Database Programming with SQL
12-1: INSERT Statements

- **User -** Someone doing "real work" with the computer, using it as a means rather than an end
- **Transaction -** Consists of a collection of DML statements that form a logical unit of work.
- **Explicit -** Fully and clearly expressed; leaving nothing implied
- **INSERT -** Adds a new row to a table

1. Give two examples of why it is important to be able to alter the data in a database.

- **Updating Inventory:** In a retail database, it's essential to update product quantities after each sale to maintain accurate inventory records.
- **Correcting Errors:** If a customer's contact information was entered incorrectly, being able to update this ensures reliable communication and record accuracy.

2. DJs on Demand just purchased four new CDs. Use an explicit INSERT statement to add each CD to the copy_d_cds table. After completing the entries, execute a SELECT * statement to verify your work.

```
INSERT INTO copy_d_cds (CD_Number, Title, Producer, Year)

VALUES (97, 'Celebrate the Day', 'R & B Inc.', 2003);


INSERT INTO copy_d_cds (CD_Number, Title, Producer, Year)

VALUES (98, 'Holiday Tunes for All Ages', 'Tunes are Us', 2004);


INSERT INTO copy_d_cds (CD_Number, Title, Producer, Year)

VALUES (99, 'Party Music', 'Old Town Records', 2004);


INSERT INTO copy_d_cds (CD_Number, Title, Producer, Year)

VALUES (100, 'Best of Rock and Roll', 'Old Town Records', 2004);


## Verify entries

SELECT * FROM copy_d_cds;
```

3. DJs on Demand has two new events coming up. One event is a fall football party and the other event is a sixties theme party. The DJs on Demand clients requested the songs shown in the table for their events. Add these songs to the copy_d_songs table using an implicit INSERT statement.

```
INSERT INTO copy_d_songs
```

VALUES (52, 'Surfing Summer', NULL, 12);

INSERT INTO copy_d_songs

VALUES (53, 'Victory Victory', '5 min', 12);

5. Add the new client’s events to the copy_d_events table. The cost of each event has not been determined at this date.

INSERT INTO copy_d_events (ID, Name, Event_Date, Description, Cost, Venue_ID, Package_Code, Theme_Code, Client_Number)

VALUES (110, 'Ayako Anniversary', '07-JUL-2004', 'Party for 50, sixties dress, decorations', NULL, 245, 79, 240, 6655);

INSERT INTO copy_d_events (ID, Name, Event_Date, Description, Cost, Venue_ID, Package_Code, Theme_Code, Client_Number)

VALUES (115, 'Neuville Sports Banquet', '09-SEP-2004', 'Barbecue at residence, college alumni, 100 people', NULL, 315, 87, 340, 6689);

6. Create a table called rep_email using the following statement:

CREATE TABLE rep_email (

id NUMBER(3) CONSTRAINT rel_id_pk PRIMARY KEY,

first_name VARCHAR2(10),

last_name VARCHAR2(10),

email_address VARCHAR2(10)

);

INSERT INTO rep_email (id, first_name, last_name, email_address)

SELECT employee_id, first_name, last_name, email

FROM employees

WHERE job_title = 'REP';

<u>12-2: Updating Column Values and Deleting Rows</u>

- **UPDATE -** Modifies existing rows in a table
- **Subquery -** Retrieves information from one table & uses the information to update another table
- **Integrity Constraint -** Ensures that the data adheres to a predefined set of rules

- **Cascading Delete -** Deletes information on a linked table based on what was deleted on the other table
- **DELETE -** Removes existing rows from a table

1. Monique Tuttle, the manager of Global Fast Foods, sent a memo requesting an immediate change in prices. The price for a strawberry shake will be raised from $3.59 to $3.75, and the price for fries will increase to $1.20. Make these changes to the copy_f_food_items table.

```
UPDATE copy_f_food_items

SET price = 3.75

WHERE item_name = 'Strawberry Shake';


UPDATE copy_f_food_items

SET price = 1.20

WHERE item_name = 'Fries';
```

