БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ . BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, 48
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, 48

София . 1999 . Sofia

CASE-based Development of Web-enabled Database Applications. The Promise and the Dead-end of the Relational Approach

Vassil T. Vassilev

Institute of Information Technologies, 1113 Sofia

1. Introduction: the Web-enabled information systems as the key to the Information Society

The first industrial technology, which solved successfully the complex task for automatic generation of Web-enabled information systems, is the one incorporated in **ORACLE DESIGNER/2000** product suite of Oracle Corp [1]. Its Web generator produces PL/SQL-coded database applications for running under **ORACLE WEB APPLICATION SERVER** [2] as a background process (see Fig. 1). The **Web Server Generator** [5] was added to the DESIGNER/2000 toolkit in 1996 as complementary to the client-server generators from the **ORACLE DEVELOPER/2000** suite – Forms, Graphics and Reports. It is an additional alternative together with the programming generators producing applications coded in Visual Basic and C/C^{++}, which were included earlier in the release. The Web Server Generator of DESIGNER/2000 is fully integrated with all visual diagrammers for developing applications with relational databases, which are the core of the relational concept of RAD. For the fist time they were producing fully featured and error free applications for remote Internet/intranet data management, entirely developed using graphical diagrammers, which employ the relational approach.

In order to access the databases over the World Wide Web using standard Web browsers, such as NETSCAPE NAVIGATOR or MICROSOFT INTERNET EXPLORER, the respective URL should be visited first. It is in fact a virtual URL since its entering leads to an execution of application procedures, which are stored on the database server. The structure of such a virtual link is the following:

http:\\<*website*>:<*port*>\<*database access descriptor*>\<*cartridge*>\<*server procedure*>

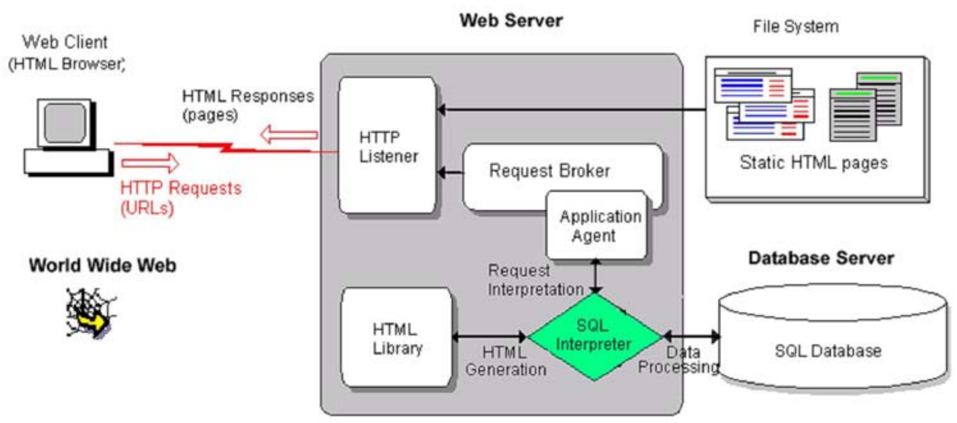


Fig. 1. Internet/intranet access to database information through Oracle Web Application Server

where the first part of the URL – http:\\<*website*>:<*port*> – specifies the Internet address of the Web server and the IP port on which the HTTP listener receives requests over WWW, while the second one is virtual only and it does not exist as a physical address. It provides specific to the WEB APPLICATION SERVER information instead. When processed it is dynamically converted into execution of respective stored database procedures; for example, the URL

http:\\owas.iinf.acad.bg:8888\owa_dba\plsql\main$.startup

points to the Web server of the Institute of Information Technologies, which is a Windows NT computer named owas.iinf.acad.bg with non-secure IP port 8888 listening to HTTP requests. Behind the curtains this virtual URL hides the fact, that the host owas.iinf.acad.bg runs the **ORACLE WEB APPLICATION SERVER FOR NT** and it also acts as a home for the **ORACLE 7 WORKGROUP SERVER FOR NT.** The HTML page supplied in response to the external requests via this URL over the World Wide Web is dynamically generated during execution of the procedure startup. It belongs to the user scheme where the PL/SQL package main is stored on the server.

Although the procedures, which WEB APPLICATION SERVER executes for generating HTML, could be written in any CGI-compliant languages – PL/SQL, Java, Perl, VRML etc., for the purpose of the CASE-based design only PL/SQL might be used. The reason for stacking to this somewhat old fashion language in the presence of much more modern, object-oriented languages like C^{++} and Java, is the clear advantage of using native database language. The new version of DESIGNER/2000 [7,4] will also allow Java to be directly generated by the Web Generator, but this will be a real advantage only when Java becomes a native language for the database server.

2. The Oracle CASE Story: 100% Authomatic Generation

DESIGNER/2000 as a CASE tool for RAD of information system is built on top of the success of RDBMS developed by Oracle and others last two decades. It is a graphical front-end to set of typical components of the design process, layered into three separate logical levels: *modellers*, *designers* and *generators*. The main two modellers from the toolkit cover both the data processing and the data management components of the design process:

Business Process Modeller, used for specification of business processes on logical level using high-level terms, such as business function, datastore, dataflow, decision point, triggering event and output report

Entity Relationship Modeller, giving the abstract model of data in terms of entities, attributes, and relations

The design layer includes several visual diagrammers, which specify the physical implementation of the information system – database and program modules - using schema diagrams and layout preferences:

Data Diagrammer is used for physical design of the database (tables, views, snapshots, etc.)

