„Arctica“?

Xbase++ meets PostgreSQL

„The world is not black and white - reality is always grey“

Overview

- Xbase++ the language
- Meeting points with PostgreSQL
- Pass-Through-SQL (P-SQL)
- PostgreSQL does ISAM and more!
- Xbase++ as a Server-Side-Language
- Into the future...
- Summary

Xbase++ the platform

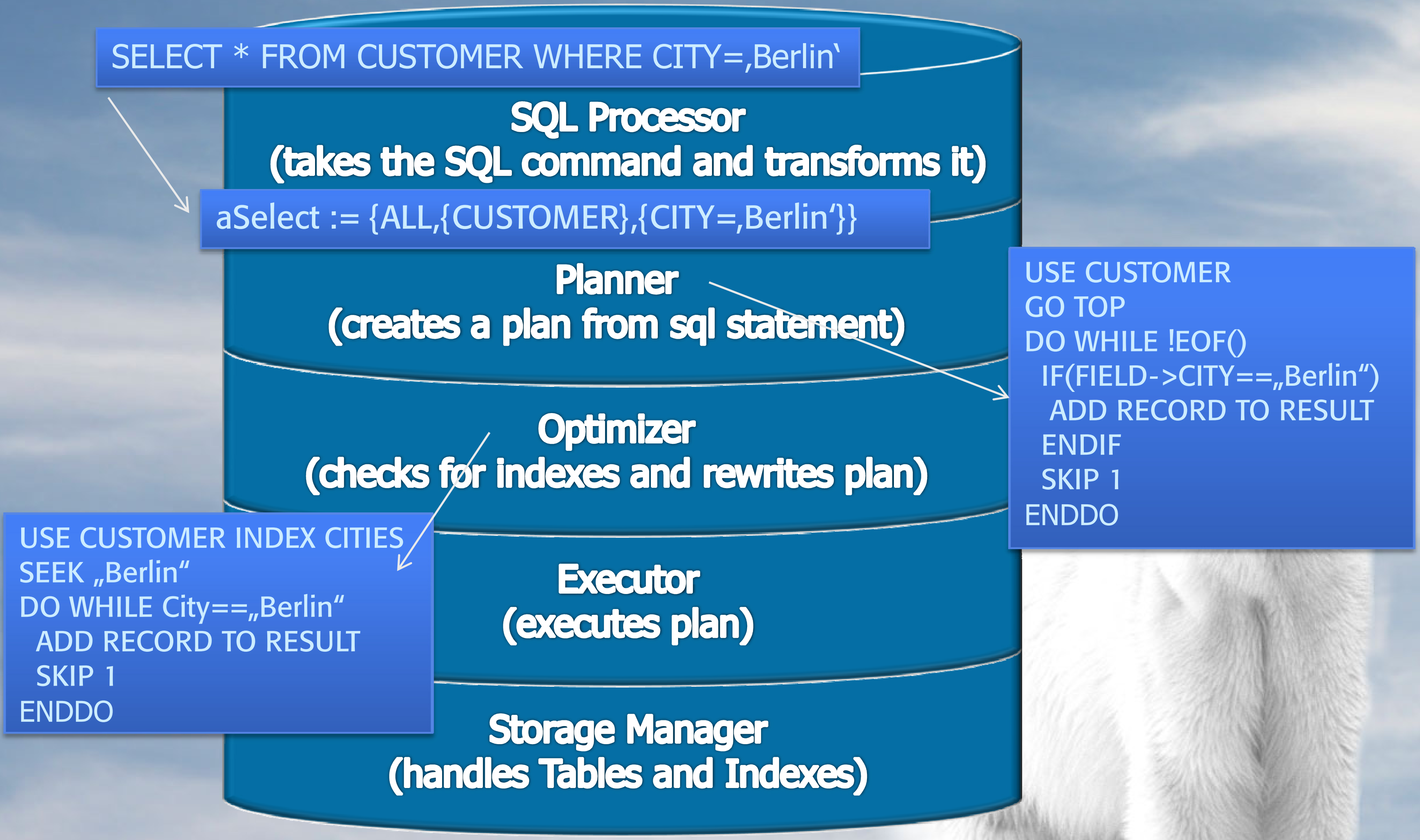
- 4GL language, 100% Clipper compatible
- With asynchronous garbage collection, automatic multi threading – no deadlock no need to sync. resource access
- OOP engine with multiple inheritance, dynamic class/object creation and full object persistence
- Lamba expressions aka. Lisp Codeblocks over the wire
- Full dynamic loadable features and copy deployment
- The language and runtime itself introduce no-limits!
- Very powerfull preprocessor to add DSL constructs!
- Supports development of Console/Textmode, Graphical User-Interface, Service, CGI, Web-Application-Server and Compiled-Web-Page application types
- Has its roots on the OS/2 platform
- Currently focuses onto the Windows 32Bit and 64 Bit Operating-Systems
- Hybrid Compiler, generates native code and VM code.

Data-Access in Xbase++ & xBase

- Rule-1: There is no impedance mismatch
  - Any table datatype is a valid language datatype this includes operators and expressive behaviour and NIL-State / NULL-behaviour
  - Only tables introduce fixed-typing, the language itself must be dynamic typed to adapt automatically to schema changes
- Rule-2: The language is the database and vice versa
  - Any expression of the development language is a valid expression on behalf of the database/table (index-key-expr., functions, methods)
- Rule-3: Isolation semantics of the database and language are the same
  - Different application access a remote data source and different threads accessing the same field/column are underlying the same isolation principles.

xBase is ISAM, data-access is done based on the navigational pradigma! How does xBase therefore fit into the world of SQL?

Zooming into a SQL Server


SQL Processor
(takes the SQL command and transforms it)
Planner
(creates a plan from sql statement)
Optimizer
(checks for indexes and rewrites plan)
Executor
(executes plan)
Storage Manager
(handles Tables and Indexes)

```
SELECT * FROM CUSTOMER WHERE CITY=‚Berlin‛
```

```
aSelect := {ALL,{CUSTOMER},{CITY=‚Berlin‘}}
```

```
USE CUSTOMER
GO TOP
DO WHILE !EOF()
  IF(FIELD->CITY==„Berlin“)
   ADD RECORD TO RESULT
  ENDIF
  SKIP 1
ENDDO
```

```
USE CUSTOMER INDEX CITIES
SEEK „Berlin“
DO WHILE City==„Berlin“
  ADD RECORD TO RESULT
  SKIP 1
ENDDO
```

Warning: This is a simplified illustration some details are omitted -☺

Comparing DML!

With xBase commands you express **how** you get it. The what is „hidden" in your logic and coding-style. Your code depends on all details of your data model.

```
USE CUSTOMER INDEX CITIES
SEEK „Berlin"
DO WHILE City==„Berlin"
  ADD RECORD TO RESULT
  SKIP 1
ENDDO
```

```
USE CUSTOMER
GO TOP
DO WHILE !EOF()
  IF(FIELD->CITY==„Berlin")
   ADD RECORD TO RESULT
  ENDIF
  SKIP 1
ENDDO
```

In SQL you express **what** you want! The how is determined dynamically depending in your concrete data-model and its state.

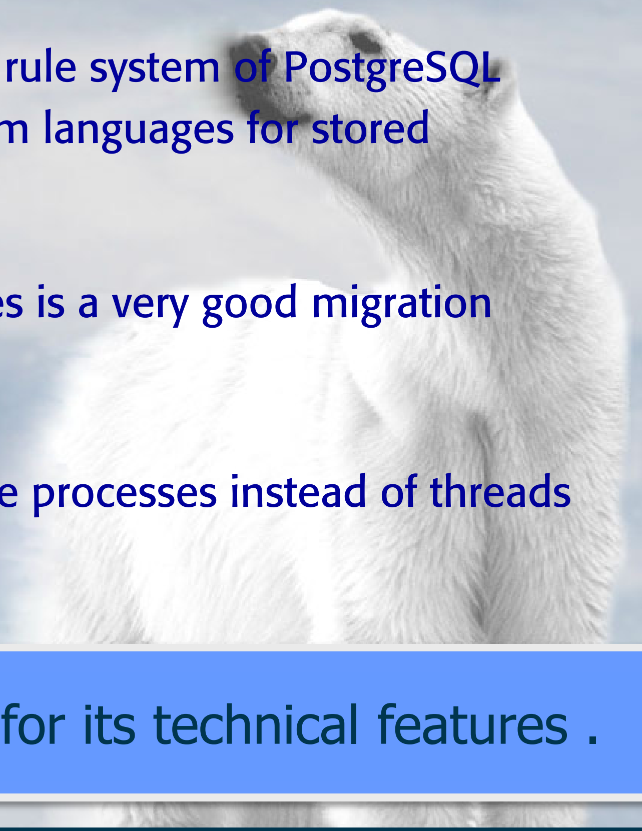
```
SELECT * FROM CUSTOMER WHERE CITY=‚Berlin'
```

Comparing apples with ....