2. Bob Miller and Sue Doe have been outstanding employees at Global Fast Foods. Management has decided to reward them by increasing their overtime pay. Bob Miller will receive an additional $0.75 per hour and Sue Doe will receive an additional $0.85 per hour. Update the copy_f_staffs table to show these new values. (Note: Bob Miller currently doesn't get overtime pay. What function do you need to use to convert a null value to 0?)

```
UPDATE copy_f_staffs

SET overtime_pay = NVL(overtime_pay, 0) + 0.75

WHERE first_name = 'Bob' AND last_name = 'Miller';


UPDATE copy_f_staffs

SET overtime_pay = overtime_pay + 0.85

WHERE first_name = 'Sue' AND last_name = 'Doe';
```

3. Add the orders shown to the Global Fast Foods copy_f_orders table:

```
INSERT INTO copy_f_orders (ORDER_NUMBER, ORDER_DATE, ORDER_TOTAL, CUST_ID,
STAFF_ID)

VALUES (5680, '12-JUN-2004', 159.78, 145, 9);


INSERT INTO copy_f_orders (ORDER_NUMBER, ORDER_DATE, ORDER_TOTAL, CUST_ID,
STAFF_ID)

VALUES (5691, '23-SEP-2004', 145.98, 225, 12);
```

INSERT INTO copy_f_orders (ORDER_NUMBER, ORDER_DATE, ORDER_TOTAL, CUST_ID, STAFF_ID)

VALUES (5701, '04-JUL-2004', 229.31, 230, 12);

4. Add the new customers shown below to the copy_f_customers table. You may already have added Katie Hernandez. Will you be able to add all these records successfully?

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');

INSERT INTO copy_f_customers (ID, FIRST_NAME, LAST_NAME, ADDRESS, CITY, STATE, ZIP, PHONE_NUMBER)

VALUES (225, 'Daniel', 'Spode', '1923 Silverado', 'Denver', 'CO', 80219, '7193343523');

INSERT INTO copy_f_customers (ID, FIRST_NAME, LAST_NAME, ADDRESS, CITY, STATE, ZIP, PHONE_NUMBER)

VALUES (230, 'Adam', 'Zurn', '5 Admiral Way', 'Seattle', 'WA', 4258879009);

5. Sue Doe has been an outstanding Global Foods staff member and has been given a salary raise. She will now be paid the same as Bob Miller. Update her record in copy_f_staffs.

UPDATE copy_f_staffs

SET salary = (SELECT salary FROM copy_f_staffs WHERE first_name = 'Bob' AND last_name = 'Miller')

WHERE first_name = 'Sue' AND last_name = 'Doe';

6. Global Fast Foods is expanding their staff. The manager, Monique Tuttle, has hired Kai Kim. Not all information is available at this time, but add the information shown here.

INSERT INTO copy_f_staffs (ID, FIRST_NAME, LAST_NAME, BIRTHDATE, SALARY, STAFF_TYPE)

VALUES (25, 'Kai', 'Kim', '03-NOV-1988', 6.75, 'Order Taker');

7. Now that all the information is available for Kai Kim, update his Global Fast Foods record to include the following: Kai will have the same manager as Sue Doe. He does not qualify for overtime. Leave the values for training, manager budget, and manager target as null.

UPDATE copy_f_staffs

SET manager_id = (SELECT manager_id FROM copy_f_staffs WHERE first_name = 'Sue' AND last_name = 'Doe'),

overtime_pay = 0,

training = NULL,

manager_budget = NULL,

manager_target = NULL

WHERE first_name = 'Kai' AND last_name = 'Kim';

8. Execute the following SQL statement. Record your results.

DELETE FROM departments

WHERE department_id = 60;

9. Kim Kai has decided to go back to college and does not have the time to work and go to school. Delete him from the Global Fast Foods staff. Verify that the change was made.

