"Thick Database" Techniques for Fusion (and other Web) Developers

Dr. Paul Dorsey,
Michael Rosenblum

Dulcian, Inc.

NYOUG Web SIG
August 7, 2008

“Thick Database” Approach (1)

- Micro-Service-Oriented-Architecture (M-SOA)
- Service Component Architecture (SCA)
- Division between the database and user interface (UI) portions.
- Two key features involved in "thick database thinking":
  - Nothing in the UI ever directly interacts with a database table. All interaction is accomplished through database views or APIs.
  - Nearly all application behavior (including screen navigation) is handled in the database.
- Thick database does not simply mean stuffing everything into the database and hoping for the best.

“Thick Database” Approach (2)

- Creating a thick database makes your application UI technology-independent.
  - Creates reusable, UI technology-independent views and APIs.
  - Reduces the complexity of UI development.
  - Database provides needed objects.
  - Reduces the burden on the UI developer

Thick Database Benefits

- Minimizes development risk
- Helps build working applications that scale well.
- Benefit Metrics:
  - Better performance (10X)
  - Less network traffic (100X)
  - Less code (2X)
  - Fewer application servers (3X)
  - Fewer database resources (2X)
  - Faster development (2X)

Easier to Refactor

- UI technology stack changes are common.
- The .Net vs. Java EE battle rages on.
- Web architecture is more volatile than the database platform.
- Defense against the chaos of a rapidly evolving standard.
- Test: What is the probability that your web UI standards will be the same in 18 months?

Answer 0%

How Thick is too Thick?

- What would happen if 100% of all UI logic were placed in the database?
  - Tabbing out of a field
  - LOV populated from database
  - Page navigation

- Pathologically complete way to implement the thick database approach.
- A system built this way would be sub-optimal.
  - But it works

How Thin is too Thin?

- Can a skilled team successfully build applications that are 100% database “thin”?
  - Requires a highly skilled team.
  - Minimize round trips
  - ANY middle tier technology (e.g. BPEL) can also be a performance killer.
- Possible but difficult

De-Normalized Views

De-Normalized Views

- The idea:
  - Convert relational data into something that will make user interface development easier.
  - Easiest way to separate data representation in the front-end from the real model.
- The solution:
  - Use a view with a set of INSTEAD-OF triggers

De-Normalized view

```
create or replace view v_customer
as
select c.cust_id,
       c.name_tx,
       a.addr_id,
       a.street_tx,
       a.state_cd,
       a.postal_cd
from customer c
left outer join address a
   on c.cust_id = a.cust_id
```

INSTEAD-OF Insert

```
create or replace trigger v_customer_ii
instead of insert on v_customer
declare
  v_cust_id customer.cust_id%rowtype;
begin
  if :new.name_tx is not null then
   insert into customer (cust_id,name_tx)
    values(object_seq.nextval,:new.name_tx)
   returning cust_id into v_cust_id;
  if :new.street_tx is not null then
   insert into address (addr_id,street_tx,
        state_cd, postal_cd, cust_id)
   values (object_seq.nextval,:new.street_tx,
    :new.state_cd,:new.postal_cd, v_cust_id);
  end if;
end;
```

Collections

Using Collections

- Sometimes it is just not possible to represent all required functionality in a single SQL statement.
- Denormalized view cannot be built.
- Oracle provides a different mechanism:
  - Collections allow you to hide the data separation, as well as all of the transformation logic.

What is a collection?

- Definition:
  - An ordered group of elements, all of the same type, addressed by a unique subscript.
- Implementation:
  - Since all collections represent data, they are defined as data types.
- Three types
  - Nested Tables
  - Associative Arrays
  - Variable-size arrays (V-Arrays)

Why use collections?

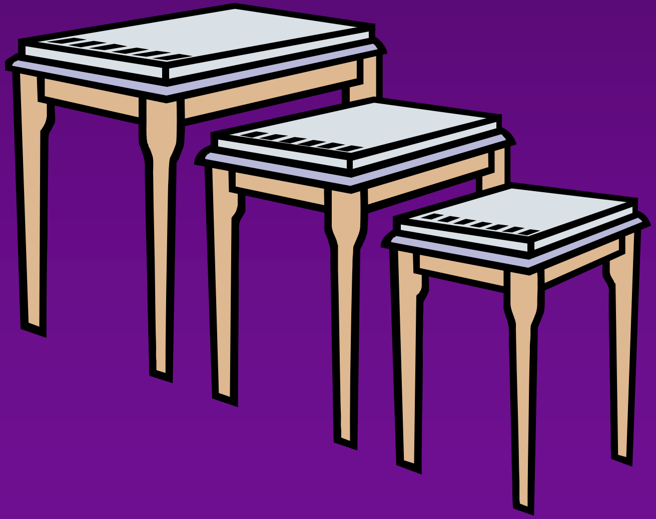
- Logical reason:
  - Collections allow you to articulate and manipulate sets of data.
- Technical reason:
  - Processing data in sets is “usually” faster than doing so one element at a time.
- Physical reason:
  - Manipulating sets in memory is “usually” 100 times faster than manipulating sets on the storage device.

Possible Issues

- Technical problem:
  - Amount of memory is limited (especially in 32-bit architecture)
- Economic problem:
  - Storage is cheap – memory is NOT.

