Nuchem Katz
nkatz01
Introduction to Database Technology (DT)
Ms Ping Brennan
FMA
01/08/2017

Queries

Information retrieval

1. Listing of children and their associated carer

```
create view Childlist as
select concat(child_fname,' ', child_sname) as 'child name', concat(carer_fname,' ',
carer_sname) as 'carer name', carer_phone as 'carer phone'
from Child, Carer
where carer_id = child_carer
order by child_sname, child_fname;
```

This query produces a view of a list of all children's full names and their carer's full names and their carer's full mobile numbers. I've used a 'where' statement to achieve a join and I've also created the view in my database. This query joins a child with their carer in my results only after assuring that the primary key (pk) carer_id and the foreign key (fk) child_carer match. I've checked manually that the output is correct because for example Carer.carer_id 2, Ms Little Weed, appears twice in the Child.child_carer column as being responsible for the two Flowerpot children 2 and 3 and indeed in my output they're both linked with her.

2. Count of children each carer is responsible for

To add myself as carer no 12, I ran the following statement:

```
insert into Carer
values ('12','Mr','Nuchem','Katz',[redacted]Null,'London',[redacted];
```

I then run the following which concatenates the f_names, s_names of the carers and counts the number of children for which each has responsibility. It counts row by row in the child where the fk child_carer in Child table matches the pk carer_id in Carer table and also where there is no match in the Child table. This is due to the use of left join gives me all the carers regardless of whether they have any children linked to them. They're grouped by carer_id, as otherwise each instance that it joins a child with a carer, would appear on a new row. I've also saved it as a spreadsheet. I've checked manually that the output is correct because for example Carer.carer_id 8, Ms Ethel Menace carer, appears twice in Child.child_carer column, meaning she's responsible for two children (child_id 13 and 16) and hence my output shows that.

activity number. They're then grouped by carer_id whose appearance is unique in - and whose details are extracted from - Carer table. These restraints also mean that when it counts the sum of activity_fee, it can only calculate them, grouped by the children that belong to a particular carer. I've checked manually that the output is correct because for example Carer_id 6, Liane Cartman, is associated in Child.child_carer with four children, child_ids 8, 9, 10 and 11. These 4 children are all linked to activity_id 3 in Childactivity table for £2.50 per activity. Multiply by 4, that's £10.00. Indeed in my output, Mrs Liane Cartman has 4 activities associated with her for the amount of £10.00 whereas someone with no children doesn't appear in my output.

```
select concat(carer_title,' ', carer_fname,' ', carer_sname) as 'Carer name' ,
count(C.activity_id) as 'No of activity',  concat('£', round(SUM(activity_fee),2)) as 'Total
Activity fees'
from Carer
join Child D on child_carer = carer_id
join Childactivity C on  D.child_id = C.child_id
join Activity A on C.activity_id = A.activity_id
group by carer_id
order by carer_sname, carer_fname;
```

Referential Integrity

Referential integrity

a) Every table in a relational database must have one or more columns where the primary key (pk) is recorded. This uniquely identifies each record in the table. When two tables are linked together in a 'one to many relationship', the "one" side of the relationship (the parent) will have a value for the pk which will be unique throughout that table. The "many" side of the relationship (the child) will link to its parent by means of a column referred to as the foreign key (fk). Any given entry in the fk column will match a value in the pk column of the parent. The fk value is not necessarily unique within its table - hence the "many" side of the relationship.

   An example in the Playscheme database is the pk Carer.carer_id that links with fk Child.child_carer. This gives me all the children that are cared for by a given carer. Another example is the pk Child.child_id and its foreign key childactivity_child_id. Childactivity helps us link a child with a given activity they're taking.

Pks and fks are referred to as constraints. Referential integrity says that we must assure that every value in a fk column of a Child matches a value of its parent's fk column so that each record in the child is linked with its associated record in the parent. Else, If a fk has no pair, to its pk column in the parent table, the link between the two tables is broken. To enforce referential integrity, the fk (eg Child.child_carer) is set to ON DELETE NO ACTION and ON UPDATE NO ACTION, so that an attempt to delete (or update the pk of) the record in the parent table related to this fk refuses to do so.

Also, to enforce constraints to remain consistent, if an attempt to enter a non-existing parent pk value, in a fk column of a child, is made, an error will occur notifying that this value cannot be found in the column of the parent that it's supposed to be linked to. (it can remain empty, in which case, its attribute will have to be set to allow NULL.)

