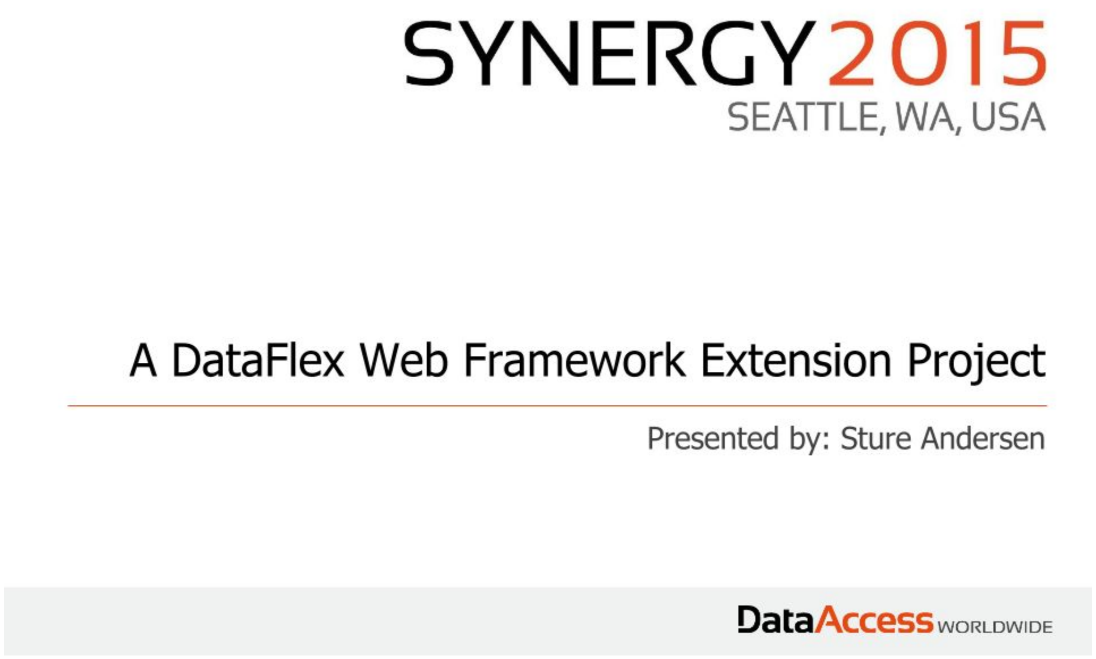


SYNERGY 2015
SEATTLE, WA, USA

A DataFlex Web Framework Extension Project

Presented by: Sture Andersen

DataAccess WORLDWIDE

Good afternoon.

I want start by saying that I am very excited about the direction of DataFlex at this moment.

We saw it several times during yesterdays presentations that you can generate mobile applications in minutes.

And I can confirm that.

On the night alpha-2 was released I read the What's New, Mobile section in the help file (takes a while, get coffee) and had a working mobile version of our time billing view within a couple of hours. With two "selects" and one "zoom". And a dashboard.

Getting our stuff on mobile devices like that will increase the exposure and the usability of our applications by a large factor.

I'm looking forward to seeing where that will bring us.

Why do need a new Web Framework Extension?

And by "we", I mean my own company, Sture ApS, and Front-IT that is run by Klaus Bethelsen and Charlotte Grønvold. We have experience in developing web applications for customers. And we all agree that it is a pleasure to work with DataFlex for these projects.

So we decided it was time to create a common library, *AppWrap*, to be used as a basis for new projects.

It should:

- add support for multi-tenancy applications (the ability to run more customers on the same database). I am happy to say that we have a solid solution for this.

- add support for multi lingual interfaces so that each user can use a different language.

  We have not achieved that yet, although some techniques have been identified.

- canonise some ad-hoc techniques that we developed during the course of the last year. Server side variables, seeding views with records.

  Some of these techniques tend to be made redundant with each new version of dataflex. But there is still one or two left.

Since both our companies needs a new hour billing systems, we might as well develop that on-the-way to test it all out. We have named it Trex to resemble the sound of "time registration".

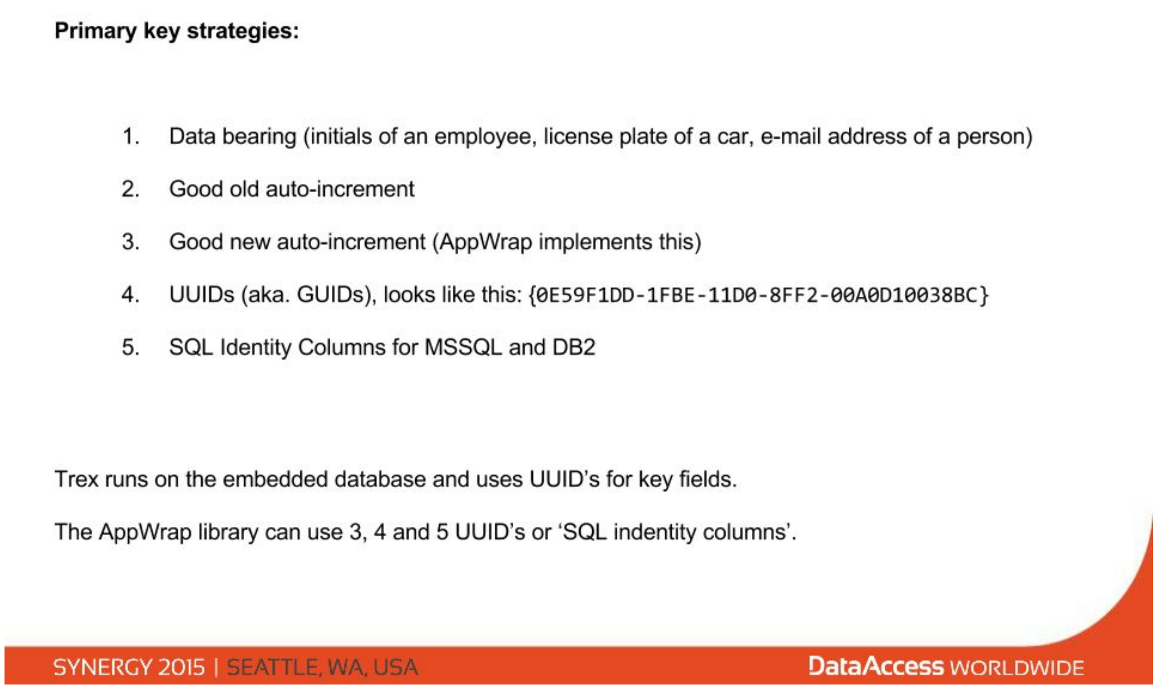
So, AppWrap is the name of the library that we use for new systems and Trex is the name of our hour billing system