- Learning curve:
  - People who are used to old habits of processing one row at a time (since COBOL days) will have problems working with sets.

Nested Tables:
Function-Based Views



Nested Tables (1)

- Nested tables – arbitrary group of elements of the same type with sequential numbers as a subscript
  - Undefined number of elements (added/removed on the fly)
  - Not dense (objects could be removed from inside)
  - Available in SQL and PL/SQL
  - Very useful in PL/SQL! (but not in tables)

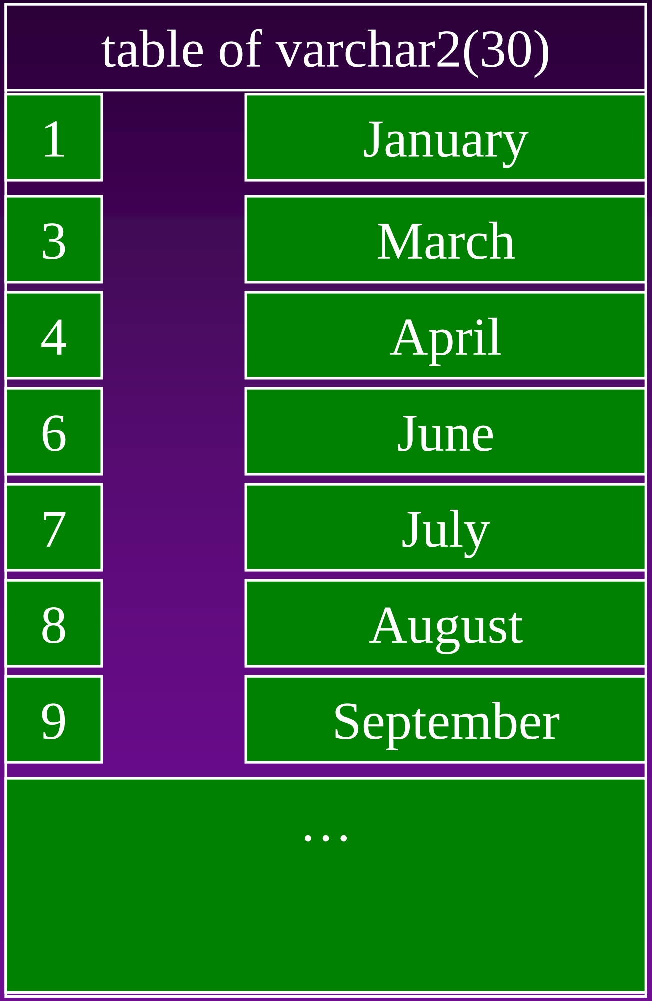


table of varchar2(30)	
1	January
3	March
4	April
6	June
7	July
8	August
9	September
…	

Nested Tables (2)

- Definition:

```
declare
   type NestedTable is
      table of ElementType;
...
create or replace type NestedTable
         is table of ElementType;
```

More About Nested Tables

- Nested tables can be used in SQL queries with the special operator: TABLE
  - Allows hiding of complex procedural logic “under the hood”
  - Nested table type must be declared as a user-defined type (CREATE OR REPLACE TYPE…)

Nested Tables – Example 1a

- Specify exactly what is needed as output and declare the corresponding collection:

```
Create type lov_oty is object
  (id_nr NUMBER,
   display_tx VARCHAR2(256));

Create type lov_nt
           as table of lov_oty;
```

Nested Tables - Example 1b

- Write a PL/SQL function to hide all required logic

```
function f_getLov_nt
 (i_table_tx,i_id_tx,i_display_tx,i_order_tx)
return lov_nt is
  v_out_nt lov_nt := lov_nt();
begin
  execute immediate
    'select lov_oty('
          ||i_id_tx||','||i_display_tx||
                    ')'||
    ' from '||i_table_tx||
    ' order by '||i_order_tx
  bulk collect into v_out_nt;
  return v_out_nt;
end;
```

Nested Tables - Example 1c

- Test SQL statement with the following code:

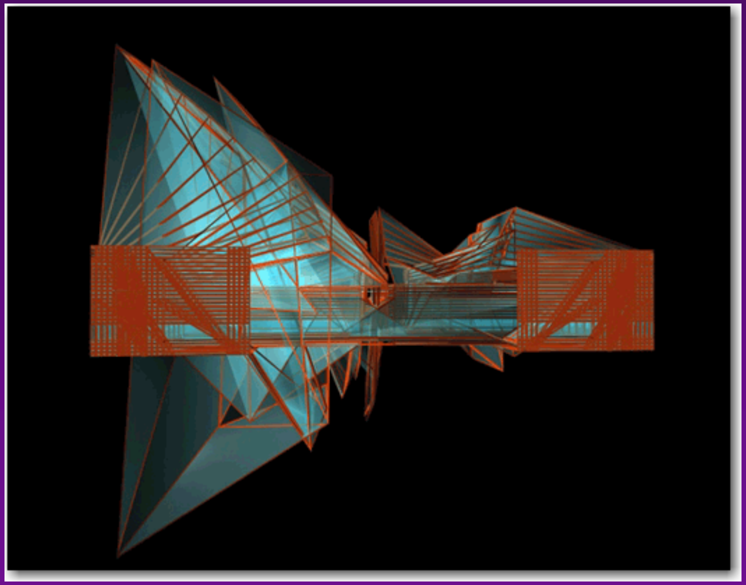
```
select id_nr, display_tx
from table(
           cast(f_getLov_nt
                  ('emp',
                   'empno',
                   'ename||''-''||job',
                   'ename')
           as lov_nt)
           )
```

