



Merging Incompatible Database Tables

Introduction

This project was concerned with the general problem of porting all records from one database site containing multiple tables and relationships to another database site with the same kind of tables and relationships. In order to respect referential integrity of the other database site during the port operation, a mathematical model has been developed to determine the order in which the records should be ported. This problem arises whenever copies of a master database are used in remote sites, and data from those sites are ported to the master database on a regular basis.

For example, UBC Medicine recently expanded its facility to accommodate satellite locations throughout the Province, and faculty appointment information maintained in those remote sites must be periodically ported to a master database in Vancouver. In particular, remote tables containing references (ie. foreign keys) to other remote tables cannot be ported to the master database until the referenced records in the remote tables have already been ported, otherwise referential integrity would be violated in the master database. We have provided a simple recipe for automatically determining the order in which all of the tables may be ported in order to preserve referential integrity. Our solution works for any database schema, and may be used to automate the script for any ETL process¹.

Technical Description

The main problem is that we need to port all records from one database site to another database site (eg. site A to site B), but the lookup tables in A may have different records than those in B. A given record in A may be missing in B or it may be stored under a different Id (ie. key). We must append the records in A which are missing in B and then re-calculate all references to their Ids in B so that the records in A that are ported to B point to the same lookup table information in B that they did in A. This preserves the information defining each record in A using those lookup tables. More importantly, referential integrity is preserved (otherwise the port would fail).

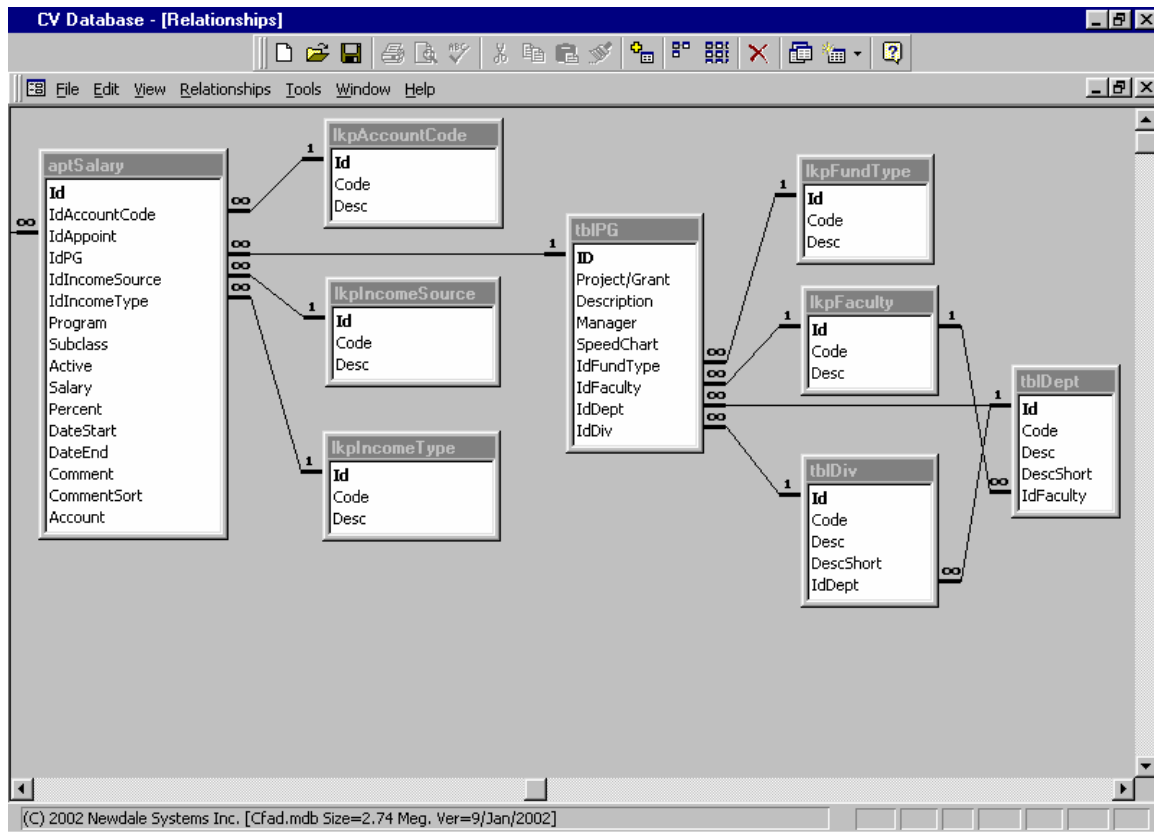
We also have the problem of what is meant by "missing". If a record in A is contained in B but one of its fields contains a slightly different value, is it missing? We use a single query to define what records are missing for a given lookup table, using an outer join that searches for null values. These queries may be revised at any time without affecting the rest of the port. That way, our definition of "missing" may be changed later by simply modifying the queries used to define that term. In effect, we are defining two records to be "equal" if they agree on a certain field. For example, if a table of Degrees has a Short field (eg. PhD) and a Long field (Doctor of Philosophy) then the Short field might be chosen to decide if two records from different sites essentially contain the "same" information.