Module Data Diagrammer designs the interactive applications built on top of the existing data stored in the database (for querying, inserting, updating and deleting)

Module Logic Diagrammer helping the process of writing programming logic (procedures, functions and triggers in PL/SQL for storing on the server) and

Module Structure Diagrammer necessary for specification of the structural links between complex sets of modules, comprising the interactive applications of the information system

All the components included in the two levels are unified and tidely integrated. They work as visual diagrammers, which allow specifying the internal components, the external links and the display preferences of the diagrams in focus in a highly simplified and helpful manner. Based on the descriptions behind the visual diagrams, on the next level the respective programming generators can produce interactive applications, which implement particular information systems for processing data from the database.

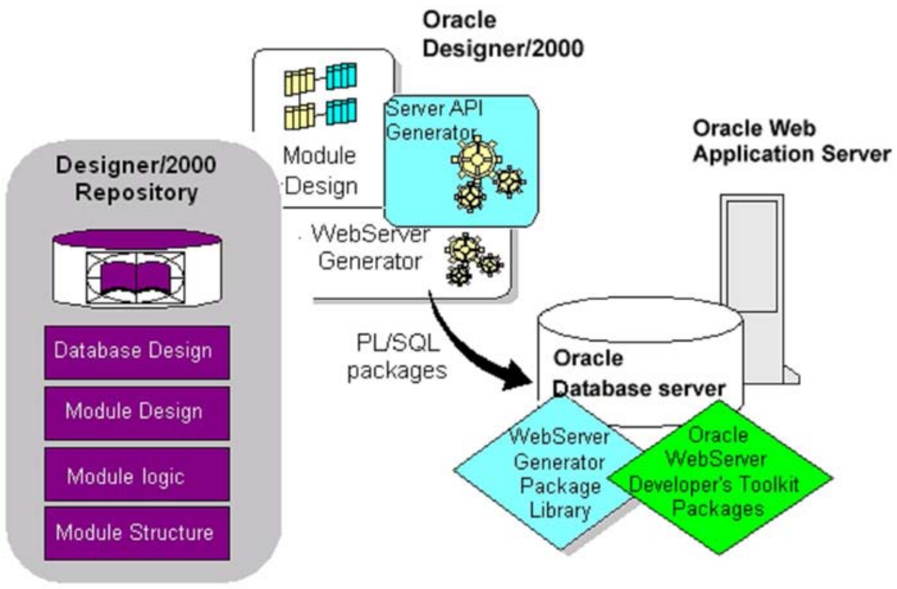


Fig. 2. CASE-based approach for development of Web-enabled database applications

Web-enabled applications could be also developed using the same CASE-based technology through the WebServer Generator (see Fig. 2). Briefly this process works in the following way. First, using the data and module diagrams prepared during the design stage, the WebServer Generator generates several PL/SQL packages for storing and manipulating data. They are then installed on the Database Server using PL/SQL interpreter. During realtime already, when the listener of the Web Application Server accepts requests for visiting particular URL's from certain Web browser, it passes them to the Web Request Broker component of the Web Application Server, which directs them to the respective cartridge for interpretation. In this case this is the PL/SQL cartridge. It interprets the code of the corresponding PL/SQL procedures stored on the database server and returns the result in the form of HTML code, which is itself returned by the HTTP server to be interpreted by the Web browser of the remote client. When this interaction includes several steps for full data processing, e.g. query-view-update sequence of actions, this procedure is repeated under the control of the client logic using standard HTML mechanisms – clicking URL's, filling forms, submitting buttons, etc.

3. An example: development for the Web using Designer/2000

Let take an example of simple office information system. It deals with information about the office departments and the staff working in them. The information will be accessed over the corporate intranet, or from the World Wide Web. On the Fig.3 is shown the screen of the Data Diagrammer with the physical data diagram of this system, converted from the corresponding Entity-Relationship model. The only two tables in this application represent the departments (DEPT) and employees (EMP) together with some relationships officially recorded by the company. They are placed in the VASSIL datascheme, and EMP_ID and DEPT_ID columns identify the records, respectively. The columns in the table DEPT which describe the information about departments are DEPT_DEPTNO, standing for its registration number, DEPT_NAME, for its name and DEPT_LOC for address information. The columns EMP_EMPNO in the EMP table stand for the number, EMP_EMPNAME for the name, EMP_JOB for the job and EMP_MGR for the duties of the employees. The two tables are related via Works-in relation (EMP_DEPT_FK is the foreign key in

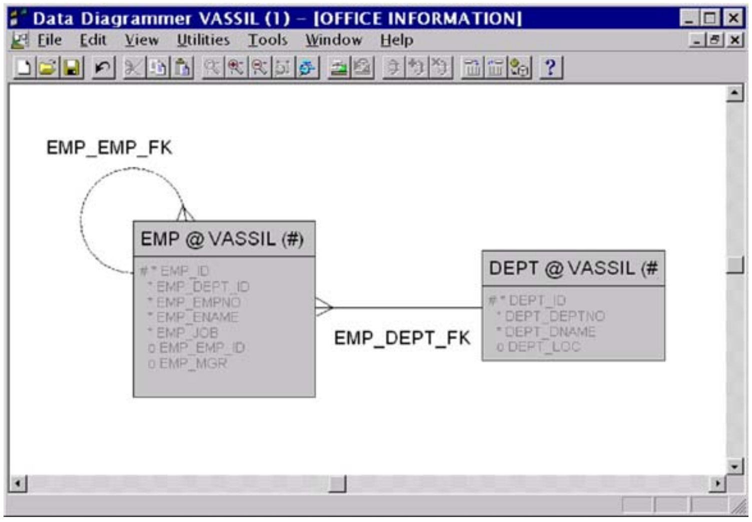


Fig. 3 Table design for simple office application

the EMP table pointing to the corresponding record in the DEPT table, while EMP_DEPT_ID is the placeholder for it in EMP). In addition, the reflective relation Supervised-by gives information about the personal chiefs of each staff member (it is denoted as EMP_EMP_FK with EMP_EMP_ID as a placeholder for it).

