

Chapter 33: Cascading Delete

Section 33.1: ON DELETE CASCADE

Assume you have a application that administers rooms.

Assume further that your application operates on a per client basis (tenant).

You have several clients.

So your database will contain one table for clients, and one for rooms.

Now, every client has N rooms.

This should mean that you have a foreign key on your room table, referencing the client table.

```
ALTER TABLE dbo.T_Room WITH CHECK ADD CONSTRAINT FK_T_Room_T_Client FOREIGN KEY(RM_CLI_ID)
REFERENCES dbo.T_Client (CLI_ID)
GO
```

Assuming a client moves on to some other software, you'll have to delete his data in your software. But if you do

```
DELETE FROM T_Client WHERE CLI_ID = x
```

Then you'll get a foreign key violation, because you can't delete the client when he still has rooms.

Now you'd have write code in your application that deletes the client's rooms before it deletes the client. Assume further that in the future, many more foreign key dependencies will be added in your database, because your application's functionality expands. Horrible. For every modification in your database, you'll have to adapt your application's code in N places. Possibly you'll have to adapt code in other applications as well (e.g. interfaces to other systems).

There is a better solution than doing it in your code.

You can just add `ON DELETE CASCADE` to your foreign key.

```
ALTER TABLE dbo.T_Room -- WITH CHECK -- SQL-Server can specify WITH CHECK/WITH NOCHECK
ADD CONSTRAINT FK_T_Room_T_Client FOREIGN KEY(RM_CLI_ID)
REFERENCES dbo.T_Client (CLI_ID)
ON DELETE CASCADE
```

Now you can say

```
DELETE FROM T_Client WHERE CLI_ID = x
```

and the rooms are automagically deleted when the client is deleted.

Problem solved - with no application code changes.

One word of caution: In Microsoft SQL-Server, this won't work if you have a table that references itself. So if you try to define a delete cascade on a recursive tree structure, like this:

```
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_T_FMS_Navigation_T_FMS_Navigation]') AND parent_object_id =
OBJECT_ID(N'[dbo].[T_FMS_Navigation]'))
ALTER TABLE [dbo].[T_FMS_Navigation] WITH CHECK ADD CONSTRAINT
[FK_T_FMS_Navigation_T_FMS_Navigation] FOREIGN KEY([NA_UID])
REFERENCES [dbo].[T_FMS_Navigation] ([NA_UID])
ON DELETE CASCADE
```

```
GO

IF EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_T_FMS_Navigation_T_FMS_Navigation]') AND parent_object_id =
OBJECT_ID(N'[dbo].[T_FMS_Navigation]'))
ALTER TABLE [dbo].[T_FMS_Navigation] CHECK CONSTRAINT [FK_T_FMS_Navigation_T_FMS_Navigation]
GO
```

it won't work, because Microsoft-SQL-server doesn't allow you to set a foreign key with **ON DELETE CASCADE** on a recursive tree structure. One reason for this is, that the tree is possibly cyclic, and that would possibly lead to a deadlock.

PostgreSQL on the other hand can do this;

the requirement is that the tree is non-cyclic.

If the tree is cyclic, you'll get a runtime error.

In that case, you'll just have to implement the delete function yourselves.

A word of caution:

This means you can't simply delete and re-insert the client table anymore, because if you do this, it will delete all entries in "T_Room"... (no non-delta updates anymore)
