Optimal use of SQL workers, modular systems for web apps optimization

Sten-Erik Björling

Enviro Data, Luleå, Sweden

s-e.bjorling@enviro.se

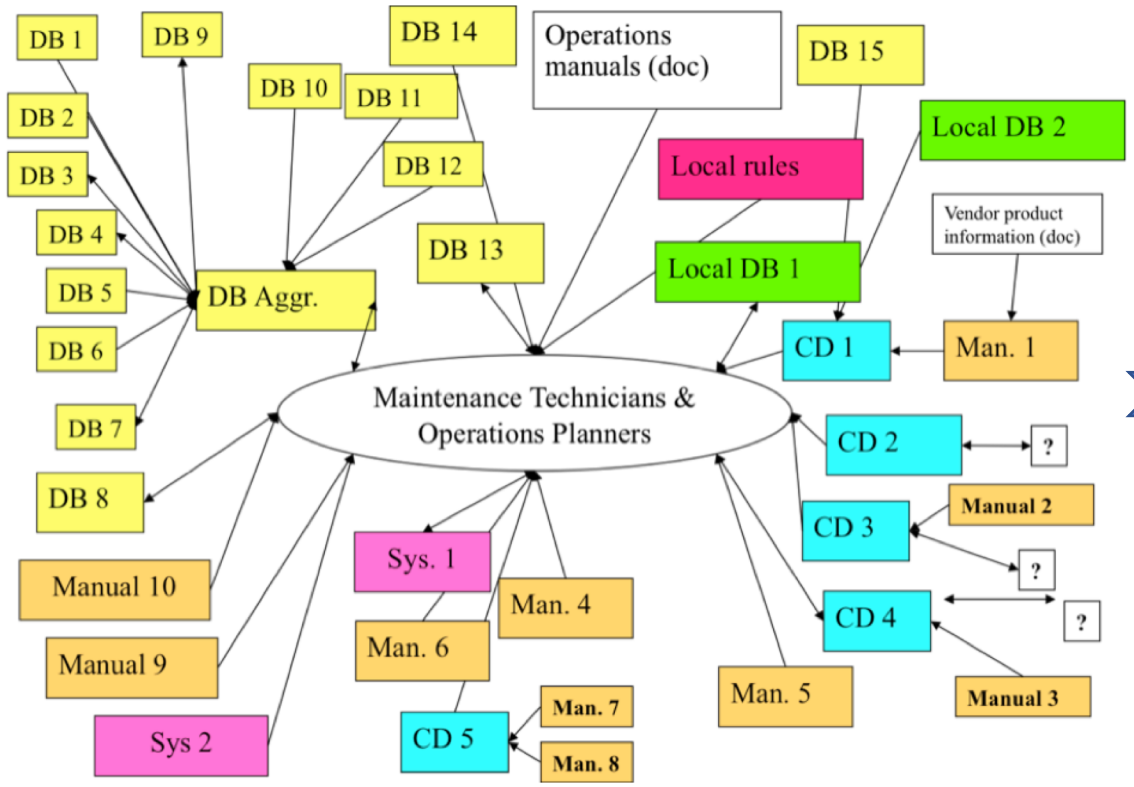
Please note…

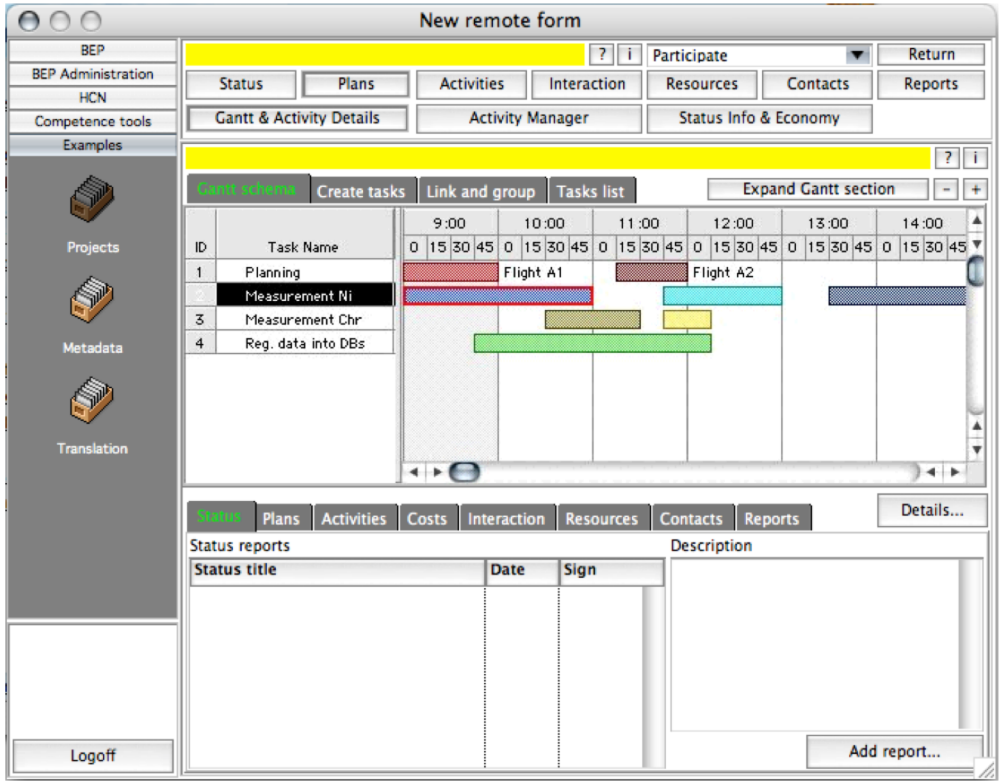
- Due to time constraints – updates to the demonstration libraries and presentation will be stored at GitHub
  - stenerikbjorling
  - Omnis-Studio-Wesel-2018
- When finding the mandatory idiocies – please modify and contribute…

Overview…

- The context -> How to effectively use SQL workers in complex systems
- The strategy -> minimize re-development and long-term maintenance costs…
- The concept -> core optimized infrastructures for current and future implementations…