Compounding these problems is the fact that some lookup tables use pointers to other, more elementary, lookup tables which need to be ported first.

Finally, when we add records to a table in B they must obey referential integrity or the port will fail.

¹ Extract, Transform and Load

Let's look at a concrete example containing appointment information for University faculty:



Here, the aptSalary table contains salary information for someone's appointment (referenced by the foreign key value IdAppoint, which points to an appointment table). This information consists of raw values like Salary, along with pointers such as IdAccountCode which point to lookup tables containing the relevant information, such as Code (short form) and Desc (long form). Often the field Code is used to decide whether two records are "equal" since the Desc field may be changed in various database sites to suit local preferences on reports.

We need to copy over the apt, tbl and lkp tables to a master database in a certain order, saving their internal Ids so that we can properly match up records in the apt tables with records in the tbl and lkp tables. That is, we want to insure that each record points to the same information in the master database as it did in the original database.

For example, look at aptSalary.

Four fields contain pointers to four other tables: lkpAccountCode, tblPG, lkpIncomeSource, lkpIncomeType. We can't simply copy the aptSalary records from site A to site B because the original pointers (IdAccountCode, IdPG, IdIncomeSource, IdIncomeType) may now point to the wrong records in the four tables in site B since there's no guarantee that these four tables are the same in both sites. For example, lkpAccountCode may contain ten records with Id=1,...,10 in each site, but contain different Code and Desc values. So a pointer value of 1 in site A may no longer point to the correct record in site B. Furthermore, site B may not even have a record with Id=1.

What we do is this:

First, create a new field called IdSave in each of the lkp tables in both sites (forget about the tbl tables for a moment, because they are built from lkp tables using pointers). In A, set IdSave to Id in all of them, so we can remember these values after some have been copied over to B.

Then copy those records from each lkp table in site A to site B where the Code or Desc fields can't be found in site B (these are the records missing in the lookup tables of site B which will be needed before aptSalary can be ported). This copy operation will build new values for the Id field automatically, since they are counter fields controlled by Access. That's why the pointer values in A can no longer depend on them when their tables are copied over to B. They must be re-calculated to point to the correct records in the lookup tables, using IdSave.