- SQL
  - limited to data access and data definition (DDL, DML)
  - using SQL means dealing with the impedance mismatch between SQL as the server language and XYZ as the client language
  - **due to its client/server nature and the SQL abstraction it scales relatively well**
- Xbase++
  - is a multi-purpose language centered around database-application development. It goes from data-access to the user-interface, formatting and validation.
  - due to its ISAM approach it can reach very high transaction rates (>20.000 „TPS“). If queries are hard-coded, local queries can be blasting fast.
  - **suffers scaleability, reliability and abstraction due to its ISAM nature and file-access-pattern**.

Why PostgreSQL

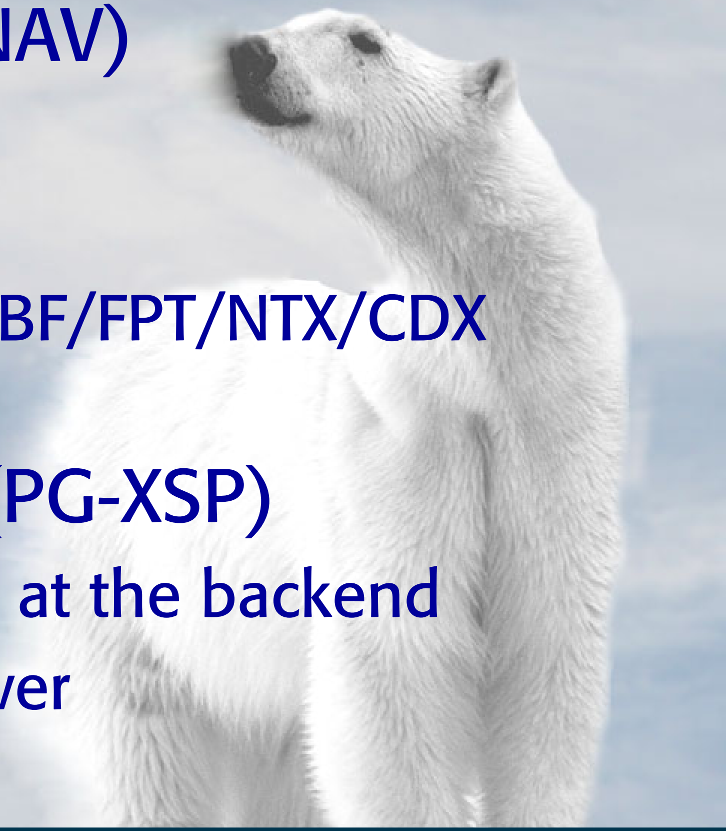
- Philosopical:
  - We don‘t like compromises and we also use the term propaganda for sales-related activites
  - We are a technical driven company and value our customers needs always first. But we allow us to educate our customers and do not support all the hype‘s out there!
- Technical wise:
  - We like the early idea of Active-Databases and the rule system of PostgreSQL
  - Ability to add user-defined-types and to add custom languages for stored procedures
  - Good SQL92/99 compliance, specifically with 8.3
  - With 8.3 the inclusion of FTS with GIST/GIN indexes is a very good migration argument
- In addition:
  - With respect to multi-core and scaleability: multiple processes instead of threads is a plus!
  - Many books, even for beginners are out there



Bottom-Line: We have chosen PostgreSQL for its technical features .

Meeting points with PostgreSQL

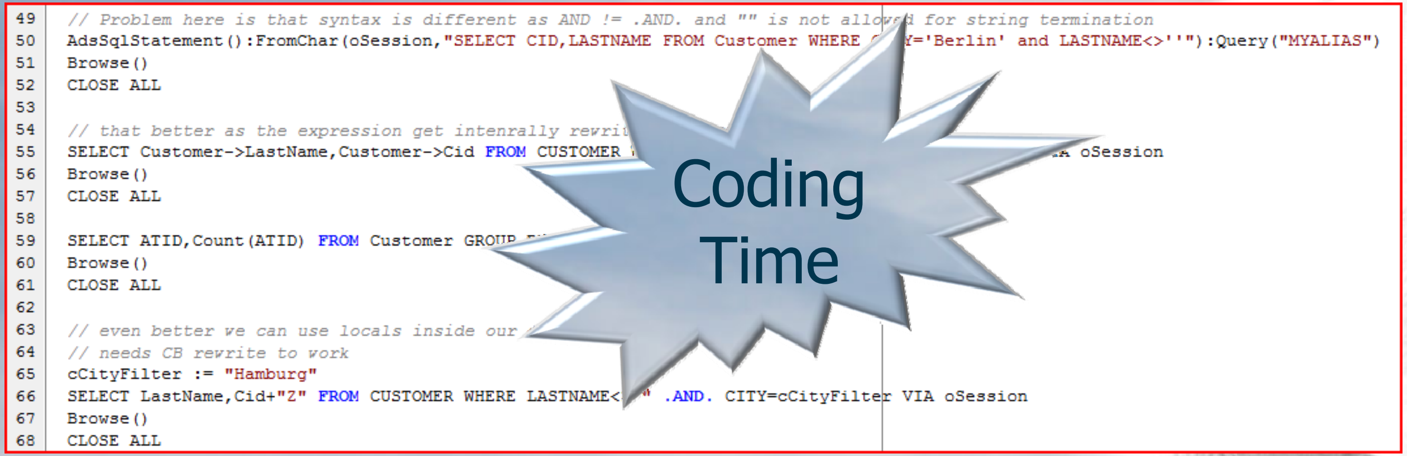
- Pass-Through-SQL (PG-PAS)
  - SELECT, INSERT, UPDATE,DELETE... as first class language element
  - PostgreSQL specific Datatypes are supported
- Navigational-Data-Access (PG-NAV)
  - Navigational data-access
  - Index, Filter, Scope support
  - 100% compatible behaviour as DBF/FPT/NTX/CDX DatabaseEngines
- Stored-Procedures in Xbase++(PG-XSP)
  - 100% Xbase++ language support at the backend
  - Code migration from client to server



Pass-Through SQL

Pass-Through SQL

- SQL Statements supported:
  - SELECT, INSERT, UPDATE and DELETE
- Specific features are:
  - Automated syntax adaption
  - LOCAL and STATIC variables can be used in SQL statements
- In fact SQL Pass-Through-Statements are first class language statements, lets see…

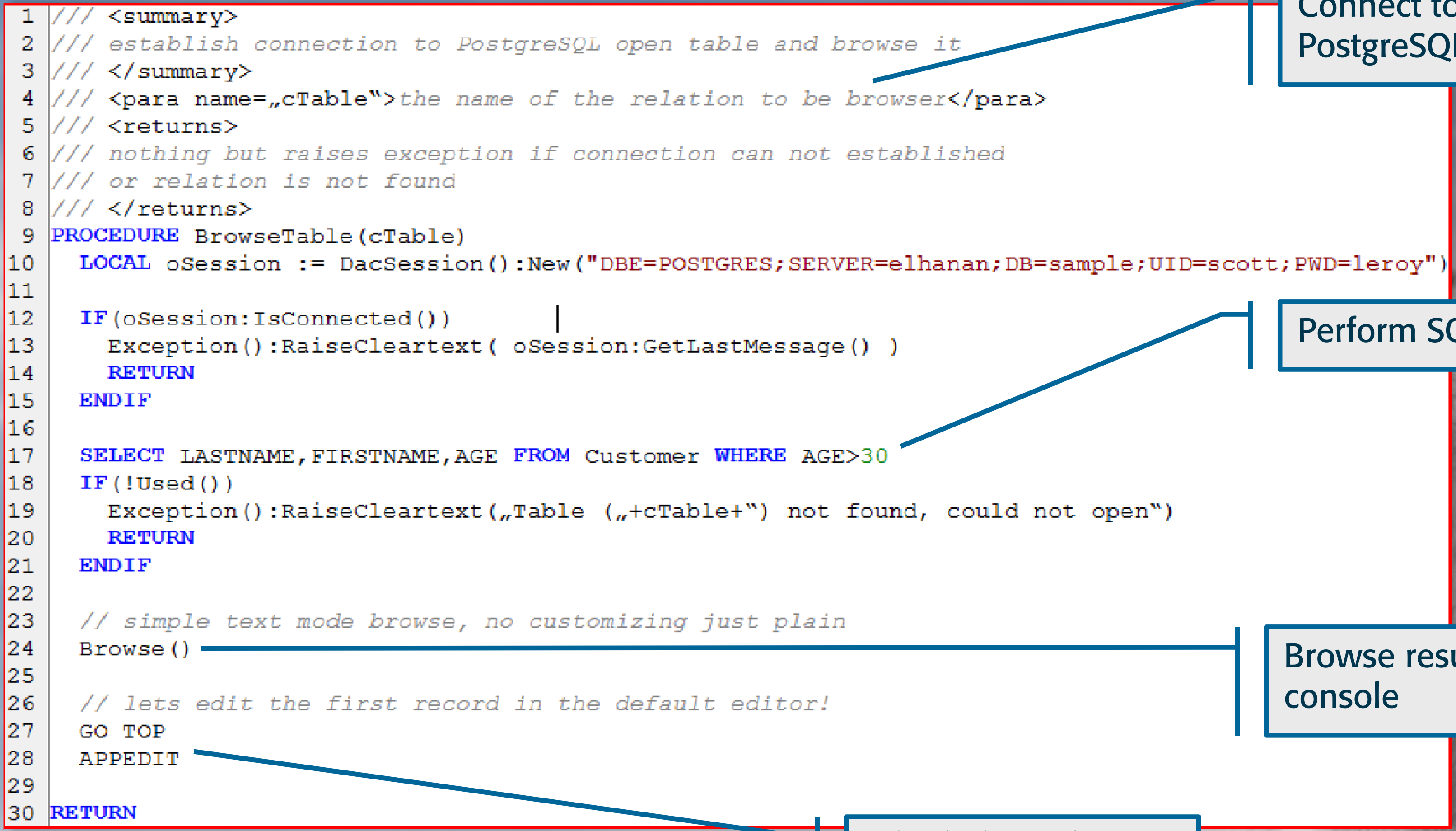


```
// Problem here is that syntax is different as AND != .AND. and "" is not allo[illegible] for string termination
AdsSqlStatement():FromChar(oSession,"SELECT CID,LASTNAME FROM Customer WHERE [illegible]Y='Berlin' and LASTNAME<>''"):Query("MYALIAS")
Browse()
CLOSE ALL

// that better as the expression get intenrally rewrit[illegible]
SELECT Customer->LastName,Customer->Cid FROM CUSTOMER [illegible] oSession
Browse()
CLOSE ALL

SELECT ATID,Count(ATID) FROM Customer GROUP [illegible]
Browse()
CLOSE ALL

// even better we can use locals inside our [illegible]
// needs CB rewrite to work
cCityFilter := "Hamburg"
SELECT LastName,Cid+"Z" FROM CUSTOMER WHERE LASTNAME<[illegible]" .AND. CITY=cCityFilter VIA oSession
Browse()
CLOSE ALL
```