- Examples! (Use of data and process objects, relatively optimized web apps – including mobile device apps…)
- Issues… ☺

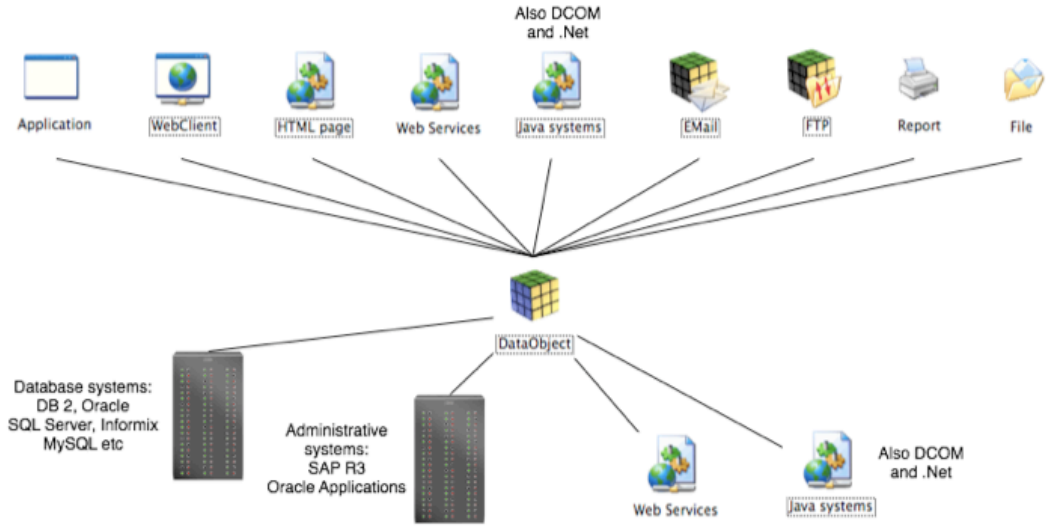
Context -> How to get from A to B…





Strategy -> Omnis Studio as the ”glue” environment… (and ONE main debugger…)

- Data & Process objects acts as adaptor cores for access & distribution of data between different sources & different end-user contexts / devices
- Key component for this is the session pools… Controls the access to relevant data source…
- Session pools allows for optimized settings depending on data access mode…
- Note the VERY efficient support for RESTful, SOAP etc.



