

SQL Programming

DML

We've spent the better part of the semester examining how we can retrieve information from tables in a database.

And in just this single lesson, we'll see how we can add, remove and change data in those tables.

Back in module 1 we discussed the genesis of database systems as having developed out of an opportunity to automate common file processing functions.

In this regard, we emphasized that database management systems were a tool primarily for the IT staff, to help them increase their productivity, by helping to automate the CRUD functions.

CRUD is an acronym that serves as a useful mnemonic for all of the essential file processing functions.

C: Create

create records, ie.
add records/rows to the table

R: Retrieve

access records, ie.
query / report on data in the DB

U: Update

to modify or change values in
the fields/columns of a record

D: Delete

remove or delete records from
the table

Up until now, we've been focused on the data query language (DQL) aspects of SQL.

All of the programs we've written have only retrieved or queried (SELECTed) the table data.

In this module we're going to see how SQL supports the rest of the CRUD matrix as we explore these three data manipulation language (DML) statements:

DELETE

UPDATE

INSERT

DELETE is used to remove records/rows from a table.

The syntax for the DELETE statement is:

```
DELETE
FROM   table-name
[WHERE predicate expression]
```

This structure should remind you of the SELECT statement. DELETE has the same basic phrases, and it functions in the same general manner.

The FROM clause identifies the table.

The WHERE clause identifies the rows.

But why aren't there any column specifications included in the DELETE clause?

When we DELETE records from the database we're interested in removing the entire row from the table.

Since 'everything goes', we don't need to provide a column list.

If it's our intention to delete, or zero out some columns in the row, then we should use the UPDATE command, but more on that, later.

Our talent agency is in the midst of an office re-organization in an attempt to better serve our clients. Foreign-born talent will be handled by a new division.

Complete copies of the database files have been shared and installed in each of the two divisions, and you've been assigned to support the foreign-born division.

Your first task is to eliminate all records that don't belong to foreign-born clients.

Module 16: DML

Page B-7 Problem 17-1 Design

You'll be using the DELETE statement to remove the unneeded records from the table.

```
DELETE
FROM   talent
WHERE  home_country = 'USA'
```

iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help Links »

ORACLE iSQL*Plus Password Log Out Help

Script Location: Browse... Load Script

Enter statements:

```
SELECT COUNT(*)
FROM   talent;

DELETE
FROM   talent
WHERE  home_country = 'USA';

SELECT COUNT(*)
FROM   talent;
```

Execute Output: Work Screen Clear Screen Save Script

COUNT(*)
25

20 rows deleted.

COUNT(*)
5

iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help Links »

ORACLE iSQL*Plus

[Password](#) [Log Out](#) [Help](#)

Script Location:

Enter statements:

```
SELECT COUNT(*)
FROM talent;

DELETE
FROM talent
WHERE home_country = 'USA';

SELECT COUNT(*)
FROM talent;
```

Output:

COUNT(*)
25

20 rows deleted.

COUNT(*)
5

One of the problems with the DELETE command is that it 'works in the dark'. The only feedback the programmer receives from this statement is a simple display of the number of rows that were affected by the statement.

When I'm developing code, I like to get a better feel for my programs. I'd like to know something about the before-image of the database and the after-image of the database. (ie. Before in the sense of before the delete command is processed, ...)

In the solution on the slide to the left, I've included two simple SQL programs that 'bracket' my DELETE program.

These programs simply display the number of rows in the table, before the delete operation, and then they display the number of rows in the table after the delete operation. I like numbers, so this gives me an idea of the scope of my command.

Simply telling me that I've deleted 20 rows is one bit of information, knowing that I've eliminated 80% (20/25) of the records in my table is a whole 'nother way of looking at that same operation.

Notice that each of the programs is terminated with a semi-colon.

The Oracle *i*SQL*Plus environment only expects to see a single SQL program inside that box.

If you want to issue multiple commands/programs all at once, then each of those programs needs to be properly 'punctuated', and each needs to be terminated with a semi-colon ';'

iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help Links »

ORACLE iSQL*Plus Password Log Out Help

Script Location: Browse... Load Script

Enter statements:

```
SELECT COUNT(*)
FROM talent;

DELETE
FROM talent
WHERE home_country = 'USA';

SELECT COUNT(*)
FROM talent;
```

Execute Output: Work Screen Clear Screen Save Script

COUNT(*)
25

20 rows deleted.

