AIMS Database Structure in SQL

Current as on 15th April 2004

The Assets and Infrastructure Management System (AIMS) software is also being ported to and developed under Microsoft® SQL 2000 Server.

The advantages of porting the Prolog-based internal databases to Microsoft® SQL 2000 Server (henceforth called SQL, in short) are many. Some of them are:

- SQL is a production-grade relational database management system (RDBMS).
- SQL is highly scalable and the databases can be distributed among different Servers.
- SQL can be easily accessed using the traditional Windows Forms, or using the browser-based Web Forms, or even using Wireless, handheld and mobile devices.
- The AIMS database, through SQL, can be made truly distributed even geographically.

While porting the AIMS database from Prolog to SQL, there are a few important points to be remembered:

- The various databases of AIMS, currently stored as Prolog's internal databases, have to be converted to SQL Tables.
- The presence of various lists, as constituents of the Prolog's internal databases, should be converted to separate Tables in SQL.
- The various IDs present in the AIMS database should be converted into Primary and Foreign Keys in SQL.
- The "Name", "Description" and "Notes" fields should declared as 'varchar' in SQL, instead of the 'string' declaration in Prolog.
- The information within existing AIMS databases must be normalised for SQL Tables to avoid data redundancy.

And so on.

To start with, let us examine a small fragment of the AIMS database in Prolog and study how this information is converted to SQL Tables. In the following code listing (Code Listing 1), the structures of only the four prominent databases (DB) of AIMS, viz., assets, systems, components and parts, are shown. For each database, the DB structure, its explanation and an example are given. The actual database declarations are shown in **bold**.

```
GLOBAL DATABASE – assets /* Asset Declaration */
/* Represents:
asset(AssetID,AssetName,AssetDescription,AssetUSINumber,AssetSerialNumber,AssetI
nclusionDate,AssetInclusionTime,AssetURL,ListOfSystemsInTheAsset,ListofEvolution
aryCyclesAssociatedWithTheAsset,ListOfNotesAssociatedWithTheAsset) */
  asset(id,name,desc,usi,serialno,date,time,url,SLIST,SLIST,SLIST)
/* Stored in: assets.dat file */
/* Asset Example - has 3 systems, 3 evocycles associated with the asset, and 3
notes */
/* Note that for the Systems, Evocycles and Notes, only the corresponding IDs
are listed */
/* asset("IA1","MyAsset","A Test
Asset",["IA1SY1","IA1SY2","IA1SY3"],["IA1LF1","IA1LF2","IA1LF3"],["IA1NT1","IA1N
T2","IA1NT3"]) */
```

```
GLOBAL DATABASE – systems /* System Declaration */
/* Represents:
system(SystemID,SystemName,SystemDescription,SystemUSINumber,SystemSerialNumber,
SystemInclusionDate,SystemInclusionTime,SystemURL,ListOfComponentsInTheSystem,Li
stofEvolutionaryCyclesAssociatedWithTheSystem,ListOfNotesAssociatedWithTheSystem
) */
  sys(id,name,desc,usi,serialno,date,time,url,SLIST,SLIST,SLIST)
/* Stored in: systems.dat file */
/* System Example, which is the first System ("IA1S1") of the Asset "IA1" */
/* Note that the example system's own ID, is the same as the fist in the list of
systems, for the asset "IA1" */
/* system("IA1SY1","MyFirstSys","A Test
System",["IA1SY1CO1","IA1SY1CO2","IA1SY1CO3"],["IA1SY1LF1","IA1SY1LF2","IA1SY1LF
3"],["IA1SY1NT1","IA1SY1NT2","IA1SY1NT3"]) */

/* Similarly Components and Parts are listed as follows */
GLOBAL DATABASE – components  /* Component Declaration */
/* Represents:
component(ComponentID,ComponentName,ComponentDescription,ComponentUSINumber,Comp
onentSerialNumber,ComponentInclusionDate,ComponentInclusionTime,ComponentURL,Lis
tOfPartsInTheComponent,ListofEvolutionaryCyclesAssociatedWithTheComponent,ListOf
NotesAssociatedWithTheComponent) */
  component(id,name,desc,usi,serialno,date,time,url,SLIST,SLIST,SLIST)
/* Stored in: components.dat file */
/* Component Example, for the first asset, first system, first component */
/* component("IA1SY1CO1","MyFirstComp","A Test
Component",["IA1SY1CO1PA1","IA1SY1CO1PA2"],["IA1SY1CO1LF1","IA1SY1CO1LF2"],["IA1
SY1CO1NT1","IA1SY1CO1NT2"]) */


GLOBAL DATABASE – parts /* Part Declaration */
/* Represents:
part(PartID,PartName,PartDescription,PartUSINumber,PartSerialNumber,PartInclusio
nDate,PartInclusionTime,PartURL,ListofEvolutionaryCyclesAssociatedWithThePart,Li
stOfNotesAssociatedWithThePart) */
  part(id,name,desc,usi,serialno,date,time,url,SLIST,SLIST)
/* Stored in: parts.dat file */
/* Part Example, for the first part of the first component of the first system
of the first asset */
/* part("IA1SY1CO1PA1","FirstPart","A Test
Part",["IA1SY1CO1PA1LF1","IA1SY1CO1PA1LF2"],["IA1SY1CO1PA1NT1","IA1SY1CO1PA1NT2"
]) */

/* Similarly, all the other DB components are declared */
```