Database access… (boring, boring… ☺)

- Centered on data objects…
  - Allows easier change of data-source -> change from DB access to REST etc.
  - Allows more efficient resource use -> use of object references (granular implementation – use no more memory than needed…)
- The ”Data Object Ecosystem”…
  - Data Object – one location to handle the data for an entity…
  - DBStandardOps object – handles standard database operations…
  - SQL Worker Object – manage SQL workers management…
  - Session Object – manages sessions in a sessions pool…

A simple example – DVD Rental access

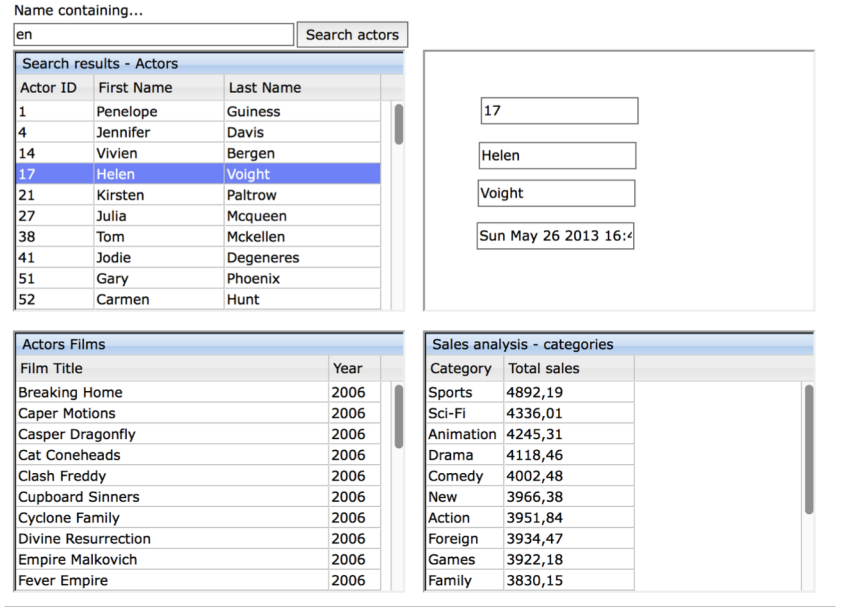
JS form for presenting data on an Actor from the DVD Rental example database (variant of mySQL Sakila).

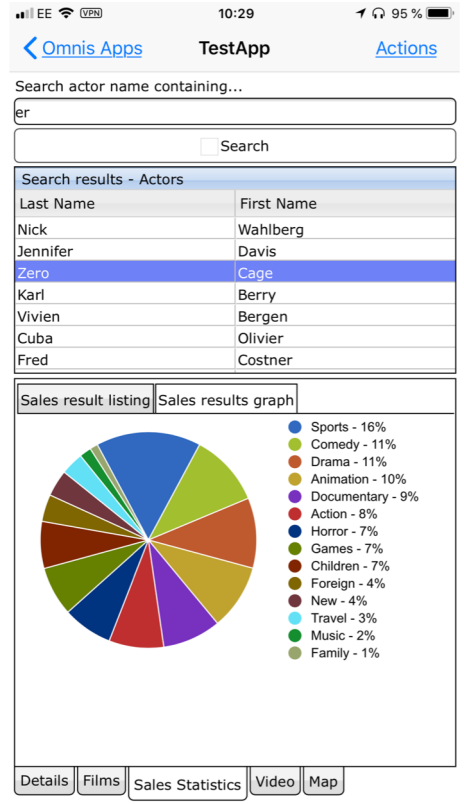
Utilizes four autonomous sub forms and SQL worker objects in combination with push…

Fat client, web form, mobile device app, RESTful server, ultrathin – most of the code in common and readable code!

