Use one of the following command:

* To list all the tables related the current user:
  + SELECT table\_name FROM user\_tables;
* List the tables accessible by the user:
  + SELECT table\_name FROM all\_tables;
* List all the tables (ADMIN):
  + SELECT table\_name FROM dba\_tables;

-

**ORACLE/PLSQL: BEFORE INSERT TRIGGER**

This Oracle tutorial explains how to **create a BEFORE INSERT Trigger** in Oracle with syntax and examples.

**DESCRIPTION**

A **BEFORE INSERT Trigger** means that Oracle will fire this trigger before the INSERT operation is executed.

**SYNTAX**

The syntax to create a **BEFORE INSERT Trigger** in Oracle/PLSQL is:

CREATE [ OR REPLACE ] TRIGGER *trigger\_name*

BEFORE INSERT

ON *table\_name*

[ FOR EACH ROW ]

DECLARE

-- variable declarations

BEGIN

-- trigger code

EXCEPTION

WHEN ...

-- exception handling

END;

Parameters or Arguments

**OR REPLACE**

Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a [DROP TRIGGER statement](http://www.techonthenet.com/oracle/triggers/drop.php).

**trigger\_name**

The name of the trigger to create.

**BEFORE INSERT**

It indicates that the trigger will fire before the INSERT operation is executed.

**table\_name**

The name of the table that the trigger is created on.

**RESTRICTIONS**

* You can not create a BEFORE trigger on a view.
* You can update the :NEW values.
* You can not update the :OLD values.

**NOTE**

* See also how to create [AFTER DELETE](http://www.techonthenet.com/oracle/triggers/after_delete.php), [AFTER INSERT](http://www.techonthenet.com/oracle/triggers/after_insert.php), [AFTER UPDATE](http://www.techonthenet.com/oracle/triggers/after_update.php), [BEFORE DELETE](http://www.techonthenet.com/oracle/triggers/before_delete.php), and [BEFORE UPDATE](http://www.techonthenet.com/oracle/triggers/before_update.php)triggers.
* See also how to [drop a trigger](http://www.techonthenet.com/oracle/triggers/drop.php).

**EXAMPLE**

Let's look at an example of how to create an BEFORE INSERT trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

CREATE TABLE orders

( order\_id number(5),

quantity number(4),

cost\_per\_item number(6,2),

total\_cost number(8,2),

create\_date date,

created\_by varchar2(10)

);

We could then use the CREATE TRIGGER statement to create an BEFORE INSERT trigger as follows:

**TIP:** When using SQLPlus, you need to enter slash on a new line after the trigger. Otherwise, the script won't execute.

CREATE OR REPLACE TRIGGER orders\_before\_insert

BEFORE INSERT

ON orders

FOR EACH ROW

DECLARE

v\_username varchar2(10);

BEGIN

-- Find username of person performing INSERT into table

SELECT user INTO v\_username

FROM dual;

-- Update create\_date field to current system date

:new.create\_date := sysdate;

-- Update created\_by field to the username of the person performing the INSERT

:new.created\_by := v\_username;

END;

/

**ORACLE/PLSQL: AFTER DELETE TRIGGER**

This Oracle tutorial explains how to **create an AFTER DELETE Trigger** in Oracle with syntax and examples.

**DESCRIPTION**

An **AFTER DELETE Trigger** means that Oracle will fire this trigger after the DELETE operation is executed.

**SYNTAX**

The syntax to create an **AFTER DELETE Trigger** in Oracle/PLSQL is:

CREATE [ OR REPLACE ] TRIGGER *trigger\_name*

AFTER DELETE

ON *table\_name*

[ FOR EACH ROW ]

DECLARE

-- variable declarations

BEGIN

-- trigger code

EXCEPTION

WHEN ...

-- exception handling

END;

Parameters or Arguments

**OR REPLACE**

Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a [DROP TRIGGER statement](http://www.techonthenet.com/oracle/triggers/drop.php).

**trigger\_name**

The name of the trigger to create.

**AFTER DELETE**

It indicates that the trigger will fire after the DELETE operation is executed.

**table\_name**

The name of the table that the trigger is created on.

