Murach Chapter 7

How to Insert, Delete, Update Data

Week 6

Knowledge Points in this lecture

- Create copies of tables
- INSERT – insert 1 row
- Database Transaction, COMMIT, ROLLBACK
- UPDATE
- DELETE
- DELETE FROM vs DROP TABLE

The syntax of the CREATE TABLE AS statement

```
CREATE TABLE table_name AS
SELECT select_list
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_list]
[HAVING search_condition]
[ORDER BY order_by_list]
```

A statement that creates a complete copy of the Invoices table

```
CREATE TABLE invoices_copy AS
SELECT *
FROM invoices
```

A statement that creates a partial copy of the Invoices table

```
CREATE TABLE old_invoices AS
SELECT *
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
```

A statement that creates a table with summary rows from the Invoices table

```
CREATE TABLE vendor_balances AS
SELECT vendor_id, SUM(invoice_total) AS sum_of_invoices
FROM invoices
WHERE (invoice_total - payment_total - credit_total) <> 0
GROUP BY vendor_id
```

A statement that deletes a table

```
DROP TABLE old_invoices
```

Warning

- When you use the SELECT statement to create a table, only the column definitions and data are copied.
- Definitions of primary keys, foreign keys, indexes, default values, and so on are not included in the new table.

A statement that adds a row to the Invoices table

```
INSERT INTO invoices
VALUES (115, 97, '456789', '01-AUG-14', 8344.50,
        0, 0, 1, '31-AUG-14', NULL)
```

The response from the system

```
1 rows inserted
```

A COMMIT statement that commits the changes

```
COMMIT
```

The response from the system

```
COMMIT succeeded
```

A statement that rolls back the changes

```
ROLLBACK
```

The response from the system

```
ROLLBACK succeeded
```

COMMIT, ROLLBACK

- Database Transaction
  - Like ATM transaction
  - A group of SQL statements that must all fail or all succeed
- ROLLBACK
  - Cancel the data modification made in the current database transaction
- COMMIT
  - Make the data modification made in the current database transaction permanent
  - Changes still exist after user logs off

The INSERT syntax for inserting a single row

```
INSERT INTO table_name [(column_list)]
    VALUES (expression_1 [, expression_2]...)
```

An INSERT statement that adds a new row without using a column list

```
INSERT INTO invoices
VALUES (115, 97, '456789', '01-AUG-14', 8344.50,
        0, 0, 1, '31-AUG-14', NULL)

(1 rows inserted)
```

Must have 1 value for each column.
Values must be listed in same order as column order in the table

The values for the new row

Column	Value
invoice_id	115
vendor_id	97
invoice_number	456789
invoice_date	8/01/2014
invoice_total	8,344.50
payment_total	0
credit_total	0
terms_id	1
invoice_due_date	8/31/2014
payment_date	null

An INSERT statement that adds the new row using a column list

```
INSERT INTO invoices
    (invoice_id, vendor_id, invoice_number,
     invoice_total, payment_total,
     credit_total, terms_id, invoice_date,
     invoice_due_date)
VALUES
    (115, 97, '456789', 8344.50, 0, 0, 1,
     '01-AUG-14', '31-AUG-14')

(1 rows inserted)
```

Value order in VALUES clause must match the column order in the column list in INSERT INTO clause.

The definition of the Color_Sample table

Column name	Data Type	Not Null	Default Value
color_id	NUMBER	Yes	
color_number	NUMBER	Yes	0
color_name	VARCHAR2		

Five INSERT statements for the Color_Sample table

```
INSERT INTO color_sample (color_id, color_number)
VALUES (1, 606)

INSERT INTO color_sample (color_id, color_name)
VALUES (2, 'Yellow')

INSERT INTO color_sample
VALUES (3, DEFAULT, 'Orange')

INSERT INTO color_sample
VALUES (4, 808, NULL)

INSERT INTO color_sample
VALUES (5, DEFAULT, NULL)
```

- Missing columns in column list of first 2 examples are given default value or NULL based on table definition.
- Missing columns must be nullable or have a default value defined.