After completion of the database design the corresponding DDL commands could be generated by the Server Generator of the toolkit (see the listing on Fig. 4). They contain SQL statements for creation of tables, for indexing of records, for checking primary, unique and foreign key constraints, etc[1].

Immediately afterwards or, alternatively, somewhat later – they should be executed on the server which will act as a database server through standard SQL interpreter, such as SQL*Plus. Directly from the data diagram the Web Server Generator generates the PL/SQL package for accessing the database tables from WWW via the WEB APPLICATION SERVER; this package is called Application Programming Interface for that table (API). One pair of files will be automatically generated per each table in the database (Fig. 5): .pks file for the forward declarations, and the .pkb file for the procedure code. The structure of the API package CG$DEPT generated for the table DEPT of our system is listed on Fig. 6, while the code is shown on Fig. 7.

```
REM
REM  This ORACLE7 command file was generated by Oracle Server Generator
REM  Version 5.5.10.0.0 on 19-MAR-98
REM
REM For application VASSIL version 1 database VASSIL
PROMPT Creating tables…
CREATE TABLE dept(dept_id        NUMBER(10,0)    NOT NULL,
                  dept_deptno    INTEGER         NOT NULL,
                  dept_dname     VARCHAR2(240)   NOT NULL,
                  dept_loc       VARCHAR2(240)   NULL);
CREATE TABLE emp(emp_id          NUMBER(10,0)    NOT NULL,
                  emp_dept_id     NUMBER(10,0)    NOT NULL,
                  emp_empno       INTEGER         NOT NULL,
                  emp_ename       VARCHAR2(240)   NOT NULL,
                  emp_job         VARCHAR2(240)   NOT NULL,
                  emp_emp_id      NUMBER(10,0)    NULL,
                  emp_mgr         VARCHAR2(240)   NULL);
PROMPT Creating sequences…
CREATE SEQUENCE dept_seq INCREMENT BY 1
   START WITH 1 MINVALUE 1 NOMAXVALUE
   NOCYCLE      NOCACHE    NOORDER;
CREATE SEQUENCE emp_seq INCREMENT BY 1
   START WITH 1 MINVALUE 1 NOMAXVALUE
   NOCYCLE      NOCACHE    NOORDER;
PROMPT Adding Primary Key constraints to the tables…
ALTER TABLE DEPT ADD (CONSTRAINT DEPT_PK PRIMARY KEY (DEPT_ID)
                      USING INDEX PCTFREE  10);
ALTER TABLE EMP ADD (CONSTRAINT EMP_PK PRIMARY KEY (EMP_ID)
                     USING INDEX PCTFREE  10);
PROMPT Adding Unique Key constraints to the tables…
ALTER TABLE DEPT ADD (CONSTRAINT DEPT_DEPT0_UK UNIQUE (DEPT_DEPTNO)
                      USING INDEX PCTFREE  10);
ALTER TABLE DEPT ADD (CONSTRAINT DEPT_DEPT1_UK UNIQUE (DEPT_DEPTNO)
                      USING INDEX PCTFREE  10);
ALTER TABLE EMP ADD (CONSTRAINT EMP_EMP0_UK UNIQUE (EMP_EMPNO)
                     USING INDEX PCTFREE  10);
ALTER TABLE EMP ADD (CONSTRAINT EMP_EMP1_UK UNIQUE (EMP_EMPNO)
                     USING INDEX PCTFREE  10);
PROMPT Adding Foreign Key constraints to the tables…
ALTER TABLE EMP ADD (CONSTRAINT EMP_DEPT_FK FOREIGN KEY (EMP_DEPT_ID)
                     REFERENCES  DEPT (DEPT_ID));
ALTER TABLE EMP ADD (CONSTRAINT EMP_EMP_FK FOREIGN KEY (EMP_EMP_ID)
                     REFERENCES  EMP (EMP_ID));
PROMPT Creating Table Indexes…
CREATE INDEX EMP_DEPT_FK_I ON EMP (emp_dept_id) PCTFREE 10;
CREATE INDEX EMP_EMP_FK_I ON EMP (emp_emp_id) PCTFREE 10;
```

Fig. 4. DDL statements in SQL for generation of the office information system database

[1] Actually the DDL statemnts necessary for creating the database are spread among different files with the same name and different extensions – one for the the tables.tab, one for the sequences .seg,one for the constraints .con and one for the indexes .ind.