**RESTRICTIONS**

* You can not create an AFTER trigger on a view.
* You can not update the :NEW values.
* You can not update the :OLD values.

**NOTE**

* See also how to create [AFTER INSERT](http://www.techonthenet.com/oracle/triggers/after_insert.php), [AFTER UPDATE](http://www.techonthenet.com/oracle/triggers/after_update.php), [BEFORE DELETE](http://www.techonthenet.com/oracle/triggers/before_delete.php), [BEFORE INSERT](http://www.techonthenet.com/oracle/triggers/before_insert.php), and [BEFORE UPDATE](http://www.techonthenet.com/oracle/triggers/before_update.php)triggers.
* See also how to [drop a trigger](http://www.techonthenet.com/oracle/triggers/drop.php).

**EXAMPLE**

Let's look at an example of how to create an AFTER DELETE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

CREATE TABLE orders

( order\_id number(5),

quantity number(4),

cost\_per\_item number(6,2),

total\_cost number(8,2)

);

We could then use the CREATE TRIGGER statement to create an AFTER DELETE trigger as follows:

**TIP:** When using SQLPlus, you need to enter slash on a new line after the trigger. Otherwise, the script won't execute.

CREATE OR REPLACE TRIGGER orders\_after\_delete

AFTER DELETE

ON orders

FOR EACH ROW

DECLARE

v\_username varchar2(10);

BEGIN

-- Find username of person performing the DELETE on the table

SELECT user INTO v\_username

FROM dual;

-- Insert record into audit table

INSERT INTO orders\_audit

( order\_id,

quantity,

cost\_per\_item,

total\_cost,

delete\_date,

deleted\_by)

VALUES

( :old.order\_id,

:old.quantity,

:old.cost\_per\_item,

:old.total\_cost,

sysdate,

v\_username );

END;

/

**ORACLE/PLSQL: AFTER UPDATE TRIGGER**

This Oracle tutorial explains how to **create an AFTER UPDATE Trigger** in Oracle with syntax and examples.

**DESCRIPTION**

An **AFTER UPDATE Trigger** means that Oracle will fire this trigger after the UPDATE operation is executed.

**SYNTAX**

The syntax to create an **AFTER UPDATE Trigger** in Oracle/PLSQL is:

CREATE [ OR REPLACE ] TRIGGER *trigger\_name*

AFTER UPDATE

ON *table\_name*

[ FOR EACH ROW ]

DECLARE

-- variable declarations

BEGIN

-- trigger code

EXCEPTION

WHEN ...

-- exception handling

END;

Parameters or Arguments

**OR REPLACE**

Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a [DROP TRIGGER statement](http://www.techonthenet.com/oracle/triggers/drop.php).

**trigger\_name**

The name of the trigger to create.

**AFTER UPDATE**

It indicates that the trigger will fire after the UPDATE operation is executed.

**table\_name**

The name of the table that the trigger is created on.

**RESTRICTIONS**

* You can not create an AFTER trigger on a view.
* You can not update the :NEW values.
* You can not update the :OLD values.

**NOTE**

* See also how to create [AFTER DELETE](http://www.techonthenet.com/oracle/triggers/after_delete.php), [AFTER INSERT](http://www.techonthenet.com/oracle/triggers/after_insert.php), [BEFORE DELETE](http://www.techonthenet.com/oracle/triggers/before_delete.php), [BEFORE INSERT](http://www.techonthenet.com/oracle/triggers/before_insert.php), and [BEFORE UPDATE](http://www.techonthenet.com/oracle/triggers/before_update.php)triggers.
* See also how to [drop a trigger](http://www.techonthenet.com/oracle/triggers/drop.php).

**EXAMPLE**

Let's look at an example of how to create an AFTER UPDATE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

CREATE TABLE orders

( order\_id number(5),

quantity number(4),

cost\_per\_item number(6,2),

total\_cost number(8,2)

);

We could then use the CREATE TRIGGER statement to create an AFTER UPDATE trigger as follows:

**TIP:** When using SQLPlus, you need to enter slash on a new line after the trigger. Otherwise, the script won't execute.

