INSERT Statement

What Will I Learn?

In this lesson, you will learn to:

- Give examples of why it is important to be able to alter the data in a database
- Construct and execute INSERT statements which insert a single row using a VALUES clause
- Construct and execute INSERT statements that use special values, null values, and date values
- Construct and execute INSERT statements that copy rows from one table to another using a subquery

Why Learn It?

Up to now you have been learning how to extract data from a database. It's time to learn how to make changes to the database.

In business, databases are dynamic. They are constantly in the process of having data inserted, updated, and deleted. Think how many times the school's student database changes from day to day and year to year. Unless changes are made, the database would quickly lose its usefulness.

In this lesson, you will begin to use data manipulation language (DML) statements to make changes to a database.

Tell Me / Show Me

You will be responsible for altering tables in your schema. You will also be responsible for restoring them just as a real Database Administrator would assume that responsibility.

To keep your schema tables in their original state you will make a copy of each table to complete the practice activities in this and later lessons

Then, if you wrongly alter a copy table, you can restore a correct version of it from the original table.

You should name each table copy_tablename. The table copies will not inherit the associated primary-to-foreign-key integrity rules (relationship constraints) of the original tables. The column data types, however, are inherited in the copied tables.

The next slide shows you how to make a copy of a table.

Tell Me / Show Me

Use this syntax to create a copy of a table:

```
CREATE TABLE copy_tablename
   AS (SELECT * FROM tablename);
```

For example:

```
CREATE TABLE copy_f_customers
  AS (SELECT * FROM f_customers);
```

To verify and view the copy of the table, use the following DESCRIBE and SELECT statements:

```
DESCRIBE copy_f_customers

SELECT * FROM copy_f_customers;
```

Tell Me / Show Me

DML statements enable users to make changes in the database. Executing one DML statement is considered a transaction.

INSERT

The INSERT statement is used to add new rows to a table. The statement requires three values:

- the name of the table
- the names of the columns in the table to populate
- corresponding values for the column

How can we INSERT the data below to create a new customer in the copy_f_customers table?

ID	FIRST_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NO
145	Katie	Hernandez	92 Chico Way	LA	CA	98008	8586667641

Tell Me / Show Me

The syntax shown uses INSERT to add a new customer to a Global Fast Foods table. This statement explicitly lists each column as it appears in the table. The values for each column are listed in the same order. Note that number values are not enclosed in single quotation marks.

```
INSERT INTO copy_f_customers
  (id, first_name, last_name, address, city, state, zip, phone_number)
VALUES
  (145, 'Katie', 'Hernandez', '92 Chico Way', 'Los Angeles', 'CA', 98008, 8586667641);
```

ID	FIRST_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NO
145	Katie	Hernandez	92 Chico Way	LA	CA	98008	8586667641

Tell Me / Show Me

Another way to insert values in a table is to implicitly add them by omitting the column names.

One precaution: the values for each column must match exactly the default order in which they appear in the table (as shown in a DESCRIBE statement), and a value must be provided for each column.

The next slide shows an example of this.

Tell Me / Show Me

The INSERT statement in this example was written without explicitly naming the columns. For clarity, however, it is best to use the column names in an INSERT clause.

INSERT INTO f_customers

VALUES (475, 'Angelina', 'Wright', '7425 Redwood St', 'San Francisco', 'CA', 94162, '4159982010');

ID	FIRST_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NO
475	Angelina	Wright	7425 Redwood St	San Francisco	CA	94162	4159982010

Tell Me / Show Me

CHECK THE TABLE FIRST

Before inserting data into a table, you must check several table details. The DESCRIBE tablename syntax will return a table summary.