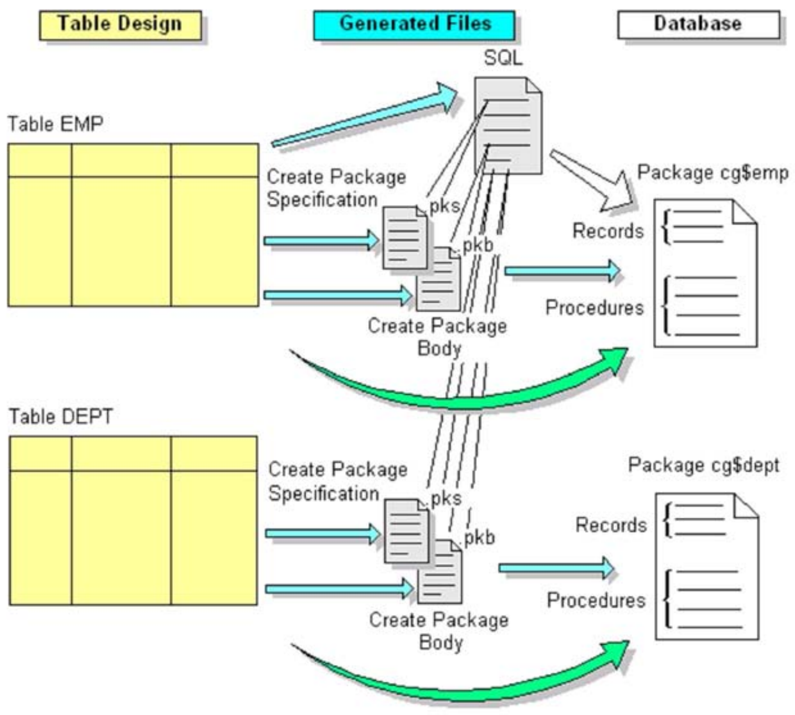


Fig. 5. Generating PL/SQL packages for Web Server Database API

```
————————————————————————————————————————
— Name:        cg$DEPT
— Description: DEPT table API package declarations
————————————————————————————————————————
CREATE OR REPLACE PACKAGE cg$DEPT IS
— Primary key of the DEPT table
DEPT_PK CONSTANT VARCHAR2(240) := ‘’;
—  Column default prompts
P2_DEPT_DEPTNO CONSTANT VARCHAR2(240) := ‘Dept No.’;
P3_DEPT_DNAME CONSTANT VARCHAR2(240) := ‘Dept Name’;
P4_DEPT_LOC CONSTANT VARCHAR2(240) := ‘Dept Location’;
— DEPT row type
cg$row DEPT%ROWTYPE;
—  DEPT row type variables
TYPE cg$row_type IS RECORD
   (DEPT_DEPTNO cg$row.DEPT_DEPTNO%TYPE
   ,DEPT_DNAME cg$row.DEPT_DNAME%TYPE
   ,DEPT_LOC cg$row.DEPT_LOC%TYPE
   ,DEPT_ID cg$row.DEPT_ID%TYPE
   ,JN_NOTES VARCHAR2(240));
TYPE cg$ind_type IS RECORD
   (DEPT_DEPTNO BOOLEAN DEFAULT FALSE
   ,DEPT_DNAME BOOLEAN DEFAULT FALSE
   ,DEPT_LOC BOOLEAN DEFAULT FALSE
   ,DEPT_ID BOOLEAN DEFAULT FALSE);
```

```
TYPE cg$pk_type IS RECORD
   (DEPT_ID cg$row.DEPT_ID%TYPE, JN_NOTES VARCHAR2(240));
```

```
— Procedures for accessing DEPT from Web Application Server side
PROCEDURE ins(cg$rec IN OUT cg$row_type
             ,cg$ind IN OUT cg$ind_type);
PROCEDURE upd(cg$rec IN OUT cg$row_type
             ,cg$ind IN OUT cg$ind_type);
PROCEDURE del(cg$pk IN cg$pk_type);
PROCEDURE lck(cg$old_rec IN cg$row_type
             ,cg$old_ind IN cg$ind_type
             ,nowait_flag IN BOOLEAN DEFAULT TRUE);
END cg$DEPT;
```

Fig. 6 Structure of the PL/SQL package for the database API

```
————————————————————————————————————————
— Name:        cg$DEPT
— Description: DEPT table API package definitions
————————————————————————————————————————
CREATE OR REPLACE PACKAGE BODY cg$DEPT IS
PROCEDURE err_msg(msg IN VARCHAR2, type IN INTEGER, loc IN VARCHAR2 DEFAULT
‘’);
————————————————————————————————————————
— Name:        err_msg
— Description: Pushes onto stack appropriate user defined error message
—              depending on the rule violated
— Parameters:  msg     Oracle error message
—             type   Type of violation e.g. check_constraint: ERR_CHECK_CON
—             loc    Place where this procedure was called for error trapping
```

<Forward specifications of other auxiliary procedures for this API package>

```
…
PROCEDURE err_msg(msg IN VARCHAR2, type IN INTEGER, loc IN VARCHAR2 DEFAULT
‘’) IS
con_name VARCHAR2(240);
BEGIN
    con_name := cg$errors.parse_constraint(msg, type);
    IF (1=2) THEN
        NULL;
    ELSIF (con_name = ‘DEPT_PK’ AND DEPT_PK IS NOT NULL) THEN
        cg$errors.push(DEPT_PK, ‘E’, ‘API’, -9999, loc);
   ELSIF (con_name = ‘EMP_DEPT_FK’ AND type = cg$errors.ERR_DELETE_RESTRICT)
THEN
       cg$errors.push(cg$errors.MsgGetText(4, cg$errors.ERR_DEL_RESTRICT,
‘Dept’, ‘Emp’),
                       ‘E’, ‘API’, -9999, loc);
    ELSE
        cg$errors.push(SQLERRM, ‘E’, ‘ORA’, SQLCODE, loc);
    END IF;
END err_msg;
```

<Body definitions of other auxiliary procedures for this API package>

```
…
————————————————————————————————————————
— Name:        ins
— Description: API insert procedure
```

```
    — Parameters:  cg$rec  Record of row to be inserted

    —              cg$ind  Record of columns specifically set

    PROCEDURE ins(cg$rec IN OUT cg$row_type,
                  cg$ind IN OUT cg$ind_type) IS
    cg$tmp_rec cg$row_type;

    BEGIN
        up_autogen_columns(cg$rec, cg$ind, 'INS'); —  Autogen + Upper
        INSERT INTO DEPT (DEPT_DEPTNO, DEPT_DNAME, DEPT_LOC, DEPT_ID)
              VALUES (cg$rec.DEPT_DEPTNO, cg$rec.DEPT_DNAME, cg$rec.DEPT_LOC,
cg$rec.DEPT_ID);
        slct(cg$rec);


    EXCEPTION
        WHEN cg$errors.mandatory_missing THEN
            validate_mandatory(cg$rec, 'cg$DEPT.ins.mandatory_missing');
            cg$errors.raise_failure;
         <Definition of other exceptional situations during inserting into the table DEPT>
         …
        WHEN OTHERS THEN
          cg$errors.push(SQLERRM, 'E', 'ORA', SQLCODE, 'cg$DEPT.ins.others');
            cg$errors.raise_failure;
    END ins;
    <Body definitions of the procedures for updating, deleting and locking of the table
DEPT>
    …
    END cg$DEPT;
```

Fig. 7. Body of the generated PL/SQL package for the table DEPT

Later on, or in parallel with the database specification, design and generation process described here, one could also develop the application for database management according to the business logic of information processing. On Fig. 8 is shown the screen of the Module Data Diagrammer containing the module design for sample office information system, which could be used for registration of the staff from the different departments.