CREATE OR REPLACE TRIGGER orders\_after\_update

AFTER UPDATE

ON orders

FOR EACH ROW

DECLARE

v\_username varchar2(10);

BEGIN

-- Find username of person performing UPDATE into table

SELECT user INTO v\_username

FROM dual;

-- Insert record into audit table

INSERT INTO orders\_audit

( order\_id,

quantity\_before,

quantity\_after,

username )

VALUES

( :new.order\_id,

:old.quantity,

:new.quantity,

v\_username );

END;

/

**ORACLE/PLSQL: AFTER UPDATE TRIGGER**

This Oracle tutorial explains how to **create an AFTER UPDATE Trigger** in Oracle with syntax and examples.

**DESCRIPTION**

An **AFTER UPDATE Trigger** means that Oracle will fire this trigger after the UPDATE operation is executed.

**SYNTAX**

The syntax to create an **AFTER UPDATE Trigger** in Oracle/PLSQL is:

CREATE [ OR REPLACE ] TRIGGER *trigger\_name*

AFTER UPDATE

ON *table\_name*

[ FOR EACH ROW ]

DECLARE

-- variable declarations

BEGIN

-- trigger code

EXCEPTION

WHEN ...

-- exception handling

END;

Parameters or Arguments

**OR REPLACE**

Optional. If specified, it allows you to re-create the trigger is it already exists so that you can change the trigger definition without issuing a [DROP TRIGGER statement](http://www.techonthenet.com/oracle/triggers/drop.php).

**trigger\_name**

The name of the trigger to create.

**AFTER UPDATE**

It indicates that the trigger will fire after the UPDATE operation is executed.

**table\_name**

The name of the table that the trigger is created on.

**RESTRICTIONS**

* You can not create an AFTER trigger on a view.
* You can not update the :NEW values.
* You can not update the :OLD values.

**NOTE**

* See also how to create [AFTER DELETE](http://www.techonthenet.com/oracle/triggers/after_delete.php), [AFTER INSERT](http://www.techonthenet.com/oracle/triggers/after_insert.php), [BEFORE DELETE](http://www.techonthenet.com/oracle/triggers/before_delete.php), [BEFORE INSERT](http://www.techonthenet.com/oracle/triggers/before_insert.php), and [BEFORE UPDATE](http://www.techonthenet.com/oracle/triggers/before_update.php)triggers.
* See also how to [drop a trigger](http://www.techonthenet.com/oracle/triggers/drop.php).

**EXAMPLE**

Let's look at an example of how to create an AFTER UPDATE trigger using the CREATE TRIGGER statement.

If you had a table created as follows:

CREATE TABLE orders

( order\_id number(5),

quantity number(4),

cost\_per\_item number(6,2),

total\_cost number(8,2)

);

We could then use the CREATE TRIGGER statement to create an AFTER UPDATE trigger as follows:

**TIP:** When using SQLPlus, you need to enter slash on a new line after the trigger. Otherwise, the script won't execute.

CREATE OR REPLACE TRIGGER orders\_after\_update

AFTER UPDATE

ON orders

FOR EACH ROW

DECLARE

v\_username varchar2(10);

BEGIN

-- Find username of person performing UPDATE into table

SELECT user INTO v\_username

FROM dual;

-- Insert record into audit table

INSERT INTO orders\_audit

( order\_id,

quantity\_before,

quantity\_after,

username )

VALUES

( :new.order\_id,

:old.quantity,

:new.quantity,

v\_username );

END;

/

**ORACLE/PLSQL: DISABLE A TRIGGER**

This Oracle tutorial explains how to **disable a trigger** in Oracle with syntax and examples.

**DESCRIPTION**

Once you have created a Trigger in Oracle, you might find that you are required to disable the trigger. You can do this with the ALTER TRIGGER statement.

**SYNTAX**

The syntax for a disabling a Trigger in Oracle/PLSQL is:

ALTER TRIGGER *trigger\_name* DISABLE;

Parameters or Arguments

**trigger\_name**

The name of the trigger that you wish to disable.

