Навчальна дисципліна

БАЗИ ДАНИХ

Лектор - к.т.н., доцент

Баклан Ігор Всеволодович

Site: baklaniv.at.ua

E-mail: iaa@ukr.net

2016-2017

Лекція №6.

Проектування користувацького інтерфейсу бази даних

- Introduction
- Deciding on the User Interface
- Steps in the User Interface Design
- User Interface Development and Implementation
- Summary and Concluding Remarks

6.1 Introduction

At this stage we have settled on all relations (and attributes). Remember, we implement relations, tuples and attributes as files, records and fields.

Designing the user interface to facilitate user access is the next step. The user interface should facilitate at least the following basic functions: data insertion, data update, data deletion, and information retrieval (query and print). This is important, as it is not acceptable to give end users direct, unfettered access to the database; were this to be done, the integrity of the system would be compromised in very short order. What is more desirable is to provide the end users with a user friendly, controlled environment that gives users all the privileges and functionalities that they need and nothing more.

The user interface will consist of menus from which user operations can be accessed. Depending on the software development tool used, it will be constructed from various building blocks, and there will be various categories of user interface objects (review your software engineering notes).

The system must also facilitate various user (external) views through logical interpretation of objects. This must be developed using the DBMS and/or whatever software development tool is being used. Note that if the O/ESG methodology (discussed in section 5.8) is employed, you will be well on your way with the user interface specification.

Example: By the way of illustration, let us revisit the O/ESG for the partial database specification of the manufacturing firm, discussed in the previous chapter (section 5.8). Figure 6-1 shows a repeat of the O/ESG for the **Employee** entity. According to the figure (adopting the conventions from section 5.8), this entity could be implemented as a relational table named RMEmployee_BR. The user interface should anticipate and support various logical views on this relational table. Following are some examples:

- Employees arranged by Name
- Employees arranged by Telephone Numbers
- Employees arranged by Departments
- Employees arranged by Social Security Number

E2 – Employee [RMEmployee_BR]

Attributes:

01. Employee Identification Number [Emp#] {N7}
02. Employee Last Name [EmpLName] {A20}
03. Employee First Name [EmpFName] {A20}
04. Employee Middle Initials [EmpMInit] {A4}
05. Employee Date of Birth [EmpDOB] N8
06. Employee's Department [EmpDept#] {N4} **Refers to E1.Dept#**
07. Employee Gender [EmpGender] {A1}
08. Employee Marital Status [EmpMStatus] {A1}
09. Employee Social Security Number [EmpSSN] {N10}
10. Employee Home Telephone Number [EmpHomeTel] {A14}
11. Employee Work Telephone Number [EmpWorkTel] {A10}

...

Comments:

This table stores standard information about all employees in the organization.

Indexes:

1. Primary Key Index: RMEmployee_NX1 on [01]; constraint RMEmployee_PK.
2. RMEmployee_NX2 on [02, 03, 04]
3. RMEmployee_NX3 on [09]
4. RMEmployee_NX4 on [10] or [11]

Valid Operations:

1. Manage Employees [RMEmployee_MO]

1.1 Add Employees [RMEmployee_AO]
1.2 Update Employees [RMEmployee_UO]
1.3 Delete Employees [RMEmployee_ZO]

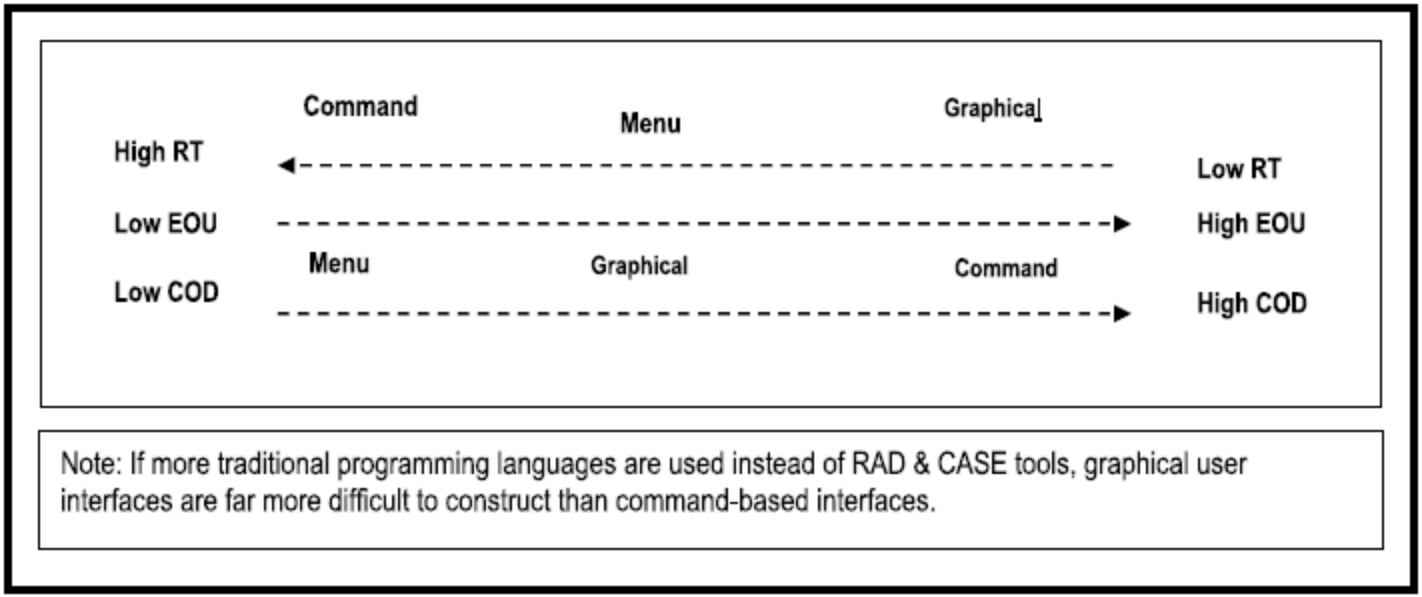
2. Inquire on Employees [RMEmployee_IO]
3. Report on Employees [RMEmployee_RO]

Figure 6-1. *Excerpt from the Partial O/ESG for Manufacturing Environment*

6.2 Deciding on User Interface

User interfaces can be put into three broad categories — m*enu-driven interface, command interface* and g*raphical user* interface (GUI). Figure 6-2 compares the approaches in terms of relative complexity of design (COD), response time (RT), and ease of use (EOU).

Command entry interfaces are the oldest type; they typify traditional operating systems, compilers and other software development tools. Up until the mid-1990s, menu driven interfaces were the most frequently used, dominating the arena of business information and application system. Since the late 1980s, graphical interfaces have become very popular, and clearly dominate user interfaces of the current era. Of course, the approaches can be combined.



Note: If more traditional programming languages are used instead of RAD & CASE tools, graphical user interfaces are far more difficult to construct than command-based interfaces.

Figure 6-2. *Comparison of User Interface Categories*

6.3 Steps in User Interface Design

How you design the user interface will depend to a large extent on the type of user interface your software requires, It will also depend on the intended users of the software (experts, knowledgeable intermittent, or novices).

6.3.1 Menu or Graphical User Interface

If the user interface is to be menu driven or graphical, the following steps are recommended (assuming object-oriented design):

1. Put system objects (structures and operations) into logical groups. At the highest level, the menu will contain options pointing to summarized logical groups.
2. For each summarized logical group, determine the component sub-groups where applicable, until all logical groups have been identified.
3. Let each logical group represent a component menu.

4. For each menu, determine options using an object-oriented strategy to structure the menu hierarchy (object first, operation last).

5. Design the menus to link the various options. Develop a menu hierarchy tree or a *user interface topology chart* (UITC).

