

Re: Which database design is better

Source:

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2004-10/0596.html>

From: David Browne (*meat_at_hotmail.com*)

Date: 10/03/04

Date: Sun, 3 Oct 2004 16:24:31 -0500

"Joe Celko" <jcelko212@earthlink.net> wrote in message
news:%23IvlhZWqEHA.952@TK2MSFTNGP10.phx.gbl...
>>> *But what's wrong with the circular foreign key reference?* <<

>
> *Ambiguous DRI actions are possible. An action in A changes tables B and*
> *C. The change in B also changes C. Which change takes effect in C?*
> *Why?*

Not here. You shouldn't be using ON UPDATE CASCADE in first place, since you shouldn't be updating primary keys. And the cascading deletes cause no ambiguity, since the FK of key_employee would never have ON DELETE CASCADE turned on. It would amount to a business rule that: whenever you fire the key employee, the store ceases to exist.

> *They can prevent tables from being updated, inserted into or deleted*
> *from -- remember the old "You cannot get a job without experience and*
> *you cannot get experience without a job" joke?*

Foreign keys exist precisely to prevent certian inserts, updates or deletes. Here the FK enforces the following rules:
-You must add an employee to a store before assigning a key employee.
-You cannot delete a key employee; first you must assign someone else.

Just FK's enforcing business rules. Nothing scary and recursive or NP-complete.

>
> *Finding general solutions to such problems is an NP-complete problem and*
> *it is a bitch even with a small number of tables and references.*
>
> *SQL Server elected to disallow them; DB2 does some fancier checking for*
> *cycles.*
>
>>> *It faithfully models what's going on here.* <<
>
> *No, it does not. A store is not an attribute of an employee, so your*

> *data model is fundamentally wrong.*

Yes it is. Each employee has exactly one store. Therefore store is an attribute of employee.

> *And you are violating the principle*
> *that a table models an entity or a relationship but never both. You*
> *need that relationship table, and it ought to look like this:*
>

That's not a principle at all. According to that, you could never have a FK relationship between two "entity tables", there would always need to be an intermediate "relationship tables". There's nothing particularly wrong with introducing relationship tables to model all your relationships, but for 1-many relationships it's generally unnecessary.

> *CREATE TABLE EmpAssignments*
> *(store_nbr INTEGER NOT NULL*
> *REFERENCES Stores(store_nbr)*
> *ON UPDATE CASCADE*
> *ON DELETE CASCADE,*
> *ssn CHAR(9) NOT NULL PRIMARY KEY*
> *REFERENCES Personnel (ssn)*
> *ON UPDATE CASCADE*
> *ON DELETE CASCADE);*
>
>>> *Using a separate linking table is fine, but will behave exactly the*
> *same as a nullable foreign key column in store. <<*
>
> *The correct term is "relationship table" -- the term "linking" is from*
> *the old IDMS and IMS network databases. Then doesn't the phrase*
> *"nullable [foreign] key" strike you as a sign of a design flaw?*
>

No it's exactly what you end up with with the relationship table. You may or may not have a row in the relationship table, just as you may or may not have a value in the "nullable foreign key" column.

David