Goals to be achieved by doing this:

- We will ourselves be using one of the systems similar to what we are trying to sell to customers and developers.
- If it turns good, we will make the AppWrap library available to the community. Either for direct use or for cherry-picking useful techniques .
- With the 'mobility' of alpha-2 it also means that we will always have an application at hand to show. This will hopefully initiate many interesting conversations with potential customers.

Anyway, that's why we need 'a new Library'. And the library is called AppWrap.

First though:



Primary key strategies:

1. Data bearing (initials of an employee, license plate of a car, e-mail address of a person)
2. Good old auto-increment
3. Good new auto-increment (AppWrap implements this)
4. UUIDs (aka. GUIDs), looks like this: `{0E59F1DD-1FBE-11D0-8FF2-00A0D10038BC}`
5. SQL Identity Columns for MSSQL and DB2

Trex runs on the embedded database and uses UUID's for key fields.

The AppWrap library can use 3, 4 and 5 UUID's or 'SQL indentity columns'.

SYNERGY 2015 | SEATTLE, WA, USA

DataAccess WORLDWIDE

John Tuohy talked some about this and I'll basically repeat what he said.

We have data bearing keys, that the user enters himself.

We have good old auto-increment which means that we have to create an extra field in a system table somewhere.

We have a new auto-increment that AppWrap assigns by opening all the tables an extra time in read-only-mode.

It reads the highest number currently used and adds one for new records. This by the way, is a bad strategy if a guarantee is required, that no ID is ever reused.

We have UUIDs that are good because they do not require an extra server round-trip upon creation.

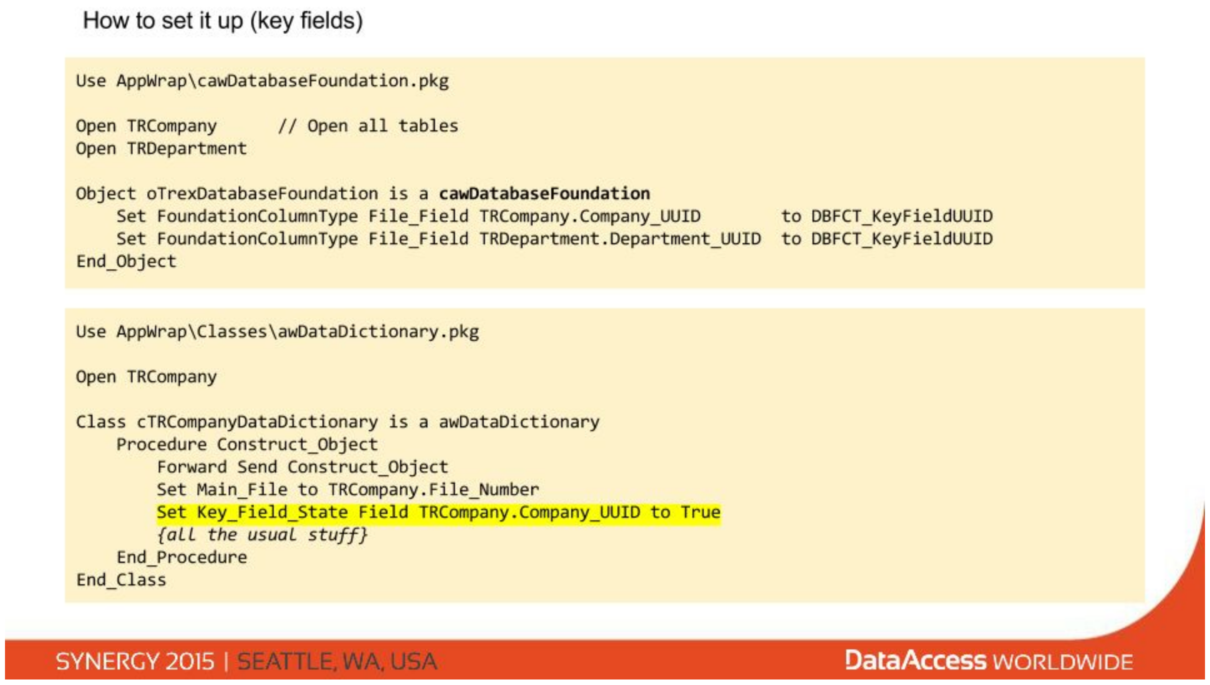
As John said yesterday, there may theoretically be a tree-balancing problem with them, but that remains to be seen.

And they are rather bulky. But who cares.

For Microsoft SQL and DB2 we have the new feature in 18.1 called SQL Identity Columns. This is actually the best strategy because it is compact, and because it does not re-use numbers upon deletion (I suppose).

Trex uses the UUID strategy because it needs to be able to run on the embedded database during development.