Look into the demonstration library…

Very simple example – but quite optimal… ☺





Some practical examples…

Look into the example library…

Some performance tips

- Design for hundreds or event thousands of simultaneous users…
- Use object references (and strive to use instantiation as locals…)
- Strive to use SQL Workers… Even when developing fat client systems – will allow for simpler re-use…
- The same for session objects – also allows for optimizing operations on the DB servers (read-only operations faster…)
- Use template approach for SQL objects management – copy 3 times faster than $definefromsqlclass…
- Do not forget to minimize sizes etc. of resources (graphics, icons etc.)

Some performance tips (cont.)

- Omnis not the only component to optimize…
  - Actually quite good – especially when running multiple instances of the application server…
- Apache
  - Modules instead of cgi-bin
  - Note the compile mode – a good idea to re-compile with different settings…
  - Evaluate optimal settings - can give a substantial boost…
- Postgres
  - Activate use of multiple instances parallel use and really work on the memory settings…
  - One example - adaptations to optimize use of SSDs – great gains!!

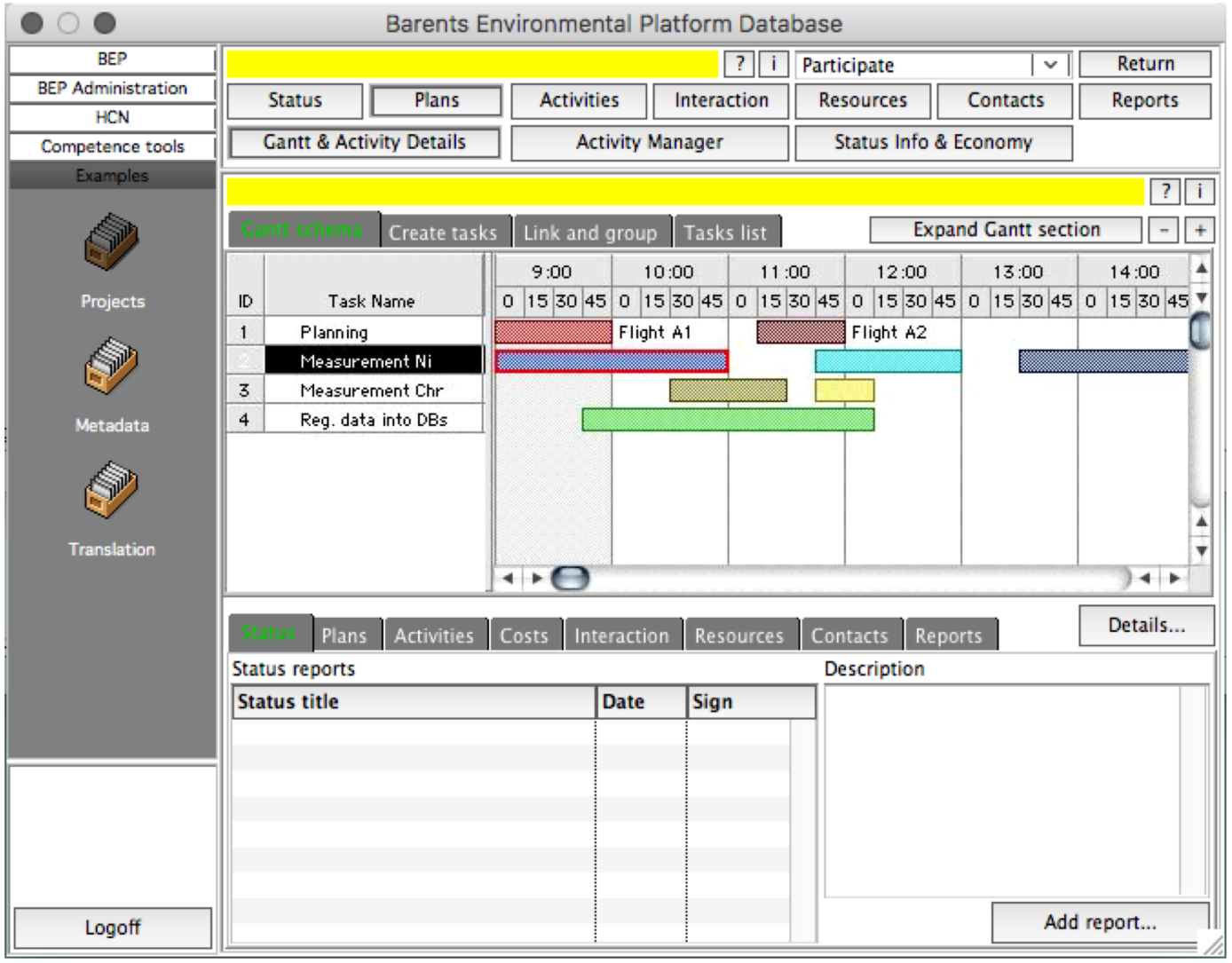
Autonomous sub forms…

- Designed to be used autonomously
  - Their own access to the data access layers…
  - Coordination of messaging between subforms managed by main form...
- Allows development of systems allowing the end-user to adapt their own system… And to use the same forms for desktop and mobile…
- Design should allow for hierarchical subform approaches – subform within a subform etc.
- Complex subforms with data originating from different tables / systems… Reason behind the concept of ”Roles”…

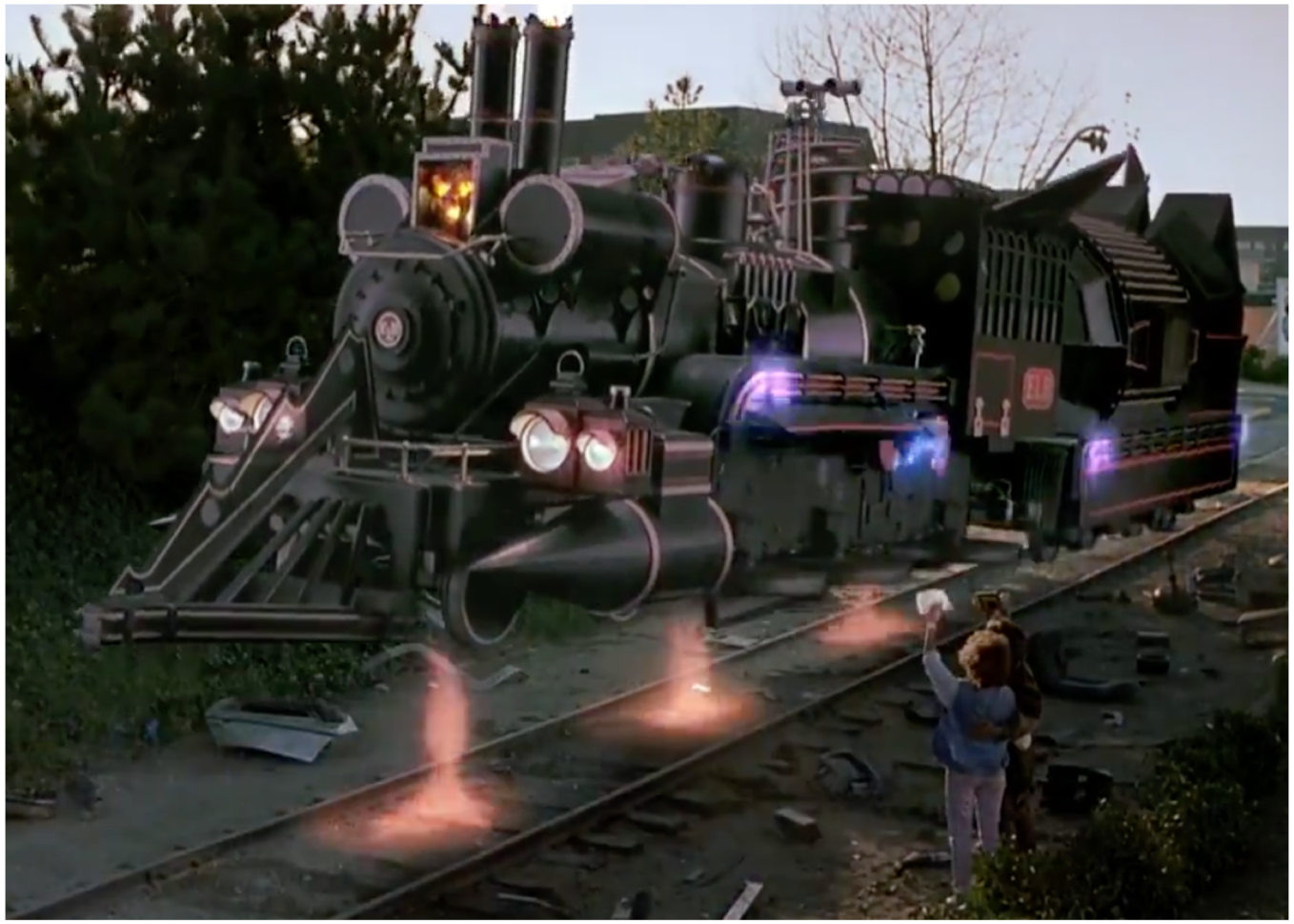
In complex systems a given subform can become very complex – containing paged panes with different subforms – often dynamically assigned… Everyone with their own data access processes running in parallel…

