DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/91M-01

TRANSFERING 4GL APPLCATIONS FROM INGRES TO ORACLE

THESIS

Adnan Altunisik
First Lieutenant, Turkish Air Force

AFIT/GCE/ENG/91M-01

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1991	*Master's Thesis*

4. TITLE AND SUBTITLE
Transfer of 4GL Application from INGRES to ORACLE

5. FUNDING NUMBERS

6. AUTHOR(S)
Adnan Altunisik, 1 Lt Turkish AF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/91M-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Col Thomas Yax
AU/CADRE/WG
Maxwell AFB, AL 36112

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Abstract

This paper documents the transfer of 4GL applications from INGRES to ORACLE. As a result of a fast and conceptual change in computer languages, 4GLs were developed and evolved. These are programming support tools whose goal is, basically, to make the programs more efficient by reducing the number of instructions. Both ORACLE and INGRES database systems use 4GLs to develop applications. Their 4GL environments and their facilities for application development are investigated and explained in depth in this effort. The TWX application, which was originally implemented in INGRES by using its 4GL, is designed and reimplemented in ORACLE. This is accomplished in ORACLE SQL*Forms. Totally eight forms are built to accommodate the new application. The data, itself, was easily transferred to ORACLE apriori. The TWX application now runs on a Sun386*i* stand-alone computer interactively.

14. SUBJECT TERMS	15. NUMBER OF PAGES
4GL, INGRES RDBMS, ORACLE RDBS, SQL*Forms, TWX Application	85
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - DOD - Leave blank
DOE - DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA - NASA - Leave blank
NTIS - NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

AFIT/GCE/ENG/91M-01

TRANSFERING 4GL APPLCATIONS FROM INGRES TO ORACLE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Adnan Altunisik, B.S.

First Lieutenant, Turkish Air Force

March, 1991

Acknowledgments

I would like to thank my advisor, Maj. Mark Roth, for the guidance and support through the course of this thesis effort. I would also thank my committee members, Dr. Thomas Hartrum and Dr Henry Potoczny. Their critiques and inputs turned on lots of lights upon this effort.

I would like to thank all the friends in the Oracle Contractor Area, especially David Roliff, for their support and 'patience' on the design phase.

I would also thank all the AFIT personnel, faculty members, and my classmates. They helped me to overcome the AFIT challenge and made these seven quarters an extraordinary experience for me.

Last, but not the least, I would like to thank my family for supporting me through these two hard years. My mother Gulten, my father Hikmet, and my brother Adil did a hard-to-believe job by coming to AFIT all the way from Ankara, Turkey to show their support.

Adnan Altunisik

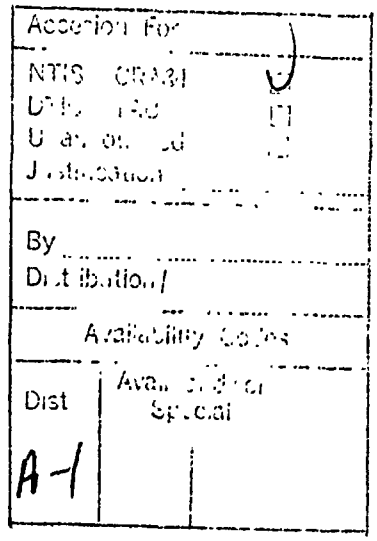


Table of Contents

List of Figures

List of Tables

AFIT/GCE/ENG/91M-01

Abstract

This paper documents the transfer of 4GL applications from INGRES to ORACLE. As a result of a fast and conceptual change in computer languages, 4GLs were developed and evolved. These are programming support tools whose goal is, basically, to make the programs more efficient by reducing the number of instructions. Both ORACLE and INGRES database systems use 4GLs to develop applications. Their 4GL environments and their facilities for application development are investigated and explained in depth in this effort. The TWX application, which was originally implemented in INGRES by using its 4GL, is designed and reimplemented in ORACLE. This is accomplished in ORACLE SQL*Forms. Totally eight forms are built to accommodate the new application. The data, itself, was easily transferred to ORACLE apriori. The TWX application now runs on a Sun386*i* stand-alone computer interactively.

TRANSFERING 4GL APPLCATIONS FROM INGRES TO ORACLE

I. Introduction

A necessary revolution is taking place in computer languages. Today, the programmers need to be able to instruct computers more easily and more quickly.

Any set of estimates of future computing power indicates that the productivity of application development must increase by at least two orders of magnitude over the next ten years (1).