AppWrap can be used with any of these strategies but in the context of multi-tenancy only strategy 3, 4, and 5 can be used.



How to set it up (key fields)

```
Use AppWrap\cawDatabaseFoundation.pkg

Open TRCompany      // Open all tables
Open TRDepartment

Object oTrexDatabaseFoundation is a cawDatabaseFoundation
    Set FoundationColumnType File_Field TRCompany.Company_UUID        to DBFCT_KeyFieldUUID
    Set FoundationColumnType File_Field TRDepartment.Department_UUID  to DBFCT_KeyFieldUUID
End_Object
```

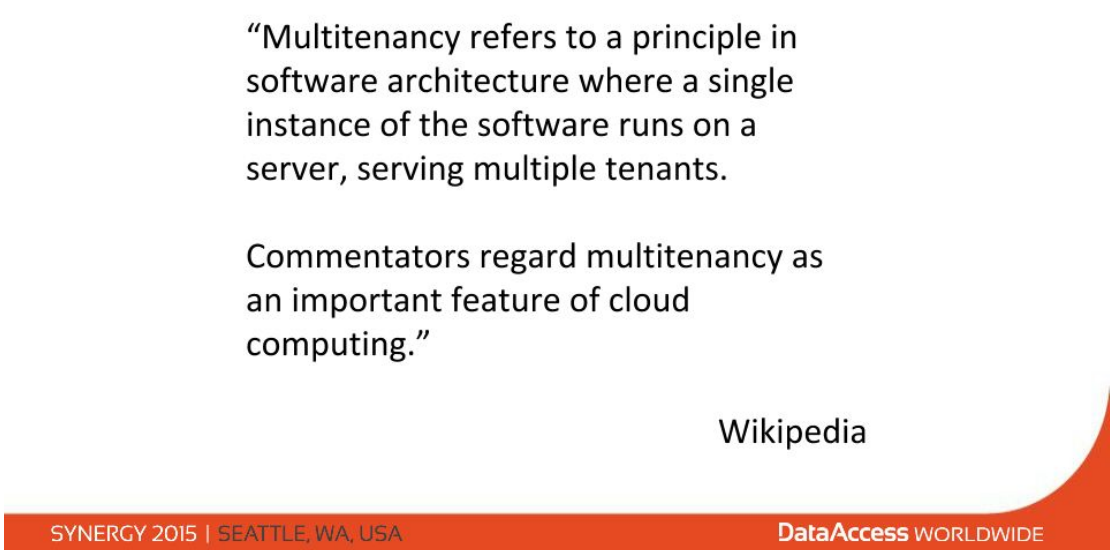
```
Use AppWrap\Classes\awDataDictionary.pkg

Open TRCompany

Class cTRCompanyDataDictionary is a awDataDictionary
    Procedure Construct_Object
        Forward Send Construct_Object
        Set Main_File to TRCompany.File_Number
        Set Key_Field_State Field TRCompany.Company_UUID to True
        {all the usual stuff}
    End_Procedure
End_Class
```

SYNERGY 2015 | SEATTLE, WA, USA

DataAccess WORLDWIDE

A awDataDictionary does not set any properties behind your back. Therefore the “Set Key_Field_State … to True” is needed even if the class has already the information to set it up itself.

With that aside,


"Multitenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple tenants.

Commentators regard multitenancy as an important feature of cloud computing."