A single subform can contain data and information from several different sources – thus the use of ”Roles” when handling the resultset when it arrives to the subform…

Several SQL worker object sessions can be responsible to deliver data to a given subform or even an object when processing more complex business logic…



I dare you to ask questions!!



Sten-Erik Björling (Stene)
Enviro Data
s-e.bjorling@enviro.se
+46-70-655 11 72

Code / performance screens…
Eventually replaced for new demo library…

One approach!!

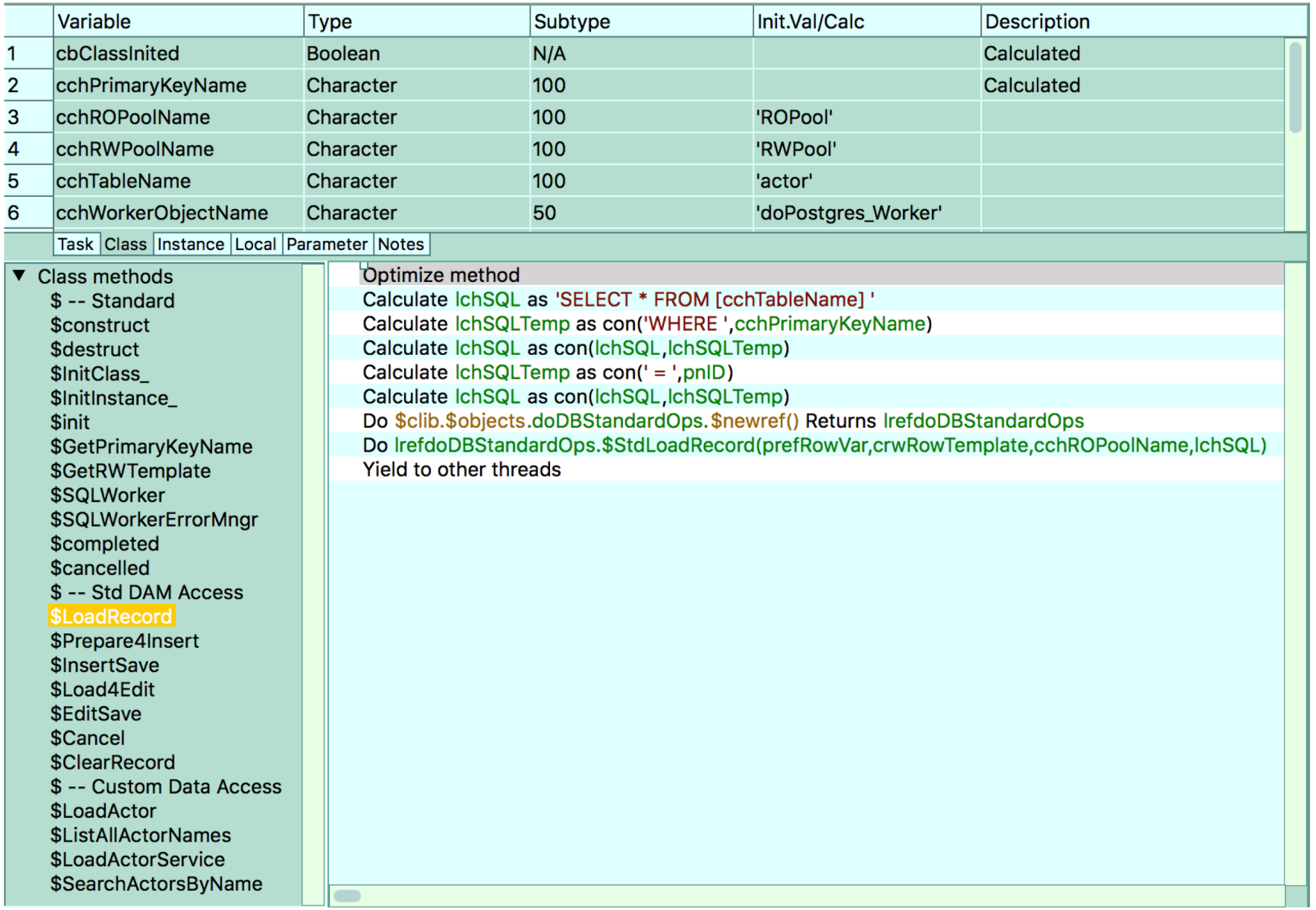
The basic architecture of a data object in the DVD rental example. Kindly note the following:

Vital parameters for the standard operations of the data object defined with class variables.

Close to all standard operations do not need modifications. - Use DB operations support object.

Use of SQL worker objects drastically simplified.

Note the use of locals!



Example of a data access method for Actor using SQL worker access.

Note the parameter for ”Role” – important for autonomous subform operation!

	Variable	Type	Subtype	Init.Val/Calc	Description
1	cbClassInited	Boolean	N/A		Calculated
2	cchPrimaryKeyName	Character	100		Calculated
3	cchROPoolName	Character	100	'ROPool'	