**NOTE**

* See also how to [disable all triggers on a table](http://www.techonthenet.com/oracle/triggers/disable_all.php).
* See also how to [enable a trigger](http://www.techonthenet.com/oracle/triggers/enable.php) on a table or [enable all triggers on a table](http://www.techonthenet.com/oracle/triggers/enable_all.php).

**EXAMPLE**

Let's look at an example that shows how to disable a trigger in Oracle.

For example:

ALTER TRIGGER orders\_before\_insert DISABLE;

This example uses the ALTER TRIGGER statement to disable the trigger called *orders\_before\_insert*.

-

**ORACLE/PLSQL: DISABLE ALL TRIGGERS ON A TABLE**

This Oracle tutorial explains how to **disable all triggers on a table** in Oracle with syntax and examples.

**DESCRIPTION**

Once you have created Triggers in Oracle, you might find that you are required to disable all of the triggers on a table.. You can do this with the ALTER TRIGGER statement.

**SYNTAX**

The syntax for a disabling all Triggers on a table in Oracle/PLSQL is:

ALTER TABLE *table\_name* DISABLE ALL TRIGGERS;

Parameters or Arguments

**table\_name**

The name of the table that all triggers should be disabled on.

**NOTE**

* See also how to [disable a trigger](http://www.techonthenet.com/oracle/triggers/disable.php).
* See also how to [enable a trigger](http://www.techonthenet.com/oracle/triggers/enable.php) on a table or [enable all triggers on a table](http://www.techonthenet.com/oracle/triggers/enable_all.php).

**EXAMPLE**

Let's look at an example that shows how to disable all triggers on a table in Oracle.

For example:

ALTER TABLE orders DISABLE ALL TRIGGERS;

This example uses the ALTER TRIGGER statement to disable all triggers on the table called *orders*.

**ORACLE/PLSQL: ENABLE A TRIGGER**

This Oracle tutorial explains how to **enable a trigger** in Oracle with syntax and examples.

**DESCRIPTION**

You may have found that you have disabled a trigger on a table and you wish to enable the trigger again. You can do this with the ALTER TRIGGER statement.

**SYNTAX**

The syntax for a enabling a Trigger in Oracle/PLSQL is:

ALTER TRIGGER *trigger\_name* ENABLE;

Parameters or Arguments

**trigger\_name**

The name of the trigger that you wish to enable.

**NOTE**

* See also how to [enable all triggers on a table](http://www.techonthenet.com/oracle/triggers/enable_all.php).
* See also how to [disable a trigger](http://www.techonthenet.com/oracle/triggers/disable.php) on a table or [disable all triggers on a table](http://www.techonthenet.com/oracle/triggers/disable_all.php).

**EXAMPLE**

Let's look at an example that shows how to enable a trigger in Oracle.

For example:

ALTER TRIGGER orders\_before\_insert ENABLE;

This example uses the ALTER TRIGGER statement to enable the trigger called *orders\_before\_insert*.

**ORACLE/PLSQL: ENABLE ALL TRIGGERS ON A TABLE**

This Oracle tutorial explains how to **enable all triggers on a table** in Oracle with syntax and examples.

**DESCRIPTION**

You may have found that you have disabled all triggers on a table and you wish to enable the triggers again. You can do this with the ALTER TRIGGER statement.

**SYNTAX**

The syntax to **enable all triggers on a table** in Oracle/PLSQL is:

ALTER TABLE *table\_name* ENABLE ALL TRIGGERS;

Parameters or Arguments

**table\_name**

The name of the table that all triggers should be enabled on.

**NOTE**

* See also how to [enable a trigger](http://www.techonthenet.com/oracle/triggers/enable.php).
* See also how to [disable a trigger](http://www.techonthenet.com/oracle/triggers/disable.php) on a table or [disable all triggers on a table](http://www.techonthenet.com/oracle/triggers/disable_all.php).

**EXAMPLE**

Let's look at an example that shows how to enable all triggers on a table in Oracle.

For example:

ALTER TABLE orders ENABLE ALL TRIGGERS;

This example uses the ALTER TRIGGER statement to enable all triggers on the table called *orders*.

**ORACLE/PLSQL: CURSORS**

In Oracle, a **cursor** is a mechanism by which you can assign a name to a SELECT statement and manipulate the information within that SQL statement.