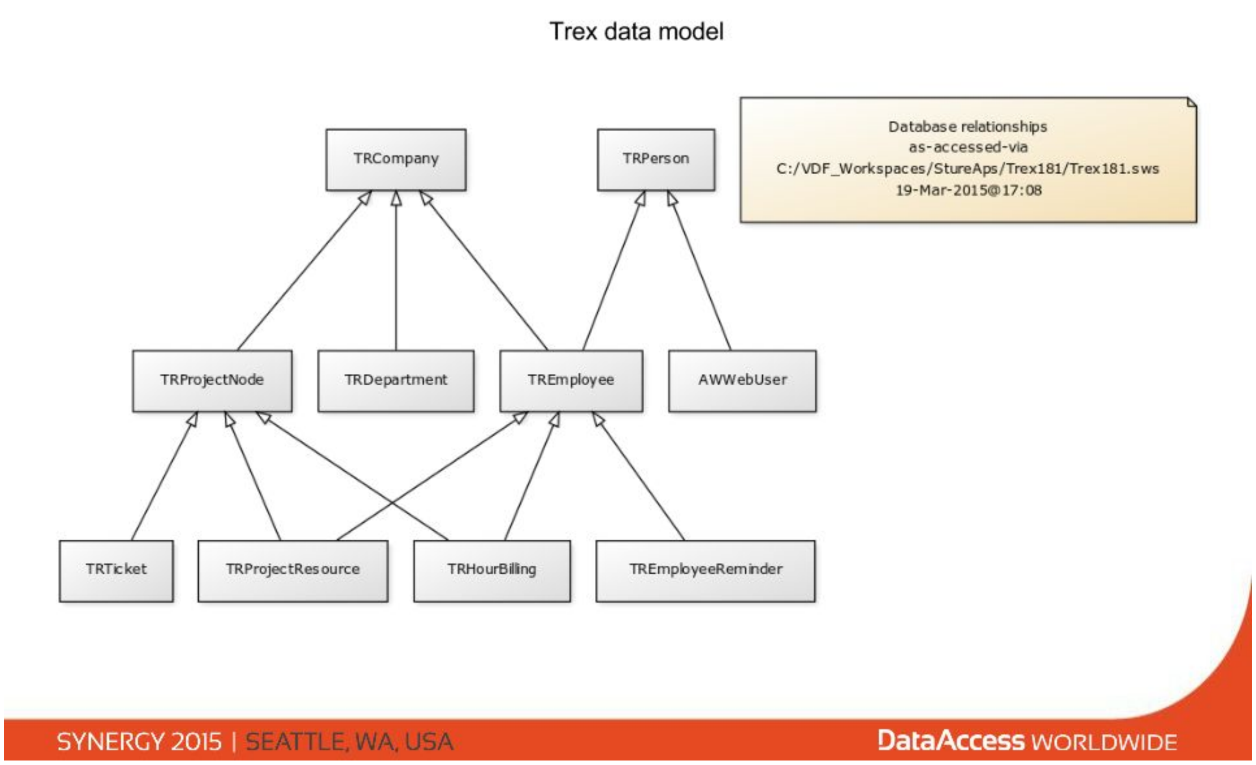
Wikipedia

SYNERGY 2015 | SEATTLE, WA, USA

DataAccess WORLDWIDE

With the AppWrap strategy for multitenancy, this means putting the data of multiple tenants (or: data owners as we call them) into the same table and then trust the application logic to keep these data sets apart.

Trex data model


Trex data model

We can see that Companies have Persons as Employees and Employees can bill hours on Projects and so forth.

Each of these tables have a single-column unique index for primary key (as we just saw) based on UUID type values.

The Company table contains both the company that writes the invoices (we call that the mother company) and then also all the customers of that company.

Trex knows which company is the mother company by storing its ID in a “system table” that can not be seen in this diagram

The other thing not evident from the diagram is the full meaning of the AWWebUser table.

Whenever an application needs to identify a logged-in webuser with an entity in the “data bearing” tables (in this case: the Person table), we introduce an extra table that holds that necessary relation - as well as a user name and password to be used for login.

This is the job of the AWWebUser table.

But the Trex system (or rather: “an AppWrap application”) also has the normal WebAppUser and WebAppSession tables to control the login and session handling.

These are created automatically by the Studio and the WebAppUser table also contains login name and password.

So how do they co-exist?

The datadictionary of the AWWebUser table takes care of synchronizing the content between itself and WebAppUser table.

So, when a login is performed the user-name and password is first checked against the WebAppUser table and then - if correct - against the AWWebUser table. Thereby identifying person, employee and company.

This creates a galvanic separation data-wise, between the login housekeeping and the logic that determines who-you-are-in-the-context-of-the-application.

We want that.

Now, how do one enable this database to run multiple tenantly?

Well, one adds a DataOwner column to each table.

With the AppWrap library, a numeric type is used for dataowner columns.

Now, here comes the thing.

First realize that a primary key - because its value has no 'meaning' - will only ever be used for find EQ operations. That is, when a 'relate' is executed on a child record.

We will not use the primary index for scanning a table for a sequence of records, because the order will be completely arbitrary (and therefore un-optimizable in any way). No meaningful scan can be performed.

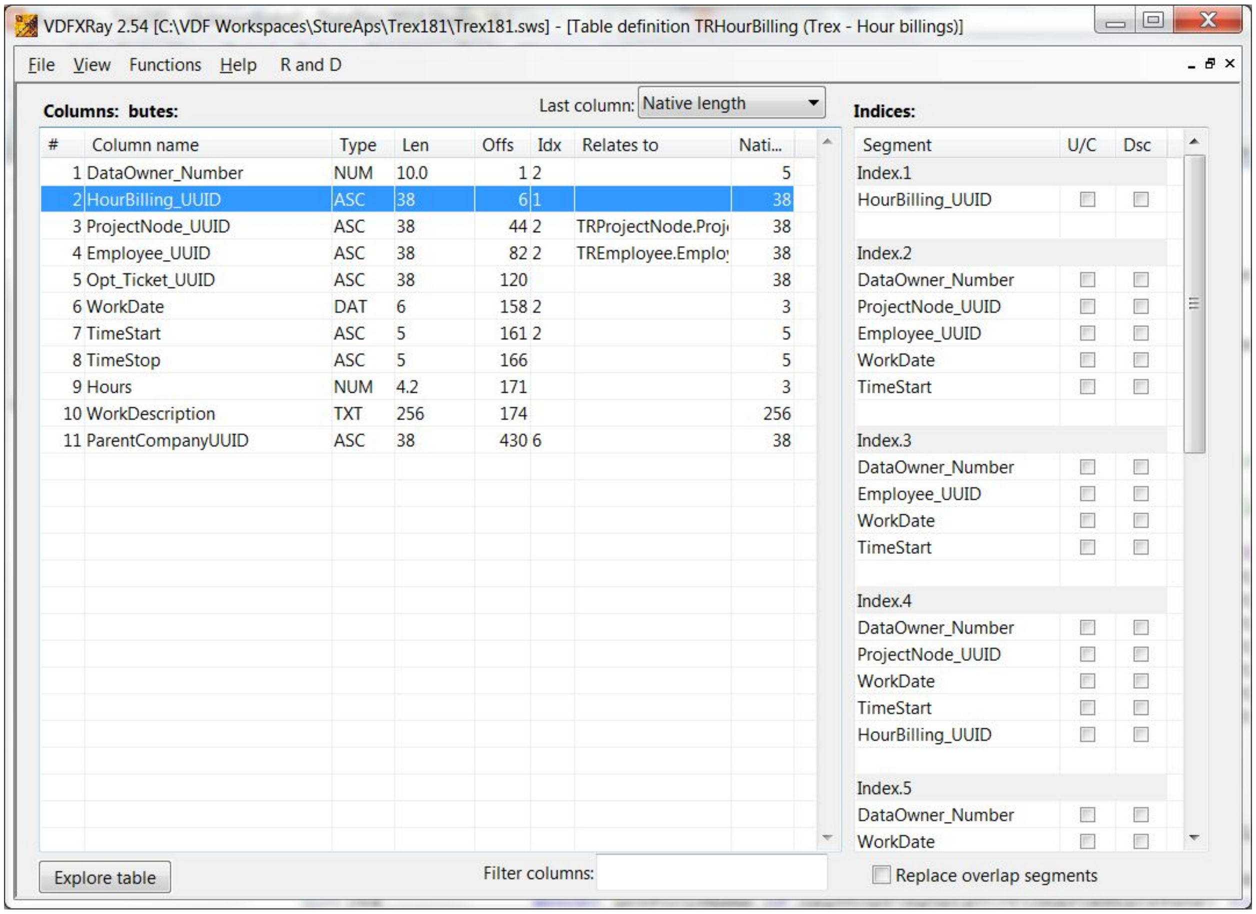
Because we want to **_avoid_** having to use the dataowner column for setting up relations, we require that the primary key be without the this column.