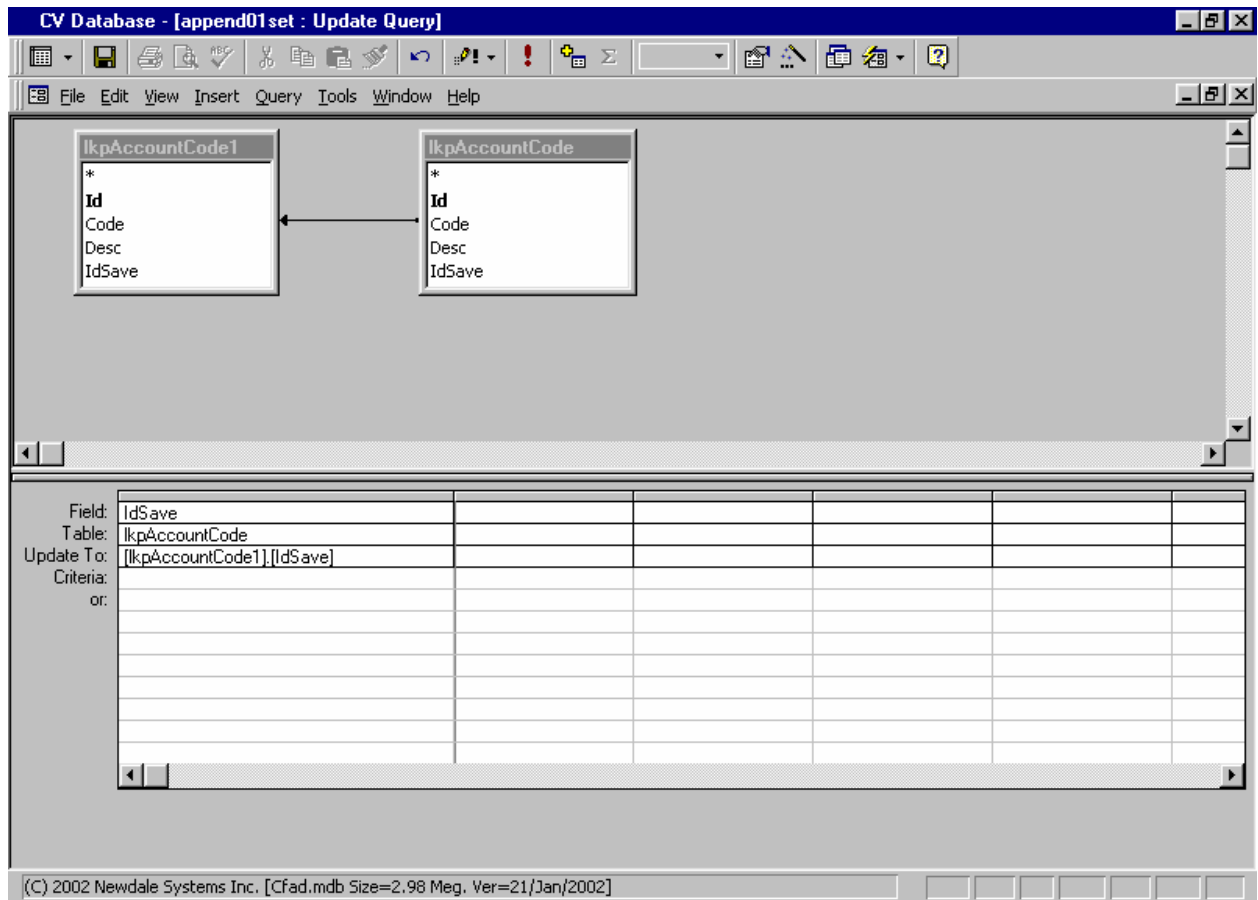
For example, the following query append01 copies missing records in lkpAccountCode1 (lookup table in A) to lkpAccountCode (lookup table in B):

The screenshot displays the Microsoft Access interface for an Append Query named 'append01'. The design grid shows two tables: 'lkpAccountCode1' and 'lkpAccountCode'. The design grid includes fields: Code, Desc, IdSave, and Code. The criteria row shows 'Is Null' for the second 'Code' field.

Field:	Code	Desc	IdSave	Code			
Table:	lkpAccountCode1	lkpAccountCode1	lkpAccountCode1	lkpAccountCode			
Sort:							
Append To:	Code	Desc	IdSave				
Criteria:				Is Null			
or:							

(C) 2002 Newdale Systems Inc. [Cfad.mdb Size=2.98 Meg. Ver=21/Jan/2002]

This query defines what's "missing" for that lookup table, using the field Code². A related query append01set for that table sets IdSave in B to its original value in A:



This is necessary because records in A which were already in B before appending occurred must be accounted for. Otherwise the query append01 would have sufficed because IdSave has already been set to Id for the records which were appended.

At this point site B has all the lookup table records that site A needs, and we've got the IdSave fields to remember their original lookup values used in site A. The is even true for the ones which were already in B before the port. So we can use them to re-calculate the pointers in Site B.