Code Listing 1 A small portion of the AIMS DB in Prolog (all lists are lists of strings; actual declarations are shown in **bold**)

Based on the Code Listing 1, there are a few important features of the AIMS (Prolog) database worth reviewing at this stage. Note that some of these features are a direct consequence of Prolog's internal database syntax and semantics, and are not specific to the design of the AIMS DB structure *per se*.

- The constituents of each DB (e.g., assets, systems and so on) are identified uniquely by an ID (e.g., for a typical part, the ID could be `IA1SY1CO1PA1`). Each ID is a `STRING` variable.
- For each such constituent, its branches are represented only by their IDs, as a list of IDs (e.g., the Notes for a given part `IA1SY1CO1PA1` is represented by the ID list `["IA1SY1CO1PA1NT1","IA1SY1CO1PA1NT2"]`.

- These IDs are generated automatically by AIMS without any User intervention.
- IDs are automatically adjusted if a branch is moved, copied, added anew or deleted.
- These IDs provide the key link in the hierarchy of databases.
- No two IDs are identical, throughout all the databases.
- The User (User, Manager or Administrator) of the AIMS software handles only the names of the database constituents, leaving the AIMS system to handle the IDs.
- In AIMS, under the current scheme, no two names of the database constituents can be identical – much like how no two files can have the same name within a folder or directory.
- The AIMS database is so structured that it is more hierarchical than relational.
- The structure of the AIMS database allows any or all of the branches to be empty. In fact, it is enough if the ID field and the Name field are filled to create a valid DB constituent.
- There is *practically* no limit on the length of the `STRING` variable in any of AIMS DB fields and this length need not be declared in the beginning.
- Similarly, there is no limit on the number of elements within a given list, and this number need not be declared in the beginning. Each list can be anywhere between an empty list (represented by []) to a list having "infinite" number of elements.
- It is possible to re-configure each AIMS DB on the fly (as it is being used by AIMS) in such a way that those constituents used more frequently, are placed on top of the DB (using `asserta`) for faster access.

So, any attempt to convert the existing AIMS DB structure into SQL Tables must take into consideration all of these issues, or at least those features that directly impact the performance of the AIMS software. Some the desirable properties of the SQL RDBMS Tables could be:

- Ease of Use.
- Data normalisation (avoiding repetition of data leading to potential errors) is the next priority.
- No IDs to be entered at all by the User (as in AIMS Prolog version).
- Each database (Assets, Systems, Components, Parts, Lifecycles, Stages, etc.) could be a single Table, each Table containing many individual constituents.
- Each constituent (e.g., a typical asset or a typical part) can have one user-defined identity slot and one serial number slot (which could later be used for Search purposes).
- The primary identification mechanism of each constituent can be by the Primary Key of the entity, which is the natural way to identify any item.
- Since each database is a separate Table, it must be possible to fix flexible security and permissions to each of them.