This is the reason why the primary key remains the same, even if we now use the table in a multitenancy environment.

Secondly, every index other than the primary index **_will_** be used for scans. And therefore, to remain efficient - in a multitenant-environment - it **_must_** have the DataOwner column as its first segment.

Take a moment to convince yourself of these 2 facts.

Lets take a look at the HourBilling table of the Trex system.


VDFXRay 2.54 [C:\VDF Workspaces\StureAps\Trex181\Trex181.sws] - [Table definition TRHourBilling (Trex - Hour billings)]
File
View
Functions
Help
R and D
Explore table
Filter columns:
Replace overlap segments

Columns: butes:

Last column: Native length

#	Column name	Type	Len	Offs	Idx	Relates to	Nati...
1	DataOwner_Number	NUM	10.0	1	2		5
2	HourBilling_UUID	ASC	38	6	1		38
3	ProjectNode_UUID	ASC	38	44	2	TRProjectNode.Proj	38
4	Employee_UUID	ASC	38	82	2	TREmployee.Emplo	38
5	Opt_Ticket_UUID	ASC	38	120			38
6	WorkDate	DAT	6	158	2		3
7	TimeStart	ASC	5	161	2		5
8	TimeStop	ASC	5	166			5
9	Hours	NUM	4.2	171			3
10	WorkDescription	TXT	256	174			256
11	ParentCompanyUUID	ASC	38	430	6		38

Indices:

Segment	U/C	Dsc
Index.1		
HourBilling_UUID	☐	☐
Index.2		
DataOwner_Number	☐	☐
ProjectNode_UUID	☐	☐
Employee_UUID	☐	☐
WorkDate	☐	☐
TimeStart	☐	☐
Index.3		
DataOwner_Number	☐	☐
Employee_UUID	☐	☐
WorkDate	☐	☐
TimeStart	☐	☐
Index.4		
DataOwner_Number	☐	☐
ProjectNode_UUID	☐	☐
WorkDate	☐	☐
TimeStart	☐	☐
HourBilling_UUID	☐	☐
Index.5		
DataOwner_Number	☐	☐
WorkDate	☐	☐

Primary key is the HourBilling_UUID column.

All other indices have the DataOwner as the first segment.

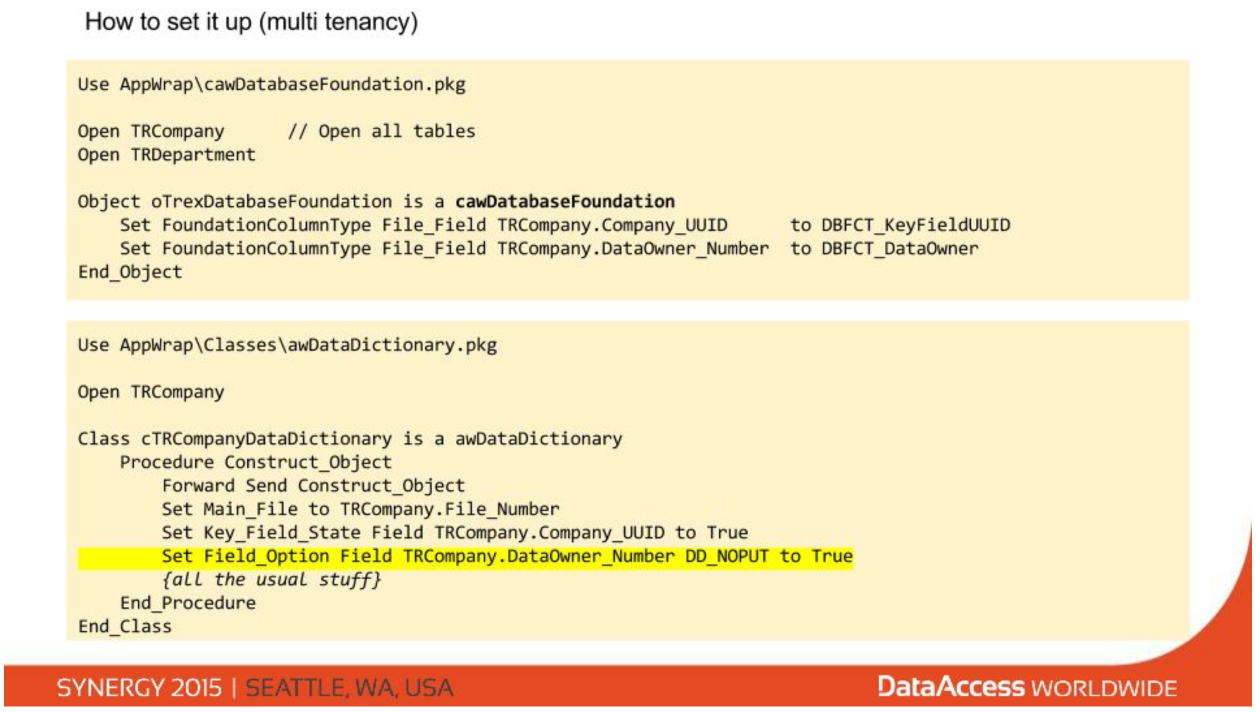
Since we always know which DataOwner we are working with this is _exactly_ as efficient as it would have been, had it not been a multitenant table.