The module has two components, which are placed on the diagram in a sequence from top to bottom. The first component contains one table usage, namely the basic usage of the database table DEPT, required for stating department before inserting any staff information. The second module contains two usages of the table EMP – one master-detail usage, corresponding to the relation *Works-in* and one lookup usage of the same table, corresponding to the reflective relation *Supervised-by.* From this diagram after clicking the proper button the Web Server Generator of DESIGNER/2000 produces three separate PL/SQL packages – one per each module component plus one master package for the whole application (see Fig. 9).

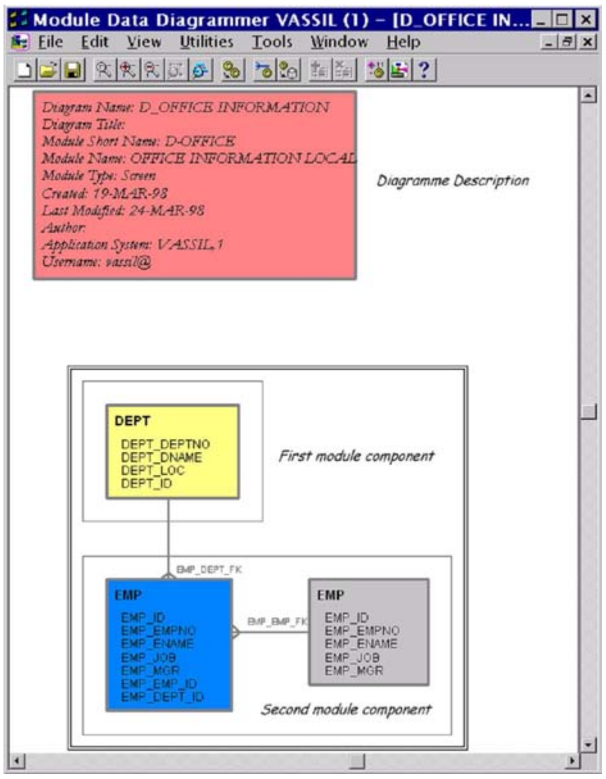

Module Data Diagrammer VASSIL (1) – [D_OFFICE IN...
File Edit View Utilities Tools Window Help

Diagram Name: D_OFFICE INFORMATION
Diagram Title:
Module Short Name: D-OFFICE
Module Name: OFFICE INFORMATION LOCAL
Module Type: Screen
Created: 19-MAR-98
Last Modified: 24-MAR-98
Author:
Application System: VASSIL, 1
Username: vassil@

Diagramme Description

DEPT
DEPT_DEPTNO
DEPT_DNAME
DEPT_LOC
DEPT_ID

First module component

EMP_DEPT_FK

EMP
EMP_ID
EMP_EMPNO
EMP_ENAME
EMP_JOB
EMP_MGR
EMP_EMP_ID
EMP_DEPT_ID

EMP_EMP_FK

EMP
EMP_ID
EMP_EMPNO
EMP_ENAME
EMP_JOB
EMP_MGR

Second module component

Fig. 8. Module design of database table usages containing both master-detail and look-up relationships

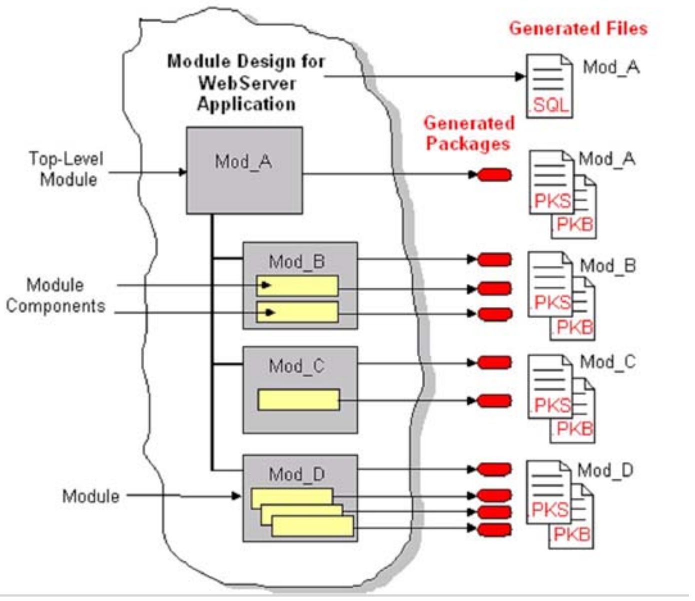


Fig. 9. Generating PL/SQL packages for Web Server Database Application

The structure of the packages is specified by the PL/SQL code generated in the .pks file for this module, produced by the Web Server Generator. It contains forward declarations of the procedures for that package (see Fig. 10). Each package has one main procedure named Startup and several other procedures corresponding to the DML statements in SQL, but customised according to the module logic. For the set of three packages generated for our office information system from module diagram OFFICE they are:

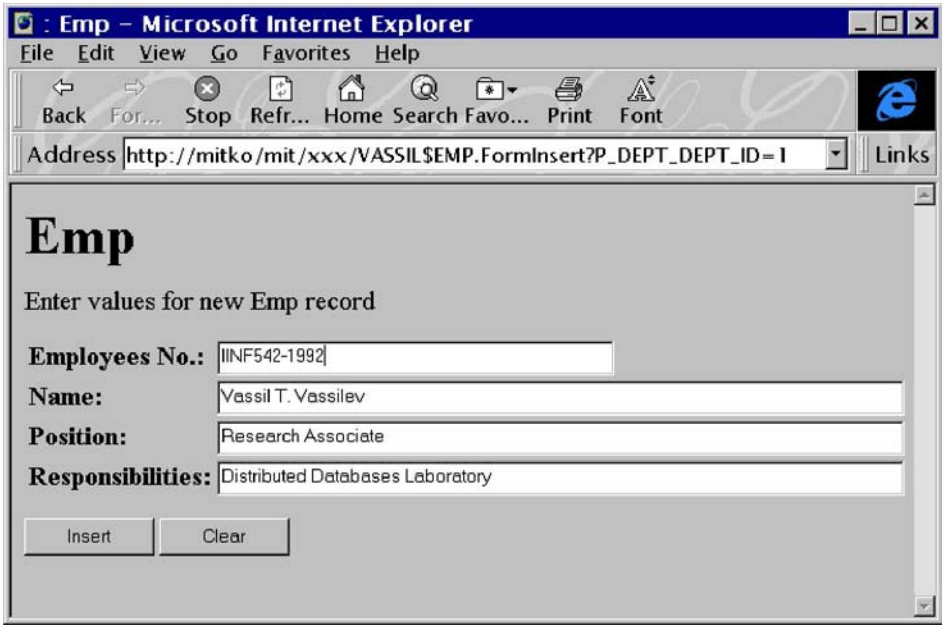
```
     create or replace package OFFICE$ is
        procedure Startup;
        procedure FirstPage;
        procedure ShowAbout;
     end;
     create or replace package OFFICE$DEPT is
        procedure Startup;
        procedure ActionQuery(
                  P_DEPT_DNAME in varchar2 default null,
                  Z_ACTION in varchar2 default null);
        procedure QueryView(
                  P_DEPT_ID in varchar2 default null,
                  Z_JUST_NON_BASE in boolean default false,
                  Z_FORM_STATUS in number default WSGL.FORM_STATUS_OK);
        procedure QueryList(
                  P_DEPT_DNAME in varchar2 default null,
                  Z_START in varchar2 default null,
                  Z_ACTION in varchar2 default null);
        procedure QueryFirst(
                  P_DEPT_DNAME in varchar2 default null,
                  Z_ACTION in varchar2 default null);
        function QueryHits(
                 P_DEPT_DNAME in varchar2 default null) return number;
     end;
     create or replace package OFFICE$EMP is
        procedure Startup(
                  P_EMP_DEPT_ID in varchar2);
        procedure FormInsert(
                  P_EMP_DEPT_ID in varchar2 default null,
                  Z_FORM_STATUS in number default WSGL.FORM_STATUS_OK);
        procedure QueryView(
                  P_EMP_ID in varchar2 default null,
                  Z_JUST_NON_BASE in boolean default false,
                  Z_FORM_STATUS in number default WSGL.FORM_STATUS_OK);
        procedure QueryList(
                  P_EMP_DEPT_ID in varchar2 default null,
                  Z_START in varchar2 default null,
                  Z_ACTION in varchar2 default null);
        procedure QueryFirst(
                  P_EMP_DEPT_ID in varchar2 default null,
                  Z_ACTION in varchar2 default null);
        function QueryHits(
                 P_EMP_DEPT_ID in varchar2 default null) return number;
        procedure ActionInsert(
                  P_EMP_EMPNO in varchar2 default null,
                  P_EMP_ENAME in varchar2 default null,
                  P_EMP_JOB in varchar2 default null,
                  P_EMP_MGR in varchar2 default null,
                  P_EMP_DEPT_ID in varchar2 default null,
                  Z_ACTION in varchar2 default null);
     end;
```

Fig. 10 Structure of the generated PL/SQL packages for the office application module

OFFICE$: Startup is the main procedure of the module. It begins the interactive session invoking the procedure FirstPage. The only work the procedure FirstPage does is to invoke proper procedures from the package corresponding to the second component, in this case **OFFICE$DEPT.**

OFFICE$DEPT: the procedure ActionQuery queries the database table DEPT using specified by the user criteria, QueryList shows the list of records which match the stated criteria, QueryFirst shows the first of them while QueryView is used for detailed description of any record in that table;

OFFICE$EMP: this package besides the procedures which are also present in **OFFICE$DEPT** package contains the procedures FormInsert and ActionInsert which organize the inserting of new records in the table EMP with information for the staff members.


: Emp – Microsoft Internet Explorer
File Edit View Go Favorites Help
Back For... Stop Refr... Home Search Favo... Print Font
Address
Links

http://mitko/mit/xxx/VASSIL$EMP.FormInsert?P_DEPT_DEPT_ID=1

Emp

Enter values for new Emp record

Employees No.:	IINF542-1992
Name:	Vassil T. Vassilev
Position:	Research Associate
Responsibilities:	Distributed Databases Laboratory

Insert Clear

Fig. 11. Working with the Office Information System over WWW using INTERNET EXPLORER

The package bodies of the packages (.pkb files) may contain also some auxiliary procedures, which are not public, but using the procedures declared in the files with forward declarations (.pks files) all the build-in functionality of the package is available. In this way, the generated packages could be used for implementing different business functions. However, only one of them is an entry point for data processing according to the designed module logic. The actual screen of a Web browser during running of the real office information system according to the above module logic at the stage of information processing specified in the second module component is shown on Fig. 11.