COUNT(*)
5

The screenshot shows the iSQL*Plus interface with three SQL queries entered in the script area:

```

SELECT COUNT(*)
FROM talent;

SELECT DISTINCT home_country
FROM talent;

SELECT home_country, COUNT(*)
FROM talent
GROUP BY home_country;

```

Below the script area are buttons for 'Execute', 'Output: Work Screen', 'Clear Screen', and 'Save Script'. The 'Execute' button has been clicked, and the results are displayed in three tables.

Table 1: COUNT(*)

COUNT(*)
25

Table 2: HOME_COUNTRY

HOME_COUNTRY
Austria
Germany
Ireland
UK
USA

Table 3: HOME_COUNTRY and COUNT(*)

HOME_COUNTRY	COUNT(*)
Austria	1
Germany	1
Ireland	1
UK	2
USA	20

Let's return to the original problem ... eliminate all records that don't belong to foreign-born clients.

Is there anything that you'd like to know about the data, as the programmer, before you write your code?

I'd like to have as good a picture of the data as I can. Counts are good, but a more detailed breakdown on the relatively limited number of home_countries would serve me well.

This helps.

What do you think?

iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help Links »

Enter statements:

```

SELECT    home_country, COUNT(*)
FROM      talent
GROUP BY  home_country;

DELETE
FROM      talent
WHERE     home_country = 'USA';

SELECT    home_country, COUNT(*)
FROM      talent
GROUP BY  home_country;

```

Execute Output: Work Screen ▼ Clear Screen Save Script

HOME_COUNTRY	COUNT(*)
Austria	1
Germany	1
Ireland	1
UK	2
USA	20

20 rows deleted.

HOME_COUNTRY	COUNT(*)
Austria	1
Germany	1
Ireland	1
UK	2

So that when it comes time to actually 'do the deed', I might write a series of programs, such as these, to accomplish the task, and to help me verify that everything ran properly.

What does this program do?

```
DELETE  
FROM   talent;
```

Go ahead and try it.

Module 16: DML

Page B-13 DELETE w/o WHERE

You didn't really type that program in, did you?

You saw the smiley face, right? You knew I was kidding, didn't you? Good!

Because as you can see from my demonstration slide, the DELETE statement without a WHERE clause eliminates every single row from the table.

Whew! Good thing you didn't type that in, or you'd have nothing to work with for the rest of the semester.

iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help Links »

ORACLE iSQL*Plus Password Log Out Help

Script Location: Browse... Load Script

Enter statements:

```
SELECT COUNT(*)
FROM talent;

DELETE
FROM talent;

SELECT COUNT(*)
FROM talent;
```

Execute Output: Work Screen Clear Screen Save Script

COUNT(*)
5

5 rows deleted.

COUNT(*)
0

Now that you're at the stage where you'll be effecting changes in the database, it's probably a good idea to tell you about SQL's undo command.

I'll save the long story for semester 2, but here's the short story.

ROLLBACK is the SQL command that you can issue to *roll back*, or undo, everything you've done during your session.

ROLLBACK is related to the COMMIT command.

COMMIT is the command you issue when you want to *commit* the changes to the database, you want to make them permanent.

Any changes that have been COMMITed cannot be ROLLBACKed.

The ROLLBACK command has some other limitations, but as long as the work you're doing is limited to the data manipulation commands (INSERT, UPDATE, and DELETE) you should be safe.

My intuitions about how SQL processes the DELETE command are simply this:

The DELETE action is accomplished by a two-step process, and I think of it as:

delete THIS!

OR

```
DELETE these rows
      (SELECT
        FROM table_name
        [WHERE condition])
```

The first step is an implicitly nested subquery that identifies the rows that are to be deleted.

The second step then deletes the matching rows from the base table.

The syntax for the UPDATE command is

```
UPDATE table-name  
SET column-name = value  
[WHERE predicate-expression]
```

And this introduces a new clause: SET

The SET clause will **always** be in the form of:

```
SET some column  
TO BE EQUAL TO some value
```

Now in computer-ese, we generally don't think of the equal sign as meaning 'equals'. Instead, we think of it as the assignment operator.

In this regard, we take whatever is on the right-hand side of the equal sign, and *assign* it to the variable that's on the left-hand side.

In SQL, that variable on the left-hand side of the equal sign is always going to be a column name.

And the value on the right-hand side of the equal sign can be a literal, or an expression.