COPY_F_STAFFS TABLE SUMMARY

Primary Key	Name	Type	Length	Precision	Scale	Nullable	De-fault	Com-ments
1	ID	Number		5	0			…
	FIRST_NAME	Varchar2	25					…
	LAST_NAME	Varchar2	35					…
	BIRTHDATE	Date	7					…
	SALARY	Number		8	2	✓		…
	OVERTIME_RATE	Number		5	2	✓		…
	TRAINING	Varchar2	50					…
	…	…	…					…

Tell Me / Show Me

As shown in the example, the table summary provides the following information about the columns in the table:

- columns that can have a NULL value
- columns that cannot have duplicate values
- allowable data type
- amount of data that can be entered in a column

Primary Key	Name	Type	Length	Precision	Scale	Nullable	De-fault	Com-ments
1	ID	Number		5	0			…
	FIRST_NAME	Varchar2	25					…
	LAST_NAME	Varchar2	35					…
	BIRTHDATE	Date	7					…
	SALARY	Number		8	2			…
	OVERTIME_RATE	Number		5	2	✓		…
	TRAINING	Varchar2	50			✓		…
	…	…	…					…

Tell Me / Show Me

Notice the Length column for characters and dates, and the Precision and Scale column for numbers. Precision is the total number of decimal digits, and the scale is the number of digits to the right of the decimal place.

The OVERTIME_RATE number column has precision 5 and scale 2. The maximum value that can be inserted into this column is 999.99.

Primary Key	Name	Type	Length	Precision	Scale	Nullable	De-fault	Com-ments
1	ID	Number		5	0			…
	FIRST_NAME	Varchar2	25					…
	LAST_NAME	Varchar2	35					…
	BIRTHDATE	Date	7					…
	SALARY	Number		8	2	✓		…
	OVERTIME_RATE	Number		5	2	✓		…
	TRAINING	Varchar2	50					…
	…	…	…					…

Tell Me / Show Me

INSERTING ROWS WITH NULL VALUES

The INSERT statement need not specify every column, as long as the columns not explicitly named in the INSERT clause can accept null values.

In our example, the SALARY column is defined as a NOT NULL column. An implicit attempt to add values to the table as shown would generate an error.

```
INSERT INTO copy_f_staffs
  ( id, first_name, last_name, birthdate, overtime_rate)
VALUES
  (15, 'Gregorz', 'Polanski‘,’25-SEP-88’,17.50);
```

ORA-01400: cannot insert NULL into (“USQA_JOHN_SQL01_S01”.“COPY_F_STAFFS”.“SALARY”)

Tell Me / Show Me

INSERTING ROWS WITH NULL VALUES

If a column can hold null values, it can be omitted from the INSERT clause. An implicit insert will automatically insert a null value in that column. To explicitly add null values to a column, use the NULL keyword in the VALUES list for those columns that can hold null values.

To specify empty strings and/or missing dates, use empty single quotation marks (' ') for missing data.

```
INSERT INTO copy_f_staffs
 ( id, first_name, last_name, birthdate, salary, overtime_rate)
VALUES
 (15, 'Gregorz', 'Polanski',’25-SEP-88’,224.25,null);
```

ID	FIRST_NAME	LAST_NAME	BIRTHDATE	SALARY	OVERTIME_RATE	TRAINING	….
15	Gregorz	Polanski	25-SEP-88	224.25	(null)	(null)	….

Tell Me / Show Me

INSERTING SPECIAL VALUES

Special values such as SYSDATE and USER can be entered in the VALUES list of an INSERT statement.

SYSDATE will put the current date and time in a column.

USER will insert the current session's username, which in Oracle Application Express, will be OAE_PUBLIC_USER.

```
INSERT INTO copy_employees
  (employee_id, last_name, email,hire_date, job_id)
VALUES
  (1001, USER, 'Test', SYSDATE,'IT_PROG');
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID
1001	(null)	HTMLDB_PUBLIC_USER	Test	(null)	12-MAR-06	IT_PROG

Tell Me / Show Me

INSERTING SPECIAL VALUES

In addition, functions and calculated expressions can also be used in the VALUES clause.

The example below illustrates two types of insertions into the Global Fast Foods copy_f_orders table. SYSDATE is used to insert the order_date and a calculated expression is used to enter the order_total.