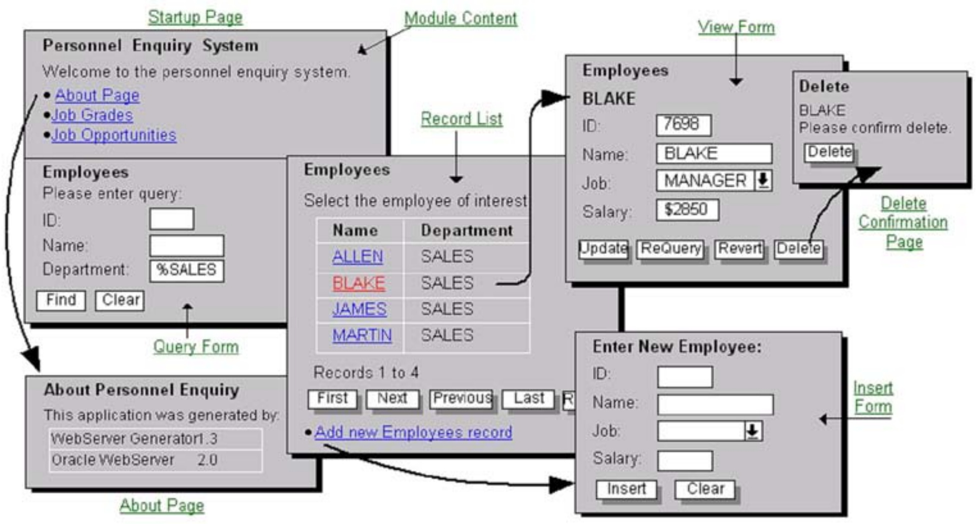
4. 100% or less?

All dynamically generated from WEB APPLICATION SERVER HTML pages, which are designed using WebServer Generator of DESIGNER/2000, belong to the following categories of interactive forms (Fig. 12):

– **Startup Page**; organises the sequence of virtual pages in one logical sequence of HTML frames or pages without data processing functions.

– **About Page**; contains only information logs with no data processing functions.
– **Query Form**; generates an interactive form for entering of combination of search criteria concerning one particular basic table usage.
– **Query First/List Form**; shows the list of records matching the entered search criteria in a table form.
– **Action View/Insert/Update Form**; shows one particular record from the list of records shown on the Record List form;
– **Delete Confirmation Page**; introduce an additional level of security for confirmation of deleting operations.

These interactive forms have been used for long time in many approaches for developing information systems before coming of the Web, e.g. in terminal-based centralised systems, in client/server distributed systems, in visual applications programmed using languages like VISUAL BASIC, VISUAL C, etc. When using the CASE tool for RAD of information systems, however, these forms could be automatically generated by the Web Generator from specifications of the table and column usages, layout preferences and display styles. They could be given completely interactively using the diagrammers and navigators included in the DESIGNER/2000 toolkit.



Personnel Enquiry System
Welcome to the personnel enquiry system.
- About Page
- Job Grades
- Job Opportunities

Employees
Please enter query:
ID:
Name:
Department: %SALES
Find | Clear

Employees
Select the employee of interest

Name	Department
ALLEN	SALES
BLAKE	SALES
JAMES	SALES
MARTIN	SALES

Records 1 to 4
First | Next | Previous | Last | R
- Add new Employees record

Employees
BLAKE
ID: 7698
Name: BLAKE
Job: MANAGER
Salary: $2850
Update | ReQuery | Revert | Delete

Delete
BLAKE
Please confirm delete.
Delete

About Personnel Enquiry
This application was generated by:

WebServer Generator1.3	
Oracle WebServer	2.0

Enter New Employee:
ID:
Name:
Job:
Salary:
Insert | Clear

Fig. 12. Dynamically generated HTML forms for Web-access to database tables

It sounds good, because these forms cover most of the known data usages in contemporary data processing, but in practice it is far from sufficient for building real Web applications although for quite different reason. Roughly speaking, the essential drawback of this approach is rooted in the way the Web Generator produces the PL/SQL packages for the designed applications from the respective module specifications (see the forward definitions of the generated procedures on Fig. 10). Each PL/SQL packages may contain procedures for generating of all the above listed data usages, but per package they could use not more than one basic table or view. This simply means, that if Your application uses data from N tables, N packages will be generated, too, and each of them will contain respective sequence of “query form-record list-view form" data usages of that table. This pattern for organising of the modules is employed here because it is especially convenient in Web setting, since this way one dynamically generated HTML form could refer to at most one basic table. From the point of view of the design process this could be regarded as very low level of data granularity. It could be much more useful, if the level of granularity changes from entire table usage to separate column usage instead. Unfortunately, it is not possible without compete revising of the generation strategy. The fact, that even 2 years after issuing of the first family of the suite the next family does not address fully this issue either, only confirms that the CASE strategy here should be completely revised.

The second fundamental shortcoming of the approach employed by DESIGNER/2000 concerns the module application logic. If we would like to program more realistic applications in accordance to the full logic, at the design level we should be able to describe the characteristics of the data manipulation module and to transfer them beyond the current computation point. But since HTTP is a stateless protocol, after completion of the data manipulation at particular virtual URL all the parameters associated with the underlying PL/SQL procedure expire. Because of this restriction the only way to extend the state beyond the current URL is to make an explicit parameter passing from the calling PL/SQL procedure to the called one. Unfortunately, the Web Generator again does not make distinction between the abstract module parameters and the programming language variables. As a consequence, the precise mapping during package generation cannot be done automatically and the module parameters stay completely useless. For this to be changed, we should be able to distinguish better the mechanisms of parameter passing between the procedures (implementation level of granularity), from the package export/import between modules (design level of granularity).

Even worse: the Web Generator does not allow during module structure design to use even very basic programming constructs besides simple sequencing and branching. The internal logic of the module also follows directly the "hardwired" patterns of query/view interactions during all data manipulation sessions. Even the very basic "if-then-else" conditional, which is necessary for implementing of the simplest logic at design level is impossible within the Designer. Here the 100% generation concept fails to respect even the basic requirements behind the design process. This means, that after finishing of the application generation we should additionally tune the intermodule logic at finer level of granularity, which in practice requires re-programming of all dynamic HTML generation afterwards, using the WEB APPLICATION SERVER PL/SQL Toolkit. For example, the only way to program cyclic sequence of dynamically generated HTML forms is to use the underlying recursive calls of the PL/SQL procedures implementing them, controlling the recursion using standard buttons. Even the new, second release of DESIGNER/2000 which is expected this year allows do not solve this issues. Although it is claimed to be possible at design time to incorporate some logic into the modules using the underlying control structures of PL/SQL [3], it does not address the intermodule logic in any way.