Not a single unnecessary find is performed when following this strategy.

Note also, that the relations to ProjectNode and Employee are based on their single segment primary keys.

How to set up multi tenancy in the DatabaseFoundation

All tables of the system are opened in the database-foundation object. But the slide had only room for the Company table:


How to set it up (multi tenancy)

```
Use AppWrap\cawDatabaseFoundation.pkg

Open TRCompany      // Open all tables
Open TRDepartment

Object oTrexDatabaseFoundation is a cawDatabaseFoundation
    Set FoundationColumnType File_Field TRCompany.Company_UUID      to DBFCT_KeyFieldUUID
    Set FoundationColumnType File_Field TRCompany.DataOwner_Number  to DBFCT_DataOwner
End_Object
```

```
Use AppWrap\Classes\awDataDictionary.pkg

Open TRCompany

Class cTRCompanyDataDictionary is a awDataDictionary
    Procedure Construct_Object
        Forward Send Construct_Object
        Set Main_File to TRCompany.File_Number
        Set Key_Field_State Field TRCompany.Company_UUID to True
        Set Field_Option Field TRCompany.DataOwner_Number DD_NOPUT to True
        {all the usual stuff}
    End_Procedure
End_Class
```

SYNERGY 2015 | SEATTLE, WA, USA

DataAccess WORLDWIDE

A quick tour of Trex

{

- Showed some features of Trex to showcase 10 datadictionaries doing seemles multitenancy-ballets.
- And that multiple owners could not access each others data.
- No reference whatsoever to the fact that we're running in multitenancy in the source code for even a very complex view. Also not so in the source for the datadictionary classes.

}

Now, this multitenancy abstraction operates with two kinds of ‘elevated’ user types:

DataOwner-user-type: Who is allowed to create other users for that data owner.

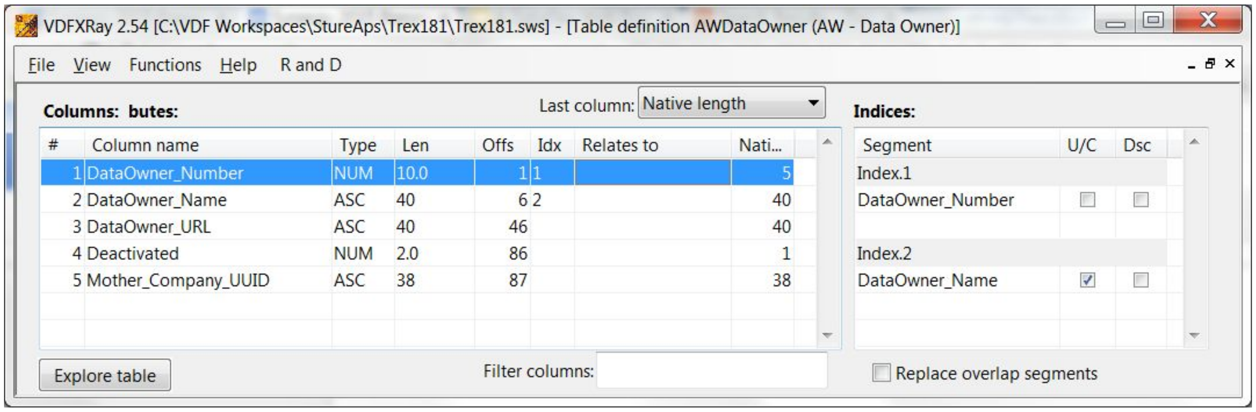
Site Administrator: Is allowed disable and enable data owners.

In addition, AppWrap has a user-rights system that I will not show in detail, but common to all of this is the notion that rights may not necessarily be assigned at login. But once the session is going, the user can elevate him or herself into having these ‘extra’ rights (by reentering their password).

Creating a new data owner

{Showed a sequence of login- and setup-type dialogs to set up a new data owner, ready for use}

The DataOwner table


VDFXRay 2.54 [C:\VDF Workspaces\StureAps\Trex181\Trex181.sws] - [Table definition AWDataOwner (AW - Data Owner)]
File View Functions Help R and D
Explore table

Columns: butes:

Last column: Native length

#	Column name	Type	Len	Offs	Idx	Relates to	Nati...
1	DataOwner_Number	NUM	10.0	1	1		5
2	DataOwner_Name	ASC	40	6	2		40
3	DataOwner_URL	ASC	40	46			40
4	Deactivated	NUM	2.0	86			1
5	Mother_Company_UUID	ASC	38	87			38

Indices:

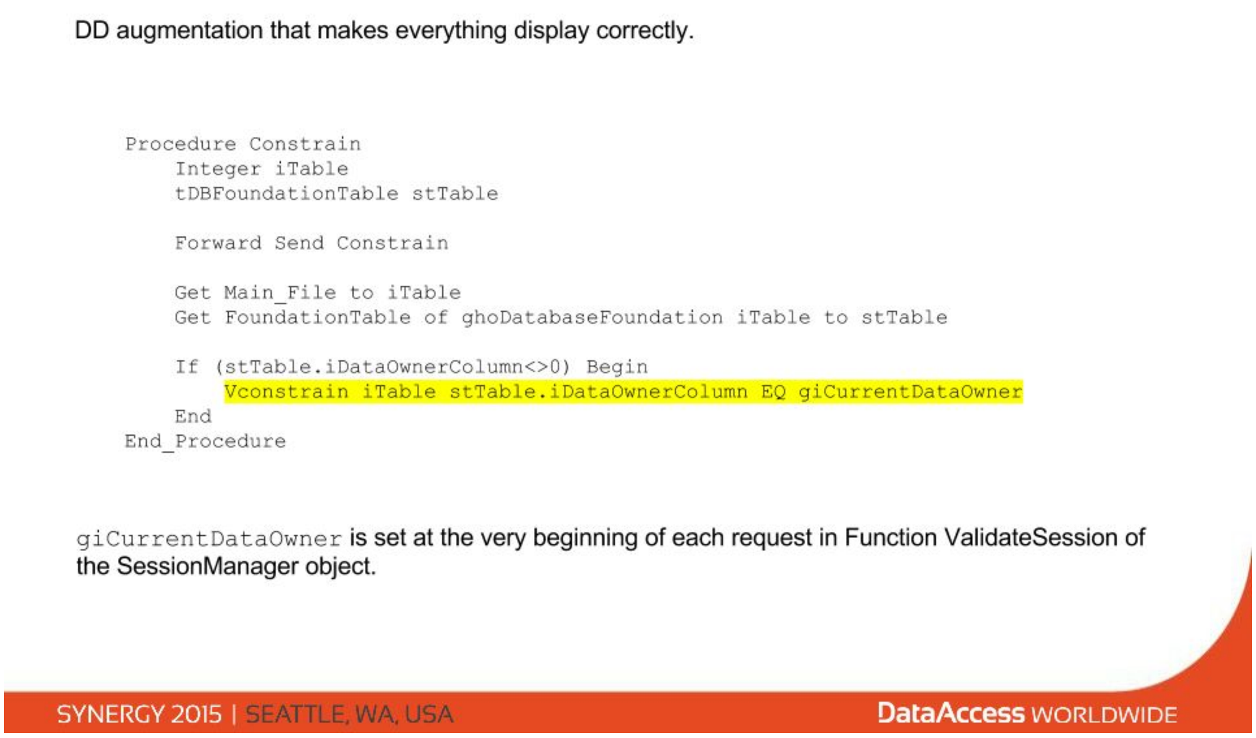
Segment	U/C	Dsc
Index.1		
DataOwner_Number	☐	☐
Index.2		
DataOwner_Name	☑	☐

Filter columns:

☐ Replace overlap segments

- The names of the data owners are stored in this table
- For Trex, it also stores the information about which company is the ‘mother company’ of a data owner.
- So it acts as our ‘system table’, since it has exactly one record per data owner.

How simple it is



DD augmentation that makes everything display correctly.

```
Procedure Constrain
    Integer iTable
    tDBFoundationTable stTable

    Forward Send Constrain

    Get Main_File to iTable
    Get FoundationTable of ghoDatabaseFoundation iTable to stTable

    If (stTable.iDataOwnerColumn<>0) Begin
        Vconstrain iTable stTable.iDataOwnerColumn EQ giCurrentDataOwner
    End
End_Procedure
```

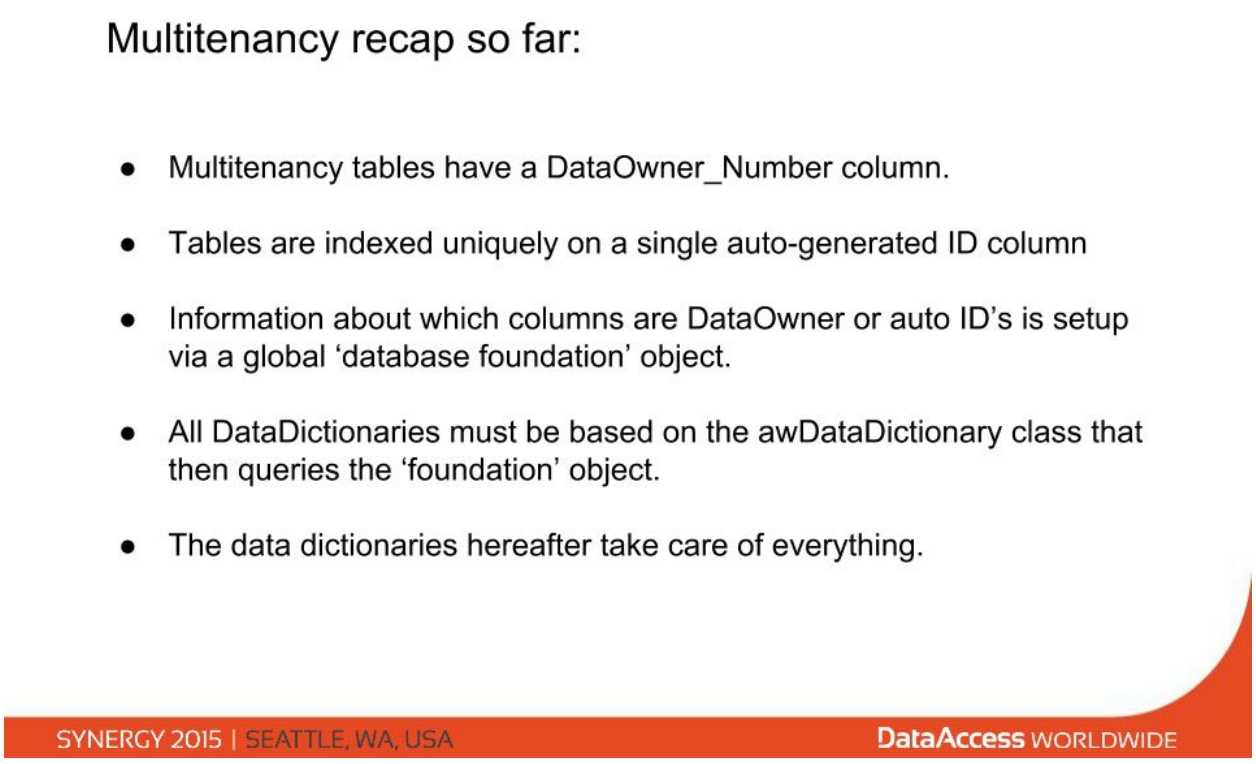
`giCurrentDataOwner` is set at the very beginning of each request in Function ValidateSession of the SessionManager object.