With these aspects in mind, the SQL Tables for the four prominent databases of AIMS were designed (Figures 1 to 4) as follows. In each Figure, the first part (a) shows the properties of the SQL Table, and the second part (b) shows a few examples in each Table.

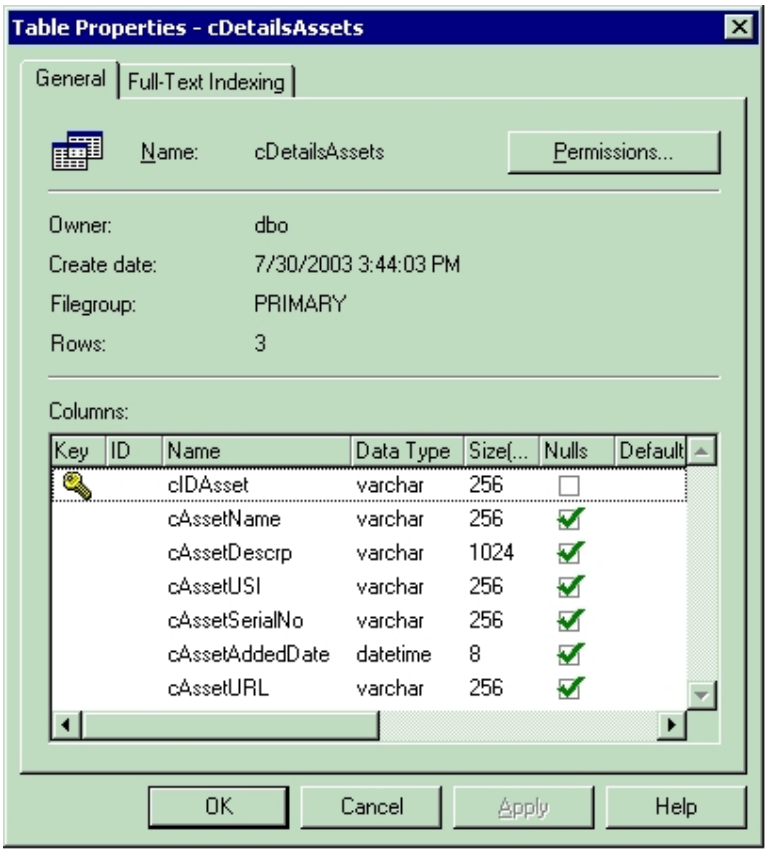

Figure 1a Properties of the "assets" SQL Table

Take a moment to compare the fields described in Figure 1a with the Code Listing 1. Unlike in Prolog, we need to fix the length of each field type right away! For example the first field `cIDAsset`, which is the primary key field for this Table has been declared as type `varchar` having a size 256. Since this is a key field, this cannot take a `NULL` value and that has been set in the Table properties. The other fields are `cAssetName` (to represent the name of the asset), `cAssetDescrp` (description of the asset, having a size of 1024), `cAssetUSI` (the USI number of the asset), `cAssetSerialNo` (the serial number of the asset), `cAssetAddedDate` (the date on which the asset was added), `cAssetURL` (the URL of the asset if any).

But where are the links to the Systems associated with each Asset, its Evolutionary Cycles and Notes? This is a major difference between the Prolog version of the AIMS DB and its SQL counterpart. In this case, the Primary Keys of the Assets SQL Table, make a direct reference to the appropriate Systems, in the Systems SQL Table, (to the EvoCycles in the EvoCycles SQL Table and so on) as can be seen by studying Figure 1b and Figure 2b. Consider the asset 'My First Asset' in Figure 1b. It has an ID value of IA1. Now refer Table 2b. There are three Systems IA1S1, IA1S2 and IA1S3 associated with the asset 'My First Asset' (IA1). Further branches of these Assets and Systems are so related, as one can observe from the SQL Tables shown in Figures 3b and 4b.