5. The next round: Object-oriented CASE

ORACLE DESIGNER/2000 is not a new product. Its predecessor, ORACLE CASE DESIGNER, failed to cover the entire process of development of information systems in all its aspects from business modelling, through system analysis, to software development, generation and documentation. Although the currently available version of the DESIGNER/2000 is somewhat out of date, it is still a valuable tool because of its extreme power and useful set of utilities. There are some other software tools and CASE environments, which could be used in the process of development of information systems as well, but they are either too restrictive, like **S-Designor** from Powersoft, or are oriented towards general software development, like **OMTool**, **JBuilder**, etc., to mention some. However, they cannot be directly used for RAD. Designer/2000 is still the only available tool on the market for rapid enterprise information system development which takes the full advantages of the re's – redesign, regeneration, reusability and reengineering. Unfortunately, its Web generator does not respect very well the specifics of the underlying HTTP protocol, which still dominates the WWW as a medium for remote access to databases. Even the last version, announced some time ago [3, 4] does not change the underlying problems. There are two complementary trends, which will soon change the picture.

First, the concept of *navigation* or the "drill-down" metaphor plays an important role here. It has recently re-appeared in software engineering as a keyword in CASE setting [4]. It could rather soon replace the concept of relating entities, the core of the relational point of view, with the concept of focusing on an object under investigation, the insight of the object-oriented paradigm. Although not new in the information retrieval research, this concept is much better placed within the contemporary object-oriented paradigm than the "entity-relationship" approach. This technological trend will be enforced more significantly with introduction of the new family of object-relational database management systems, based on **POSTGRESS** and **GEM** models [10], already incorporated in the new families of industrial DBMS like **ILLUSTRA** from Informix and **Oracle 8** from Oracle itself. These systems only extend the relational mechanisms, widely used in the contemporary relational databases, using subtyping and classification mechanisms, but this allows also building truly hypertext structure of the discourse of data processing.

Second, the *component approach*, which promises significant change in the face of the contemporary software engineering, will probably reach better level of granularity of the building blocks to be generated by the CASE tools then the concrete relations. The intensive growth of Java as a programming language for Web development and CORBA as an industrial standard for building distributed applications using component structures over distributed networks are the most important milestones of this change. Oracle, Inc. as a leading industrial software company for information systems development, has tried to tune its global strategy in order to reflect these changes [5]. But the new Designer/2000 [6] fails to make this transition in a smooth and consistent way due to its very relational nature, and its improvements are not substantial from this point of view.

6. Conclusion

The object-oriented paradigm is more than 25 years old, and 10 years old is the World Wide Web itself. Nevertheless, only few CASE based RAD tools on the software market addresses this combination of technologies in an adequate way. The new industrial CASE tools for RAD of information systems, expected in days after invention of the new development environments like **Java AppBuilder** from Oracle itself [9], as well as the analogous tools from other companies, like **JavaStudio**, for example, will be based on completely new set of building blocks. They will be more flexible to reflect the natural object-orientation of the software development technologies and they will probably use industrial standards – e.g. , JavaBeans could be used as a component library standard, IIP as an abstract brokerage service protocol, JDBC as an non-native protocol for accessing relational databases, etc. The long waited appearance of such tools on the market will probably end up the relational domination in the CASE domain.

References

1. D o r s e y, P., P. K o l e t z k e. Oracle Designer/2000 Handbook. Berkeley: McGraw-Hill, 1997.
2. G r e e n w a l d, R. Using Oracle Web Application Server 3. Indianapolis. — In: Que Corp., 1997.
3. P i r i e, M. Designer/2000 Release 2.0: Rapid Model-Driven Development. — Oracle Magazine, Vol. XI (6), 1997, 15-16.
4. G w y e r, M. Using Designer/2000 for 100% Generation of Advanced Web Applications. — In: Proc.Oracle Developer Conference, Stockholm, 5-6 March 1998. URL http://technet.oracle.com~/events/wd104.pdf.
5. Oracle, Corp. Application Development for the Web.— Oracle Developer Programme Technical Report , 1997. URL http://technet.oracle.com~/product/tools/des2k/info/appdev.pdf.
6. G w y e r, M. Oracle Designer/2000 WebServer Generator Technical Overview. —Oracle White Paper, February 1996. URL http://technet.oracle.com~/product/tools/des2k/info/wwwgen.pdf.
7. S t o w, S., M. P i r i e. Designer/2000 Release 2.0 Product Overiew. — Oracle White Paper, March 1997. URL http://technet.oracle.com~/product/tools/des2k/info/des2k2.0.pdf.
8. Power Soft, Inc. The Drill Down Metaphor. — Powerbuilder Journal, June 1996. URL http://www.powersoft.com~/pbj-06-96.pdf.
9. L o e n n r o t h, M. Building Database Applications in Java. — In: Proc. Oracle Developer Conference, Stockholm, 5-6 March 1998. URL http://technet.oracle.com~/events/jo102.pdf.
10. Readings in Object-Oriented database Systems. S. Zdonik, D. Maier (eds.). San Mateo, CA:Morgan Kauffman, 1990.

Применение систем автоматизированного проектирования (САПР): возможности и ограничения реляционного подхода

Васил Т. Василев

Институт информационных технологий, 1113 София

(Р е з ю м е)

Представлен анализ софтверных технологий для автоматизированной разработки информационных систем доступа к базам данных при помощи Интернета. На примере информационной системы, использующей реляционную базу данных как основу управления бизнес-информацией, демонстрируются удобства САПР. Показан специфический для фирмы ORACLE подход.