4	cchRWPoolName	Character	100	'RWPool'	
5	cchTableName	Character	100	'actor'	
6	cchWorkerObjectName	Character	50	'doPostgres_Worker'	

Task | Class | Instance | Local | Parameter | Notes

- Class methods
  - $ -- Standard
  - $construct
  - $destruct
  - $InitClass_
  - $InitInstance_
  - $init
  - $GetPrimaryKeyName
  - $GetRWTemplate
  - $SQLWorker
  - $SQLWorkerErrorMngr
  - $completed
  - $cancelled
  - $ -- Std DAM Access
  - $LoadRecord
  - $Prepare4Insert
  - $InsertSave
  - $Load4Edit
  - $EditSave
  - $Cancel
  - $ClearRecord
  - $ -- Custom Data Access
  - $LoadActor
  - $ListAllActorNames
  - $LoadActorService
  - $SearchActorsByName

```
Calculate ichResultRole as pchResultRole
Set reference irefCallingForm to prefCallingObject

Calculate lrwParams as crwWOParamsNoBindTempl

Calculate lchSrchTmp as con("'%",pchSrchStr,"%'")

Calculate lchSQL as con('SELECT actor_id, first_name, last_name FROM actor WHERE first_name LIKE ',lchSrchTmp)
Calculate lchSQL as con(lchSQL,' OR last_name LIKE ',lchSrchTmp)

Calculate lrwParams.poolname as cchROPoolName
Calculate lrwParams.query as lchSQL

Do method $SQLWorker (lrwParams,$cinst)

Quit method
```

Performance analysis covers the use of the JS form DVD Rental example.