And everything ought to work out just fine, so long as the value on the right-hand side is drawn from the same domain as the column that is specified on the left-hand side.

SET

```
    leng = 40,  
    name = 'Bil',  
    birthdate =  
        TO_DATE('23-Jan-1980', 'dd-mon-yyyy'),  
    ...
```

The talent agency just renegotiated everybody's contract. All of our clients will be charged the same percentage rate for representation.

You've been assigned to update the database so that it reflects this new policy of charging everyone a flat 6% percentage rate.

This task is pretty straightforward.

The change applies across the board, so you won't need to code a WHERE clause.

```
UPDATE talent  
    SET perc = 6;
```

Let's consider another problem.

The talent agency has renegotiated contracts with all of their clients. Everyone will be charged a flat percentage rate of 7%.

Clients who do television work will be charged 5% (not just for their TV work, but for all of their work).

Clients who work only in Theater, will be charged 10%.

Let's break the problem down into pieces:

Everyone will be charged a flat percentage rate of 7%.

```
UPDATE talent
    SET perc = 7;
```

Clients who do television work will be charged 5%.

```
UPDATE talent
    SET perc = 5
    WHERE UPPER(tv) = 'YES';
```

Clients who only work in Theater, will be charged 10%.

```
UPDATE talent
    SET perc = 10
    WHERE UPPER(theatre) = 'YES'
        AND (UPPER(film) = 'NO' AND UPPER(tv) = 'NO');
```

So after all of that, we're left with these three statements:

```
UPDATE talent  
  SET perc = 7;
```

```
UPDATE talent  
  SET perc = 5  
WHERE UPPER(tv) = 'YES';
```

```
UPDATE talent  
  SET perc = 10  
WHERE UPPER(theatre) = 'YES'  
  AND (UPPER(film) = 'NO' AND UPPER(tv) = 'NO');
```

What do you think of this solution?

Focus on the last UPDATE statement, and consider what happens when NULL values are encountered?

Is this correct? Is this particular statement doing what it's supposed to do?

In all honesty it's hard to say.

But as the programmer, you're expected to notice any inconsistencies in the specifications that you've been given, and it's your job to bring these inconsistencies to the attention of the analyst.

If the specs are vague, or open to interpretation, don't take it upon yourself to make any assumptions. Route an inquiry back to the users (thru the systems analyst) for a clarification on the company policy.

As this solution is coded now, if we don't know whether or not the talent has worked in film or TV, we give them the benefit of the doubt.

We've seen that we can change data and remove data from the database. The final feature of CRUD that we need to examine is how we go about adding data/rows to a table with the INSERT statement.

The syntax for the INSERT command is:

```
INSERT
  INTO    table_name [column-list]
          VALUES(value-list)
```

And you should notice that the column-list is an optional component.

Module 16: DML

Page F-2 INSERT w/ Column List

One way of expressing an insert operation is to completely list all of the elements (columns) in the table, for which you are providing values.

INSERT

```
INSERT INTO talent(id, last_name, first_name, birthdate,
                    gender, home_town, home_state, home_country,
                    perc, theatre, film, tv)
VALUES(1908, 'Williams', 'Robin', TO_DATE('21-july-1952', 'DD-MON-YYYY'),
       'M', 'Chicago', 'Illinois', 'USA',
       7, 'No', 'Yes', 'Yes');
```

With this technique, the values are arranged in the same order as the column names are listed.

The screenshot shows the iSQL*Plus web interface in a Microsoft Internet Explorer browser. The page title is "iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer". The interface includes a menu bar (File, Edit, View, Favorites, Tools, Help) and a toolbar with icons for Password, Log Out, and Help. Below the toolbar, there is a "Script Location:" field with a "Browse..." button and a "Load Script" button. The main area is labeled "Enter statements:" and contains a text box with the following SQL script:

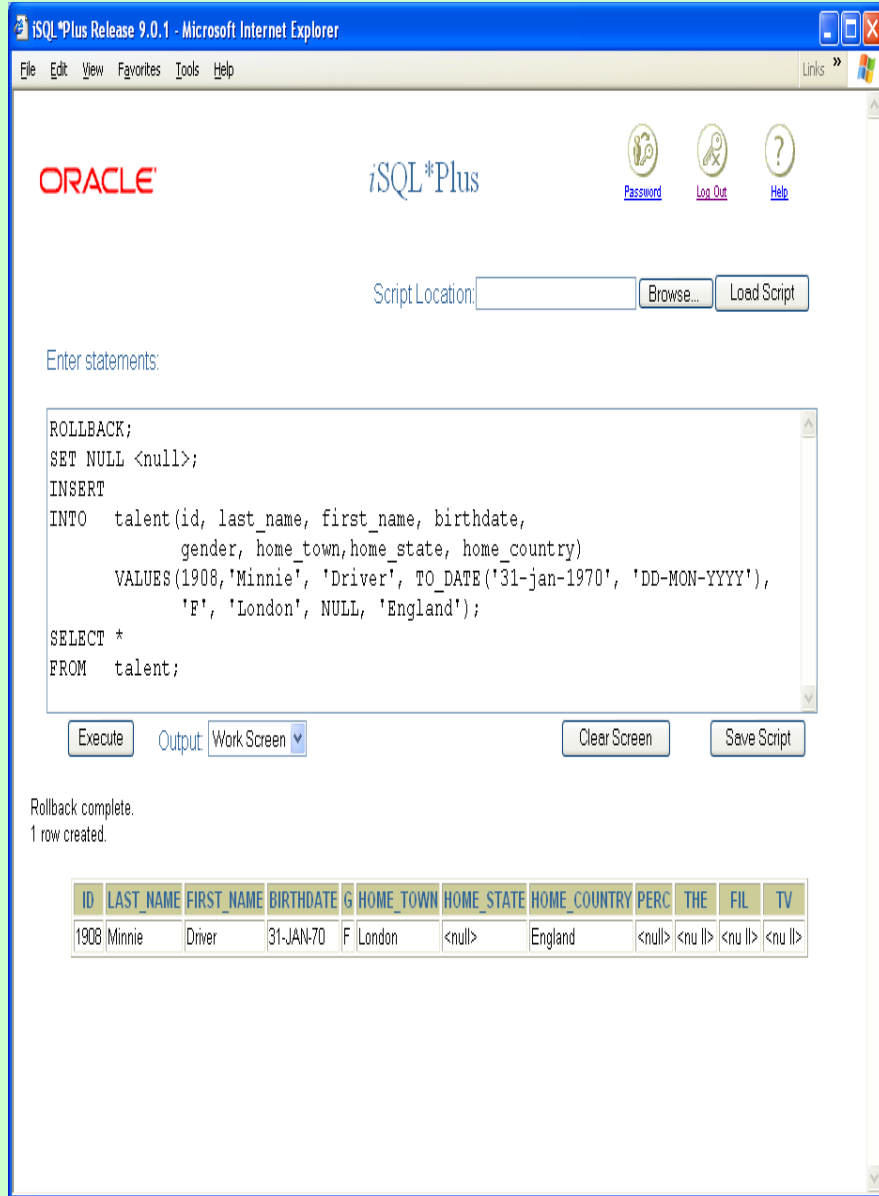
```
ROLLBACK;
INSERT
INTO talent(id, last_name, first_name, birthdate,
            gender, home_town, home_state, home_country,
            perc, theatre, film, tv)
VALUES(1908, 'Williams', 'Robin', TO_DATE('21-july-1952', 'DD-MON-YYYY'),
       'M', 'Chicago', 'Illinois', 'USA',
       7, 'No', 'Yes', 'Yes');

SELECT *
FROM talent;
```

Below the text box are buttons for "Execute", "Output", "Work Screen" (with a dropdown arrow), "Clear Screen", and "Save Script". The "Execute" button is highlighted. Below the buttons, the status message reads: "Rollback complete. 1 row created." At the bottom, there is a table showing the result of the query:

ID	LAST_NAME	FIRST_NAME	BIRTHDATE	G	HOME_TOWN	HOME_STATE	HOME_COUNTRY	PERC	THE	FIL	TV
1908	Williams	Robin	21-JUL-52	M	Chicago	Illinois	USA	7	No	Yes	Yes

Module 16: DML



iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

ORACLE iSQL*Plus

Script Location: Browse... Load Script

Enter statements:

```
ROLLBACK;
SET NULL <null>;
INSERT
INTO talent(id, last_name, first_name, birthdate,
            gender, home_town, home_state, home_country)
VALUES(1908, 'Minnie', 'Driver', TO_DATE('31-jan-1970', 'DD-MON-YYYY'),
       'F', 'London', NULL, 'England');
SELECT *
FROM talent;
```

Execute Output Work Screen Clear Screen Save Script

Rollback complete.
1 row created.

