Creating a Simple Table Using SQL

-- create a simple table for keeping track of birthdays

CREATE TABLE my\_birthdays

( first\_name VARCHAR2(20),

last\_name VARCHAR2(25),

bday\_date DATE

);

Creating a Table With NOT NULL Constraints Using SQL

-- create a table with NOT NULL constraints in the HR schema

CREATE TABLE personal\_info (

employee\_id NUMBER(6,0) NOT NULL,

birth\_date DATE NOT NULL,

social\_security\_id VARCHAR2(12) NOT NULL,

marital\_status VARCHAR2(10),

dependents\_claimed NUMBER(2,0) DEFAULT 1,

contact\_name VARCHAR2(45) NOT NULL,

contact\_phone VARCHAR2(20) NOT NULL,

contact\_address VARCHAR2(80) NOT NULL

);

-- add a new column

ALTER TABLE personal\_info ADD

(contact\_email VARCHAR2(30) NULL);

-- modify a column

ALTER TABLE personal\_info MODIFY

(contact\_email VARCHAR2(40) NOT NULL);

-- drop a column

ALTER TABLE personal\_info DROP COLUMN

contact\_address;

ALTER TABLE personal\_info

ADD CONSTRAINT personal\_info\_pkey

PRIMARY KEY (employee\_id);

-- add a foreign key constraint

ALTER TABLE personal\_info

ADD CONSTRAINT personal\_info\_fkey

FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id)

ON DELETE CASCADE;

-- add a unique constraint

ALTER TABLE personal\_info

ADD CONSTRAINT personal\_info\_unique\_con

UNIQUE (social\_security\_id);

-- add a check constraint

ALTER TABLE personal\_info

ADD CONSTRAINT personal\_info\_check\_con

CHECK ( dependents\_claimed > 0);

-- disable a constraint

ALTER TABLE personal\_info

DISABLE CONSTRAINT personal\_info\_check\_con;

-- enable a constraint

ALTER TABLE personal\_info

ENABLE CONSTRAINT personal\_info\_check\_con;

-- drop a constraint

ALTER TABLE personal\_info

DROP CONSTRAINT personal\_info\_check\_con;

ALTER TABLE my\_birthdays RENAME to birthdays;

Dropping a Table Using SQL

-- drop tables from the database

-- use caution when use the DROP statement!

DROP TABLE birthdays;

DROP TABLE personal\_info;

CREATE OR REPLACE VIEW my\_emp\_view AS

SELECT d.department\_id, d.department\_name, d.manager\_id,

e.employee\_id, e.first\_name, e.last\_name

FROM employees e, departments d

WHERE d.manager\_id = e.employee\_id;

Dropping a View Using SQL

-- drop the view with the DROP VIEW statement

DROP VIEW my\_emp\_view;

Using a Simple PL/SQL Block

-- the following is an optional declarative part

DECLARE

monthly\_salary NUMBER(6);

number\_of\_days\_worked NUMBER(2);

pay\_per\_day NUMBER(6,2);

-- the following is the executable part, from BEGIN to END

BEGIN

monthly\_salary := 2290;

number\_of\_days\_worked := 21;

pay\_per\_day := monthly\_salary/number\_of\_days\_worked;

-- the following displays output from the PL/SQL block

DBMS\_OUTPUT.PUT\_LINE('The pay per day is ' || TO\_CHAR(pay\_per\_day));

-- the following is an optional exception part that handles errors

EXCEPTION

WHEN ZERO\_DIVIDE THEN

pay\_per\_day := 0;

END;

/

Using DBMS\_OUTPUT.PUT\_LINE to Display PL/SQL Output

-- enable SERVEROUTPUT in SQL Command Line (SQL\*Plus) to display output with

-- DBMS\_OUTPUT.PUT\_LINE, this enables SERVEROUTPUT for this SQL\*Plus session only

SET SERVEROUTPUT ON

DECLARE

answer VARCHAR2(20); -- declare a variable

BEGIN

-- assign a value to a variable

answer := 'Maybe';

-- use PUT\_LINE to display data from the PL/SQL block

DBMS\_OUTPUT.PUT\_LINE( 'The answer is: ' || answer );

END;

/

Assigning Values to Variables With the PL/SQL Assignment Operator

DECLARE -- declare and assiging variables

wages NUMBER(6,2);

hours\_worked NUMBER := 40;

hourly\_salary NUMBER := 22.50;

bonus NUMBER := 150;

country VARCHAR2(128);

counter NUMBER := 0;

done BOOLEAN := FALSE;

valid\_id BOOLEAN;