In other words, unless its fk column is set to allow Null, to protect Referential integrity, fk constraints does not allow to add a record in a child table, eg. a 'child' in the 'Child table' in the Playscheme database, before assuring that there is a Carer in the Carer parent table, available to be associated with this Child through the fk child_carer column. Similarly, unless its fk column is set to Cascade or NULL, a record in a parent table will not allow itself to be changed or deleted so long as there is a related column in a child table. For example in the Playscheme database, a record in the parent Carer table is not deletable, or its pk changeable so long as there is a child in the child 'Child table' related to it. Say it were allowed, the impact would be that for eg if you queried for all the children together with their Carer, you'd have a child or more with no carer. Or in the other eg above, you'd have a child who'd lost their activity(s) linked to them.

b) An attempt to delete a record in a parent, that has children related to it in its child table, will not work with a 'On Delete: NO ACTION' setting, which is the default setting, the way then to delete a record in the parent table would be as followed: First to either delete all children in its Child table and then delete the parent or simply change the fk of all related records in the child table so that they don't anymore link to this particular record that you would like to delete in the parent table. Eg. delete child 8, 9, 10, 11 from Child table or change their fk child_carer to Null or to any other pk in the Parent table Carer.carer_id, so that they don't link anymore to Liane Cartman, should she be the one you'd like to delete.

c) CASCADE would allow to delete a record in the parent table, eg Liane Cartman with carer_id 6 in the carer table but would also automatically delete all related records, child 8, 9, 10 and 11 - in the child table, namely, all children for which this deleted carer is responsible for. Similarly, if the pk Carer.carer_id is changed, its fk child_carer in the Child table will automatically be updated accordingly, in our case if Liane's pk in Carer.carer_id is changed, the fk in Child.child_carer will be updated with that changed pk so that her children remain linked to her. And if the children have themselvs children through their own fk in a third table, the grand-children would have to be deleted too (unless ON DELETE RESTRICT is set). This is all so that no orphans that link to nowhere are left when a parent is deleted as otherwise this may impede a one to many relationship that has been introduced with a linking table. Eg. say several children are deleted in our Child table and but their fk remained in the Childactivity table, when queriying the Activity table to see how many activities are taken, we would get the wrong answer because some of the Childactivity.child_id that link with Childactivity.activity_id are not relevant anymore since these children do not exist. However, if a related child's record is deleted altogether, the parent can remain as this doesn't affect data integrity.

An advantage of this is that it allows you to delete rows in one table if certain rows of information become redundant, say for eg if Liane Cartman is not a carer anymore, and there would be no chance of having left orphaned rows in other tables whose fk link to nowhere, because child 8, 9, 10 11 in the Child table, who all have 6 in their child_carer column, will be deleted along with her. Also, if you want to delete a number of children, you can do it by deleting just the parent they all relate to, rather than deleting child by child.

However, a disadvantage is that you don't get to keep the information about its related children in the other table/s that may be important to keep as they may be relevant even without their parent. Eg. the details of the children that Liane was responsible for may still be important to keep.

Similarly, with editing a primary key, since the CASCADING setting is done on the fk, people may change a pk of the Carer and not realise that the fk is also going to change.

```
select concat(carer_fname,' ', carer_sname) as 'Carer name', count(child_carer) as 'No of
children'
from Carer
left join Child  on child_carer = carer_id
group by carer_id;
```

3. A register for a specified activity

This query sets a variable = to activity ‘Art’ , looks up that activity and assigns its activity_id to another variable named @activitynum. It then selects all the listed columns from the tables specified, and joins their rows, on the following conditions: The fk Childactivity.child_id matches the pk Child.child_id. This in turn matches up the pk Activity.activity_id associated, with its fk, Childactivity.activity_id and in particular it is the same as the value of @activitynum . But it also commends that in addition, Activity.activity_name column has also got to match with the @activityname (i.e. ‘Art’) variable. I’ve checked manually that the output is correct because for example Child.child_id 6, Jemima Puddleduck is only linked up with 1 activity in the Childactivity table, activity_id 1, which is ART. Hence in my output she only appears once, whereas Anglica Pickles who is linked up with Activity_id 2 doesn’t appear in my output. Also, Fred Drack who is Carer_id 4 appears in Child.child_carer in the same row as Jemima (child_id 6) and so is my output.