6. Program the implementation.

Figure 6-3 illustrates a partial user interface topology chart (UITC) for a college/university administrative information system (CUAIS). The UITC displays the main user interface structure for the system; it is fully discussed in [Foster, 2010]; however, it is intuitive enough for you to understand it. The CUAIS project is also described in [Foster, 2010]; like the UITC, a full discussion is not necessary here.

Figure 6-3 illustrates a partial user interface topology chart (UITC) for a college/university administrative information system (CUAIS). The UITC displays the main user interface structure for the system; it is fully discussed in [Foster, 2010]; however, it is intuitive enough for you to understand it. The CUAIS project is also described in [Foster, 2010]; like the UITC, a full discussion is not necessary here.

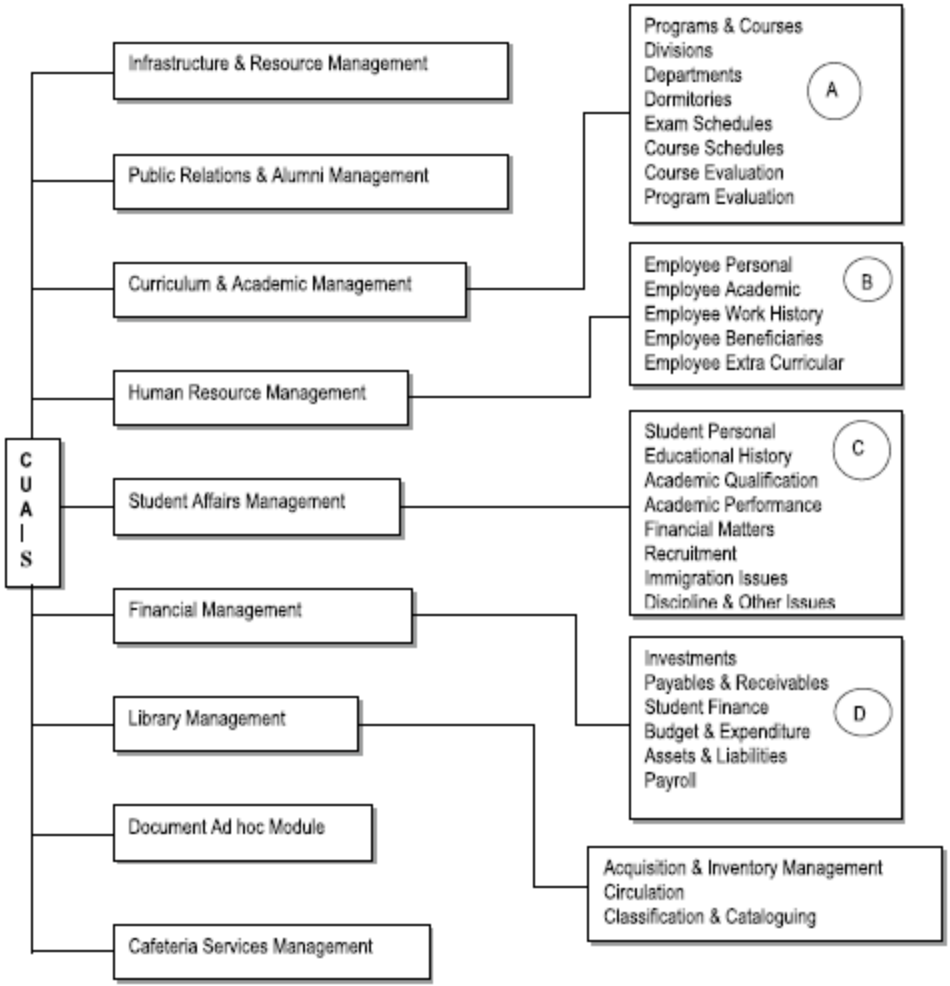


Figure 6-3. Partial UITC for a CUAIS Project

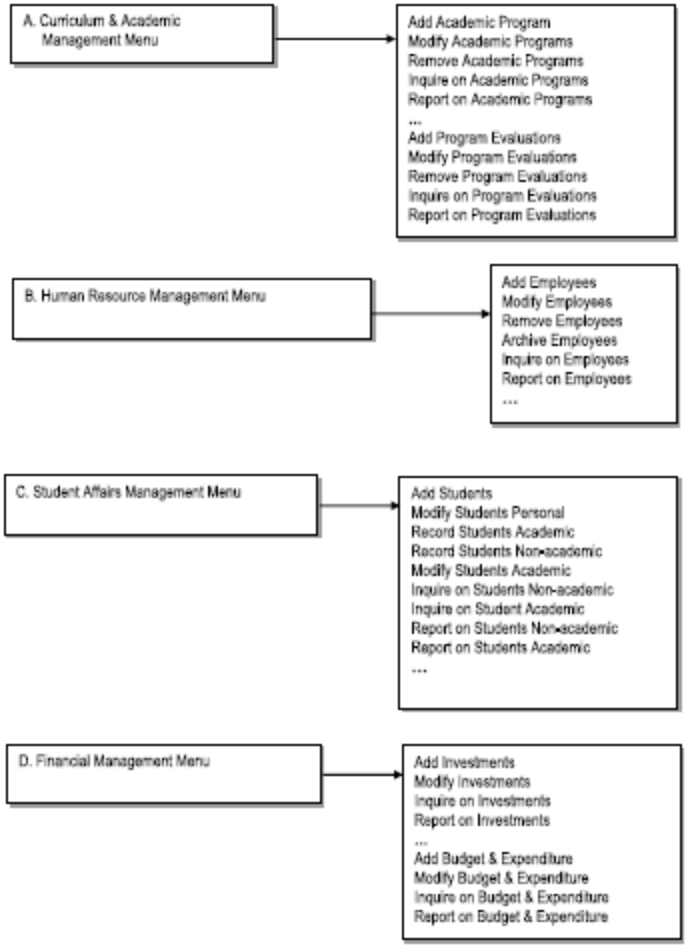


Figure 6-3. *Partial UITC for a CUAIS Project (continued)*

6.3.2 Command-Based User Interface

If the user interface is command-driven, the following steps are recommended:

1. Develop an *operations-set* i.e. a list of operations that will be required.
2. Categorize the operations — user operations as opposed to system operations.
3. Develop a mapping of operations with underlying database objects.
4. Determine required parameters for each operation.
5. Develop a list of commands (may be identical to operations set). If this is different from the operations set, each command must link to its corresponding system operations.
6. Define a syntax for the command.
7. Develop a user interface support for each command (and by extension each operation). This interface support must be consistent with the defined command syntax.
8. Program the implementation of each operation.