Nested Tables - Example 1d

- Create a VIEW on the top of the SQL statement.
  - Completely hides the underlying logic from the UI
  - INSTEAD-OF triggers make logic bi-directional
  - Minor problem: There is still no way of passing parameters into the view other than some kind of global.

```
Create or replace view v_generic_lov as
select id_nr, display_tx
from table( cast(f_getLov_nt
     (GV_pkg.f_getCurTable,
      GV_pkg.f_getPK(GV_pkg.f_getCurTable),
      GV_pkg.f_getDSP(GV_pkg.f_getCurTable),
      GV_pkg.f_getSORT(GV_pkg.f_getCurTable))
            as lov_nt)
           )
```

Associative Arrays:
Optimizing Database Processing



Associative Arrays (1)

- An associative array is a collection of elements that uses arbitrary numbers and strings for subscript values
  - PL/SQL only
  - Still useful

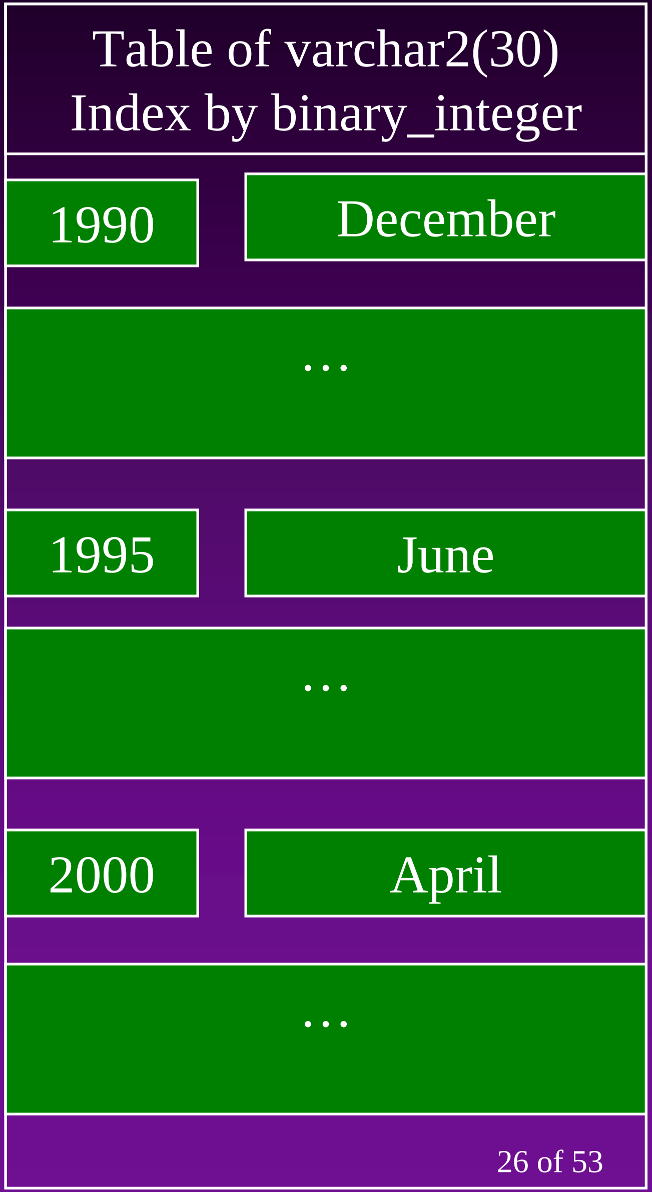


Table of varchar2(30)
Index by binary_integer

1990 December

…

1995 June

…

2000 April

…

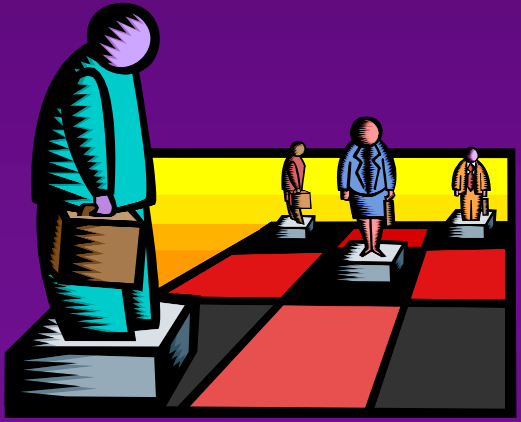
Associative Arrays (2)

- Definition:

```
declare
  type NestedTable is
    table of ElementType
        index by Varchar2([N]);
...
 type NestedTable is
    table of ElementType
        index by binary_integer;
```

Key New Feature

- Index by VARCHAR2 instead of by BINARY_INTEGER
  - Cannot be used in a FOR-loop
  - Allow creation of simple composite keys with direct access to the row in memory