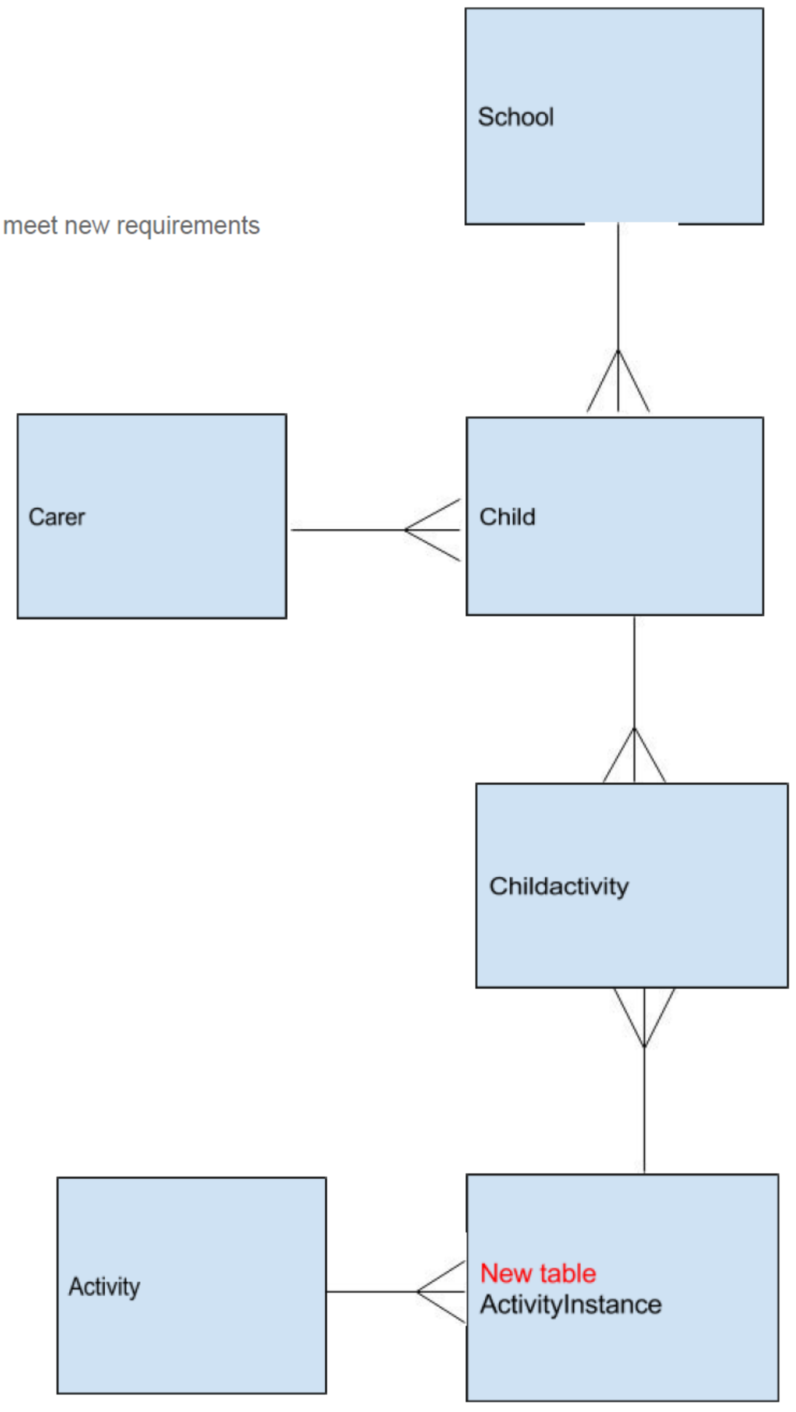
```
set @activityname = 'Art';
set @activitynum =
(select activity_id
from Activity
where activity_name = @activityname);
select activity_name, concat(child_fname,' ', child_sname) as 'Child name' ,
concat(carer_fname,' ', carer_sname) as 'Carer name', carer_phone
from Activity A, Child D, Childactivity C, Carer
where D.child_id = C.child_id and C.activity_id = @activitynum and carer_id = child_carer
having activity_name =  @activityname
order by child_sname, child_fname;
```

4. Total amount payable by each carer.

The following statement concatenates a carers title, fname and sname. It also counts the no of entries in childactivity_activity_id but only where the fk Childactivity.child_id in the adjacent column matches with its pk Child.child_id whose Child.child_carer column also matches with Carer.carer_id. In turn, it would count each childactivity_activity_id row as an

Database extension

Fig 1: ER Diagram to meet new requirements



Explanation

There is a many to many relationship between Child and Activity as one child can take several activities and one activity can take several children. This was broken by introducing the Childactivity table that paris a given child_id with a given activity_id.

But now with the idea of instances introduced, there is still a many to many relationship between Child and Activity as the same activity can be taken by the same child on several instances and several children can take the same instance of the same activity.

To solve this, we create a new table named ActivityInstance that records info about a given instance. This has a fk column that links each instance with the pk in Activity table to which it belongs to. We then change the fk column Childactivity_activity_id to be named instead instance_id, linking to pk ActivityInstance.instance_id.

This way, we now have a one to many relationship between ActivityInstance Childactivity and also between Activity and ActivityInstance because although an instance can take many children, a child can only pair with an instance once. Similarly, although one activity can take many instances, a given instance relates only to one activity.

Table 1: New table's columns listing

Table: ActivityInstance			
Column	**Datatype**	**Attribute**	**Default**
ActivityInstance.instance_id	INT	PK, NN, AI	
ActivityInstance.start_date	DATE	NN	
ActivityInstance.end_date	DATE	NN	
ActivityInstance.activity_id	INT	FK, NN FK reference Activity.activity_id	ON DELETE NO ACTION ON UPDATE NO ACTION

Other changes

FK activity_id column in **Childactivity** table to be changed:

Table 2: Modifications to Childactivity table

Table: Childactivity			
Column	**Datatype**	**Attributer**	**Default**
Childactivity.instance_id	INT	COMPOSITE PK, FK, NN FK reference ActivityInstance.instance_id	ON DELETE NO ACTION ON UPDATE NO ACTION

Comments

I have used the MYSQL manual to lookup data types, functions and mysql syntax.