DELETE FROM copy_f_staffs

WHERE first_name = 'Kai' AND last_name = 'Kim';

## Verify the deletion

SELECT * FROM copy_f_staffs WHERE first_name = 'Kai' AND last_name = 'Kim';

10. Create a copy of the employees table and call it lesson7_emp; Once this table exists, write a correlated delete statement that will delete any employees from the lesson7_employees table that also exist in the job_history table

-- Create a copy of the employees table

CREATE TABLE lesson7_emp AS

SELECT * FROM employees;

-- Write a correlated delete statement to delete employees in lesson7_emp that also exist in job_history

DELETE FROM lesson7_emp

WHERE employee_id IN (

SELECT employee_id

FROM job_history

);

## Verify the deletion

SELECT * FROM lesson7_emp;

12-3: DEFAULT Values, MERGE, and Multi-Table Inserts

1. When would you want a DEFAULT value?

To ensure that a column always has a sensible initial value even if no value is explicitly provided during an INSERT statement.

2. Currently, the Global Foods F_PROMOTIONAL_MENUS table START_DATE column does not have SYSDATE set as DEFAULT. Your manager has decided she would like to be able to set the starting date of promotions to the current day for some entries. This will require three steps:
a. In your schema, Make a copy of the Global Foods F_PROMOTIONAL_MENUS table using the following SQL statement:

```
CREATE TABLE copy_f_promotional_menus
AS (SELECT * FROM f_promotional_menus) ;
```

b. Alter the current START_DATE column attributes using:

```
ALTER TABLE copy_f_promotional_menus
MODIFY(start_date DATE DEFAULT SYSDATE) ;
```

c. INSERT the new information and check to verify the results.
INSERT a new row into the copy_f_promotional_menus table for the manager's new promotion. The promotion code is 120. The name of the promotion is 'New Customer.' Enter DEFAULT for the start date and '01-Jun-2005' for the ending date. The giveaway is a 10% discount coupon. What was the correct syntax used?

```
INSERT INTO copy_f_promotional_menus (promotion_code, promotion_name, start_date,
end_date, giveaway)

VALUES (120, 'New Customer', DEFAULT, '01-JUN-2005', '10% discount coupon');



## Verify the results

SELECT * FROM copy_f_promotional_menus WHERE promotion_code = 120;
```

3. Allison Plumb, the event planning manager for DJs on Demand, has just given you the following list of CDs she acquired from a company going out of business. She wants a new updated list of CDs in inventory in an hour, but she doesn't want the original D_CDS table changed. Prepare an updated inventory list just for her.
a. Assign new cd_numbers to each new CD acquired.

```
CREATE TABLE manager_copy_d_cds

AS (SELECT * FROM d_cds);
```

b. Create a copy of the D_CDS table called manager_copy_d_cds. What was the correct syntax used?

```
CREATE TABLE manager_copy_d_cds

AS (SELECT * FROM d_cds);
```

c. INSERT into the manager_copy_d_cds table each new CD title using an INSERT statement. Make up one example or use this data:

20, 'Hello World Here I Am', 'Middle Earth Records', '1998'
What was the correct syntax used?

```
INSERT INTO manager_copy_d_cds (cd_number, title, producer, year)

VALUES (20, 'Hello World Here I Am', 'Middle Earth Records', 1998);
```

d. Use a merge statement to add to the manager_copy_d_cds table, the CDs from the original table. If there is a match, update the title and year. If not, insert the data from the original table. What was the correct syntax used?

```
MERGE INTO manager_copy_d_cds mc

USING d_cds d

ON (mc.cd_number = d.cd_number)

WHEN MATCHED THEN

  UPDATE SET mc.title = d.title, mc.year = d.year

WHEN NOT MATCHED THEN

  INSERT (cd_number, title, producer, year)

VALUES (d.cd_number, d.title, d.producer, d.year);
```

4. Run the following 3 statements to create 3 new tables for use in a Multi-table insert statement. All 3 tables should be empty on creation, hence the WHERE 1=2 condition in the WHERE clause.

```
CREATE TABLE sal_history (employee_id, hire_date, salary)

AS SELECT employee_id, hire_date, salary

FROM employees

WHERE 1=2;


CREATE TABLE mgr_history (employee_id, manager_id, salary)

AS SELECT employee_id, manager_id, salary

FROM employees

WHERE 1=2;


CREATE TABLE special_sal (employee_id, salary)

AS SELECT employee_id, salary

FROM employees

WHERE 1=2;
```