ID	LAST_NAME	FIRST_NAME	BIRTHDATE	G	HOME_TOWN	HOME_STATE	HOME_COUNTRY	PERC	THE	FIL	TV
1908	Minnie	Driver	31-JAN-70	F	London	<null>	England	<null>	<nu ll>	<nu ll>	<nu ll>

Page F-3 INSERT w/ Column List

Another way to use the INSERT command is with a partial set of values for the row.

INSERT

```
INTO talent(id, last_name, first_name, birthdate,
            gender, home_town, home_state, home_country)
VALUES(1908, 'Minnie', 'Driver', TO_DATE('31-jan-1970',
'DD-MON-YYYY'),
       'F', 'London', NULL, 'England');
```

Notice how null values are both explicitly and implicitly assigned in this example.

The NULL value can be explicitly assigned by simply being 'named' in the value list.

The NULL value is implicitly assigned to every column not specified in the column list.

Module 16: DML

Page F-4 INSERT w/ Column List

Since the column_list is optional, the column names don't need to be specified.

But how will SQL be able to figure out which value should be assigned to which column?

SQL (at least SQL in the Oracle world, presumes that the values are presented in the same order as the column names are listed when the table is DESCRIBED.

For the discussion forum: What can o wrong, if you don't list the column names??

iSQL*Plus Release 9.0.1 - Microsoft Internet Explorer

File Edit View Favorites Tools Help Links »

ORACLE iSQL*Plus

Script Location:

Enter statements:

```
ROLLBACK;
INSERT
INTO talent
VALUES (1908, 'Williams', 'Robin', TO_DATE('21-july-1952', 'DD-MON-YYYY'),
        'M', 'Chicago', 'Illinois', 'USA',
        7, 'No', 'Yes', 'Yes');
SELECT *
FROM talent;
```

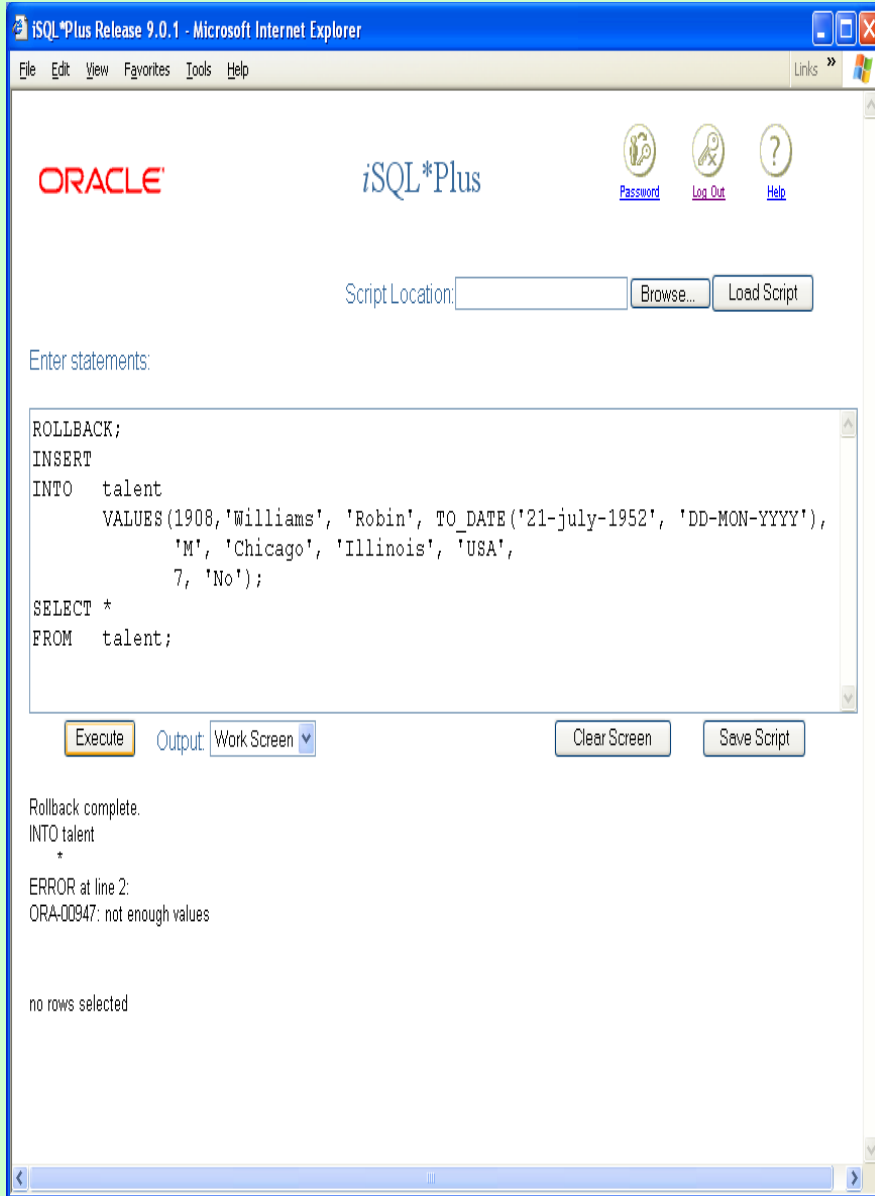
Rollback complete.
1 row created.