This shows how simple this multitenancy-mechanism really is.

This is all it takes to make that complex view - you saw before - display correctly and **at speed** and never navigate to a record of another data owner.

I am impressed.

There are additional augmentations to ensure that the DD does not save or delete records of other data owners . They are equally simple.

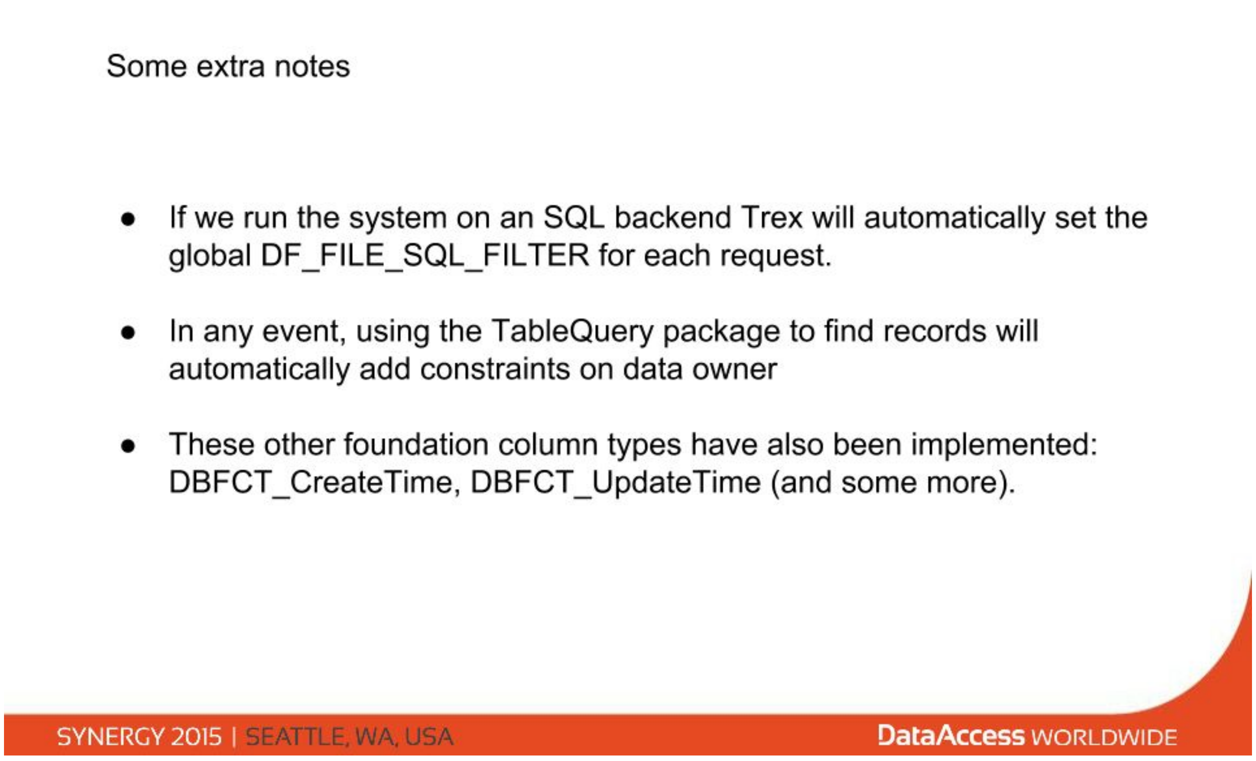


Multitenancy recap so far:

- Multitenancy tables have a DataOwner_Number column.
- Tables are indexed uniquely on a single auto-generated ID column
- Information about which columns are DataOwner or auto ID's is setup via a global 'database foundation' object.
- All DataDictionaries must be based on the awDataDictionary class that then queries the 'foundation' object.
- The data dictionaries hereafter take care of everything.

SYNERGY 2015 | SEATTLE, WA, USA DataAccess WORLDWIDE

Some extra notes


Some extra notes

- If we run the system on an SQL backend Trex will automatically set the global DF_FILE_SQL_FILTER for each request.
- In any event, using the TableQuery package to find records will automatically add constraints on data owner
- These other foundation column types have also been implemented: DBFCT_CreateTime, DBFCT_UpdateTime (and some more).

SYNERGY 2015 | SEATTLE, WA, USA DataAccess WORLDWIDE

Some (externally implemented) check rutines.

Once a multitenant system is running, you would like to do things like:

- Make a list of data owners currently in the system
- Import and export dataowner data
- Check for illegal relations (one dataowner to another)
- Delete a dataowner all together

These functions have been implemented in a general purpose workspace utility called VDFXray.
Question: How does VDFXray know about which columns have been setup for dataowner-id?

Answer: The first time an executable runs after it has been updated (by the compiler or by copying a new exe file into a production environment) it serializes the DatabaseFoundation object and writes it to a file.

This file is automatically picked up by VDFXray when a workspace is opened. Therefore it has the information needed to perform the required operations.

(
Theoretically, a routine could be added to VDFXray that would generate a "clean" version of the WebAppUser table (which is the one that is synchronized with the AWWebUser table)
)

Multi-tenancy end

The presented strategy has proved stable so far, and as you have seen, it draws precious little attention from the developer.

Thank you for your time.

Sture Andersen
Synergy 2015