Once the tables exist in your account, write a Multi-Table insert statement to first select the

employee_id, hire_date, salary, and manager_id of all employees. If the salary is more than 20000 insert the employee_id and salary into the special_sal table. Insert the details of employee_id, hire_date, and salary into the sal_history table. Insert the employee_id, manager_id, and salary into the mgr_history table. You should get a message back saying 39 rows were inserted. Verify you get this message and verify you have the following number of rows in each table:
Sal_history: 19 rows
Mgr_history: 19 rows
Special_sal: 1

```
INSERT ALL
 WHEN salary > 20000 THEN
  INTO special_sal (employee_id, salary)
  VALUES (employee_id, salary)
 INTO sal_history (employee_id, hire_date, salary)
 VALUES (employee_id, hire_date, salary)
 INTO mgr_history (employee_id, manager_id, salary)
 VALUES (employee_id, manager_id, salary)
SELECT employee_id, hire_date, salary, manager_id
FROM employees;


## Verify the number of rows in each table
SELECT COUNT(*) AS sal_history_count FROM sal_history;   -- Expected: 19 rows
SELECT COUNT(*) AS mgr_history_count FROM mgr_history;   -- Expected: 19 rows
SELECT COUNT(*) AS special_sal_count FROM special_sal;   -- Expected: 1 row
```

<u>13-1: Creating Tables</u>

- **Data Dictionary -** Created and maintained by the Oracle Server and contains information about the database
- **Schema -** A collection of objects that are the logical structures that directly refer to the data in the database
- **DEFAULT -** Specifies a preset value if a value is omitted in the INSERT statement
- **Table -** Stores data; basic unit of storage composed of rows and columns
- **CREATE TABLE -** Command use to make a new table

1. Complete the GRADUATE CANDIDATE table instance chart. Credits is a foreign-key column referencing the requirements table.
2. Write the syntax to create the grad_candidates table.

CREATE TABLE grad_candidates (

student_id NUMBER(6) PRIMARY KEY,

last_name VARCHAR2(30) NOT NULL,

first_name VARCHAR2(30) NOT NULL,

credits NUMBER,

graduation_date DATE,

CONSTRAINT fk_credits FOREIGN KEY (credits) REFERENCES requirements (credits)

);

3. Confirm creation of the table using DESCRIBE.

DESCRIBE grad_candidates;

4. Create a new table using a subquery. Name the new table your last name -- e.g., smith_table. Using a subquery, copy grad_candidates into smith_table.

CREATE TABLE smith_table AS

SELECT * FROM grad_candidates;

5. Insert your personal data into the table created in question 4.

INSERT INTO smith_table (student_id, last_name, first_name, credits, graduation_date)

VALUES (123456, 'Smith', 'John', 20, '31-MAY-2024');

6. Query the data dictionary for each of the following:

• USER_TABLES

SELECT * FROM USER_TABLES;

This query returns a list of all the tables owned by the current user, including information like table names, tablespace used, and the number of rows.

• USER_OBJECTS

SELECT * FROM USER_OBJECTS;

This query returns a list of all schema objects (e.g., tables, views, indexes, sequences) owned by the current user. It provides details such as the object's name, type, status, and creation date.

• USER_CATALOG or USER_CAT

SELECT * FROM USER_CATALOG;

This query returns a summary of all types of database objects (e.g., tables, views, synonyms, sequences) owned by the user, providing an overview of the catalog of objects in the user's schema.

13-2: Using Data Types

- **INTERVAL YEAR TO MONTH -** Allows time to be stored as an interval of years and months

- **TIMESTAMP WITH LOCAL TIME ZONE -** When a column is selected in a SQL statement the time is automatically converted to the user's timezone
- **BLOB -** Binary large object data up to 4 gigabytes
- **TIMWSTAMP WITH TIME ZONE -** Stores a time zone value as a displacement from Universal Coordinated Time or UCT
- **INTERVAL DAY TO SECOND -** Allows time to be stored as an interval of days to hours, minutes, and seconds
- **CLOB -** Character data up to 4 gigabytes
- **TIMESTAMP -** Allows the time to be stored as a date with fractional seconds