Coding Time

Pass-Through SQL (sample-1)

Pass-Through SQL



```
/// <summary>
/// establish connection to PostgreSQL open table and browse it
/// </summary>
/// <para name=„cTable“>the name of the relation to be browser</para>
/// <returns>
/// nothing but raises exception if connection can not established
/// or relation is not found
/// </returns>
PROCEDURE BrowseTable(cTable)
  LOCAL oSession := DacSession():New("DBE=POSTGRES;SERVER=elhanan;DB=sample;UID=scott;PWD=leroy")

  IF(oSession:IsConnected())
    Exception():RaiseCleartext( oSession:GetLastMessage() )
    RETURN
  ENDIF

  SELECT LASTNAME,FIRSTNAME,AGE FROM Customer WHERE AGE>30
  IF(!Used())
    Exception():RaiseCleartext(„Table („+cTable+“) not found, could not open“)
    RETURN
  ENDIF

  // simple text mode browse, no customizing just plain
  Browse()

  // lets edit the first record in the default editor!
  GO TOP
  APPEDIT

RETURN
```

Connect to PostgreSQL server

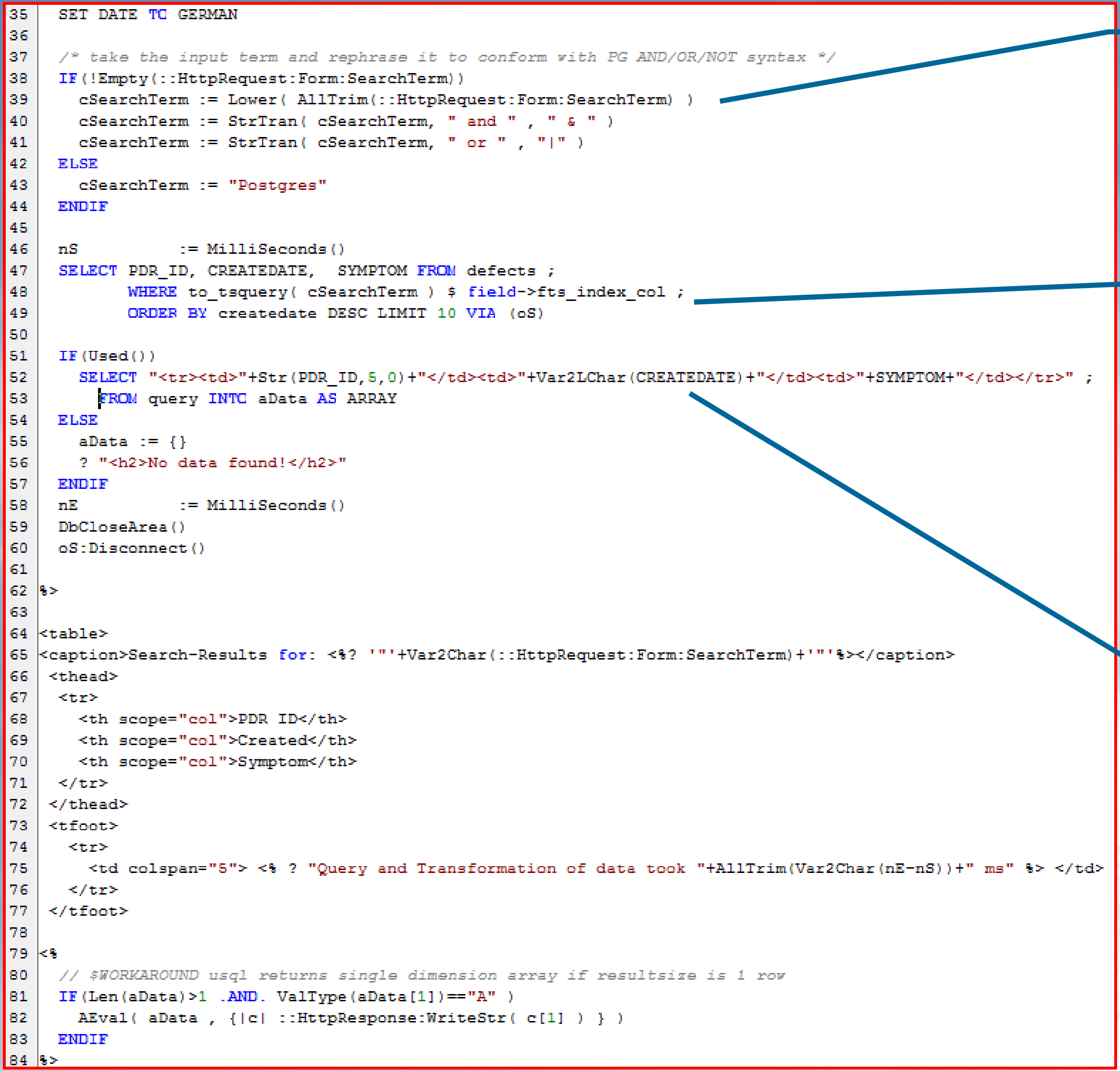
Perform SQL SELECT

Browse result in console

Edit dedicated record

Pass-Through SQL (sample-2)

Pass-Through SQL



```
  SET DATE TC GERMAN

  /* take the input term and rephrase it to conform with PG AND/OR/NOT syntax */
  IF(!Empty(::HttpRequest:Form:SearchTerm))
    cSearchTerm := Lower( AllTrim(::HttpRequest:Form:SearchTerm) )
    cSearchTerm := StrTran( cSearchTerm, " and " , " & " )
    cSearchTerm := StrTran( cSearchTerm, " or " , "|" )
  ELSE
    cSearchTerm := "Postgres"
  ENDIF

  nS          := MilliSeconds()
  SELECT PDR_ID, CREATEDATE,  SYMPTOM FROM defects ;
         WHERE to_tsquery( cSearchTerm ) $ field->fts_index_col ;
         ORDER BY createdate DESC LIMIT 10 VIA (oS)

  IF(Used())
    SELECT "<tr><td>"+Str(PDR_ID,5,0)+"</td><td>"+Var2LChar(CREATEDATE)+"</td><td>"+SYMPTOM+"</td></tr>" ;
      FROM query INTC aData AS ARRAY
  ELSE
    aData := {}
    ? "<h2>No data found!</h2>"
  ENDIF
  nE          := MilliSeconds()
  DbCloseArea()
  oS:Disconnect()

%>

<table>
<caption>Search-Results for: <%? '"'+Var2Char(::HttpRequest:Form:SearchTerm)+'"'%></caption>
 <thead>
  <tr>
    <th scope="col">PDR ID</th>
    <th scope="col">Created</th>
    <th scope="col">Symptom</th>
  </tr>
 </thead>
 <tfoot>
   <tr>
     <td colspan="5"> <% ? "Query and Transformation of data took "+AllTrim(Var2Char(nE-nS))+" ms" %> </td>
   </tr>
 </tfoot>

<%
  // $WORKAROUND usql returns single dimension array if resultsize is 1 row
  IF(Len(aData)>1 .AND. ValType(aData[1])=="A" )
    AEval( aData , {|c| ::HttpResponse:WriteStr( c[1] ) } )
  ENDIF
%>
```

Transform a search term to conform with FTS syntax.

Just use the local variable with the search term in your PostgreSQL select command. The IN operator is used to match FTS

Client side Universal SQL is used to transform the PG result-set into HTML code according to the client locale settings

How does Pass-Through SQL work?