6.4 User Interface Development and Implementation

Designing, constructing and implementing the user interface really belongs to the realm of software engineering, not database systems. However, as you are aware (and as has been emphasized in this course), the two fields are closely related. In order to construct the user interface, you will need to have an appropriate set of software development tools. Of course, development and testing of the user interface must proceed according to established software development standards.

The software development tool used to develop the user interface will depend to a large extent on the user requirements (another software engineering matter). Figure 6-4 provides some possible scenarios.

Scenario	Solution Alternative
1. It is desirable to have front-end and back-end based on the same DBMS.	Use a DBMS suite that provides facilities for both front-end and back-end systems. Oracle, DB2 and Informix are excellent examples.
2. Front-end and back-end can be based on different software development tools.	The alternative of an object-oriented RAD tool, superimposed on a relational or object database has become the norm. Products such as DB2, Oracle, Informix, Sybase and MS SQL Server are good candidates for back-end systems. DB2, Oracle, and Informix are regarded as universal databases, supporting both relational and object databases. With respect to the front-end system, your choice of development tool will depend on whether or not web access is critical.
3. Front-end must support web access.	Products such as Cold Fusion, Poet Software, Delphi, WebSphere, etc. are very popular.
4. Front-end need not support World Wide Web (WWW) access.	Earlier versions of Delphi, C++ Builder, Visual Basic, etc. will suffice.
5. A purely object oriented environment is desired.	The product Rational Rose comes readily to mind.

