipulation Language**

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a transaction.

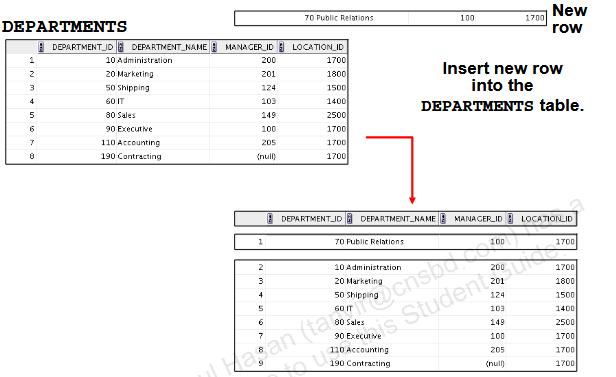
Note:

* Most of the DML statements in this lesson assume that no constraints on the table are violated. Constraints are discussed later in this course.
* In SQL Developer, click the Run Script icon or press [F5] to run the DML statements. The feedback messages will be shown on the Script Output tabbed page.

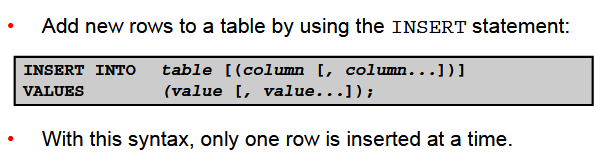
**Data Manipulation Language**

* **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**



* 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.
* Enclose character and date values within single quotation marks.



**Inserting Rows with Null Values:**

|  |
| --- |
| * Implicit method: Omit the column from the column list.   **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**    **INSERT INTO dept(department\_id, department\_name)**  **VALUES(30, 'Purchasing');**    **select \* from dept;** |

**Inserting Special Values**

|  |
| --- |
| * The SYSDATE function records the current date and time.   **drop table emp;**  **create table emp as select \* from employees where 1=2;**  **select \* from emp;**  **DESC emp;**    **INSERT INTO emp (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);**    **select \* from emp;**    **SELECT employee\_id, last\_name, job\_id, hire\_date,commission\_pct**  **FROM emp**  **WHERE employee\_id = 113;** |

**Inserting Specific Date and Time Values**

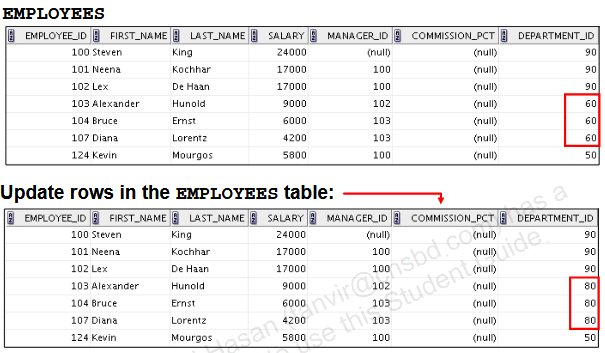
|  |
| --- |
| * Add a new employee.   **DROP table emp;**  **CREATE table emp as select \* from employees where 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**    **INSERT INTO emp**  **VALUES (114, 'Den', 'Raphealy', 'DRAPHEAL', '515.127.4561',**  **TO\_DATE('FEB 3, 1999', 'MON DD, YYYY'),**  **'SA\_REP', 11000, 0.2, 100, 60);**     * Verify your addition.   **select \* from emp;** |

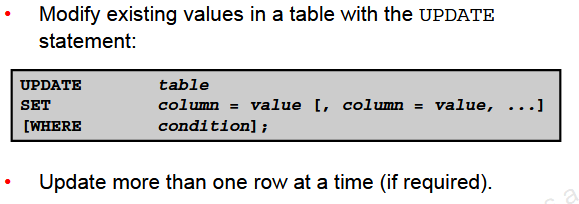
|  |
| --- |
| **Creating a Script**   * Use the & substitution in a SQL statement to prompt for values. * & is a placeholder for the variable value.   **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**    **INSERT INTO departments**  **(department\_id, department\_name, location\_id)**  **VALUES (&department\_id, '&department\_name',&location);** |