SQL Language statement gets broken down into a sequence of methods (grammar is transparent as the PreProcessor does this)

↓

The method chain gets executed and uses callbacks of the DatabaseEngine to rewrite the SQL syntax depending on the PostgreSQL specific needs

↓

Expressions are codeblocks, the are now executed by the SafeExecutionEngine.
(This step removes all sideeffects from a expression, in other words it detaches the expression from the process)

↓

Expression Identifiers, Constants and Operators get rewritten by the DatabaseEngine to conform with the PostgreSQL syntax.

↓

Resultset is available as a true workarea with all features such as navigation, filters, scopes, field-variable access

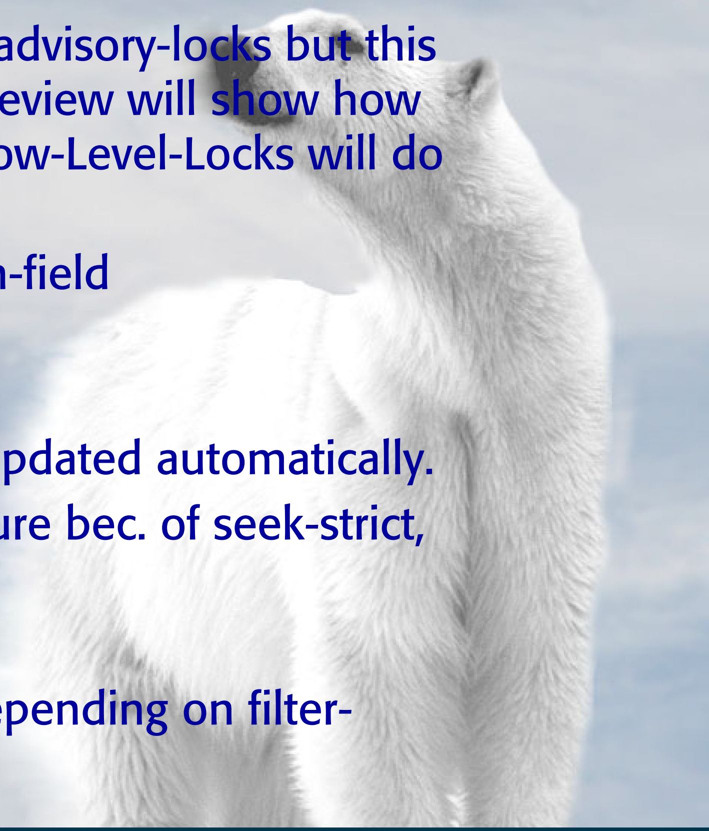
Data-Type-Mappings Xbase++ & PostgreSQL

- Smallint, int, bigint become Xbase++ Numeric-Integer
- Decimal and Numeric become Xbase++ Numeric-Decimal (arbitary numeric runtime datatype)
- Real and double become Xbase++ Numeric-Float
- Char, VarChar, Text are mapped to Xbase++ Character. Padding and Truncation as SQL defines in done silently by the runtime.
- Bytea is mapped to character w/o locale – aka. Binary type.
- Arrays are supported by Xbase++ arrays, however the type constraints introduces a impedance mismatch
- Composite types are mapped to Object of the a class which matches by classname==composite-typename, if the class is not found a hash-array is used.
- Geometric types are mapped to dedicated classes supported by the 2D primitives of Xbase++ GraphicsEngine.
- **Notes:**
  - NULL becomes NIL, in addition runtime can mask NIL-Value to empty-value to avoid three-valued logic if required.
  - The Xbase++ Numeric type is a morhping type which automatically morphes between int->bigint, int->float if required.
  - The Xbase++ character-type is locale dependend. The runtime automatically performs transformantions and collation/locale specific comparison/ordering. However we see here a clear deficit of the PostgreSQL by not supporting NCHAR types.

Navigational Data-Access (PG-NAV)

Navigational Data Access

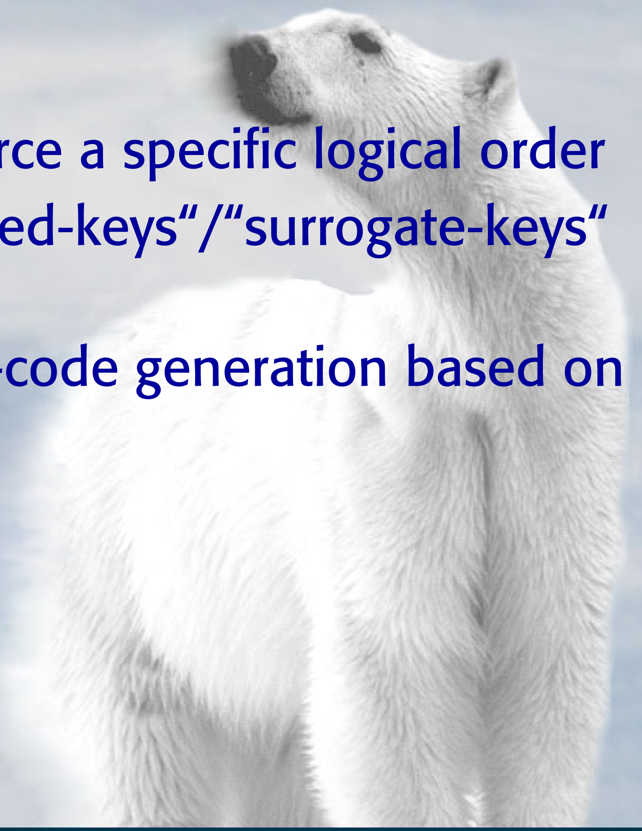
- A Workarea reflects a result-set
- REPLACE operations lead to UPDATE statement
- APPEND operation leads to deferred INSERT operation with the next REPLACE
- DELETED stated is managed with a hidden-field __DELETED
- Record-Id is emulated with a hidden-field __RECORDID
- Record-Locking is simply ignored as of yet. We tried advisory-locks but this may create interoperability problems. Community-Preview will show how to deal with that. Eventually semantics of existing Row-Level-Locks will do the job.
- Lost-Update-Detection is done implicit with a hidden-field __UPDATECOUNT
- Table-Locks are stright forwarded EXCLUSIVE locks.
- Index-Expressions become hidden fields which get updated automatically.
- Seek operations are transformed into stored-procedure bec. of seek-strict, seek-soft, seek-first, seek-last semantics.
- Scopes are transformed into a WHERE clause.
- Filters are handled dynamically on client or server depending on filter-expression complexitiy.