The Color_Sample table after the rows are inserted

COLOR_ID	COLOR_NUMBER	COLOR_NAME
1	606	(null)
2	0	Yellow
3	0	Orange
4	808	(null)
5	0	(null)

The syntax of the INSERT statement for inserting rows selected from another table

```
INSERT [INTO] table_name [(column_list)]
SELECT column_list
FROM table_source
[WHERE search_condition]
```

[A]: A is optional

An INSERT statement that inserts paid invoices in the Invoices table into the Invoice_Archive table

```
INSERT INTO invoice_archive
SELECT *
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0

(74 rows inserted)
```

If no column list:

- Subquery must return values for all columns in the table
- The columns must be returned in the same order as they appear in that table

The same INSERT statement with a column list

```
INSERT INTO invoice_archive
    (invoice_id, vendor_id, invoice_number,
     invoice_total, credit_total,
     payment_total, terms_id, invoice_date,
     invoice_due_date)
SELECT
    invoice_id, vendor_id, invoice_number, invoice_total,
    credit_total, payment_total, terms_id,
    invoice_date, invoice_due_date
FROM invoices
WHERE invoice_total - payment_total - credit_total = 0

(74 rows inserted)
```

If having column list:

- Subquery can omit columns with default values and columns that accept null values
- The columns must be returned in the same order as they appear in the column list

The syntax of the UPDATE statement

```
UPDATE table_name
SET column_name_1 = expression_1 [, column_name_2 =
expression_2]...
[WHERE search_condition]
```

An UPDATE statement that assigns new values to two columns of a single row in the Invoices table

```
UPDATE invoices
SET payment_date = '21-SEP-14',
    payment_total = 19351.18
WHERE invoice_number = '97/522'

(1 rows updated)
```

An UPDATE statement that assigns a new value to one column of all invoices for a vendor

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id = 95

(6 rows updated)
```

An UPDATE statement that uses an arithmetic expression to assign a value to a column

```
UPDATE invoices
SET credit_total = credit_total + 100
WHERE invoice_number = '97/522'

(1 rows updated)
```

Warning

- If you omit the WHERE clause, all rows in the table will be updated.

An UPDATE statement that assigns the maximum due date in the Invoices table to a specific invoice

```
UPDATE invoices
SET credit_total = credit_total + 100,
    invoice_due_date =
        (SELECT MAX(invoice_due_date)
         FROM invoices)
WHERE invoice_number = '97/522'

(1 rows updated)
```

An UPDATE statement that updates all invoices for a vendor based on the vendor's name

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id =
   (SELECT vendor_id
    FROM vendors
    WHERE vendor_name = 'Pacific Bell')

(6 rows updated)
```

An UPDATE statement that changes the terms of all invoices for vendors in three states

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id IN
    (SELECT vendor_id
     FROM vendors
     WHERE vendor_state IN ('CA', 'AZ', 'NV'))
(51 rows updated)
```

The syntax of the DELETE statement

```
DELETE [FROM] table_name
[WHERE search_condition]
```

A DELETE statement that deletes one row

```
DELETE FROM invoice_line_items
WHERE invoice_id = 100 AND invoice_sequence = 1

(1 rows deleted)
```

A DELETE statement that deletes four rows

```
DELETE FROM invoice_line_items
WHERE invoice_id = 100

(4 rows deleted)
```

A DELETE statement that uses a subquery to delete all invoice line items for a vendor

```
DELETE FROM invoice_line_items
WHERE invoice_id IN
    (SELECT invoice_id
    FROM invoices
    WHERE vendor_id = 115)

(4 rows deleted)
```

Warning

- If you omit the WHERE clause from a DELETE statement, all the rows in the table will be deleted.

DELETE FROM vs DROP TABLE

- DELETE FROM table_name
  - Remove all rows in a table
  - Do not remove the table definition
- DROP TABLE table_name
  - Remove all rows in a table
  - And remove the table definition