The following is a list of topics that explain how to use Cursors in Oracle/PLSQL:

**CREATE CURSOR**

* [Declare a Cursor](http://www.techonthenet.com/oracle/cursors/declare.php)
* [OPEN Statement](http://www.techonthenet.com/oracle/cursors/open.php)
* [FETCH Statement](http://www.techonthenet.com/oracle/cursors/fetch.php)
* [CLOSE Statement](http://www.techonthenet.com/oracle/cursors/close.php)
* [Cursor Attributes (%FOUND, %NOTFOUND, etc)](http://www.techonthenet.com/oracle/cursors/attributes.php)
* [SELECT FOR UPDATE Statement](http://www.techonthenet.com/oracle/cursors/for_update.php)
* [WHERE CURRENT OF Statement](http://www.techonthenet.com/oracle/cursors/current_of.php)

**EXAMPLES**

* [Procedure that outputs a dynamic PLSQL cursor](http://www.techonthenet.com/oracle/questions/cursor1.php)
* [Cursor within a cursor](http://www.techonthenet.com/oracle/questions/cursor2.php)
* [Cursor with variable in an "IN CLAUSE"](http://www.techonthenet.com/oracle/questions/cursor3.php)

-

# ORACLE/PLSQL: DECLARE A CURSOR

This Oracle tutorial explains how to **declare a cursor** in Oracle/PLSQL with syntax and examples.

## DESCRIPTION

A cursor is a SELECT statement that is defined within the declaration section of your PLSQL code. We'll take a look at three different syntaxes to declare a cursor.

## CURSOR WITHOUT PARAMETERS (SIMPLEST)

Declaring a cursor without any parameters is the simplest cursor. Let's take a closer look.

### Syntax

The syntax for a cursor without parameters in Oracle/PLSQL is:

CURSOR cursor\_name

IS

SELECT\_statement;

### Example

For example, you could define a cursor called c1 as below.

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

The result set of this cursor is all course\_numbers whose course\_name matches the variable called name\_in.

Below is a function that uses this cursor.

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

BEGIN

OPEN c1;

FETCH c1 INTO cnumber;

if c1%notfound then

cnumber := 9999;

end if;

CLOSE c1;

RETURN cnumber;

END;

## CURSOR WITH PARAMETERS

As we get more complicated, we can declare cursors with parameters.

### Syntax

The syntax for a **cursor with parameters** in Oracle/PLSQL is:

CURSOR cursor\_name (parameter\_list)

IS

SELECT\_statement;

### Example

For example, you could define a cursor called c2 as below.

CURSOR c2 (subject\_id\_in IN varchar2)

IS

SELECT course\_number

FROM courses\_tbl

WHERE subject\_id = subject\_id\_in;

The result set of this cursor is all course\_numbers whose subject\_id matches the subject\_id passed to the cursor via the parameter.

## CURSOR WITH RETURN CLAUSE

Finally, we can declare a cursor with a return clause.

### Syntax

The syntax for a cursor with a return clause in Oracle/PLSQL is:

CURSOR cursor\_name

RETURN field%ROWTYPE

IS

SELECT\_statement;

### Example

For example, you could define a cursor called c3 as below.

CURSOR c3

RETURN courses\_tbl%ROWTYPE

IS

SELECT \*

FROM courses\_tbl

WHERE subject = 'Mathematics';

The result set of this cursor is all columns from the course\_tbl where the subject is Mathematics.

# ORACLE/PLSQL: OPEN STATEMENT

This Oracle tutorial explains how to use the Oracle/PLSQL **OPEN statement** with syntax and examples.

## DESCRIPTION

Once you've [declared your cursor](http://www.techonthenet.com/oracle/cursors/declare.php), the next step is to use the OPEN statement to open the cursor.

## SYNTAX

The syntax to open a cursor using the OPEN statement in Oracle/PLSQL is:

OPEN cursor\_name;

### Parameters or Arguments

**cursor\_name**

The name of the cursor that you wish to open.