Associative Arrays - Example 1a

- Prepare memory structure

```
declare
  type list_aa is table of VARCHAR2(2000)
        index by VARCHAR2(256);
  v_list_aa list_aa;
  cursor c_emp is
  select ename, deptno,to_char(hiredate,'q') q_nr
  from emp;
  v_key_tx VARCHAR2(256);
begin
  for r_d in (select deptno from dept order by 1) loop
    v_list_aa(r_d.deptno||'|1'):=
      'Q1 Dept#' ||r_d.deptno||':';
    v_list_aa(r_d.deptno||'|2'):=
      'Q2 Dept#' ||r_d.deptno||':';
    ...
end loop;
```

Associative Arrays - Example 1b

- Process data and present results

```
  ...
 for r_emp in c_emp loop
   v_list_aa(r_emp.deptno||'|'||r_emp.q_nr):=
      list_aa(r_emp.deptno||'|'||r_emp.q_nr)||
      ' '||r_emp.ename;
 end loop;

 v_key_tx:=v_list_aa.first;
 loop
   DBMS_OUTPUT.put_line
     (v_list_aa(v_key_tx));
   v_key_tx:=v_list_aa.next(v_key_tx);
   exit when v_key_tx is null;
 end loop;
end;
```

Bulk Operations

BULK COLLECT (1)

- BULK COLLECT clause
  - The idea:
    - Fetch a group of rows all at once to the collection
    - Control a number of fetched rows (LIMIT)
  - Risks:
    - Does not raise NO_DATA_FOUND
    - Could run out of memory

BULK COLLECT (2)

- Syntax:

```
select …
bulk collect into Collection
 from Table;

update …
returning … bulk collect into
Collection;

fetch Cursor
bulk collect into Collection;
```

BULK COLLECT example

```
declare
  type emp_nt is table of emp%rowtype;
  v_emp_nt emp_nt;

  cursor c_emp is select * from emp;
begin
  open c_emp;
  loop
    fetch c_emp
    bulk collect into v_emp_nt limit 100;
    p_proccess_row (v_emp_nt);
    exit when c_emp%NOTFOUND;
  end loop;
  close c_emp;
end;
```

FORALL (1)

- FORALL command
  - The idea:
    - Apply the same action for all elements in the collection.
    - Have only one context switch between SQL and PL/SQL
  - Risks:
    - Special care is required if only some actions from the set succeeded

FORALL (2)

- Syntax:

```
  forall Index in lowerupper
   update … set … where id = Collection(i)
...
  forall Index in lower..upper
   execute immediate '…'
  using Collection(i);
```

FORALL (3)

- Restrictions:
  - Only a single command can be executed.
  - Must reference at least one collection inside the loop
  - All subscripts between lower and upper limits must exist.
  - Cannot work with associative array INDEX BY VARCHAR2
  - Cannot use the same collection in SET and WHERE
  - Cannot refer to the individual column on the object/record (only the whole object)

FORALL Example

```
declare
    type number_nt is table of NUMBER;
    v_deptNo_nt number_nt:=number_nt(10,20);
begin
    forall i in v_deptNo_nt.first()
                            ..v_deptNo_nt.last()
      update emp
        set sal=sal+10
      where deptNo=v_deptNo_nt(i);
end;
```

Conclusions

- The #1 critical success factor for any web development is effective utilization of the database.
- PL/SQL is not irrelevant (and it continues to improve).
- Code that needs to access the database is faster if it is placed in the database.
- Database independence is irrelevant
  - UI technology independence is more important.
- Just because everyone is moving logic to the middle tier, does not make it a smart idea.

100% Repository-Based Application Development

BRIM® Web 3.0
User Interface (BRIM_UI)

- Complete Thick Database
- Minimal web traffic required
  - Fastest web applications ever
- Full client/server functionality on the web (Forms-like)
- 2 days of training to learn
  - Basic XML
  - Coding is all PL/SQL
  - Easier than Oracle Forms

- Deployment stack-independent (Java EE, .Net)
- Rapid development
- Ultra-secure