As computers spread, many people who are not data processing professionals must be able to put computers to work. Application development without professional programmers is becoming a vigorous trend in computing (8).

End users should also build their own computer applications. They need languages that are easy to use and do not require the memorization of mnemonics, formats, sequences, and complex constructs.

The new generation of computer languages, then, needs to be much more powerful than the previous generation so that results can be obtained much faster. These languages are referred as fourth generation languages (4GLs).

The objectives of 4GLs (1) are to

- Greatly speed up the application-building process
- Make applications easy and quick to change, hence reducing maintenance costs
- Minimize debugging problems
- Generate bug-free code from high-level expressions of requirements
- Make languages easy to use so that end users can solve their own problems and put computers to work

4GLs need far fewer lines of code than would be needed with languages like COBOL, PL/1, and Ada to perform the same functions. They might be referred to as high-productivity languages. 4GLs vary greatly in their power and capabilities. Some are merely query languages; some are report generators or graphics packages; some can be used to create complete applications; some are very high level programming languages; some are highly restrictive in their range; others can handle a variety of applications as well. 'In the fourth generation, much more than in the third, the languages have to be selected to fit the application' (1).

In the scope of this thesis, 4GLs are reviewed in general. The focus in this review is on two different 4GLs, namely, INGRES and ORACLE.

The rising trend of ORACLE has been waiting to be tested and proven to have one of the best 4GLs in it. This development on the market had an effect on the Air Force as to try ORACLE's 4GL. An application first developed in INGRES, namely, the Theater War Exercise (TWX), is transferred from INGRES to ORACLE in this thesis effort.

1.1 Background of TWX

The Theater Warfare Exercise is a two sided, theater level, air power employment decision making exercise conducted by the Air Force Wargaming Center. The decisions, once made by the exercise participants, are fed into TWX's air and land battle simulation programs, which then simulate the employment of the air power. The players receive the battle results, air/land orders of battle, logistics, status, weather forecasts, and other information from the computer.

The requirement for an exercise such as TWX originated in 1976 when the current, USAF Chief of Staff directed the development of 'rigorous courses of study instructing operator and planners in the threat and application of force'. To accomplish this task, TWX was developed in Air War College between 1976 and 1977.

The TWX databases and algorithms were intentionally drawn from unclassified sources and are, therefore, only representative of real world force postures and capabilities. The bottom line is TWX provides the educational opportunity to employ airpower

strategies and doctrine and the principle of war in a simulated, but hopefully realistic, situation.

From 1977 to 1987, the TWX was run on a Honeywell 6000 series mainframe computer. The interaction between the game controller and the user was very limited. In 1987, two Air Force Institute of Technology (AFIT) students, Michael Brooks (2) and Mark Kross (7), rehosted the exercise from Honeywell to a DEC Micro Vax 3600 series computer using Zenith Z-158 microcomputers as remote terminals.

Michael Brooks and, later, Ken Wilcox (20) rewrote the game controller to allow multiple seminars of the game to be controlled concurrently and redesigned the database from application-specific files to a commercially available INGRES relational database management system.

Kross developed a new input interface on the Z-158's using a fourth generation language provided by INGRES. The new interface allowed immediate validation, feedback, and forecast on the map.

In 1988, another AFIT student, Darrell Quick (16), developed a map-based graphical display to replace old computer printouts. Again, in 1989, Peter J. Gordon went into more detail in the development of the graphical player interface. In his effort, the TWX graphic output display interface and the form-based input interface were combined into one interactive interface (5).

The TWX is currently running on a DEC Micro Vax 3600 series computer. The data manipulation for the game is accomplished through INGRES RDBMS. The user interface is written in INGRES fourth generation language (4GL).

1.2 Problem

Another database, ORACLE, has a rising trend on the database market in the last decade. It has new features, two of which are its procedural structured query language (PL/SQL) and fourth generation language.

The sponsor has requested that the ORACLE RDBMS be substituted for the INGRES RDBMS in this application. The relational data, itself, is easily transferred. How-

ever, the 4GL applications built with INGRES need to be reimplemented in ORACLE's forms-based system. This effort will determine a baseline for 4GL comparison in general and be an opportunity to compare and contrast the ORACLE and INGRES 4GLs.

The possible benefits from this thesis effort include:

- Development of a canonical set of 4GL functions which can be used to compare other systems
- Development of automated methods to transfer INGRES 4GL to ORACLE 4GL.