BEGIN

wages := (hours\_worked \* hourly\_salary) + bonus; -- compute wages

country := 'France'; -- assign a string literal

country := UPPER('Canada'); -- assign an uppercase string literal

done := (counter > 100); -- assign a BOOLEAN, in this case FALSE

valid\_id := TRUE; -- assign a BOOLEAN

END;

/

DECLARE -- declare and assign variables

number1 PLS\_INTEGER := 32000; -- numeric literal

number2 NUMBER(8,3);

BEGIN

number2 := 3.125346e3; -- numeric literal

number2 := -8300.00; -- numeric literal

number2 := -14; -- numeric literal

END;

/

Assigning Values to Variables Using PL/SQL SELECT INTO

DECLARE -- declare and assign values

bonus\_rate CONSTANT NUMBER(2,3) := 0.05;

bonus NUMBER(8,2);

emp\_id NUMBER(6) := 120; -- assign a test value for employee ID

BEGIN

-- retreive a salary from the employees table, then calculate the bonus and

-- assign the value to the bonus variable

SELECT salary \* bonus\_rate INTO bonus FROM employees

WHERE employee\_id = emp\_id;

-- display the employee\_id, bonus amount, and bonus rate

DBMS\_OUTPUT.PUT\_LINE ( 'Employee: ' || TO\_CHAR(emp\_id)

|| ' Bonus: ' || TO\_CHAR(bonus) || ' Bonus Rate: ' || TO\_CHAR(bonus\_rate));

END;

/

DECLARE -- declare variables using %TYPE attribute

empid employees.employee\_id%TYPE; -- employee\_id datatype is NUMBER(6)

emplname employees.last\_name%TYPE; -- last\_name datatype is VARCHAR2(25)

BEGIN

empid := 100301; -- this is OK because it fits in NUMBER(6)

-- empid := 3018907; -- this is too large and will cause an overflow

emplname := 'Patel'; -- this is OK because it fits in VARCHAR2(25)

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || empid); -- display data

DBMS\_OUTPUT.PUT\_LINE('Employee name: ' || emplname); -- display data

END;

/

Using %ROWTYPE with a PL/SQL Record

DECLARE -- declare variables

-- declare record variable that represents a row fetched from the employees table

emp\_rec employees%ROWTYPE; -- declare variable with %ROWTYPE attribute

BEGIN

SELECT \* INTO emp\_rec FROM EMPLOYEES WHERE employee\_id = 120; -- retrieve record

DBMS\_OUTPUT.PUT\_LINE('Employee name: ' || emp\_rec.first\_name || ' '

|| emp\_rec.last\_name); -- display

END;

/

Using a Simple IF-THEN Statement in PL/SQL

DECLARE

sal NUMBER(8,2);

bonus NUMBER(6,2);

hiredate DATE;

empid NUMBER(6) := 128; -- use employee 120 for testing

BEGIN

-- retrieve the salary and the date that employee was hired, the date is checked

-- to calculate the amount of the bonus for the employee

SELECT salary, hire\_date INTO sal, hiredate FROM employees

WHERE employee\_id = empid;

IF hiredate > TO\_DATE('01-JAN-00') THEN

bonus := sal/20;

DBMS\_OUTPUT.PUT\_LINE('Bonus for employee: ' || empid || ' is: ' || bonus );

END IF;

END;

/

Using the IF-THEN-ELSEIF Statement in PL/SQL

DECLARE

bonus NUMBER(6,2);

empid NUMBER(6) := 120;

hiredate DATE;

BEGIN

-- retrieve the date that employee was hired, the date is checked

-- to determine the amount of the bonus for the employee

SELECT hire\_date INTO hiredate FROM employees WHERE employee\_id = empid;

IF hiredate > TO\_DATE('01-JAN-98') THEN

bonus := 500;

ELSIF hiredate > TO\_DATE('01-JAN-96') THEN

bonus := 1000;

ELSE

bonus := 1500;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Bonus for employee: ' || empid || ' is: ' || bonus );

END;

/

Using the FOR-LOOP in PL/SQL

BEGIN

-- use a FOR loop to process a series of numbers

FOR loop\_counter IN 1..10 LOOP

DBMS\_OUTPUT.PUT\_LINE('Number: ' || TO\_CHAR(loop\_counter)

|| ' Square: ' || TO\_CHAR(loop\_counter\*\*2));

END LOOP;

END;

/

Using WHILE-LOOP for Control in PL/SQL

DECLARE -- declare variables

i NUMBER := 1; -- loop counter, initialize to one

i\_cubed NUMBER;