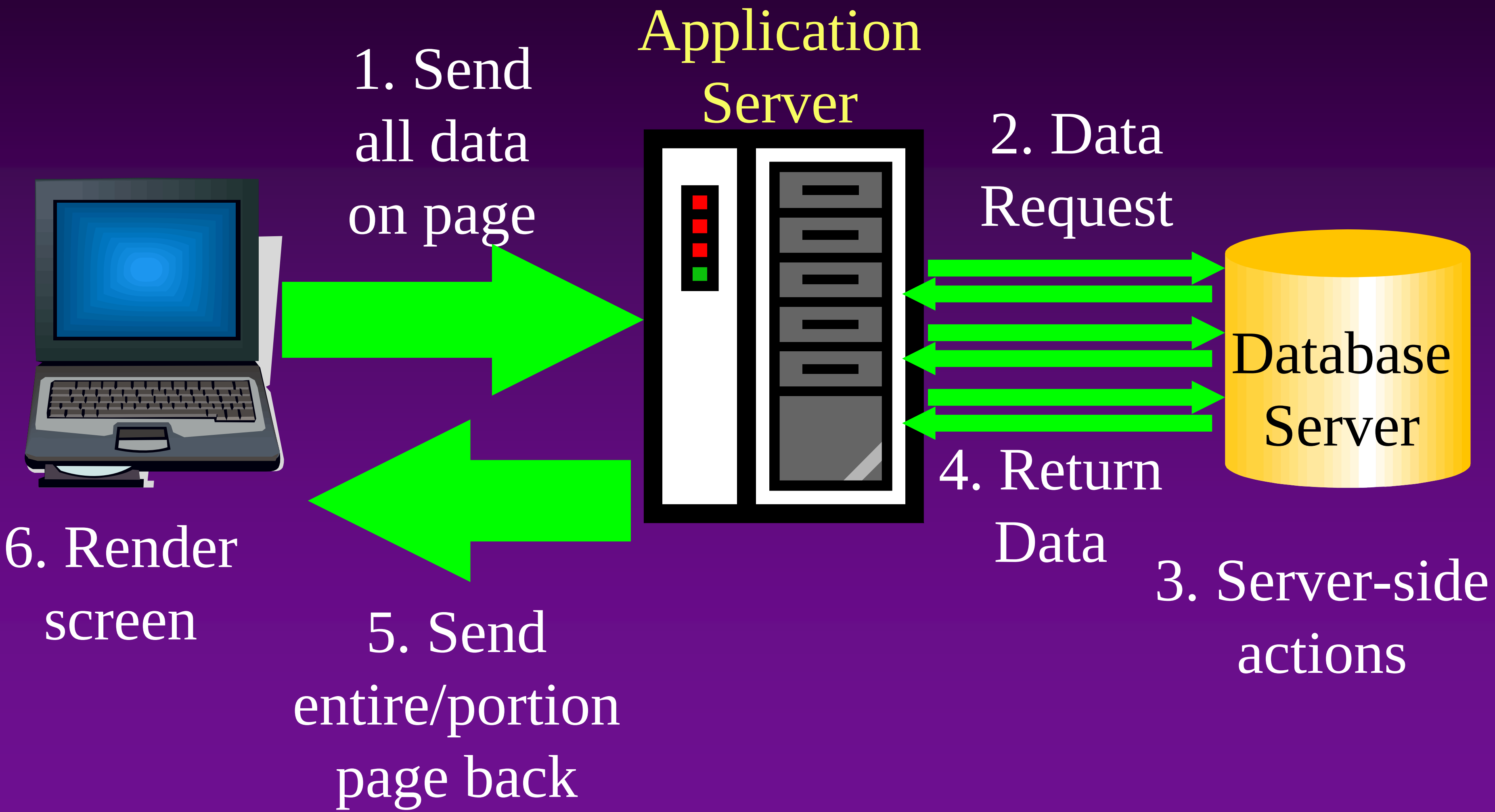
Part of the Total BRIM® Solution

- BRIM® Objects
  - Data Model
  - Process Flow
  - Data Validation
- BRIM ® Mapper
  - ETL, Web Service generation
- BRIM ® Web 3.0
  - User interface

Two Big Ideas

- A totally new web architecture
  - Event – Action Model
  - Enabling technology
- Repository-based UI tool
  - Simple repository
  - PL/SQL is the scripting language