Now, look at tblDept. It uses lkpFaculty for part of its definition but we know those records are now present in site B (because we're doing this process hierarchically). We do for tblDept what we did for the lookup tables, using a new field IdSave as before, because tblDiv needs it. The important point here is that IdFaculty is changed to the (possibly different) Id value in lkpFaculty using the IdSave field to locate the correct record. This happens when we copy over the records.

We now do for tblDiv what we just did for tblDept.

Then we do for tblIPG what we just did for the other tbl tables in the above diagram.

Then we copy over the aptSalary records in the same way, using the IdSave values to correctly re-set the values of IdAccountCode, IdPG, IdIncomeSource, IdIncomeType.

² It may be replaced by more sophisticated computations defining "missing".

Then we copy over the aptAppoint records, after doing the same preliminary work on the other lkp and tbl tables which it references.

Note that there is a natural hierarchy on the order that we copy over the tbl tables, based on which ones are used to define the others.

Some special things about these queries are:

1. Outer joins are used to avoid legacy values for IdSave from previous ports. If we use inner joins we may not change IdSave values that were set in previous runs because the above update query might not select the records in which they sit.
2. Temporary tables are used for the apt tables to avoid duplicated records when lookup tables have duplicate values for Code or Desc (this is a subtle point).

The above process is general, and may be applied to any pair of databases with the same database schema.

First, we define three types of tables: lkp1 tables (tables with no pointers but which are pointed to by at least one other table), lkp2 tables (tables with at least one pointer to another table and which are pointed to by at least one other table), and general tables (tables which are not pointed to by any other table). It follows that any table is exactly one of these kinds of tables. The first two types are called lookup tables because they are used by other tables to "look up" information using pointers to them, rather than storing it directly in the table itself. This is the essence of relational database design. It saves space and avoids the problem of maintaining the same information contained in several records or tables (such as the Long form of a given Degree).

The general idea is to port lkp1 tables first, then lkp2 tables in a certain order, and finally the general tables. Before each table is ported, all references to its lookup tables are re-calculated using the ported versions of those lookup tables. Also, a special query defines which records from the original table need to be ported, based on a special field for each table which we use to define record "equality".

The porting order of the lkp2 tables is determined mathematically in the following way:

Define the binary relationship $X \rightarrow Y$ on any pair of tables X, Y in lkp2 as follows:

$X \rightarrow Y$ if and only if at least one field in X always contains a key value of a record in Y

The the following may be proved:

*For any X , there is no sequence (possibly empty) Y_1, \dots, Y_n such that:
 $X \rightarrow Y_1 \rightarrow Y_2 \rightarrow \dots \rightarrow Y_n \rightarrow X$*

In other words, $X \rightarrow X$ is not possible, nor is $X \rightarrow Y_1 \rightarrow X$ for any Y , etc.

This theorem rests upon the assumption that in any lookup table no pointer field can behave like a key, for otherwise different records would not be allowed to use the same value.

This result says that a lookup table can never point to another lookup table, which points to another lookup table, etc. so that the last lookup table points back to the original one.

Now define the binary relationship

$$Y < X$$

if and only if

$X \rightarrow Y_1 \rightarrow \dots \rightarrow Y_n \rightarrow Y$ for some sequence (possibly empty) Y_1, \dots, Y_n

Then $<$ is a strict partial order on the set lkp2, ie. it satisfies the axioms:

$X < X$ is not possible

if $X < Y$ and $Y < Z$ then $X < Z$

A porting order of lkp2 tables is defined as any distinct sequence X_1, \dots, X_n of all tables in lkp2 such that:

For all i, j : if $X_i < X_j$ then $i < j$

In other words, a porting order lists the tables in lkp2 in such a way that each table appears after all the lookup tables it points to.

Finally, the following simple observation on partially ordered sets provides an ordering of the lookup table porting which will respect referential integrity:

Porting orders always exist

To show this, simply use induction on the set's cardinality.

A script implementing this for SQL Server 2005 may be found here:

www.newdalesystems.com/articles/Article008.htm

To build random graphs online, use this:

<http://www.newdalesystems.com/Resources/Resource01/Resource01.htm>