BEGIN

-- use WHILE LOOP to process data

WHILE i <= 10 LOOP

i\_cubed := i\*\*3;

DBMS\_OUTPUT.PUT\_LINE('Number: ' || TO\_CHAR(i)

|| ' Cube: ' || TO\_CHAR(i\_cubed));

i := i + 1;

END LOOP;

END;

/

Using the EXIT-WHEN Statement in PL/SQL

DECLARE -- declare and assign values to variables

total NUMBER(9) := 0;

counter NUMBER(6) := 0;

BEGIN

LOOP

counter := counter + 1; -- increment counter variable

total := total + counter \* counter; -- compute total

-- exit loop when condition is true

EXIT WHEN total > 25000; -- LOOP until condition is met

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Counter: ' || TO\_CHAR(counter)

|| ' Total: ' || TO\_CHAR(total)); -- display results

END;

/

[2 Day Developer](http://127.0.0.1:8080/i/doc/XEDEV/preface.htm) > [Using PL/SQL](http://127.0.0.1:8080/i/doc/XEDEV/xedev_plsql.htm) > [Using the Main Features of ...](http://127.0.0.1:8080/i/doc/XEDEV/xedev_plsql_utilizing.htm) > Using Cursors To Retrieve Data

|  |  |  |
| --- | --- | --- |
| **Using Cursors To Retrieve Data** | [[Previous](http://127.0.0.1:8080/i/doc/XEDEV/xedev_plsql013.htm) Previous](http://127.0.0.1:8080/i/doc/XEDEV/xedev_plsql013.htm) | [[Next](http://127.0.0.1:8080/i/doc/XEDEV/xedev_plsql015.htm) Next](http://127.0.0.1:8080/i/doc/XEDEV/xedev_plsql015.htm) |

A cursor is a name for a private SQL area in which information for processing the specific statement is kept. PL/SQL uses both implicit and explicit cursors. PL/SQL implicitly creates a cursor for all SQL data manipulation statements on a set of rows, including queries that return only one row.

You can explicitly declare a cursor for one row or multiple rows. For queries that return multiple rows, you can process the rows individually.

Cursor attributes return useful information about the status of cursors in the execution of SQL statements.

Fetching a Single Row With a Cursor in PL/SQL

DECLARE

-- declare variables for first\_name and last\_name fetched from the employees table

firstname employees.first\_name%TYPE; -- variable for first\_name

lastname employees.last\_name%TYPE; -- variable for last\_name

-- declare a cursor to fetch data from a row (employee 120) in the employees table

CURSOR cursor1 IS

SELECT first\_name, last\_name FROM employees WHERE employee\_id = 120;

BEGIN

OPEN cursor1; -- open the cursor

FETCH cursor1 INTO firstname, lastname; -- fetch data into local variables

DBMS\_OUTPUT.PUT\_LINE('Employee name: ' || firstname || ' ' || lastname);

CLOSE cursor1; -- close the cursor

END;

/

Fetching Multiple Rows With a Cursor in PL/SQL

DECLARE

-- declare variables for data fetched from cursors

empid employees.employee\_id%TYPE; -- variable for employee\_id

jobid employees.job\_id%TYPE; -- variable for job\_id

lastname employees.last\_name%TYPE; -- variable for last\_name

rowcount NUMBER;

-- declare the cursors

CURSOR cursor1 IS SELECT last\_name, job\_id FROM employees

WHERE job\_id LIKE '%CLERK';

CURSOR cursor2 is SELECT employee\_id, last\_name, job\_id FROM employees

WHERE job\_id LIKE '%MAN' OR job\_id LIKE '%MGR';

BEGIN

-- start the processing with cursor1

OPEN cursor1; -- open cursor1 before fetching

DBMS\_OUTPUT.PUT\_LINE( '---------- cursor 1-----------------' );

LOOP

FETCH cursor1 INTO lastname, jobid; -- fetches 2 columns into variables

-- check the cursor attribute NOTFOUND for the end of data

EXIT WHEN cursor1%NOTFOUND;

-- display the last name and job ID for each record (row) fetched

DBMS\_OUTPUT.PUT\_LINE( RPAD(lastname, 25, ' ') || jobid );

END LOOP;

rowcount := cursor1%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('The number of rows fetched is ' || rowcount );

CLOSE cursor1;

-- start the processing with cursor2

OPEN cursor2;

DBMS\_OUTPUT.PUT\_LINE( '---------- cursor 2-----------------' );

LOOP

-- fetch 3 columns into the variables

FETCH cursor2 INTO empid, lastname, jobid;