Traditional Web Applications



1. Send all data on page

Application Server

2. Data Request

Database Server

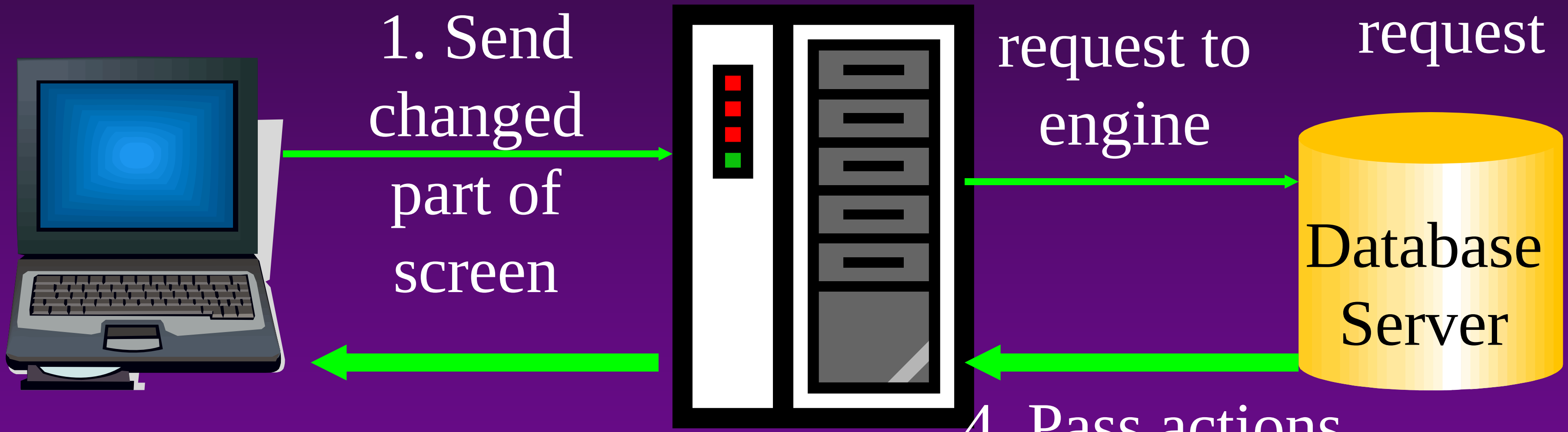
3. Server-side actions

4. Return Data

5. Send entire/portion page back

6. Render screen

BRIM® Web 3.0 Architecture



Application Server

1. Send changed part of screen

2. Pass request to engine

3. Process request

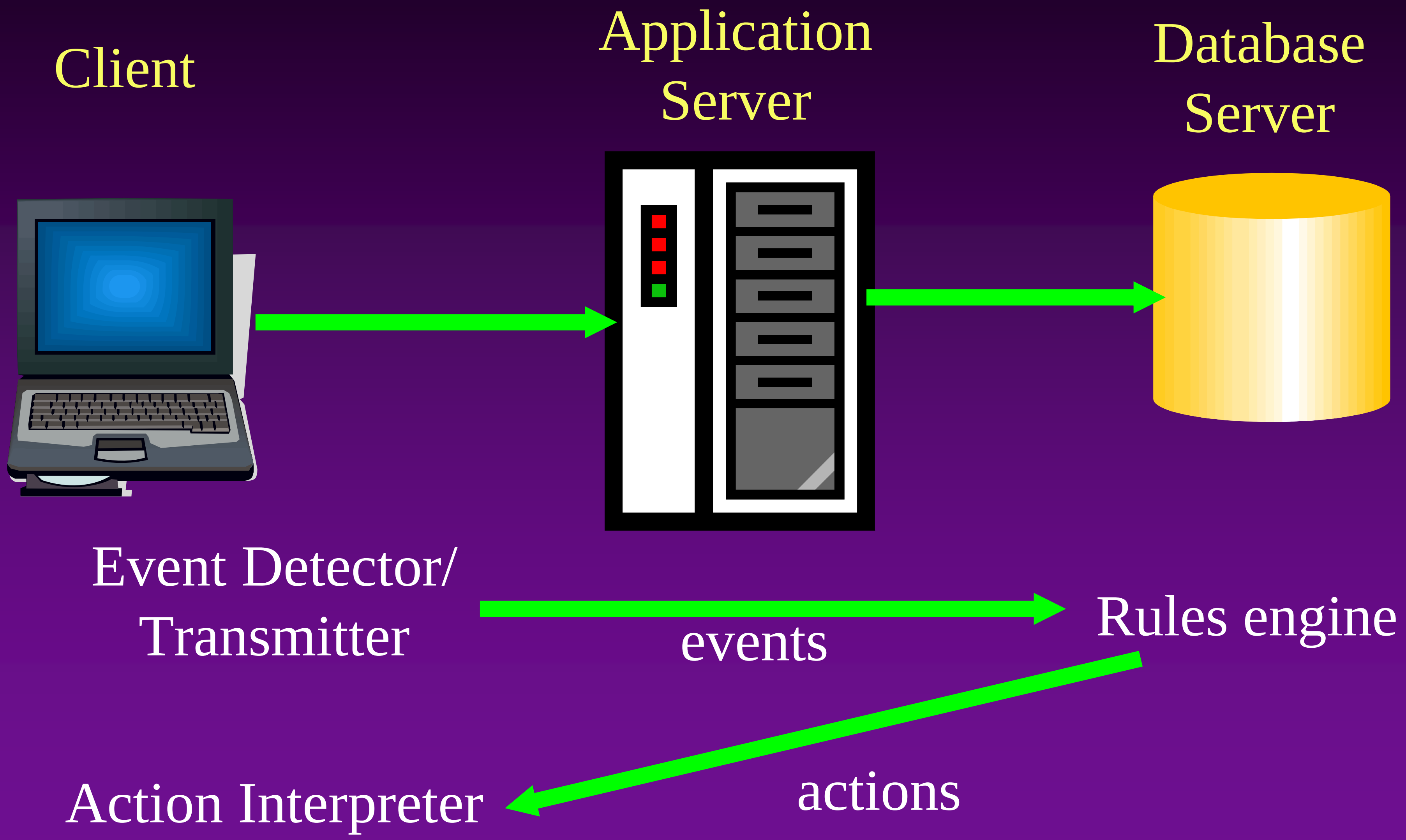
Database Server

4. Pass actions as XML

