**Triggers**

A trigger is a procedure that is run (or fired ) automatically by the database when a specified DML statement (INSERT, UPDATE, or DELETE) is run against a certain database table. Triggers are useful for doing things like advanced auditing of changes made to column values in a table

**When Trigger Fires?**

A trigger may fire before or after a DML statement runs. Also, because a DML statement can affect more than one row, the code for the trigger may be run once for every row affected (a row-level trigger), or just once for all the rows (a statement-level trigger). For example, if you create a rowlevel trigger that fires for an UPDATE on a table, and you run an UPDATE statement that modified ten rows of that table, then that trigger would run ten times. If, however, your trigger was a statement-level trigger, the trigger would fire once for the whole UPDATE statement, regardless of the number of rows affected.

There is another difference between a row-level trigger and a statement-level trigger: A rowlevel trigger has access to the old and new column values when the trigger fires as a result of an UPDATE statement on that column. The firing of a row-level trigger may also be limited using a trigger condition; for example, you could set a condition that limits the trigger to fire only when a column value is less than a specified value.

**Set Up For the Example Trigger ?**

As mentioned, triggers are useful for doing advanced auditing of changes made to column values. In the next section, you’ll see a trigger that records when a product’s price is lowered by more than 25 percent; when this occurs, the trigger will add a row to the product\_price\_audit table. The product\_price\_audit table is created by the following statement in the store\_ schema.sql script:

CREATE TABLE product\_price\_audit ( product\_id INTEGER CONSTRAINT price\_audit\_fk\_products REFERENCES products(product\_id), old\_price NUMBER(5, 2), new\_price NUMBER(5, 2) );

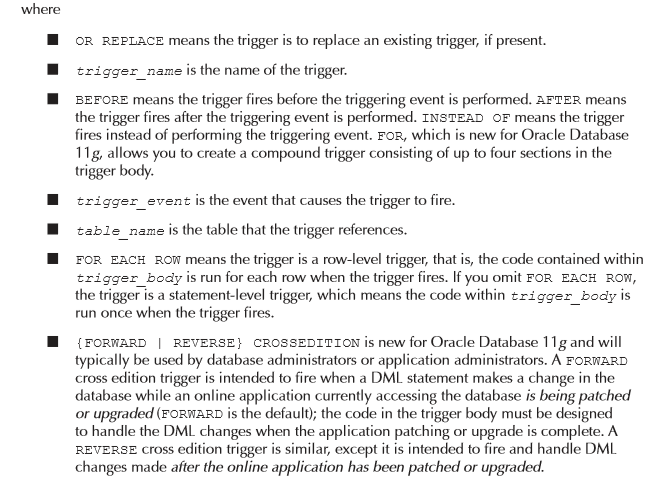
As you can see, the product\_id column of the product\_price\_audit table is a foreign key to the product\_id column of the products table. The old\_price column will be used to store the old price of a product prior to the change, and the new\_price column will be used to store the new price after the change.

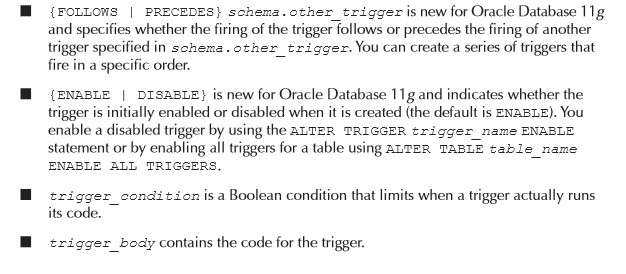
**Creating a Trigger**

You create a trigger using the CREATE TRIGGER statement. The simplified syntax for the CREATE TRIGGER statement is as follows:

CREATE [OR REPLACE] TRIGGER trigger\_name {BEFORE | AFTER | INSTEAD OF | FOR} trigger\_event ON table\_name [FOR EACH ROW] [{FORWARD | REVERSE} CROSSEDITION]

[{FOLLOWS | PRECEDES} schema.other\_trigger} [{ENABLE | DISABLE}] [WHEN trigger\_condition]] BEGIN trigger\_body END trigger\_name;





CREATE TRIGGER before\_product\_price\_update BEFORE UPDATE OF price ON products FOR EACH ROW WHEN (new.price < old.price \* 0.75) BEGIN dbms\_output.put\_line('product\_id = ' || :old.product\_id); dbms\_output.put\_line('Old price = ' || :old.price); dbms\_output.put\_line('New price = ' || :new.price); dbms\_output.put\_line('The price reduction is more than 25%'); -- insert row into the product\_price\_audit table INSERT INTO product\_price\_audit ( product\_id, old\_price, new\_price ) VALUES ( :old.product\_id, :old.price, :new.price ); END before\_product\_price\_update;

There are five things you should notice about this statement: BEFORE UPDATE OF price means the trigger fires before an update of the price column. FOR EACH ROW means this as a row-level trigger, that is, the trigger code contained within the BEGIN and END keywords runs once for each row modified by the update. The trigger condition is (new.price < old.price \* 0.75), which means the trigger fires only when the new price is less than 75 percent of the old price (that is, when the price is reduced by more than 25 percent). The new and old column values are accessed using the :old and :new aliases in the trigger. The trigger code displays the product\_id, the old and new prices, and a message stating that the price reduction is more than 25 percent. The code then adds a row to the product\_price\_audit table containing the product\_id and the old and new prices.