xBase/ISAM specific issues

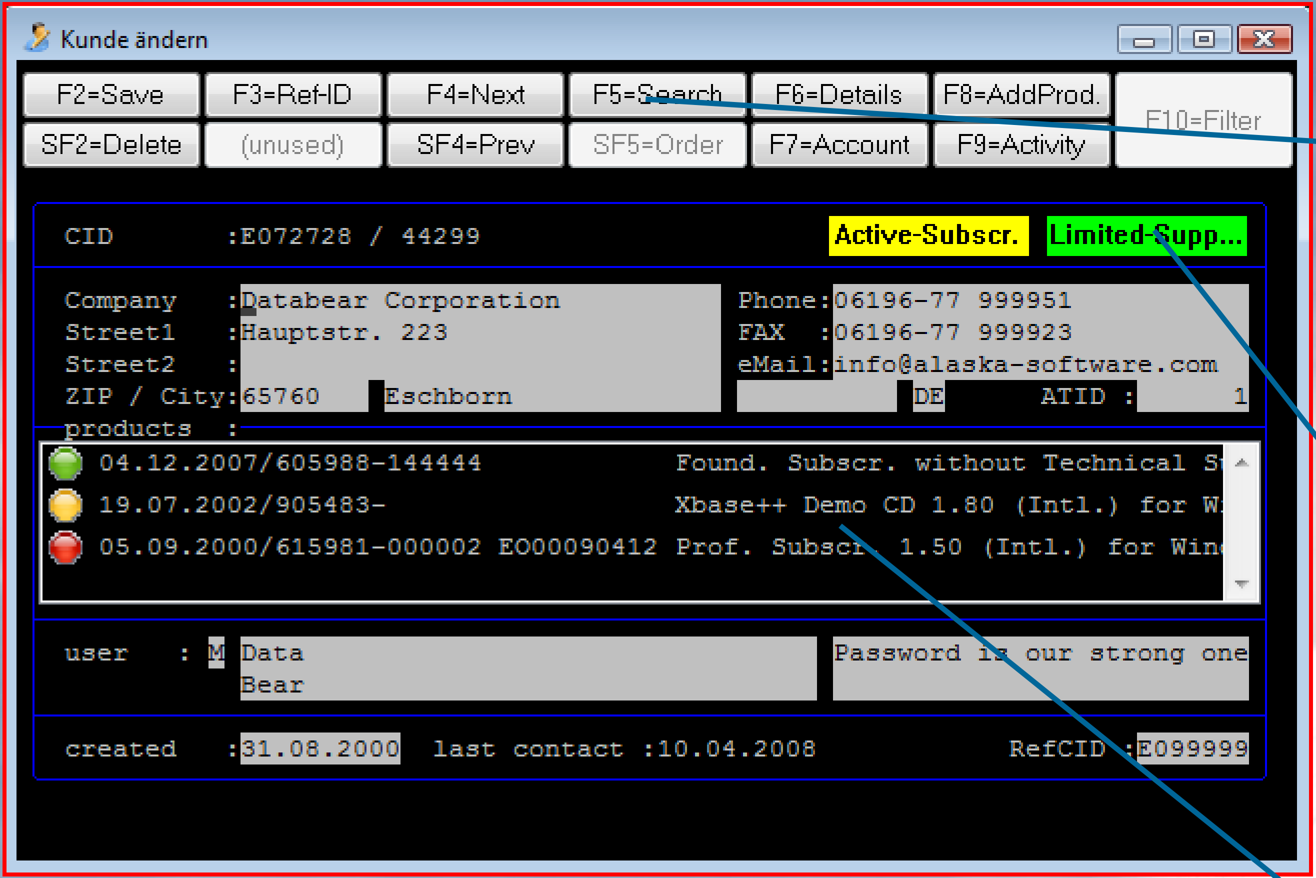
Navigational Data Access

- Delete/Recall/Pack/Zap
  - In xBase terms a record is marked for deletion
  - It can be recalled until a pack happend
  - A pack removes all marked for deletion records from a table
  - A Zap removes all record from a table
- Index have different semantics
  - In xBase terms indexes are used to enforce a specific logical order
  - Indexes are often used to build „calculated-keys“/“surrogate-keys“ used inherintly in the data-model.
  - Indexes are also used to perform match-code generation based on fields.



Application Evolution

Navigational Data Access



```
CID        :E072728 / 44299

Company    :Databear Corporation          Phone:06196-77 999951
Street1    :Hauptstr. 223                 FAX  :06196-77 999923
Street2    :                              eMail:info@alaska-software.com
ZIP / City:65760    Eschborn                   DE      ATID :       1
products   :
● 04.12.2007/605988-144444             Found. Subscr. without Technical Su
● 19.07.2002/905483-                   Xbase++ Demo CD 1.80 (Intl.) for Wi
● 05.09.2000/615981-000002 EO000090412 Prof. Subscr. 1.50 (Intl.) for Win

user    : M Data                          Password is our strong one
            Bear

created    :31.08.2000  last contact :10.04.2008          RefCID :E099999
```

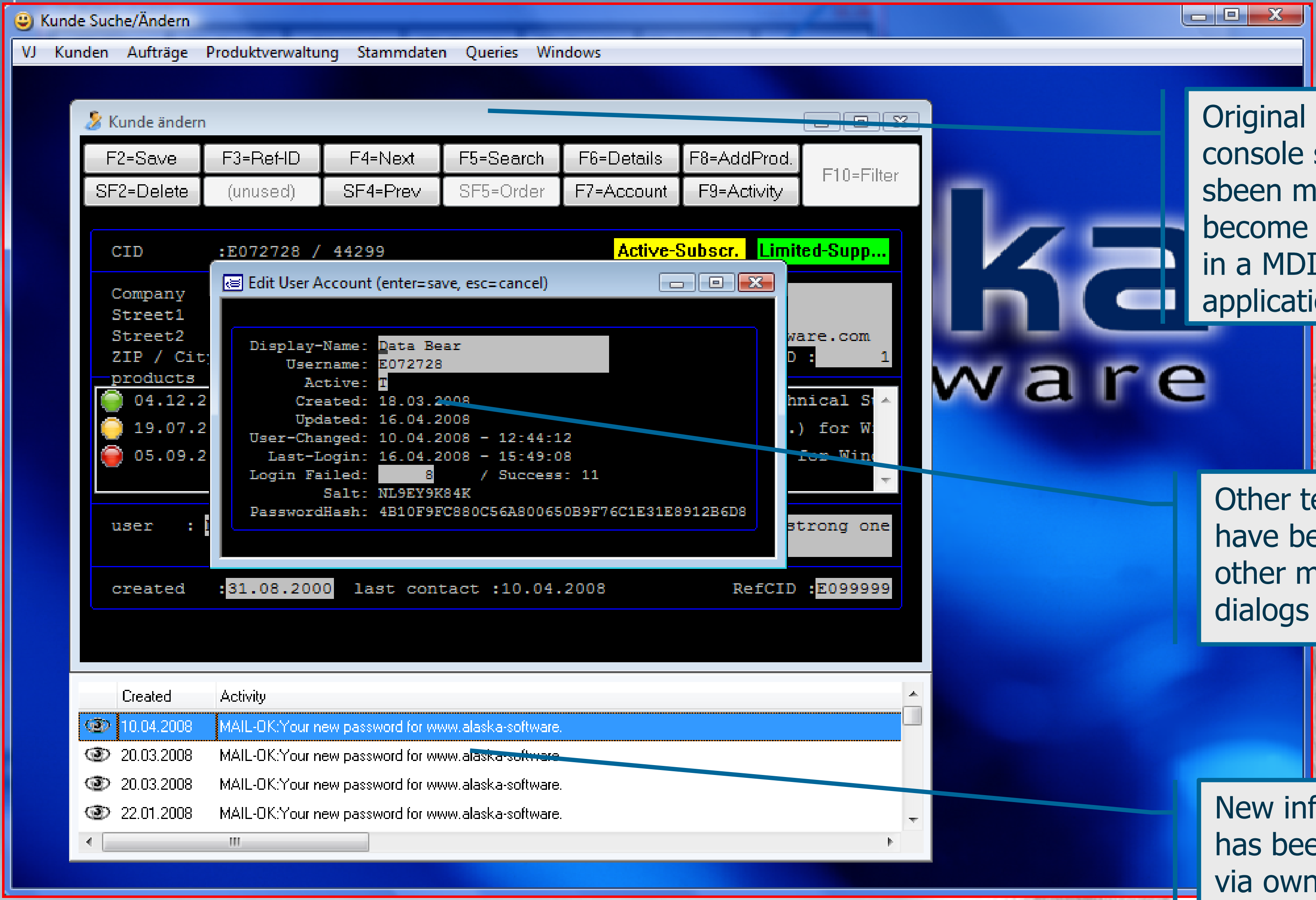
Pushbutton have been added

Static controls to view state added

Listbox with owner-drawing added

Application Evolution

Navigational Data Access



```
CID          :E072728 / 44299

Company
Street1
Street2
ZIP / Cit
products
   04.12.2
   19.07.2
   05.09.2

user    :

created      :31.08.2000   last contact :10.04.2008          RefCID :E099999
```

```
 Display-Name: Data Bear
     Username: E072728
       Active: T
      Created: 18.03.2008
      Updated: 16.04.2008
 User-Changed: 10.04.2008 - 12:44:12
   Last-Login: 16.04.2008 - 15:49:08
 Login Failed:        8     / Success: 11
         Salt: NL9EY9K84K
 PasswordHash: 4B10F9FC880C56A800650B9F76C1E31E8912B6D8
```

Created	Activity
10.04.2008	MAIL-OK:Your new password for www.alaska-software.
20.03.2008	MAIL-OK:Your new password for www.alaska-software.
20.03.2008	MAIL-OK:Your new password for www.alaska-software.
22.01.2008	MAIL-OK:Your new password for www.alaska-software.