5. Pass actions to client

6. Process actions

System Architecture



Client

Application Server

Database Server

Event Detector/ Transmitter

events

Rules engine

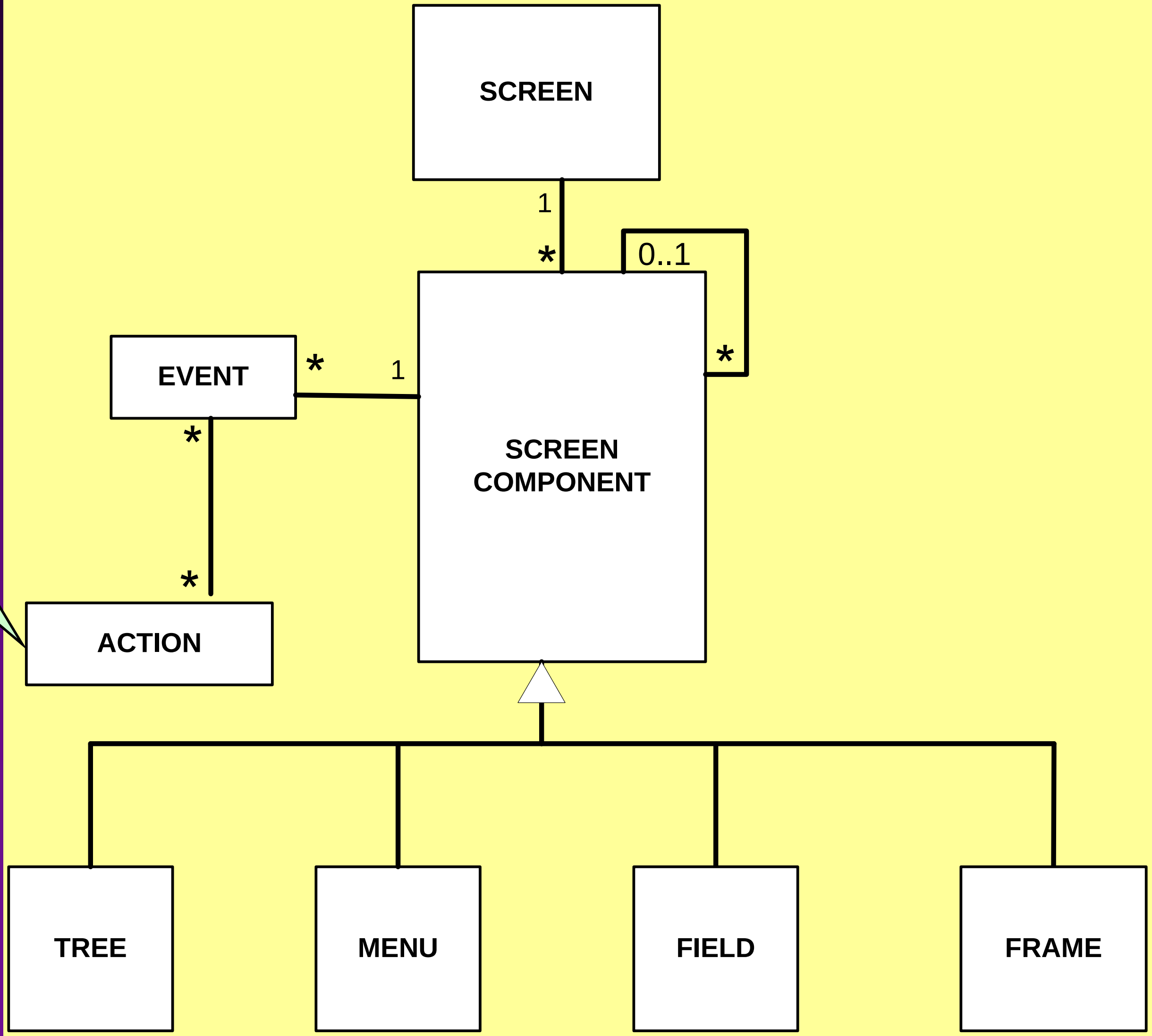
actions

Action Interpreter

What makes it work?

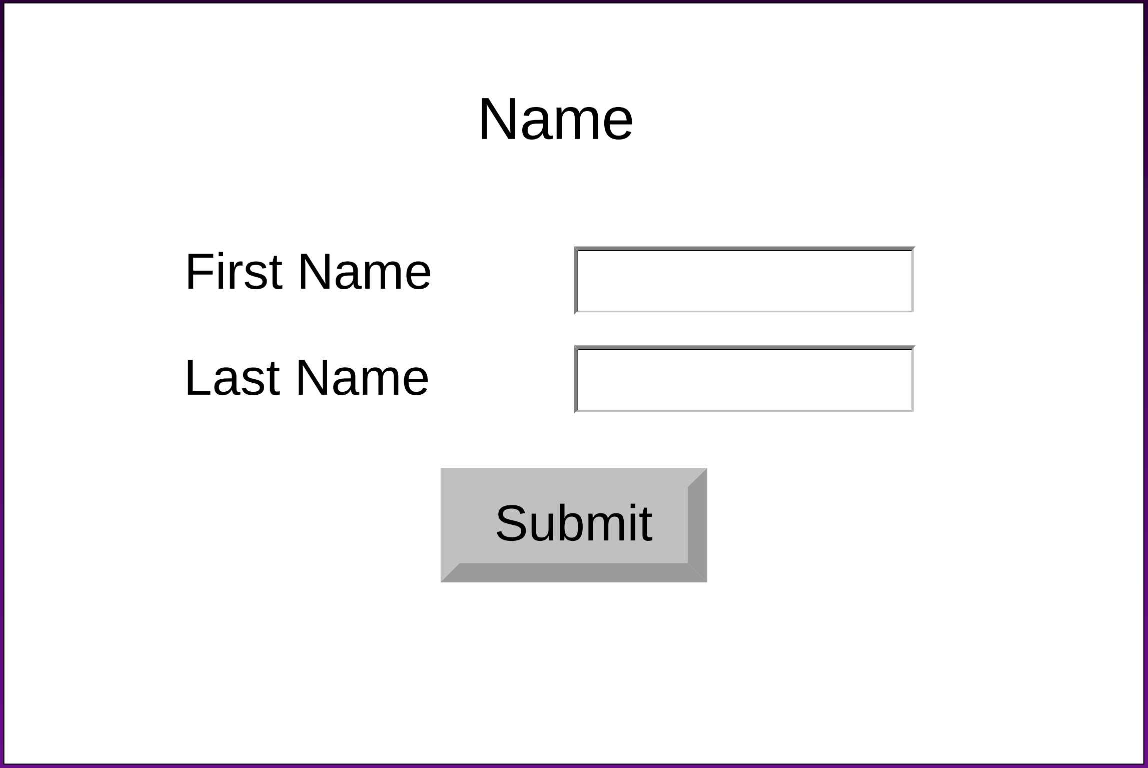
- Little code on the client
- Repository on the database
- Copy of the application state on the database
- Nothing in the application server
- Client-side code engine