While accessing a root and its branches, the SQL version of the AIMS software's logic should then consult these Tables in succession to arrive at the right constituent. Such access and modification of SQL Tables will eventually be performed using a combination of ASP.NET and Internet Information Server (IIS), the mechanism of which would be discussed later in another document.

	cIDAsset	cAssetName	cAssetDescrp	cAssetUSI	cAssetSerialNo	cAssetAddedDate	cAssetURL
▶	IA1	My First Asset	First Example Asset	<NULL>	<NULL>	<NULL>	<NULL>
	IA2	My Second Asset	Second Example Asset	<NULL>	<NULL>	<NULL>	<NULL>
	IA3	Third Asset	Third Asset Example	<NULL>	<NULL>	<NULL>	<NULL>
*							

Figure 1b The "assets" SQL Table – Some Examples

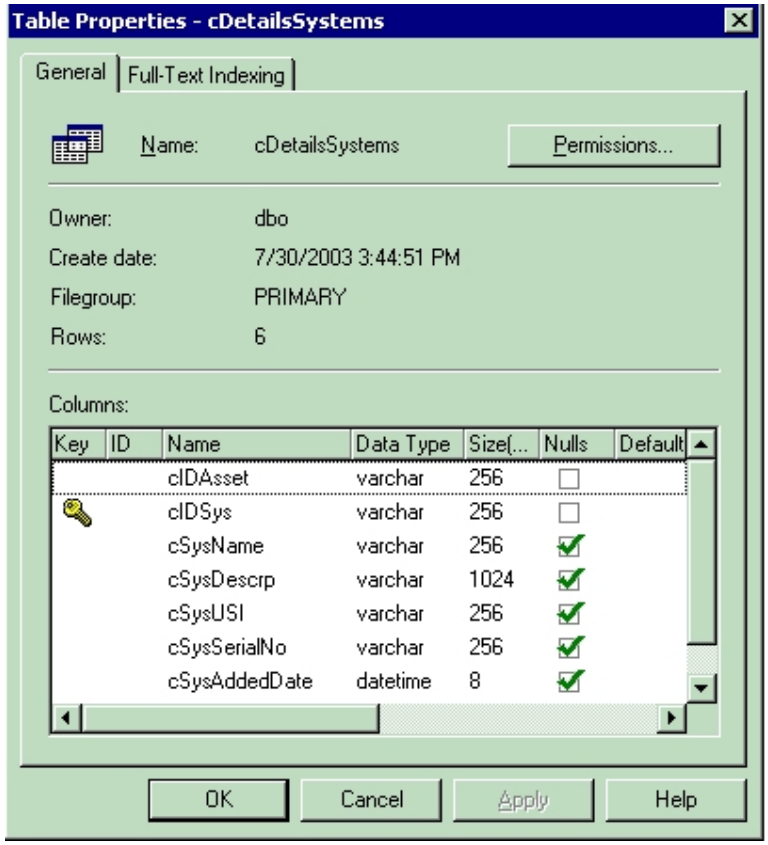

Figure 2a Properties of the "systems" SQL Table

	cIDAsset	cIDSys	cSysName	cSysDescrp	cSysUSI	cSysSerialNo	cSysAddedDate	cSysURL
▶	IA1	IA1S1	My First System in Asset One	Asset One's System One	<NULL>	<NULL>	<NULL>	<NULL>
	IA1	IA1S2	My Second Sys in Asset One	Asset One's System Two	<NULL>	<NULL>	<NULL>	<NULL>
	IA1	IA1S3	SysThree in AssetOne	Asset One's System Three	<NULL>	<NULL>	<NULL>	<NULL>
	IA2	IA2S1	My First Sys in Asset Two	Asset Two's System One	<NULL>	<NULL>	<NULL>	<NULL>
	IA2	IA2S2	My Second Sys in Asset Two	Asset Two's System Two	<NULL>	<NULL>	<NULL>	<NULL>
	IA3	IA3S1	SysOne in AssetThree	Asset Three's System One	<NULL>	<NULL>	<NULL>	<NULL>
*								

