http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@ebt-link;pt=47865?target=%25N%15_17866_START_RESTART_N%25

How triggers work

A trigger is a stored procedure that goes into effect when you insert, delete, or update data in a table. You can use triggers to perform a number of automatic actions, such as cascading changes through related tables, enforcing column restrictions, comparing the results of data modifications, and maintaining the referential integrity of data across a database.

Triggers are *automatic*. They work no matter what caused the data modification--a clerk's data entry or an application action. A trigger is specific to one or more of the data modification operations, **update**, **insert**, and **delete** and is executed once for each SQL statement.

For example, to prevent users from removing any publishing companies from the *publishers* table, you could use this trigger:

```
create trigger del_pub
on publishers
for delete
as
begin
    rollback transaction
    print "You cannot delete any publishers!"
end
```

The next time someone tries to remove a row from the *publishers* table, the *del_pub* trigger cancels the deletion, rolls back the transaction, and prints a message to that effect.

A trigger "fires" only after the data modification statement has completed and Adaptive Server has checked for any datatype, rule, or integrity constraint violation. The trigger and the statement that fires it are treated as a single transaction that can be rolled back from within the trigger. If Adaptive Server detects a severe error, the entire transaction is rolled back.

Triggers are most useful in these situations:

- Triggers can cascade changes through related tables in the database. For example, a delete trigger on the *title_id* column of the *titles* table can delete matching rows in other tables, using the *title_id* column as a unique key to locating rows in *titleauthor* and *roysched*.
- Triggers can disallow, or roll back, changes that would violate referential integrity, canceling the attempted data modification transaction. Such a trigger might go into effect when you try to insert a foreign key that does not match its primary key. For example, you could create an insert trigger on *titleauthor* that rolled back an insert if the new *titleauthor.title_id* value did not have a matching value in *titles.title_id*.

- Triggers can enforce restrictions that are much more complex than those that are defined with rules. Unlike rules, triggers can reference columns or database objects. For example, a trigger can roll back updates that attempt to increase a book's price by more than 1 percent of the advance.
- Triggers can perform simple "what if" analyses. For example, a trigger can compare the state of a table before and after a data modification and take action based on that comparison.

Using triggers vs. integrity constraints

As an alternative to using triggers, you can use the referential integrity constraint of the **create table** statement to enforce referential integrity across tables in the database. However, referential integrity constraints *cannot*:

- Cascade changes through related tables in the database
- Enforce complex restrictions by referencing other columns or database objects
- Perform "what if" analyses

Also, referential integrity constraints do not roll back the current transaction as a result of enforcing data integrity. With triggers, you can either roll back or continue the transaction, depending on how you handle referential integrity.

If your application requires one of the above tasks, use a trigger. Otherwise, use a referential integrity constraint to enforce data integrity. Adaptive Server checks referential integrity constraints before it checks triggers so that a data modification statement that violates the constraint does not also fire the trigger. For more information about referential integrity constraints, see Chapter 7, "Creating Databases and Tables."