Repository Data Model



Pointer to PL/SQL Program Unit

Sample Screen



XML to Create Screen

Code to build screen:

```
<actionSet Session = 12345>
<Screen ID=”1” Title="Name" Modal="Y" Position="center"
   Resize="N" Height="200" Width="440" FontData="Tahoma"
   FontLabel="Dialog" FontDataSize="11" FontLabelSize="11"
   FontDataBold="N" FontLabelBold="Y"
FontDataItalic="N" FontLabelItalic="N" FontDataColor="black"
   FontLabelColor="black">


<ScreenElement Type="Field" Value="John"  ID=”111”
Label="First Name" LabelPosition="Left" Editable="Y"
   PositionX="230" PositionY="100" Width="80"/>
<ScreenElement Type="Field" Value="Jones"  ID=”222”
Label="Last Name" LabelPosition="Left" Editable="Y"
   PositionX="230" PositionY="200" Width="80"/>
<ScreenElement Type="Button" PositionX="120" PositionY="300"
   Width="80" Label="Submit"  ID=”333” LabelPosition="Center"
   Action="Press"/>
      </Screen>
</actionSet>
```

Why does it work?

- Transmit

```
<session 12345 >
   <Button ID="10"
             Event = "Press" />
   <Field ID = "20"
             Value = "MyNewValue" />
<session>
```

- Return

```
<Actions>
   <Field    ID = "30"
             Value = "Update successful“/>
</Actions>
```

Performance Comparisons

	BRIM Web 3.0	ADF Faces
Initial load	350 KB (V0)	177 KB
Load screen	2KB	41 KB
Update screen	0.4KB	41 KB
Tree control	.1 KB – 10 KB Only changed nodes	200 KB Whole tree each time

Dulcian's BRIM® UI

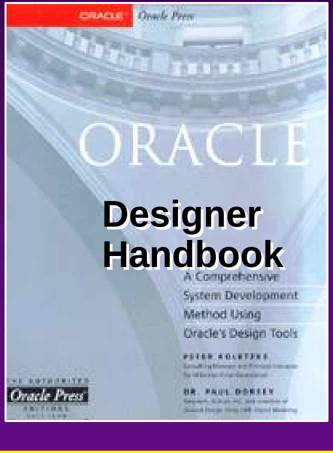
- 100% generation and maintenance of user interfaces
  - No hand coding except for views and complex routines
- 10-100X better performance
- Platform-independent
- Full client/server functionality on the web
- 90% auto-conversion from Oracle Forms

Contact Info

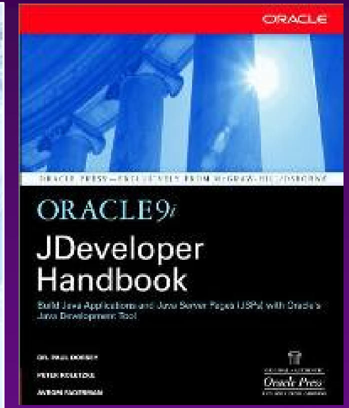
- Dr. Paul Dorsey – paul_dorsey@dulcian.com
- Michael Rosenblum – mrosenblum@dulcian.com
- Dulcian website - www.dulcian.com

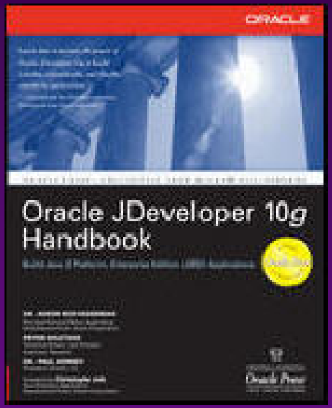
Design Using UML Object Modeling

Developer Advanced Forms & Reports

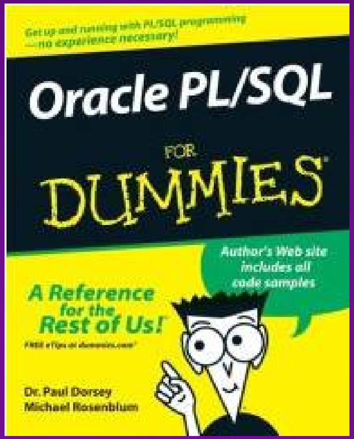

Designer Handbook

ORACLE
A Comprehensive System Development Method Using Oracle's Design Tools
Oracle Press





Available now!
Oracle PL/SQL for Dummies


Get up and running with PL/SQL programming —no experience necessary!
Oracle PL/SQL
FOR
DUMMIES
Author's Web site includes all code samples
A Reference for the Rest of Us!
Dr. Paul Dorsey
Michael Rosenblum