## EXAMPLE

For example, you could open a cursor called c1 with the following command:

OPEN c1;

Below is a function that demonstrates how to use the OPEN statement:

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

BEGIN

OPEN c1;

FETCH c1 INTO cnumber;

if c1%notfound then

cnumber := 9999;

end if;

CLOSE c1;

RETURN cnumber;

END;

# ORACLE/PLSQL: FETCH STATEMENT

This Oracle tutorial explains how to use the Oracle/PLSQL **FETCH statement** with syntax and examples.

## DESCRIPTION

The purpose of using a cursor, in most cases, is to retrieve the rows from your cursor so that some type of operation can be performed on the data. After [declaring](http://www.techonthenet.com/oracle/cursors/declare.php) and [opening](http://www.techonthenet.com/oracle/cursors/open.php) your cursor, the next step is to use the FETCH statement to fetch rows from your cursor.

## SYNTAX

The syntax for the FETCH statement in Oracle/PLSQL is:

FETCH cursor\_name INTO variable\_list;

### Parameters or Arguments

**cursor\_name**

The name of the cursor that you wish to fetch rows.

**variable\_list**

The list of variables, comma delimited, that you wish to store the cursor result set in.

## EXAMPLE

For example, you could have a cursor defined as:

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

The command that would be used to fetch the data from this cursor is:

FETCH c1 into cnumber;

This would fetch the first course\_number into the variable called cnumber.

Below is a function that demonstrates how to use the FETCH statement.

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

BEGIN

OPEN c1;

FETCH c1 INTO cnumber;

if c1%notfound then

cnumber := 9999;

end if;

CLOSE c1;

RETURN cnumber;

END;

# ORACLE/PLSQL: CLOSE STATEMENT

This Oracle tutorial explains how to use the Oracle/PLSQL **CLOSE statement** with syntax and examples.

## DESCRIPTION

The final step of working with cursors is to close the cursor once you have finished using it.

## SYNTAX

The syntax to close a cursor in Oracle/PLSQL using the CLOSE statement is:

CLOSE cursor\_name;

### Parameters or Arguments

**cursor\_name**

The name of the cursor that you wish to close.

## EXAMPLE

For example, you could close a cursor called c1 with the following command:

CLOSE c1;

Below is a function that demonstrates how to use the CLOSE statement:

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

BEGIN

OPEN c1;

FETCH c1 INTO cnumber;

if c1%notfound then

cnumber := 9999;

end if;

CLOSE c1;

RETURN cnumber;

END;

# ORACLE/PLSQL: CURSOR ATTRIBUTES

While dealing with cursors, you may need to determine the status of your cursor. The following is a list of the cursor attributes that you can use.

|  |  |
| --- | --- |
| **Attribute** | **Explanation** |
| %ISOPEN | - Returns TRUE if the cursor is open, FALSE if the cursor is closed. |
| %FOUND | - Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.  - Returns NULL if cursor is open, but fetch has not been executed.  - Returns TRUE if a successful fetch has been executed.  - Returns FALSE if no row was returned. |
| %NOTFOUND | - Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.  - Return NULL if cursor is open, but fetch has not been executed.  - Returns FALSE if a successful fetch has been executed.  - Returns TRUE if no row was returned. |
| %ROWCOUNT | - Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.  - Returns the number of rows fetched.  - The ROWCOUNT attribute doesn't give the real row count until you have iterated through the entire cursor. In other words, you shouldn't rely on this attribute to tell you how many rows are in a cursor after it is opened. |

Below is an example of how you might use the %NOTFOUND attribute.

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

BEGIN

open c1;

fetch c1 into cnumber;

if c1%notfound then

cnumber := 9999;

end if;

close c1;

RETURN cnumber;

END;

# ORACLE/PLSQL: CURSOR ATTRIBUTES

While dealing with cursors, you may need to determine the status of your cursor. The following is a list of the cursor attributes that you can use.

|  |  |
| --- | --- |
| **Attribute** | **Explanation** |
| %ISOPEN | - Returns TRUE if the cursor is open, FALSE if the cursor is closed. |
| %FOUND | - Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.  - Returns NULL if cursor is open, but fetch has not been executed.  - Returns TRUE if a successful fetch has been executed.  - Returns FALSE if no row was returned. |
| %NOTFOUND | - Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.  - Return NULL if cursor is open, but fetch has not been executed.  - Returns FALSE if a successful fetch has been executed.  - Returns TRUE if no row was returned. |
| %ROWCOUNT | - Returns INVALID\_CURSOR if cursor is declared, but not open; or if cursor has been closed.  - Returns the number of rows fetched.  - The ROWCOUNT attribute doesn't give the real row count until you have iterated through the entire cursor. In other words, you shouldn't rely on this attribute to tell you how many rows are in a cursor after it is opened. |

Below is an example of how you might use the %NOTFOUND attribute.

