UPDATE-INSERT-DELETE

After completing this module, you will be able to:

- Insert rows into a table one-at-a-time.
- Insert rows using a bulk operation like “insert-select”.
- Update one or more values for one or more rows of a table.
- Delete one or more rows from a table.
- Determine the effect of CASESPECIFIC and NOT CASESPECIFIC on DML.
- Insert, update or delete rows from tables using joins and subqueries.

Updating Data via SQL

The following syntax is referred to as DML – Data Manipulation Language.

Other SQL syntax that also updates table data that are not covered in this course.

- CREATE TABLE AS
- MERGE

. . .

INSERT	Inserts a single row into a table.
INSERT-SELECT	Inserts zero or more rows to a table using values selected from one or more existing source tables.
UPDATE	Changes column values in existing rows of a table.
DELETE	Removes rows from a table.

Inserting a Single Row

Insert a new employee into the employee table.
Value order assumes table column order.

```
INSERT INTO employee
VALUES      (1210, NULL, 401, 412101, 'Smith', 'James', 890303, 460421, 41000);
```

Insert a new employee with only partial data.
Column list may be any order.
Value list must match order of column list.
Inserts default values (discussed later) for missing columns.

```
INSERT INTO employee
   (last_name, first_name, hire_date, birthdate, salary_amount, employee_number)
VALUES      ('Garcia', 'Maria', 861027, 541110, 76500.00, 1291);
```

As a Teradata Extension to ANSI:

- INSERT can be abbreviated as INS.
- INTO and VALUES are optional keywords.

There is no typing shortcut facility for explicitly inserting multiple rows with this form.

Inserting an Apostrophe

To insert an apostrophe into a character string, use the following notation.

Two back-to-back single quotes

```
INSERT INTO Department
VALUES   (111, 'President''s Club', 400000.00, 222);
```

You can retrieve this row using the same notation like this.

```
SELECT * FROM department
WHERE department_name = 'president''s club'
```

department_number	department_name	budget_amount	manager_employee_number
111	President's Club	400000.00	222

Inserting Default Values

```
CREATE TABLE Example
(C1 INT,
 C2 CHAR(20),
 C3 CHAR(20) DEFAULT USER,
 C4 CHAR(20) WITH DEFAULT,
 C5 INT,
 C6 INT DEFAULT 10,
 C7 INT WITH  DEFAULT);
```

DEFAULT is ANSI SQL-2003-compliant.

WITH DEFAULT is a Teradata extension to the ANSI SQL-2003 standard.

The ANSI default values are:

- **Null for numeric and character columns.**

Using "WITH DEFAULT" changes this to:

- **Character default is spaces.**
- **Numeric default is zero.**

```
INS Example ( ,  ,  ,  ,  ,  , );
INS Example (1,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT);
INS Example (2,NULL,NULL,NULL,NULL,NULL,NULL);
```

```
SELECT *
FROM Example
ORDER BY 1;
```

C1	C2	C3	C4	C5	C6	C7
?	?	DLM		?	10	0
1	?	DLM		?	10	0
2	?	?	?	?	?	?

Default Values and NOT NULL

Inserting a null into a non-null column fails.

Other examples of various success and failure are illustrated.

```
CREATE TABLE T1
(C2 CHAR(20) NOT NULL)

INS T1 (  );  -- Fails
INS T1 (DEFAULT); -- Fails
INS T1 (NULL); -- Fails
```

```
CREATE TABLE T1
(C2 CHAR(20) NOT NULL WITH DEFAULT)

INS T1 (  );  -- Inserts a space
INS T1 (DEFAULT); -- Inserts a space
INS T1 (NULL); -- Fails
```

```
CREATE TABLE T1
(C2 CHAR(20) NOT NULL DEFAULT  USER)

INS T1 (  );  -- Inserts the user name for the insert
INS T1 (DEFAULT); -- Inserts the user name for the insert
INS T1 (NULL); -- Fails
```

Insert-Select

INSERT-SELECT:

- Is used to copy rows from one table to another.
- Uses a SELECT statement to define a subset of rows and/or a subset of columns to be inserted.
- Silently *(does not fail)* discards any duplicate rows into a SET table.
- Retains any duplicate rows into a MULTISET table.

```
INSERT INTO emp_copy
SELECT * FROM emp;
```

Assumes:

- emp and emp_copy both exist with the same definition.
- A complete replica of emp is required.
- Into an empty table is optimized! *(no transient journaling)*

A more complex INSERT SELECT

First, create the 'birthdays' table:

```
CREATE TABLE birthdays
   (empno   INTEGER  NOT NULL
   ,lname   CHAR(20) NOT NULL
   ,fname   VARCHAR(30)
   ,birth   DATE)
   UNIQUE PRIMARY INDEX (empno);
```

Populate the 'birthdays' table from 'employee':

```
INSERT INTO birthdays
SELECT    employee_number
          ,last_name
          ,first_name
          ,birthdate
FROM      employee
WHERE     department_number = 403;
```

CASESPECIFIC and SET Tables

```
CREATE SET TABLE DLM.T1 ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     C2 CHAR(20) CHARACTER SET LATIN
                    NOT CASESPECIFIC)
PRIMARY INDEX ( C2 );
```

```
INS T1 ('ABC');
 *** Insert completed. One row added.

INS T1 ('abc');
 *** Failure 2802 Duplicate row error in DLM.T1.

INS T1 SEL * FROM T1;
 *** Insert completed. No rows added.
```

```
C2
--------------------
ABC
```

```
CREATE SET TABLE DLM.T2 ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
     C2 CHAR(20) CHARACTER SET LATIN
                    CASESPECIFIC)
PRIMARY INDEX ( C2 );
```

```
INS T2 ('ABC');
 *** Insert completed. One row added.

INS T2 ('abc');
 *** Insert completed. One row added.

INS T2 SEL * FROM T2;
 *** Insert completed. No rows added.
```

```
C2
--------------------
ABC
abc
```

CASESPECIFIC and MULTISET Tables

```
CREATE MULTISET TABLE DLM.T1
(     C2 CHAR(20) CHARACTER SET LATIN
                    NOT CASESPECIFIC)
PRIMARY INDEX ( C2 );
```

```
INS T1 ('ABC');
 *** Insert completed. One row added.

INS T1 ('abc');
 *** Insert completed. One row added.

INS T1 SEL * FROM T1;
 *** Insert completed. 2 rows added.
```

```
SEL *
FROM t1
WHERE c2 = 'abc';
```

```
C2
--------------------
ABC
abc
ABC
abc
```

```
CREATE MULTISET TABLE DLM.T2
(     C2 CHAR(20) CHARACTER SET LATIN
                    CASESPECIFIC)
PRIMARY INDEX ( C2 );
```

```
INS t2 SEL * FROM t1;
 *** Insert completed. 4 rows added.
```

```
SEL *
FROM t2
WHERE c2 = 'abc';
```

```
C2
--------------------
abc
abc
```

UPDATE

UPDATE modifies one or more rows in a single table.

EMPLOYEE

EMP NUM	MGR EMP NUM	DEPT NUM	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1003	401	412101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

For employee 1004, change their:

- Department to 403
- Job to 432101
- Manager to 1005

```
UPDATE   Employee [ FROM Employee ]
SET      Department_Number            =   403
         ,Job_Code                    =   432101
         ,Manager_Employee_Number =   1005
WHERE    Employee_Number              =   1004;
```

EMPLOYEE

EMP NUM	MGR EMP NUM	DEPT NUM	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1005	403	432101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

Updating with Joins

Updates with joins and subqueries allow a table's rows to be updated based on information in another table.

Give everyone in all the support departments a 10% raise.
(Assume we don't know the department numbers for all of the support departments.)