Figure 2b The "systems" SQL Table – Some Examples

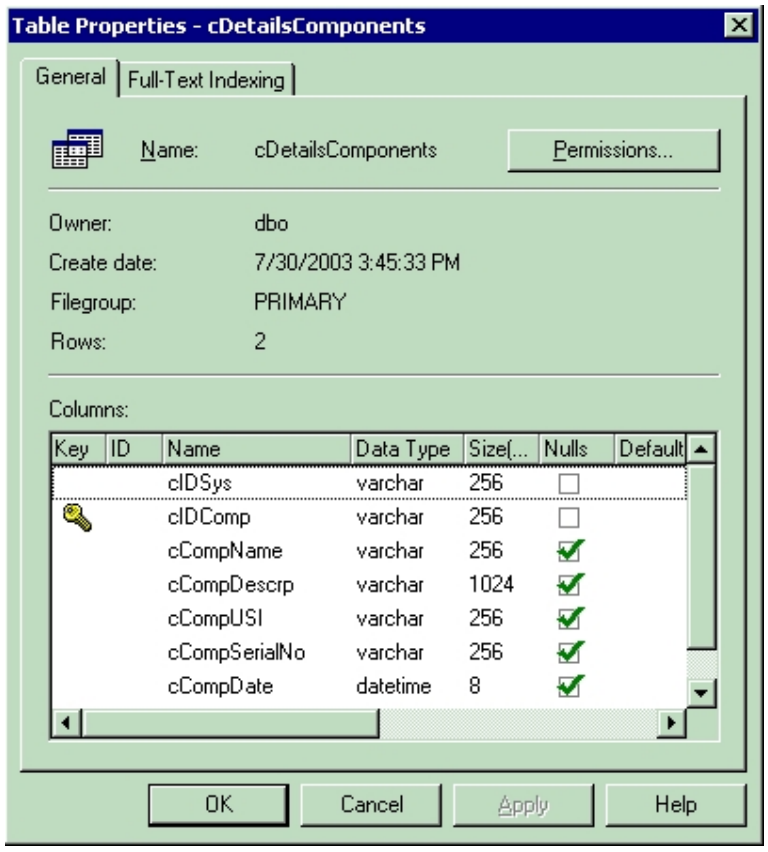

Figure 3a Properties of the "components" SQL Table

cIDSys	cIDComp	cCompName	cCompDescrp	cCompUSI	cCompSerialNo	cCompDate	cCompURL
IA1S1	IA1S1C1	Comp1 in Sys1 in Asset1	Component Example	<NULL>	<NULL>	<NULL>	<NULL>
IA1S2	IA1S2C1	Comp1 in Sys2 in Asset1	Another Example	<NULL>	<NULL>	<NULL>	<NULL>