|  |
| --- |
| **Copying Rows from Another Table**   * Write your INSERT statement with a subquery:   **EXAMPLE -1:**  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**    **INSERT INTO emp**  **SELECT \* FROM employees;**  EXAMPLE- 2:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **DESC emp**    **INSERT INTO emp(employee\_id,last\_name,email,**  **hire\_date, job\_id, salary, commission\_pct)**  **SELECT employee\_id, last\_name, email,**  **hire\_date, job\_id, salary, commission\_pct FROM employees;**  **SELECT \* FROM emp;**    EXAMPLE-3:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **DESC emp**    **INSERT INTO emp(employee\_id,last\_name,email,**  **hire\_date, job\_id, salary, commission\_pct)**  **SELECT employee\_id, last\_name, email,**  **hire\_date, job\_id, salary, commission\_pct FROM employees**  **WHERE job\_id LIKE "%REP%";**  **SELECT \* FROM emp;**     * Do not use the VALUES clause. * Match the number of columns in the INSERT clause to those in the subquery. * Inserts all the rows returned by the subquery in the table, sales\_reps. |

UPDATE

Changing Data in a Table

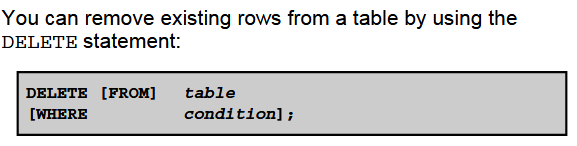




|  |
| --- |
| **Updating Rows in a Table**   * Values for a specific row or rows are modified if you specify the WHERE clause:   **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**  **INSERT INTO dept**  **SELECT \* FROM departments;**  **SELECT \* FROM dept WHERE department\_id=50;**    **SELECT \* FROM emp WHERE employee\_id=113;**    **UPDATE emp**  **SET department\_id = 50**  **WHERE employee\_id = 113;**  **SELECT \* FROM emp WHERE employee\_id=113;**     * Values for all the rows in the table are modified if you omit the WHERE clause:   **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **UPDATE emp**  **SET department\_id = 110;**  **SELECT \* FROM emp;**     * Specify SET column\_name= NULL to update a column value to NULL.   **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **UPDATE emp**  **SET job\_id ='%REP%',commission\_pct=null**  **where employee\_id=144;**  **SELECT job\_id FROM emp WHERE job\_id='%REP%';**    **select job\_id,commission\_pct from emp**  **where employee\_id=114;**    **update emp**  **set job\_id='IT\_PROG',commission\_pct=null**  **where employee\_id=114;** |

|  |
| --- |
| Updating Two Columns with a Subquery   * Update employee 113’s job and salary to match those of emp 205.   EXAPMLE-1:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **SELECT job\_id FROM emp WHERE employee\_id=205;**    **SELECT salary FROM emp WHERE employee\_id=205;**    **UPDATE emp**  **SET job\_id = (SELECT job\_id**  **FROM emp**  **WHERE employee\_id = 205),**  **salary = (SELECT salary**  **FROM emp**  **WHERE employee\_id = 205)**  **WHERE employee\_id = 113;**  EXAPMLE-2:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **SELECT job\_id FROM emp WHERE employee\_id=205;**    **SELECT salary FROM emp WHERE employee\_id=205;**    **UPDATE emp**  **SET (job\_id, salary) = (SELECT job\_id, salary**  **FROM employees**  **WHERE employee\_id = 205)**  **WHERE employee\_id = 113;** |

DELETE



|  |
| --- |
| Deleting Rows from a Table   * Specific rows are deleted if you specify the WHERE clause:   **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**  **SELECT \* FROM dept**  **WHERE department\_name = 'Finance';**    **DELETE FROM dept**  **WHERE department\_name = 'Finance';**    **SELECT \* FROM dept**  **WHERE department\_name = 'Finance';**     * All rows in the table are deleted if you omit the WHERE clause:   EXAMPLE-1:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DELETE FROM emp;**    EXAMPLE-2:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DELETE FROM emp WHERE employee\_id = 114;**  EXAMPLE-3:  **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**  **INSERT INTO dept**  **SELECT \* FROM departments;**  **DELETE FROM dept WHERE department\_id IN (30, 40);**    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 emp**  **WHERE department\_id =(SELECT department\_id**  **FROM departments**  **WHERE department\_name**  **LIKE '%Public%');** |