```
INSERT INTO copy_f_orders
  (order_number, order_date, order_total, cust_id, staff_id)
VALUES
  (1889, SYSDATE, 87.92*1.08, 123, 19)
```

Tell Me / Show Me

INSERTING SPECIFIC DATE VALUES

The default format model for date is DD-MON-RR. With this format, recall that the century defaults to the nearest century (nearest to SYSDATE) with the default time of midnight (00:00:00).

In an earlier section we learned how to use the TO_CHAR function to convert a date to a character string when we want to retrieve and display a date value in a non-default format. Here is a reminder of TO_CHAR:

```
SELECT first_name, TO_CHAR(birthdate, 'Month fmDD, RRRR')
FROM  f_staffs
WHERE id = 12;
```

FIRST_NAME	TO_CHAR(BIRTHDATE, 'MONTHFMDD', RRRR')
Sue	July 1, 1980

Tell Me / Show Me

Similarly, if we want to INSERT a row with a non-default format for a date column, we must use the TO_DATE function to convert the date value (a character string) to a date.

```
INSERT INTO f_staffs
   (first_name, TO_DATE(birthdate, 'Month fmDD, RRRR')
VALUES
   ('Sue', 'July 1, 1980');
```

Tell Me / Show Me

A second example of TO_DATE allows the insertion of a specific time of day, avoiding the default time of midnight.

```
INSERT INTO f_staffs
   (first_name, TO_DATE(birthdate, 'Month fmDD, RRRR HH24:MI')
VALUES
   (‘Sue’, ‘July 1, 1980 17:20’);
```

Tell Me / Show Me

USING A SUBQUERY TO COPY ROWS

Each INSERT statement we have seen so far adds only one row to the table. But suppose we want to copy 100 rows from one table to another ?

We do not want to have to write and execute 100 INSERT statements, one after the other. That would be very time-consuming.

Fortunately, SQL allows us to use a subquery within an INSERT statement. All the results from the subquery are inserted into the table. So we can copy 100 rows – or 1000 rows – with one multiple-row subquery within the INSERT.

As you would expect, you don't need a VALUES clause when using a subquery to copy rows, because the inserted values will be exactly the values returned by the subquery.

Tell Me / Show Me

USING A SUBQUERY TO COPY ROWS

In the example shown, a new table called SALES_REPS is being populated with copies of some of the rows and columns from the EMPLOYEES table.

The WHERE clause is selecting those employees that have job IDs like '%REP%'.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
   SELECT employee_id, last_name, salary, commission_pct
   FROM   employees
   WHERE  job_id LIKE '%REP%';
```

The number of columns and their data types in the column list of the INSERT clause must match the number of columns and their data types in the subquery.

The subquery is not enclosed in parentheses as is done with subqueries in a WHERE clause of a SELECT statement.

Tell Me / Show Me

USING A SUBQUERY TO COPY ROWS

If we want to copy all the data – all rows and all columns – the syntax is even simpler.

To select all rows from the EMPLOYEES table and insert them into the SALES_REPS table, the statement would be written as shown:

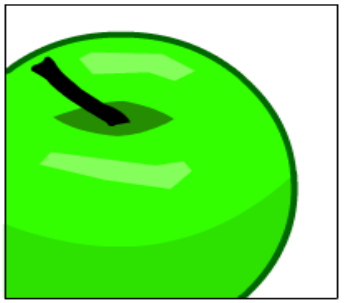
```
INSERT INTO sales_reps
  SELECT *  FROM  employees;
```

Again, this will work only if both tables have the same number of columns, in the same order, with the same data types.

Tell Me / Show Me

Terminology

Key terms used in this lesson include:



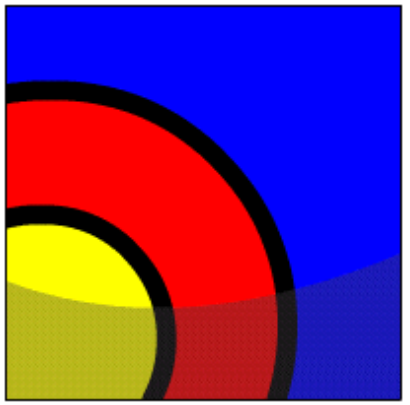
INSERT INTO

USER

Transaction

Explicit

Summary



In this lesson you have learned to:

- Give examples of why it is important to be able to alter the data in a database
- Construct and execute INSERT statements which insert a single row using a VALUES clause
- Construct and execute INSERT statements that use special values, null values, and date values
- Construct and execute INSERT statements that copy rows from one table to another using a subquery

Summary

Practice Guide

The link for the lesson practice guide can be found in the course resources in Section 0.