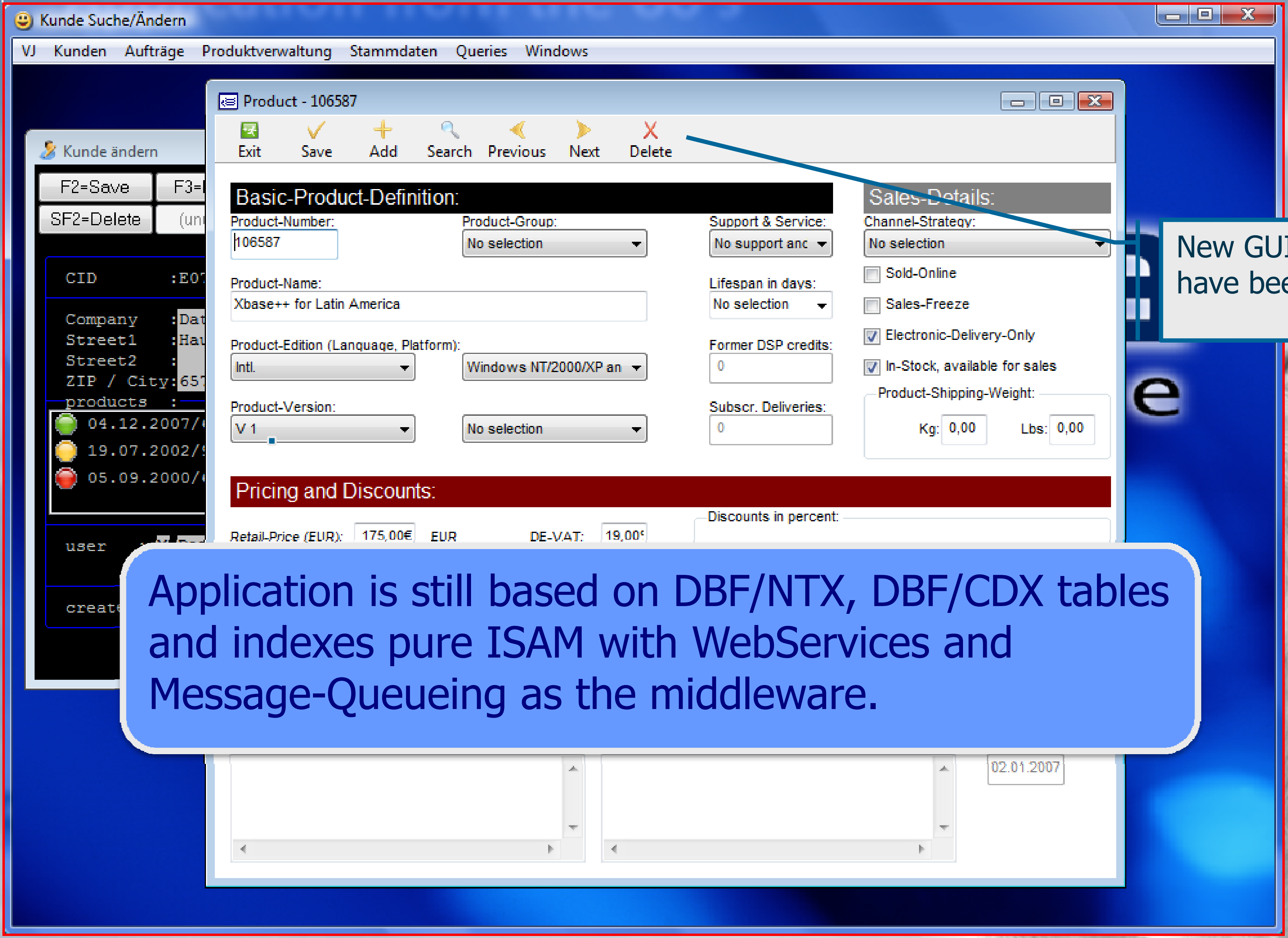
Original text-console screen ha sbeen mode to become a window in a MDI application

Other text screens have become other modal dialogs

New information has been added via owned windows

Application Evolution

Navigational Data Access


Kunde Suche/Ändern
VJ Kunden Aufträge Produktverwaltung Stammdaten Queries Windows
Kunde ändern
F2=Save
SF2=Delete
Product - 106587
Exit Save Add Search Previous Next Delete
Basic-Product-Definition:
Sales-Details:
Pricing and Discounts:

Product-Number: 106587
Product-Group: No selection
Support & Service: No support anc
Channel-Strategy: No selection
Product-Name: Xbase++ for Latin America
Lifespan in days: No selection
Product-Edition (Language, Platform): Intl. / Windows NT/2000/XP an
Former DSP credits: 0
Product-Version: V 1 / No selection
Subscr. Deliveries: 0
☐ Sold-Online
☐ Sales-Freeze
☑ Electronic-Delivery-Only
☑ In-Stock, available for sales
Product-Shipping-Weight: Kg: 0,00 Lbs: 0,00
Retail-Price (EUR): 175,00€ EUR DE-VAT: 19,00
Discounts in percent:

```
CID          :E07
Company      :Dat
Street1      :Hau
Street2      :
ZIP / City:657
products     :
 04.12.2007/
 19.07.2002/
 05.09.2000/
user
create
```

New GUI dialogs have been added

Application is still based on DBF/NTX, DBF/CDX tables and indexes pure ISAM with WebServices and Message-Queueing as the middleware.

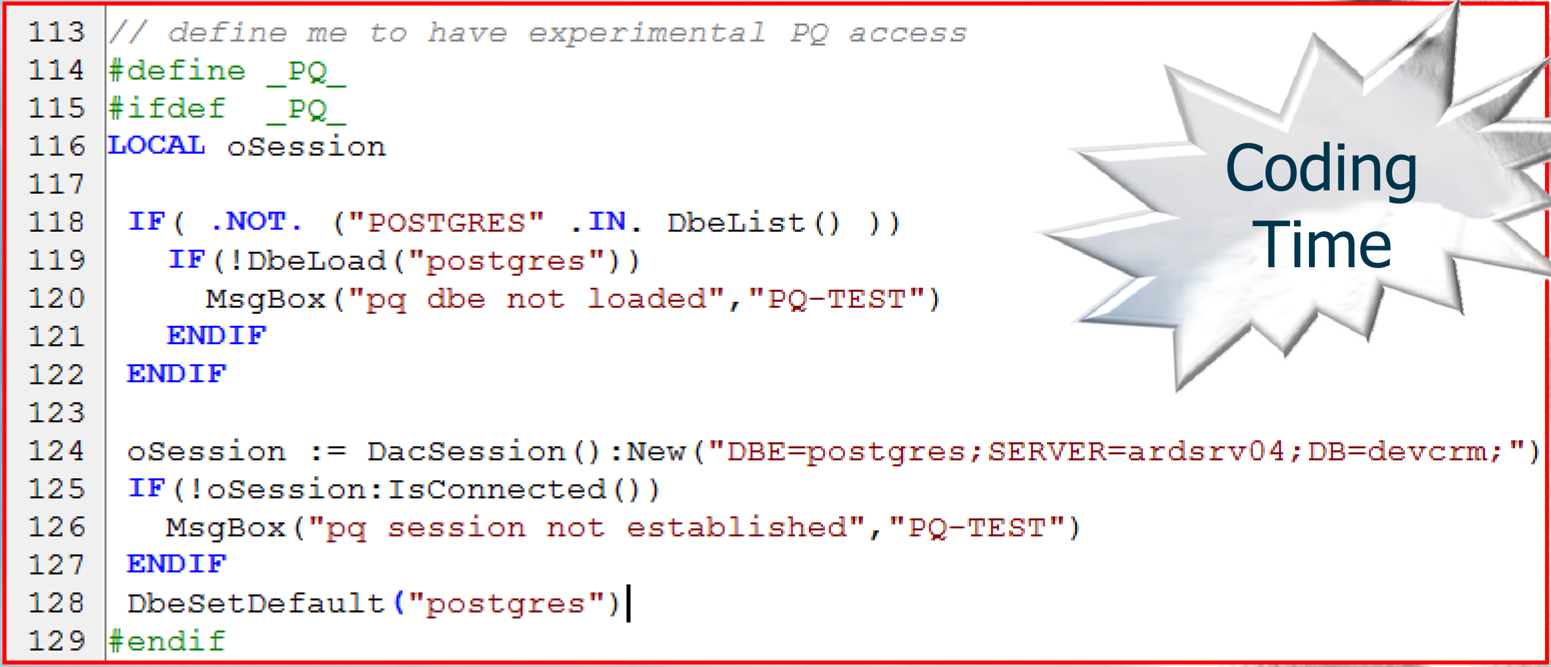
Application metrics

Navigational Data Access

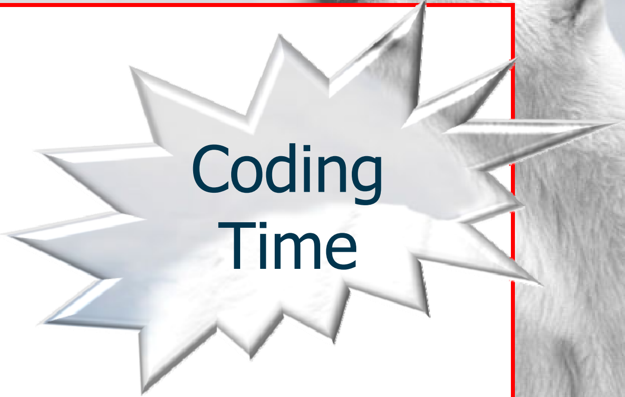
- Single Executable with one DLL
- ~20.000 lines of code in 63 source-files
- With hundreds of Seeks, Skips, Filters, Scopes
- ~ 12 tables plus 8 secondary lookup tables such as countries
- ~ 40 indexes, ~ 10 are complex expressions
- User-Defined-Functions in Index-Expressions and Filters
- Lexical-Rules to match Maier and Mayer

Migration to PostgreSQL

- The only code changes needed are at a single place:
  - establish the connection to the PostgreSQL server
  - ensure the postgres-database engine becomes the default one.
  - Table and index migration is done with the dbf2sql utility/wizard.



```
// define me to have experimental PQ access
#define _PQ_
#ifdef  _PQ_
LOCAL oSession

 IF( .NOT. ("POSTGRES" .IN. DbeList() ))
   IF(!DbeLoad("postgres"))
     MsgBox("pq dbe not loaded","PQ-TEST")
   ENDIF
 ENDIF

 oSession := DacSession():New("DBE=postgres;SERVER=ardsrv04;DB=devcrm;")
 IF(!oSession:IsConnected())
   MsgBox("pq session not established","PQ-TEST")
 ENDIF
 DbeSetDefault("postgres")
#endif
```



Performance issues

Navigational Data Access

- Navigational data-access means
  - moving relative-forward or relative-backward
  - Jumping between absolute positions
  - Working with real-time-data
- To implement navigational access the PostgreSQL DatabaseEngine performs result-set partitioning and result-set caching
- Realtime-Data-Behaviour is achieved by making use of the notify-mechanism and by data-access-pattern related heuristics.
- The current PG-NAV Engine for Xbase++ outperforms file-based data-access in 5+ user scenario over the network.