EXIT WHEN cursor2%NOTFOUND;

-- display the employee ID, last name, and job ID for each record (row) fetched

DBMS\_OUTPUT.PUT\_LINE( empid || ': ' || RPAD(lastname, 25, ' ') || jobid );

END LOOP;

rowcount := cursor2%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('The number of rows fetched is ' || rowcount );

CLOSE cursor2;

END;

/

Passing Parameters to a Cursor in PL/SQL

DECLARE

-- declare variables for data fetched from cursor

empid employees.employee\_id%TYPE; -- variable for employee\_id

hiredate employees.hire\_date%TYPE; -- variable for hire\_date

firstname employees.first\_name%TYPE; -- variable for first\_name

lastname employees.last\_name%TYPE; -- variable for last\_name

rowcount NUMBER;

bonusamount NUMBER;

yearsworked NUMBER;

-- declare the cursor with a parameter,

CURSOR cursor1 (thismonth NUMBER)IS

SELECT employee\_id, first\_name, last\_name, hire\_date FROM employees

WHERE EXTRACT(MONTH FROM hire\_date) = thismonth;

BEGIN

-- open and pass a parameter to cursor1, select employees hired on this month

OPEN cursor1(EXTRACT(MONTH FROM SYSDATE));

DBMS\_OUTPUT.PUT\_LINE('----- Today is ' || TO\_CHAR(SYSDATE, 'DL') || ' -----');

DBMS\_OUTPUT.PUT\_LINE('Employees with yearly bonus amounts:');

LOOP

-- fetches 4 columns into variables

FETCH cursor1 INTO empid, firstname, lastname, hiredate;

-- check the cursor attribute NOTFOUND for the end of data

EXIT WHEN cursor1%NOTFOUND;

-- calculate the yearly bonus amount based on months (years) worked

yearsworked := ROUND( (MONTHS\_BETWEEN(SYSDATE, hiredate)/12) );

IF yearsworked > 10 THEN bonusamount := 2000;

ELSIF yearsworked > 8 THEN bonusamount := 1600;

ELSIF yearsworked > 6 THEN bonusamount := 1200;

ELSIF yearsworked > 4 THEN bonusamount := 800;

ELSIF yearsworked > 2 THEN bonusamount := 400;

ELSIF yearsworked > 0 THEN bonusamount := 100;

END IF;

-- display the employee Id, first name, last name, hire date, and bonus

-- for each record (row) fetched

DBMS\_OUTPUT.PUT\_LINE( empid || ' ' || RPAD(firstname, 21, ' ') ||

RPAD(lastname, 26, ' ') || hiredate || TO\_CHAR(bonusamount, '$9,999'));

END LOOP;

rowcount := cursor1%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('The number of rows fetched is ' || rowcount );

CLOSE cursor1;

END;

/

|  |
| --- |
| See Also:  [Oracle Database PL/SQL User's Guide and Reference](http://www.oracle.com/pls/xe102/to_xlink?xlink=LNPLS00602) for information about managing cursors with PL/SQL |

**Cursor Attributes**

Cursor attributes return information about the execution of DML and DDL statements, such INSERT, UPDATE, DELETE, SELECT INTO, COMMIT, or ROLLBACK statements. The cursor attributes are %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. These attributes return useful information about the most recently executed SQL statement. When using an explicit cursor, add the explicit cursor or cursor variable name to the beginning of the attribute, such as cursor1%FOUND, to return information for the most recently executed SQL statement for that cursor.

The attributes provide the following information:

* %FOUND Attribute: Has a Row Been Fetched?

After a cursor or cursor variable is opened but before the first fetch, %FOUND returns NULL. After any fetches, it returns TRUE if the last fetch returned a row, or FALSE if the last fetch did not return a row.

* %ISOPEN Attribute: Is the Cursor Open?

If a cursor or cursor variable is open, then %ISOPEN returns TRUE ; otherwise, %ISOPEN returns FALSE.

Note that implicit cursors are automatically opened before and closed after executing the associated SQL statement so %ISOPEN always returns FALSE.

* %NOTFOUND Attribute: Has a Fetch Failed?

If the last fetch returned a row, then %NOTFOUND returns FALSE. If the last fetch failed to return a row, then %NOTFOUND returns TRUE. %NOTFOUND is the logical opposite of %FOUND.

* %ROWCOUNT Attribute: How Many Rows Fetched So Far?

After a cursor or cursor variable is opened, %ROWCOUNT returns 0 before the first fetch. Thereafter, it returns the number of rows fetched so far. The number is incremented if the last fetch returned a row.