ID	LAST_NAME	FIRST_NAME	BIRTHDATE	G	HOME_TOWN	HOME_STATE	HOME_COUNTRY	PERC	THE	FIL	TV
1908	Williams	Robin	21-JUL-52	M	Chicago	Illinois	USA	7	No	Yes	Yes

Module 16: DML

Page F-5 INSERT w/ Column List

And if there aren't enough values to fill the row, SQL generates this informative error message.



I've provided you with some examples in this module that were designed to illustrate the techniques for DELETEing, UPDATEing and INSERTing records into a table.

But I need to tell you yet again, that some of these examples were merely teaching illustrations and are unlike anything you should find in the 'real world'.

Especially that first example where I suggested that the talent agency would build and maintain two separate databases, one database for natives, and one for foreign-born talent. That's a really, really, REALLY BAD example of the way things work in the real world.

Most of the value that derives from database development comes from the opportunity to integrate data and record-keeping operations from across the enterprise. There's a synergy that comes from having all of your data in one place. The trend is to bring the corporate information assets together, not to break them up.

DQL, data query language

DML, data manipulation language

CRUD

SELECT

DELETE

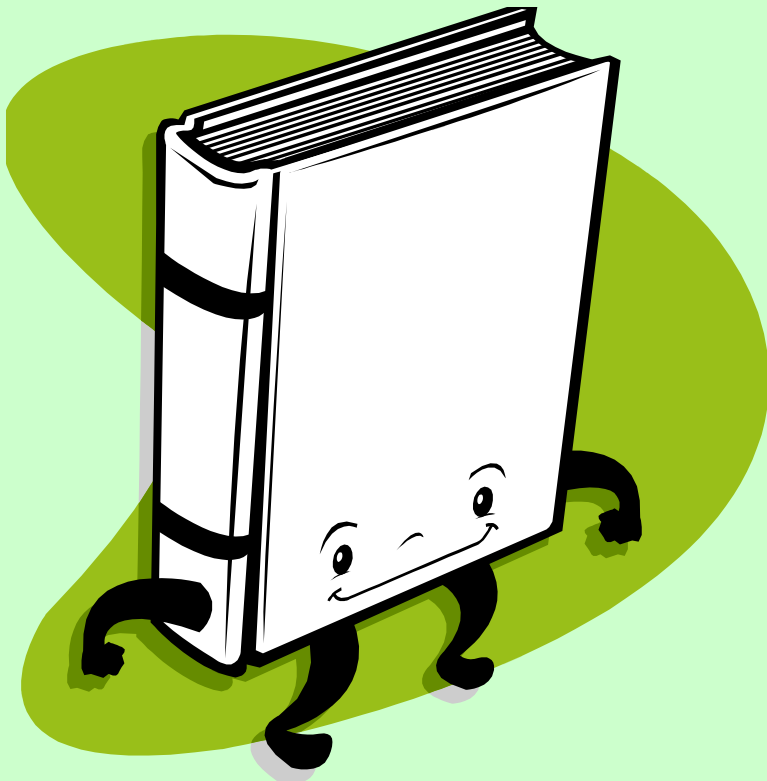
INSERT

UPDATE

Before-image, after-image

Transaction

ROLLBACK, COMMIT



Please drop me an email if you noticed any errors in this module. I'd also appreciate reading your comments, criticisms, and or suggestions as to how this module could be improved.

Thanks,

bil



That's All