TRUNCATE

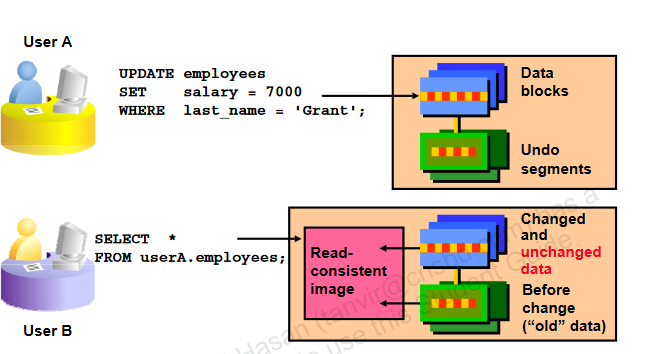
|  |
| --- |
| **TRUNCATE Statement**  A more efficient method of emptying a table is by using the TRUNCATE statement.You can use the TRUNCATE statement to quickly remove all rows from a table or cluster. Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:   * The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information * Truncating a table does not fire the delete triggers of the table. * 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     **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **TRUNCATE TABLE emp;**    **SELECT \* FROM emp;** |

|  |
| --- |
| **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 Statements**  With COMMIT and ROLLBACK statements, you can:   * Ensure data consistency * Preview data changes before making changes permanent * Group logically-related operations         Implicit Transaction Processing   * An automatic commit occurs in the following circumstances: * –A DDL statement issued * –A DCL statement 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  Every data change made during the transaction is temporary until the transaction is committed. The state of the data before COMMIT or ROLLBACK statements are issued can be described as follows:   * Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered. * The current user can review the results of the data manipulation operations by querying the tables. * Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit. * The affected rows are locked; other users cannot change the data in the affected rows.  1. The previous state of the data can be recovered. 2. The current user can review the results of the DML operations by using the SELECT statement. 3. Other users cannot view the results of the DML statements issued by the current user. 4. The affected rows are locked; other users cannot change the data in the affected rows.   **State of the Data After COMMIT**  Make all pending changes permanent by using the COMMIT statement. Here is what happens after a COMMIT statement:   * Data changes are written to the database. * The previous state of the data is no longer available with normal SQL queries. * All users can view the results of the transaction. * The locks on the affected rows are released; the rows are now available for other users to perform new data changes. * All savepoints are erased.   EXAMPLE-1:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DELETE FROM emp WHERE employee\_id = 114;**  **COMMIT;**  EXAMPLE-2:  **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**  **INSERT INTO dept**  **SELECT \* FROM departments;**  **DELETE FROM dept WHERE department\_id IN (30, 40);**  **COMMIT;**  **State of the Data After ROLLBACK**   * Data changes are undone. * Previous state of the data is restored. * Locks on the affected rows are released.   EXAMPLE-1:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DELETE FROM emp;**  **ROLLBACK;**    EXAMPLE-2:  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **DELETE FROM emp;**  **ROLLBACK;**    **DELETE FROM emp WHERE employee\_id = 100;**    **SELECT \* FROM emp WHERE employee\_id = 100;**    **COMMIT;** |

**Read Consistency**

* Read consistency guarantees a consistent view of the 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



|  |
| --- |
| **FOR UPDATE Clause in a SELECT Statement**  **EXAMPLE -1:**   * Locks the rows in the EMPLOYEES table where job\_id is SA\_REP.   **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **SELECT employee\_id, salary, commission\_pct, job\_id**  **FROM emp**  **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.   **EXAMPLE -2:**   * You can use the FOR UPDATE clause in a SELECT statement against multiple tables.   **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**  **INSERT INTO dept**  **SELECT \* FROM departments;**  **SELECT e.employee\_id, e.salary, e.commission\_pct**  **FROM emp e JOIN dept 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 you intend to change, then only the rows from that specific table are loc   **EXAMPLE-3:**  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **DROP table dept;**  **CREATE table dept as SELECT \* FROM departments WHERE 1=2;**  **SELECT \* FROM dept;**  **DESC dept;**  **INSERT INTO dept**  **SELECT \* FROM departments;**  **SELECT e.employee\_id, e.salary, e.commission\_pct**  **FROM emp e**  **JOIN dept d**  **USING (department\_id)**  **WHERE job\_id = 'ST\_CLERK' AND location\_id = 1500**  **FOR UPDATE OF e.salary**  **ORDER BY e.**  **employee\_id;**  **EXAMPLE-4:**  **DROP table emp;**  **CREATE table emp as SELECT \* FROM employees WHERE 1=2;**  **SELECT \* FROM emp;**  **DESC emp;**  **INSERT INTO emp**  **SELECT \* FROM employees;**  **SELECT employee\_id, salary, commission\_pct, job\_id**  **FROM emp**  **WHERE job\_id = 'SA\_REP'**  **FOR UPDATE WAIT 5**  **ORDER BY employee\_id;** |