1. Create tables using each of the listed time-zone data types, use your time-zone and one other in your examples. Answers will vary.
   a. TIMESTAMP WITH LOCAL TIME ZONE

```
CREATE TABLE event_log_local (
   event_id NUMBER PRIMARY KEY,
   event_name VARCHAR2(100),
   event_time TIMESTAMP WITH LOCAL TIME ZONE
);


## Insert example data
INSERT INTO event_log_local (event_id, event_name, event_time)
VALUES (1, 'Local Event', TIMESTAMP '2024-11-12 14:30:00');


## Verify entries
SELECT * FROM event_log_local;
```

b. INTERVAL YEAR TO MONTH

```
CREATE TABLE project_duration (
   project_id NUMBER PRIMARY KEY,
   project_name VARCHAR2(100),
   duration INTERVAL YEAR TO MONTH
);


## Insert example data
INSERT INTO project_duration (project_id, project_name, duration)
```

```
VALUES (1, 'Construction Project', INTERVAL '2-6' YEAR TO MONTH);


## Verify entries

SELECT * FROM project_duration;
```

c. INTERVAL DAY TO SECOND

```
CREATE TABLE task_duration (

  task_id NUMBER PRIMARY KEY,

  task_name VARCHAR2(100),

  duration INTERVAL DAY TO SECOND

);


## Insert example data

INSERT INTO task_duration (task_id, task_name, duration)

VALUES (1, 'Data Processing', INTERVAL '3 12:30:15' DAY TO SECOND);


## Verify entries

SELECT * FROM task_duration;
```

2. Execute a SELECT * from each table to verify your input.

```
## Verify data in event_log_local table

SELECT * FROM event_log_local;


## Verify data in project_duration table

SELECT * FROM project_duration;


## Verify data in task_duration table

SELECT * FROM task_duration;
```

3. Give 3 examples of organizations and personal situations where it is important to know to which time zone a date-time value refers

- **Airline Companies: Scheduling flights and ensuring all departure and arrival times are accurate across different time zones.**
- **International Banking: Processing transactions across branches in different countries, ensuring that timestamps align to avoid discrepancies.**

- **Remote Work Meetings: Planning virtual meetings between team members located in different time zones to ensure no one misses a meeting due to time zone confusion.**

<u>13-3: Modifying a Table</u>

1. Why is it important to be able to modify a table?

To ensure the database can accommodate changing requirements. For example, new business requirements may necessitate adding new columns, updating existing data types, or removing obsolete columns.

2. CREATE a table called Artists.

a. Add the following to the table:

```
CREATE TABLE artists (
  artist_id NUMBER PRIMARY KEY,
  first_name VARCHAR2(30),
  last_name VARCHAR2(30),
  band_name VARCHAR2(50),
  email VARCHAR2(50),
  hourly_rate NUMBER(6, 2)
);
```

b. INSERT one artist from the d_songs table.

```
INSERT INTO artists (artist_id, first_name, last_name, band_name, email, hourly_rate)
SELECT artist_id, first_name, last_name, band_name, email, hourly_rate
FROM d_songs
WHERE artist_id = 1;  -- Assuming there is an artist with `artist_id` = 1
```

c. INSERT one artist of your own choosing.

```
INSERT INTO artists (artist_id, first_name, last_name, band_name, email, hourly_rate)
VALUES (2, 'John', 'Doe', 'The Rockers', 'jdoe@rockers.com', 50.00);
```

d. Give an example how each of the following may be used on the table that you have created:

1) ALTER TABLE

```
ALTER TABLE artists
ADD (genre VARCHAR2(30));
```

2) DROP TABLE

```
DROP TABLE artists;
```

3) RENAME TABLE

RENAME artists TO band_artists;

4) TRUNCATE

TRUNCATE TABLE artists;

5) COMMENT ON TABLE

COMMENT ON TABLE artists IS 'Table of artists with band information and rates';

3. In your o_employees table, enter a new column called "Termination." The datatype for the new column should be VARCHAR2. Set the DEFAULT for this column as SYSDATE to appear as character data in the format: February 20th, 2003.