Benefits of PG-NAV

Navigational Data Access

- Immediate
  - Higher reliability of your data
  - Better scaleability
  - Lesser network saturation
  - Remote-Applicatons due to TCP/IP layer between Client and PostgreSQL server
- Mediate
  - Higher security – hopefully -☺
  - Full text search capabilities
  - Geo data support
  - Other platform choices for the DBMS than WINTEL
  - Central place for backups

Stored-Procedures

Xbase++ on the server

- **Defintion in the context of this lecture:** Code executed on the database-server. Implementation flavours are Stored Functions, Stored Procedures, and Triggers but also Datatypes and Operators.
- **The Pros and Cons:**

Single place for business logic. Move logic to data leads to increase performance	If you have code in both the application and the server you have two code-bases to manage
Stored Procedure can be seen as a „Service“ in SOA land	Writing stored-procedures needs knowledge of a different language, dififcult debugging.
Decouple application logic from application.	They are not dbms independant, vendor lock in
Fewer data needs to be shuffled between client and server	SP languages are somewhat limited. Leading to a lack of integration with other middle ware components

The pros and cons are basically balanced. There is no clear winner – but the cons are centered around ...!?

Stored-Procedures Cons revised

Xbase++ on the server

- All the cons. are related to the language/toolset around stored-procedure development
- The vendor-lock in argument was never true, as even with choosing specific datatypes or NULL behaviour led to a vendor lock-in the past.

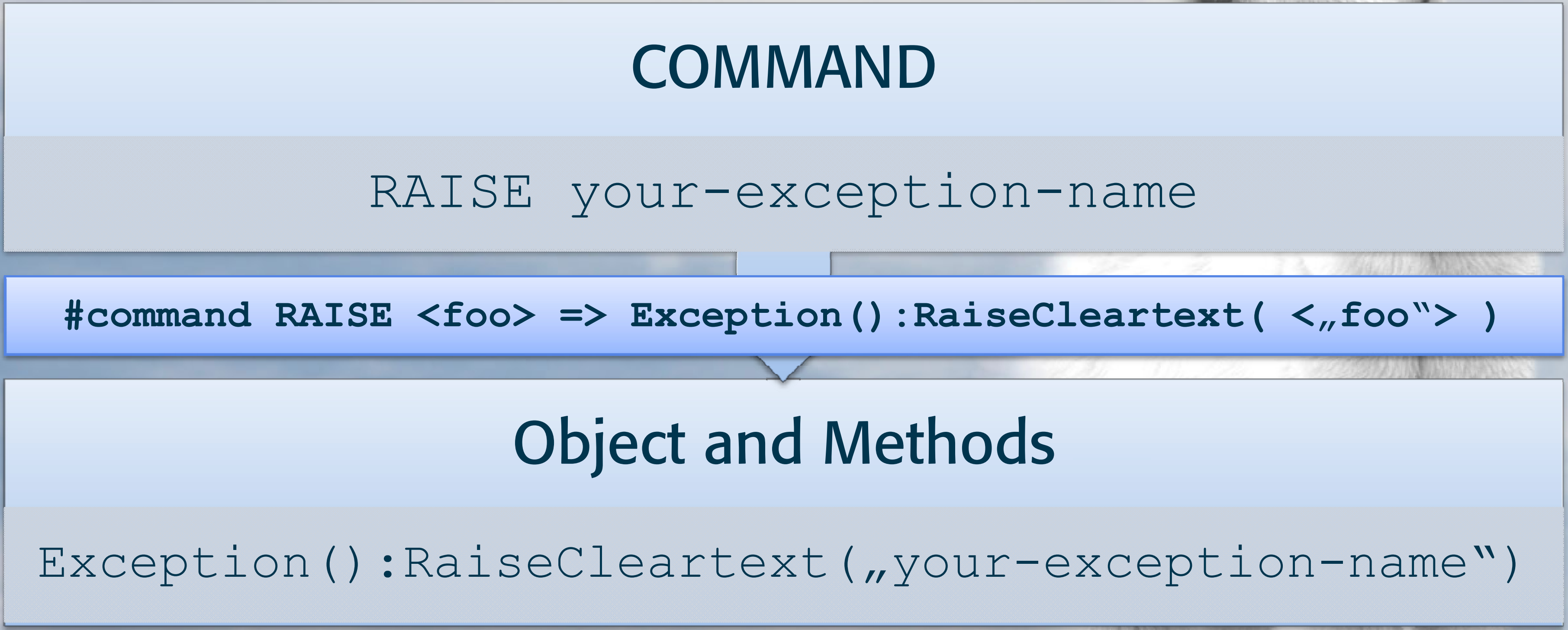
Why not!

- use Xbase++ as the development language for client and server development
- Write and debug your business logic on the client and let the runtime move them to the SQL server
- Let the Pass-Through-SQL rewrite your SQL code to conform with the syntax of the SQL server

Xbase++ a three-tier language

Xbase++ on the server

- Commands
- Functions
- Classes / Objects / Methods and Members



COMMAND

```
RAISE your-exception-name
```

```
#command RAISE <foo> => Exception():RaiseCleartext( <„foo“> )
```

Object and Methods

```
Exception():RaiseCleartext(„your-exception-name“)
```

Xbase++ Stored-Procedures (XSP)

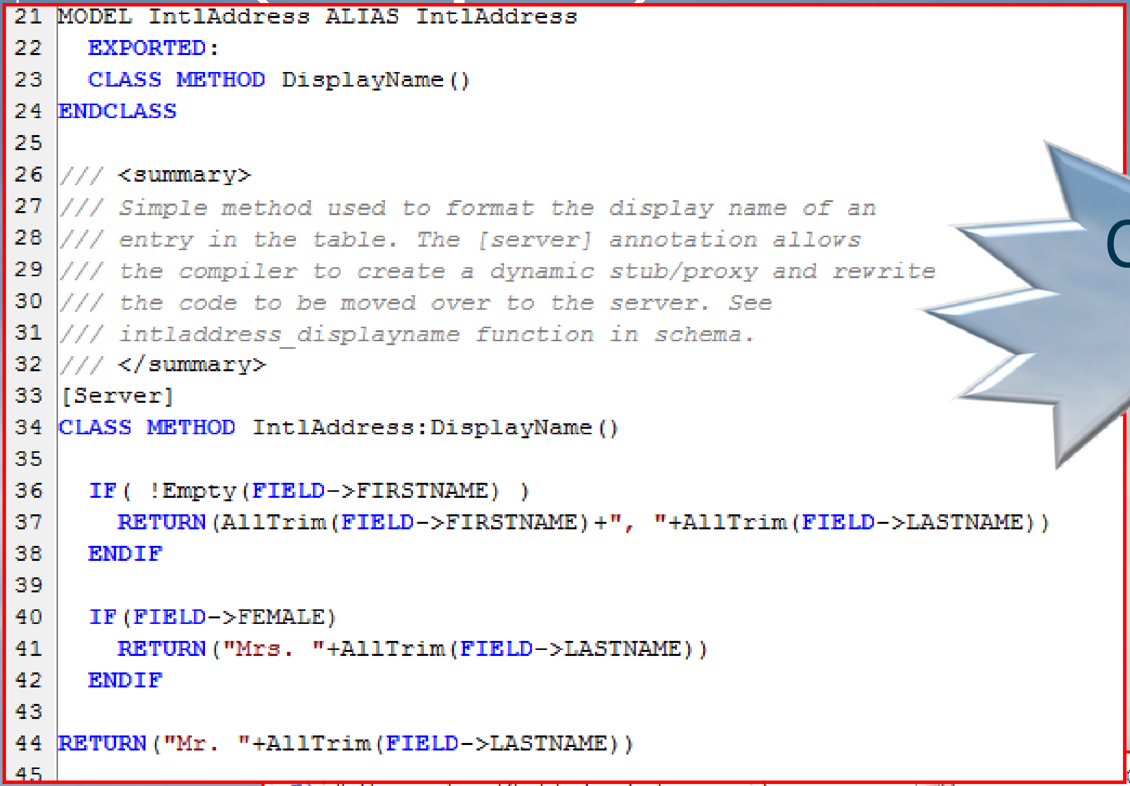
Xbase++ on the server

- The entire Xbase++ language commands with pre-processor, functions/procedures, classes and objects is transparent available on the PostgreSQL server
- The PostgreSQL Pass-through SQL engine automatically uses the SPI interface instead of LibPQ. Behaviour against the user-level stays unchanged. It is therefore transparent to the business logic if the code is executed on the client or on the server.
- Server-Side-Functions can be created explicit or implicit by using the [server] annotation in your source code. The later way allows you to debug or unit-test your application without even thinking about that this portions of your code are executed on behalf of the server.
- Code compilation, binary-creation and caching is done automatically.
- The Xbase++ profiler can be used to profile execution which allows easy and quick performance bottleneck detection.