Using a subquery:

```
UPDATE   employee
SET   salary_amount = salary_amount * 1.10
WHERE        department_number IN
   (SELECT   department_number
   FROM      department
   WHERE     department_name LIKE '%Support%');
```

Using an inner join:

```
UPDATE       employee [ FROM Department ]
SET          salary_amount = salary_amount * 1.10
WHERE        employee.department_number =
             department.department_number
AND          department_name LIKE '%Support%';
```

Using a correlated subquery:

```
UPDATE    employee e
SET salary_amount = salary_amount * 1.10
WHERE     department_number =
(SELECT      department_number
 FROM        department d
 WHERE       e.department_number =
             d.department_number
 AND  department_name LIKE '%Support%');
```

UPDATE and FROM

UPDATE with a FROM clause is used to update columns of a table with values from another table.

Update the last_call_number in the contact table to reflect the most recent call made by contact number 8005. (Call activity is carried in the call_log table.)

This is an example of a "scalar-subquery".
A scalar-subquery is one that returns a single value *(i.e. single row and single value).*

```
UPDATE  contact FROM call_log AS cl
SET     last_call_date = cl.call_date
WHERE   contact.contact_number = 8005
AND     cl.call_date =
        (SELECT MAX(call_date)
         FROM   call_log
         WHERE contact_number = 8005);
```

DELETE

DELETE removes one or more rows from a table.

EMPLOYEE

EMP NUM	MGR EMP NUM	DEPT NUM	JOB CODE	LAST NAME	FIRST NAME	HIRE DATE	BIRTH DATE	SAL AMT
PK	FK	FK	FK					
1006	1019	301	312101	Stein	John	761015	531015	2945000
1008	1019	301	312102	Kanieski	Carol	770201	580517	2925000
1005	0801	403	431100	Ryan	Loretta	761015	550910	3120000
1004	1003	401	412101	Johnson	Darlene	761015	460423	3630000
1007	1005	403	432101	Villegas	Arnando	770102	370131	4970000
1003	0801	401	411100	Trader	James	760731	470619	3785000

Remove employees in department 301 from the employee table.

```
DELETE  FROM  employee
WHERE  department_number = 301;
```

All of these are equivalent and empty a table.

```
DELETE FROM  emp_data ALL;
DELETE FROM  emp_data;
DELETE emp_data;
DEL emp_data;
```

Deleting with Joins

Remove all of the employees who are assigned to a temporary department. (i.e. for which the department name is 'Temp'.)

Using a Subquery:

```
DELETE FROM  employee
WHERE        department_number   IN
             (SELECT        department_number
             FROM           department
             WHERE          department_name  = 'Temp');
```

Using a Join:

```
DELETE FROM employee
WHERE        employee.department_number =
             department.department_number
AND          department.department_name = 'Temp';
```

Using a Correlated Subquery:

```
DELETE FROM  employee e
WHERE     department_number =
          (SELECT department_number
            FROM    department d
            WHERE   e.department_number = d.department_number
            AND     d.department_name = 'Temp');
```

Module 14: Summary

- DML syntax includes options for changing the data in a table.
- INSERT can be used to add rows to a table, either singly or in bulk.
- UPDATE can be used to change (update) column values either singly or in bulk.
- DELETE can be used to remove rows either singly or in bulk.
- Case sensitivity can affect what the DML taught in this module.
- Default values may be referenced in inserts, updates, and deletes.

Module 14: Review Questions

True or False:

1. **INSERT-SELECT can only insert rows into empty tables.**

2. **The DELETE can be used drop a table.**

3. **Updating with joins may not always be performed equally using subqueries.**

4. **By using single explicit value list, you can only insert only one row per insert.**

5. **The keywords “INTO” and “VALUES” are noise keywords.**

6. **You can include a single quote into a character string by using double-quotes.**

7. **NULL is the system default value for any column.**

Module 14: Lab Exercise

1) **Insert all of the rows from the employee table in database Employee_Sales into a copy of that table found in your database.**

2) **In your Employee table, use a subquery to give all employees working in "Support" departments a ten percent raise.**

3) **In your own Employee table, use a join to remove employees that have the word "manager" in their job description. Be sure to use your own Department table for this exercise.**

Make sure to reload your initial employee and department tables are available for the following labs. Do not use the tables from this lab.