ALTER TABLE o_employees

ADD (termination VARCHAR2(30) DEFAULT TO_CHAR(SYSDATE, 'Month DDth, YYYY'));

4. Create a new column in the o_employees table called start_date. Use the TIMESTAMP WITH LOCAL TIME ZONE as the datatype.

ALTER TABLE o_employees

ADD (start_date TIMESTAMP WITH LOCAL TIME ZONE);

5. Truncate the o_jobs table. Then do a SELECT * statement. Are the columns still there? Is the data still there?

TRUNCATE TABLE o_jobs;

6. What is the distinction between TRUNCATE, DELETE, and DROP for tables?

- **TRUNCATE: Removes all rows from a table, but the table structure remains. It is faster and uses fewer resources because it does not generate individual row delete operations.**
- **DELETE: Removes rows from a table based on a condition. You can delete specific rows, and it generates undo logs.**
- **DROP: Completely removes the table structure and all its data from the database.**

7. List the changes that can and cannot be made to a column.

- **Can Be Made: Change data type (to a compatible type), add default values, allow/disallow nulls, add comments, set column unused.**
- **Cannot Be Made: Decrease column size if data will be truncated, change data type to an incompatible type, rename a column directly in some cases without using ALTER TABLE... RENAME.**

8. Add the following comment to the o_jobs table: "New job description added" View the data dictionary to view your comments.

COMMENT ON TABLE o_jobs IS 'New job description added';

SELECT * FROM USER_TAB_COMMENTS WHERE table_name = 'O_JOBS';

9. Rename the o_jobs table to o_job_description.

RENAME o_jobs TO o_job_description;

10. F_staffs table exercises:

a. Create a copy of the f_staffs table called copy_f_staffs and use this copy table for the remaining labs in this lesson.

```
CREATE TABLE copy_f_staffs AS

SELECT * FROM f_staffs;
```

b. Describe the new table to make sure it exists.

```
DESCRIBE copy_f_staffs;
```

c. Drop the table.

```
DROP TABLE copy_f_staffs;
```

d. Try to select from the table.

```
SELECT * FROM copy_f_staffs;  -- Should fail
```

e. Investigate your recyclebin to see where the table went.

```
SHOW RECYCLEBIN;
```

f. Try to select from the dropped table by using the value stored in the OBJECT_NAME column.

```
SELECT * FROM "BIN$Q+x1nJdcUnngQESYELVIdQ==$0";
```

g. Undrop the table.

```
FLASHBACK TABLE copy_f_staffs TO BEFORE DROP;
```

h. Describe the table.

```
DESCRIBE copy_f_staffs;
```

11. Still working with the copy_f_staffs table, perform an update on the table.

a. Issue a select statement to see all rows and all columns from the copy_f_staffs table;

```
SELECT * FROM copy_f_staffs;
```

b. Change the salary for Sue Doe to 12 and commit the change.

```
UPDATE copy_f_staffs

SET salary = 12

WHERE first_name = 'Sue' AND last_name = 'Doe';



COMMIT;
```

c. Issue a select statement to see all rows and all columns from the copy_f_staffs table;

```
SELECT * FROM copy_f_staffs;
```

d. For Sue Doe, update the salary to 2 and commit the change.

```
UPDATE copy_f_staffs

SET salary = 2

WHERE first_name = 'Sue' AND last_name = 'Doe';
```

```
COMMIT;
```

e. Issue a select statement to see all rows and all columns from the copy_f_staffs table;

```
SELECT * FROM copy_f_staffs;
```

f. Now, issue a FLASHBACK QUERY statement against the copy_f_staffs table, so you can see all the changes made.

```
SELECT *

FROM copy_f_staffs AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '10' MINUTE)

WHERE first_name = 'Sue' AND last_name = 'Doe';
```

g. Investigate the result of f), and find the original salary and update the copy_f_staffs table salary column for Sue Doe back to her original salary.

```
UPDATE copy_f_staffs

SET salary = <original_salary_value>

WHERE first_name = 'Sue' AND last_name = 'Doe';


COMMIT;
```