Figure 3b The "components" SQL Table – Some Examples

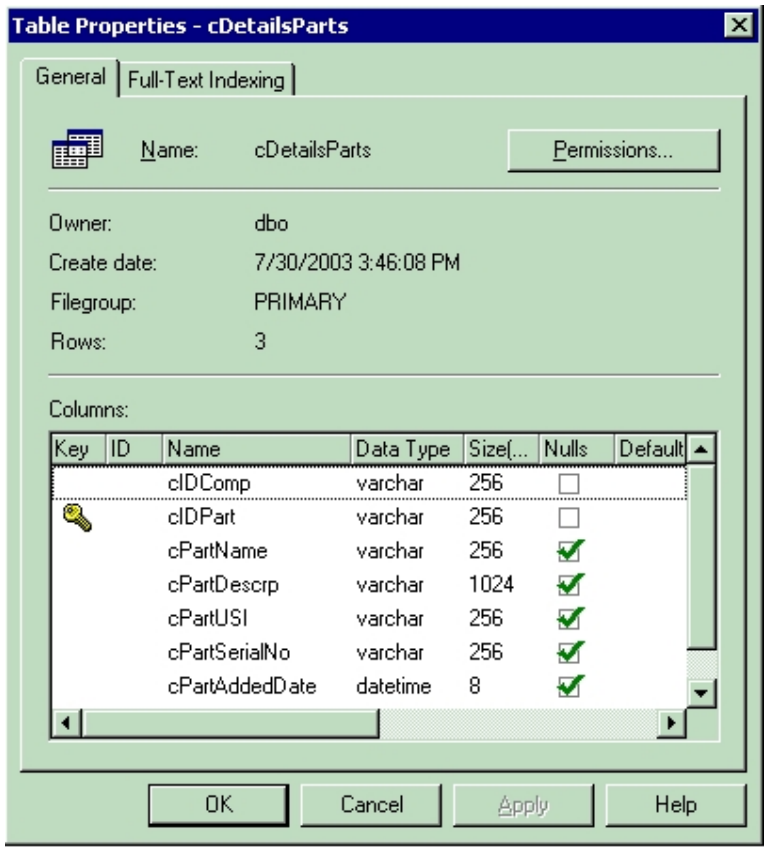

Figure 4a Properties of the "parts" SQL Table

cIDComp	cIDPart	cPartName	cPartDescrp	cPartUSI	cPartSerialNo	cPartAddedDate	cPartURL
IA151C1	IA151C1P1	Part1 in Comp1	A New Part	<NULL>	<NULL>	<NULL>	<NULL>
IA151C1	IA151C1P2	Part2 in Comp1	Another Part	<NULL>	<NULL>	<NULL>	<NULL>
IA152C1	IA152C1P1	Part1 in Comp1	Third Part	<NULL>	<NULL>	<NULL>	<NULL>

Figure 4b The "parts" SQL Table – Some Examples

We can summarise the following observations from these Tables (Figures 1 to 4):

- The Properties of each of these four prominent databases, represented here as SQL Tables, have several columns each.
- One column in each SQL Table, has been declared as the Primary Key, represented by the symbol .
- In the case of the "assets" SQL Table, `cIDAsset` is the Primary Key (Figure 1a).
- Note in Figure 1b the column represented by this Primary Key viz., `cIDAsset` has a separate ID earmarked for each Asset. (There are three examples given in Figure 1b).
- Unlike the case of the Prolog's internal database declaration we saw above in Code Listing 1, note that the type of each of the Property in Figure 1a (and in all the Property Figures 2a, 3a and 4a), is **not** a STRING. It varies from property to property. For example, in the Prolog's case, even the date and time fields are STRINGS. Here in SQL Tables, these are special types called `datetime` occupying a size of 8 bytes. Other fields are of type `varchar`.

- Each of the Property figures (Figures 1a, 2a, 3a and 4a) show which of the fields in each of these SQL Tables can take a `NULL` value. These are equivalent to empty STRING values in the Prolog scheme.
- The equivalent of lists in the Prolog scheme, is the Table itself in the SQL scheme. As a list can keep adding items to it, the SQL Table will add rows to it. The columns of the SQL Table, which represent the fields, are fixed however.
- In the case of Prolog's internal database structure of AIMS, the relationship between assets ("parent") and systems ("child"), systems ("parent") and components ("child"), etc. flowed from the appropriate declaration of correct IDs. In the case of SQL Tables, notice that this connection happens rather subtly as a combination of the Primary Keys of the "parents" and the "children". For example, consider Figure 2b – examples of "systems". The combination of cIDAsset and cIDSys define the unique relationship between assets ("parent") and systems ("child"). This happens for every parent-child relationship. These relationships are schematically shown in Figure 5.