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in;

BEGIN

open c1;

fetch c1 into cnumber;

if c1%notfound then

cnumber := 9999;

end if;

close c1;

RETURN cnumber;

END;

# ORACLE/PLSQL: SELECT FOR UPDATE STATEMENT

This Oracle tutorial explains how to use the Oracle/PLSQL **SELECT FOR UPDATE statement** with syntax and examples.

## DESCRIPTION

The SELECT FOR UPDATE statement allows you to lock the records in the cursor result set. You are not required to make changes to the records in order to use this statement. The record locks are released when the next commit or rollback statement is issued.

## SYNTAX

The syntax for the SELECT FOR UPDATE statement in Oracle/PLSQL is:

CURSOR cursor\_name

IS

select\_statement

FOR UPDATE [OF column\_list] [NOWAIT];

### Parameters or Arguments

**cursor\_name**

The name of the cursor.

**select\_statement**

A SELECT statement that will populate your cursor result set.

**column\_list**

The columns in the cursor result set that you wish to update.

**NOWAIT**

Optional. The cursor does not wait for resources.

## EXAMPLE

For example, you could use the SELECT FOR UPDATE statement as follows:

CURSOR c1

IS

SELECT course\_number, instructor

FROM courses\_tbl

FOR UPDATE OF instructor;

If you plan on updating or deleting records that have been referenced by a SELECT FOR UPDATE statement, you can use the[WHERE CURRENT OF](http://www.techonthenet.com/oracle/cursors/current_of.php) statement.

**ORACLE/PLSQL: WHERE CURRENT OF STATEMENT**

This Oracle tutorial explains how to use the Oracle/PLSQL **WHERE CURRENT OF statement** with syntax and examples.

**DESCRIPTION**

If you plan on updating or deleting records that have been referenced by a [SELECT FOR UPDATE](http://www.techonthenet.com/oracle/cursors/for_update.php) statement, you can use the WHERE CURRENT OF statement.

**SYNTAX**

The syntax for the WHERE CURRENT OF statement in Oracle/PLSQL is either:

UPDATE table\_name

SET set\_clause

WHERE CURRENT OF cursor\_name;

OR

DELETE FROM table\_name

WHERE CURRENT OF cursor\_name;

**NOTE**

* The WHERE CURRENT OF statement allows you to update or delete the record that was last fetched by the cursor.

**EXAMPLE**

Updating using the WHERE CURRENT OF Statement

Here is an example where we are updating records using the *WHERE CURRENT OF Statement*:

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

FROM courses\_tbl

WHERE course\_name = name\_in

FOR UPDATE of instructor;

BEGIN

OPEN c1;

FETCH c1 INTO cnumber;

if c1%notfound then

cnumber := 9999;

else

UPDATE courses\_tbl

SET instructor = 'SMITH'

WHERE CURRENT OF c1;

COMMIT;

end if;

CLOSE c1;

RETURN cnumber;

END;

Deleting using the WHERE CURRENT OF Statement

Here is an example where we are deleting records using the *WHERE CURRENT OF Statement*:

CREATE OR REPLACE Function FindCourse

( name\_in IN varchar2 )

RETURN number

IS

cnumber number;

CURSOR c1

IS

SELECT course\_number

from courses\_tbl

where course\_name = name\_in

FOR UPDATE of instructor;

BEGIN

open c1;

fetch c1 into cnumber;

if c1%notfound then

cnumber := 9999;

else

DELETE FROM courses\_tbl

WHERE CURRENT OF c1;

COMMIT;

end if;

close c1;

RETURN cnumber;

END;