Both the Omnis system and the Postgres DB on the same computer…

MacBook Pro, 16 GB RAM, i7 processor, Postgres 9.6.3…

All data access is done using SQL Worker approach.

Note the timescale – MILLISECONDS.

Aggregated time includes operation at the Postgres database.

Conclusion? The Omnis layer in the total system is comparably thin compared with DB processing and latencies in networks and web servers…

Benchmark Result Sort: Percent (Ascending) ☐ Method Summary

Item	Percent	Total (ms)	Calls	Avg (ms)	Min (ms)	Max (ms)
Overall	99.99	939.608	1665	20.88	0	8.256
DVD Rental 8_0x	99.99	939.608	1665	20.88	0	8.256
rfActorsOverviewMain	21.36	200.699	45	4.46	3.984	8.256
$UpdateSubForms	21.36	200.699	45	4.46	3.984	8.256
doPostgres_Worker	13.19	123.933	540	0.918	.003	2.232
$completed	12.59	118.342	135	0.877	0.629	2.232
$setCallingInst	0.38	3.542	135	.026	.021	0.102
$construct	0.15	1.409	135	.01	.007	0.126
$destruct	.07	0.64	135	.005	.003	.03
doFilm_Actor	13.07	122.809	270	2.729	0	2.128
$CompactListActorFilms	5.7	53.528	45	1.19	1.006	2.128
$SQLWorker	3.85	36.203	45	0.805	0.652	1.315
$completed	3.18	29.853	45	0.663	0.549	1.038
$construct	0.32	2.972	45	.066	.045	0.141
$destruct	.02	0.232	45	.005	.004	.01
$InitInstance_	0	.021	45	0	0	.008
doActor	12.75	119.866	270	2.664	.001	1.856
$LoadActor	5.07	47.604	45	1.058	0.911	1.856
$completed	3.73	35.086	45	0.78	0.647	1.456
$SQLWorker	3.61	33.93	45	0.754	0.641	1.205
$construct	0.3	2.822	45	.063	.056	0.128
$destruct	.04	0.379	45	.008	.005	.044
$InitInstance_	0	.045	45	.001	.001	.001
poSalesAnalysis	11.73	110.258	270	2.45	0	1.878
$SalesByFilmCategory	4.34	40.75	45	0.906	0.755	1.367
$completed	3.87	36.394	45	0.809	0.659	1.878
$SQLWorker	3.2	30.107	45	0.669	0.569	1.076
$construct	0.28	2.655	45	.059	.041	0.112
$destruct	.04	0.335	45	.007	.005	.033
$InitInstance_	0	.017	45	0	0	.009
rfActorsFilmsListing	9.73	91.388	90	2.031	0.419	2.62
$ActorFilmsListing	7.34	68.965	45	1.533	1.297	2.62
$completed	2.39	22.423	45	0.498	0.419	0.797
rfActorDetails	9.42	88.526	90	1.967	0.471	2.407
$LoadActorDetails	6.7	62.932	45	1.398	1.229	2.407
$completed	2.72	25.594	45	0.569	0.471	1.141
rfSalesAnalysis	8.74	82.129	90	1.825	0.492	1.836
$SalesAnalysisResults	5.84	54.902	45	1.22	0.998	1.836
$completed	2.9	27.227	45	0.605	0.492	1.462

The anatomy of an autonomous subform…

The subform is responsible for all its own access to data and business process management.

The integration of the subform into different contexts is quite simple…

Since push is implemented the different subforms in a system gets refreshed once the data arrives – demands manual control of redraws if complex queries are made at servers…

Note the use of “Roles” in the $completed method