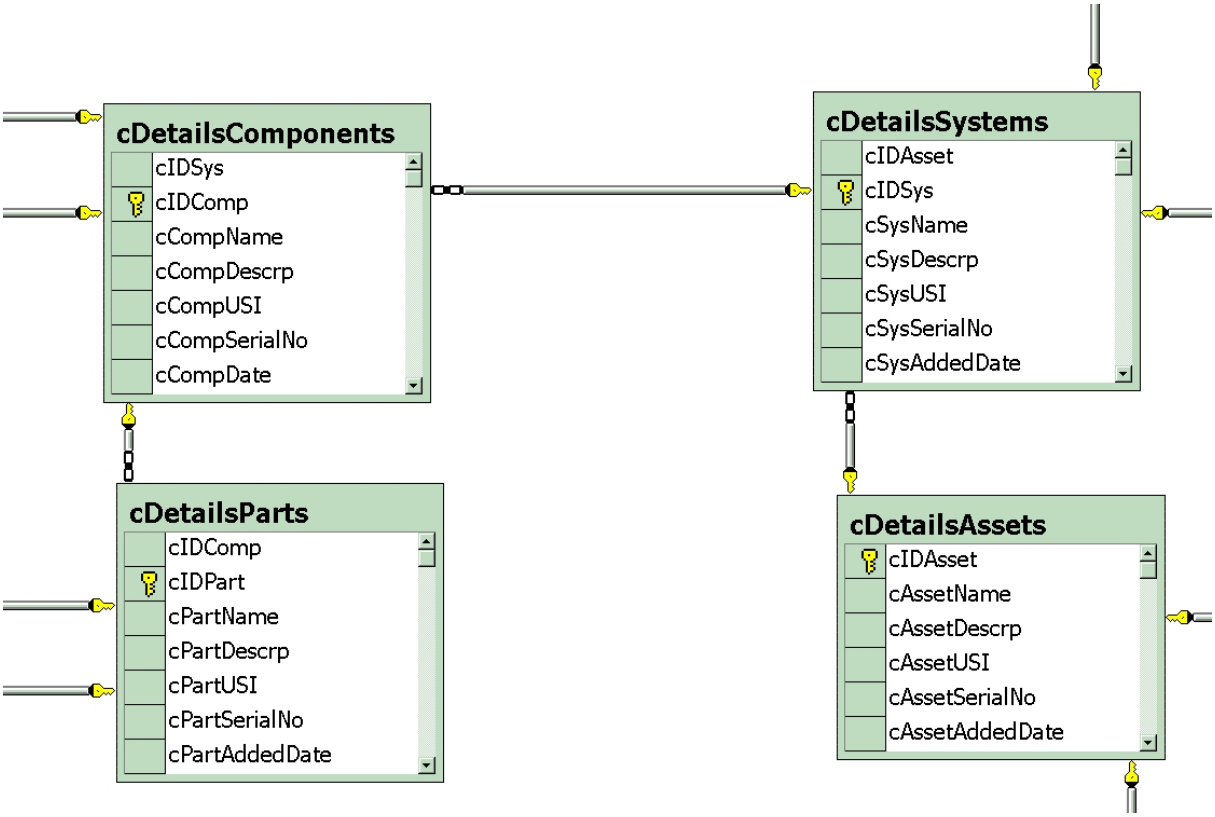

Figure 5 Schematic Relationship between the four prominent databases of AIMS in the SQL Scheme

- Note that unlike in the case of the Prolog's internal database scheme for AIMS, in the case of SQL Tables, the Primary Keys – at least in this version – need to be entered manually by the User.
- As can be seen from the Properties figures (Figures 1a, 2a, 3a and 4a), we can set the Permissions for each of these Tables – something that can be done even in the Prolog scheme, if the DB is present in an NTFS partition.
- Access to these Tables for creating new values, editing, moving and deleting values, one can use an ASP.NET front-end, so that entire SQL DB is accessible across the Intranet or even the Internet, with any browser / mobile device as the target.

In the case of the SQL version of the AIMS software, what could be a typical train of action from a User's point of view?

A User could

- first define an asset (adds a new entry to the Assets Table)
- adds a system to the asset (adds a new entry to the Systems Table)
- adds a component to the system (adds a new entry to the Components Table)
- adds a part to the component (adds a new entry to the Parts Table)
- adds another system to the asset (adds a second entry to the Systems Table)
- adds another asset (adds a second entry to the Assets Table)

and so on, all of which can be performed from a browser-like front-end.