Figure 6-4. *Possible Scenarios for User Interface*

It must be emphasized that user interface design and development can and often occur independent of database design and development. This is one of the potent results of data independence (review sections 1.2 and 2.9): the user interface applications are immune to structural and/or physical changes in the database. In fact, as mentioned in section 2.9, the user interface (which is part of the front-end system) may reside on a different machine (with a different operating system) from the actual database (which is part of the back-end system). So in addition to data independence, we can have *platform independence*.

Two prominent protocols that facilitate platform independence between database and user interface are *open database connectivity* (ODBC) and *Java database connectivity* (JDBC).

Open Database Connectivity: ODBC is an open standard *application programming interface* (API) for accessing a database. A software product that desires to access an external database must include in its suite, an ODBC driver for that database. The ODBC driver converts the database objects into a generic format that is understood by the software. The target DBMS must also support ODBC. This facilitates communication and transfer of data among heterogeneous databases, irrespective of the platform that they reside on. Microsoft is a strong proponent of ODBC; in fact, the ODBC software is typically bundled with the Windows operating system (under Control Panel Ȥ Administrative Tools). ODBC is also supported by other leading operating systems (Unix, Linux, Windows, System i, etc.).

Java Database Connectivity: JDBC is a Sun Microsystems product that allows Java programs to access heterogeneous databases, irrespective of their platforms. This API is included in J2SE and J2EE releases. JDBC cooperates with the ODBC protocol; as such, a program running JDBC can reach ODBC-accessible databases.

ODBC and JDBC may be considered as subsets of the wider set of protocols described as Common Object Request Broker Architecture (CORBA). CORBA will be further discussed in chapter 22 (section 22.6). For the purpose of illustration, Figure 6-5 provides a summary of the steps you would take in order to configure an Oracle database server to be accessed from Delphi 7.0 (or some later version) through ODBC (assuming a Windows environment).

1. Install Oracle DBMS Server on the database server. This installation will automatically include installation of Oracle's ODBC driver.
2. Install Oracle DBMS Client on client machine(s). This typically includes components such as Oracle Net Manager, Network Transport, and Oracle ODBC Driver.
3. Install front-end system (e.g. Delphi) on client machine(s).
4. On each client machine:
   a. Configure the Network Client Access file (Oracle **C:\Oracle\ora10g\network\admin\tnsnames**) to include a service that connects to the database on the DB Server.
   b. Configure ODBC (via the Control Panel) to connect to the DB server through the client service established in 4a.
5. In your front-end system (Delphi), select **Database** from the main menu; then select from the list, the DB service name established in 4b, and log on the foreign DB server.
6. From this point, you can now create front-end (Delphi) datasets and data sources that connect to the foreign DB server through the (local) Service name established above.

Figure 6-5. *Accessing an Oracle Database from Delphi via ODBC*

6.5 Summary and Concluding Remarks

Here is a summary of what we have covered in this chapter:

- The user interface for a database system should provide the end users with a user friendly, controlled environment that gives users all the privileges and functionalities that they need and nothing more.
- In planning the user interface, you must first decide what type of interface will be provided. The interface may be command-based, menu-driven, or GUI-based.
- Next, you should design the user interface using established principles of user interface design.
- The final step is the development and implementation of the user interface. Features that the user interface will provide will influence the tools used in developing the user interface. For instance, if the database is to be accessible from the WWW, then the tools used must facilitate such capability.

So now you know how to design a database and a user interface for that database. Later in the course, you will learn Structured Query Language (SQL), the universal database language. In preparation for this, the next two chapters discuss the relational algebra and relational calculus respectively — two subjects areas that will enhance your appreciation of SQL.