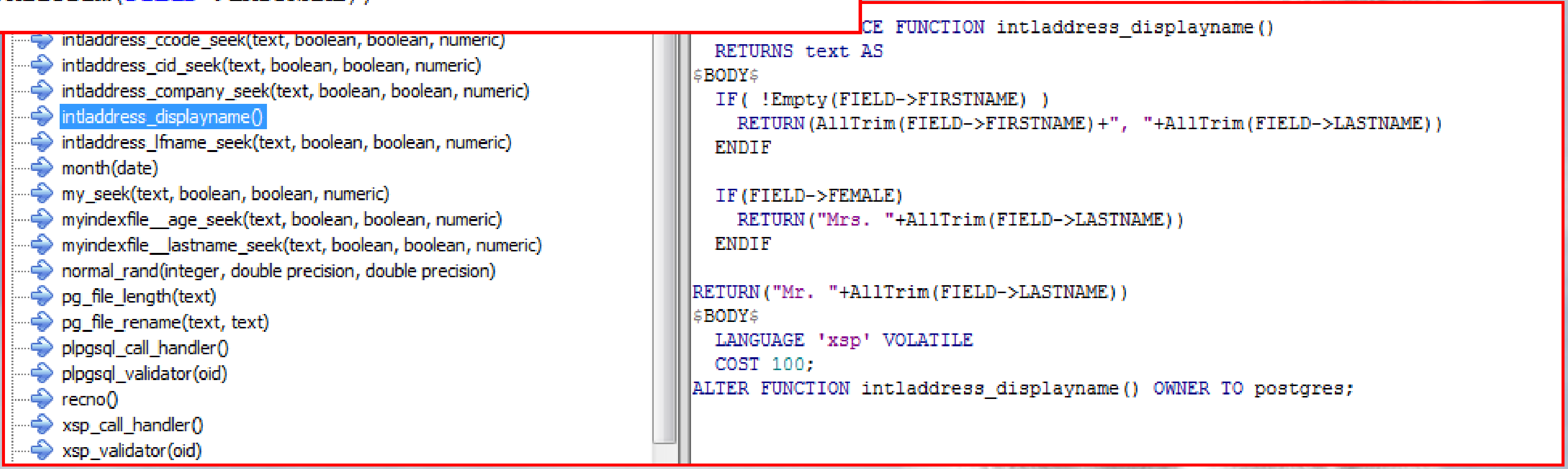
XSP (sample1)

Xbase++ on the server

Coding Time



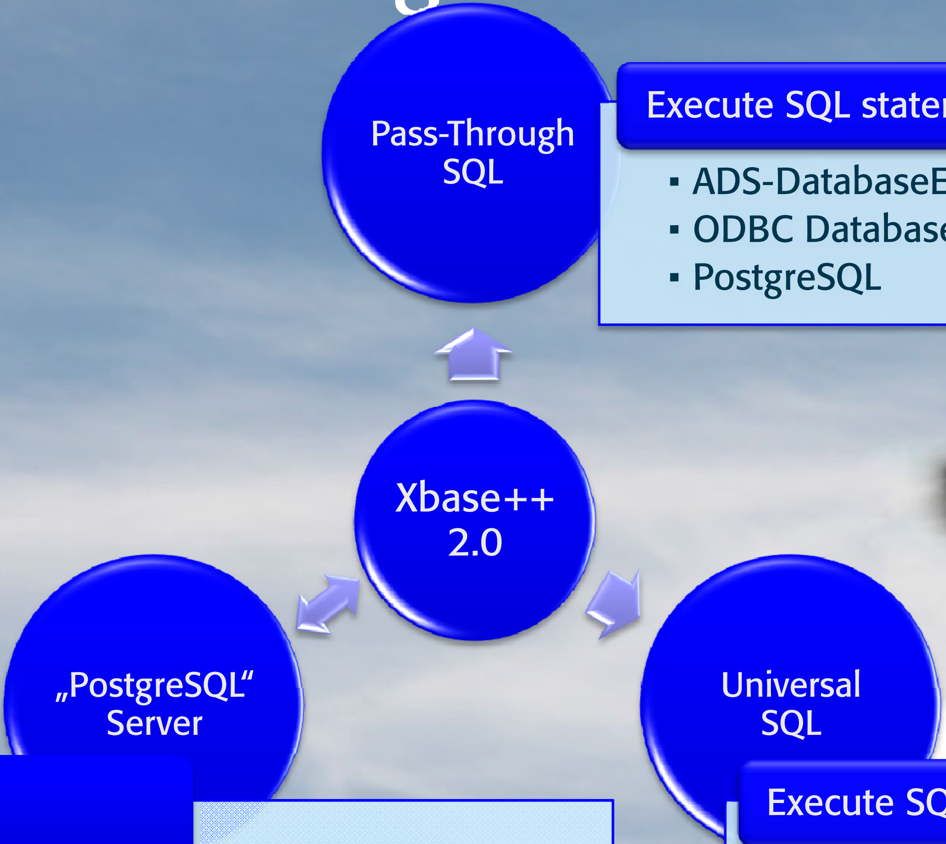
```
MODEL IntlAddress ALIAS IntlAddress
  EXPORTED:
  CLASS METHOD DisplayName()
ENDCLASS

/// <summary>
/// Simple method used to format the display name of an
/// entry in the table. The [server] annotation allows
/// the compiler to create a dynamic stub/proxy and rewrite
/// the code to be moved over to the server. See
/// intladdress_displayname function in schema.
/// </summary>
[Server]
CLASS METHOD IntlAddress:DisplayName()

  IF( !Empty(FIELD->FIRSTNAME) )
    RETURN(AllTrim(FIELD->FIRSTNAME)+", "+AllTrim(FIELD->LASTNAME))
  ENDIF

  IF(FIELD->FEMALE)
    RETURN("Mrs. "+AllTrim(FIELD->LASTNAME))
  ENDIF

RETURN("Mr. "+AllTrim(FIELD->LASTNAME))
```



```
intladdress_ccode_seek(text, boolean, boolean, numeric)
intladdress_cid_seek(text, boolean, boolean, numeric)
intladdress_company_seek(text, boolean, boolean, numeric)
intladdress_displayname()
intladdress_lfname_seek(text, boolean, boolean, numeric)
month(date)
my_seek(text, boolean, boolean, numeric)
myindexfile__age_seek(text, boolean, boolean, numeric)
myindexfile__lastname_seek(text, boolean, boolean, numeric)
normal_rand(integer, double precision, double precision)
pg_file_length(text)
pg_file_rename(text, text)
plpgsql_call_handler()
plpgsql_validator(oid)
recno()
xsp_call_handler()
xsp_validator(oid)
```

```
CE FUNCTION intladdress_displayname()
  RETURNS text AS
$BODY$
  IF( !Empty(FIELD->FIRSTNAME) )
    RETURN(AllTrim(FIELD->FIRSTNAME)+", "+AllTrim(FIELD->LASTNAME))
  ENDIF

  IF(FIELD->FEMALE)
    RETURN("Mrs. "+AllTrim(FIELD->LASTNAME))
  ENDIF

RETURN("Mr. "+AllTrim(FIELD->LASTNAME))
$BODY$
  LANGUAGE 'xsp' VOLATILE
  COST 100;
ALTER FUNCTION intladdress_displayname() OWNER TO postgres;
```

„Arctica“ technologies



Execute SQL statements against:

- ADS-DatabaseEngine
- ODBC Database-Sources
- PostgreSQL

Plays a special role:

- As it understands all native xBase/ISAM functions commands
- Supports all native Xbase++ datatyes including objects and array
- Supports Xbase++ as the language for stored procedures

Execute SQL statements against:

- Any Workarea
  - DBF/NTX/CDX/DEL/SDF/...
  - ADS-DatabaseEngine
  - ODBC Database-Sources
- Objects
- Arrays
- PostgreSQL

Summary

- PostgreSQL is a unique DBMS, thanks to the way it
  - supports alternate languages on the server side
  - Enables customizing in term of datatypes and operators
  - Enhances the scope of user-defined-function in terms of index management
  - isolates connection contexts with the process model
- Alaska Software as a company
  - We have around 25.000 customers for our development toolset with more than a million installation sites of end-user applications
  - With the release of our PostgreSQL related technologies we want to establish PostgreSQL as the preferred SQL Server solution at our customers.
  - We currently assume that approx. 1.000 of our customers will move to the PostgreSQL dbms with the help of our technologies. Leading to a 6 digit number of Server installation sites for mission critical LOB applications.

Final Note!

- We are currently in a early stage of getting in touch with the PostgreSQL community
- We are monitoring the community, we donate but don‘t contribute
- As time will come, Alaska Software may actively contribute to the progress of the PostgreSQL dbms.
- Alaska Software has no plans to sell the server, we make money with the Xbase++ platform, services and technologies. We want to add value!

Thank you all for such an open and customizable SQL DBMS as PostgreSQL is.

Thanks for your attention

Please visit us at
www.alaska-software.com

alaska software