1.3 Approach and Methodology

The implementation of the TWX in a new environment, ORACLE RDBMS, has never been realized before. This new initiative is accomplished in a systematic approach. The approach taken to reach the solution space is as follows:

1. The two database systems are compared and contrasted by their 4GLs, preceded by a detailed 4GL discussion.
2. The application that was implemented in INGRES RDBMS is reviewed.
3. The application that will be built in ORACLE is planned and designed.

1.4 Sequence of Presentation

Chapter II is a review of fourth generation languages (4GL) in general. The database-4GL connection, 4GL environment facilities, and an analysis for high-performance 4GL/RDBMS application requirements are some sections which are included in this chapter. Chapter III is where the ORACLE and INGRES 4GL environments are reviewed in detail. In order to create an application, the tools that are going to be used in these applications and how they work are explained in this chapter. In Chapter IV, the discussion is about the implementation of TWX in the ORACLE RDBMS. However, instead of explaining the implementation details, in this chapter, the focus is rather on creating an application in ORACLE SQL*Forms, using examples chosen from the TWX application.

Finally Chapter V is the section which concludes the thesis. Here, the work is summarized and criticized by considering the objectives set forth in this chapter.

II. Fourth Generation Languages

2.1 Fourth Generation Languages in General

There is a fast and conceptual change in computer languages. Today, more than ever, computers need to be programmed more practically and quickly. There are, mainly, two reasons for this:

1. Increasing computer power
2. Application development by end users

The first reason is directly related to an issue called *software versus hardware costs.* Hardware used to be expensive but not so complex. Now, it's considerably less expensive, however, more complex. As with the software trend, it continues to be labor-intensive.

The second reason refers to new computer users. Nowadays, not only professional programmers, but also end users are becoming increasingly involved in application development.

There has evolved a wide language spectrum, starting from the first generation languages. Table 2.1 summarizes orientations, uses, and product examples for each language level in this spectrum.

In addition to languages, new development tools evolved to meet the software productivity requirements. These tools are in the category *automated software development tools*, which also include programming support tools, design technique tools, and project management tools. Among these, programming support tools automate the process of writing applications.

Fourth Generation Languages are programming support tools whose goal is, basically, to make programs more efficient by reducing the number of instructions. To accomplish this, 4GLs can (6:14-15):

- Help with application definition.

Table 2.1. Summary of the Languages Spectrum(6:23)

Generation	Level	Orientation	Uses	Examples
First & Second (Pre-1950s & 1960s)	Machine Assembler (Minimal overhead)	Hardware dependent (problem → algorithm → machine operations → coding)	• For frequent highly efficient use, for example, compilers	proprietary machine and assembly language
Third (1960s & 1970s)	Algorithm (Medium overhead)	Hardware independent (problem → algorithm → code)	• Numeric calculations • Business use • General-Purpose applications • Special-purpose applications	ALGOL COBOL PL/I PASCAL FORTRAN BASIC
Fourth (1980s)	Between algorithm and problem (High overhead)	Mostly hardware independent (problem → high level algorithm → code)	• End-user computing • Decision support systems • Information center language • Rapid system development	NOMAD EXPRESS IFPS IMAGINE
Fifth (1980s & 1990s)	Object (Medium overhead)	Hardware dependent (LISP machine), but mostly hardware independent (Object → code)	• Commercial artificial intelligence systems • Expert systems • Object-oriented database management systems	LISP PROLOG GEMSTONE

- Permit definition of data in terms meaningful to the user.
- Permit the entry, modification, and deletion of data either interactively or from existing files on traditional storage media.
- Provide simple ways to specify the reports that are to be derived from the stored data with flexible formats, useful summaries, and ways to make only subsets of data available to users not authorized to see all of it.

4GLs are beneficial for the development of systems whose program flow will change frequently. This change can be in calculation, in report, or in program logic. Substantially, they reduce the complexity of development cycles and database inquiry systems making software developers more productive. Thus, they allow software developers to review system models with end users to verify such items as format and functionality of the system functions and input/output formats.

One of the functions of 4GLs, which is important in this context, is to generate applications. With 4GL, the application creator can specify how files or databases will be updated, what calculations and logic are performed, and what output is achieved. Machine

performance is often a concern with routine data processing. Application generators need compilers that create optimized code modules and organize the data access as efficiently as possible (6:20).

2.2 Databases and 4GLs

In Figure 2.1, the evolution of database systems is shown in terms of the computer languages.

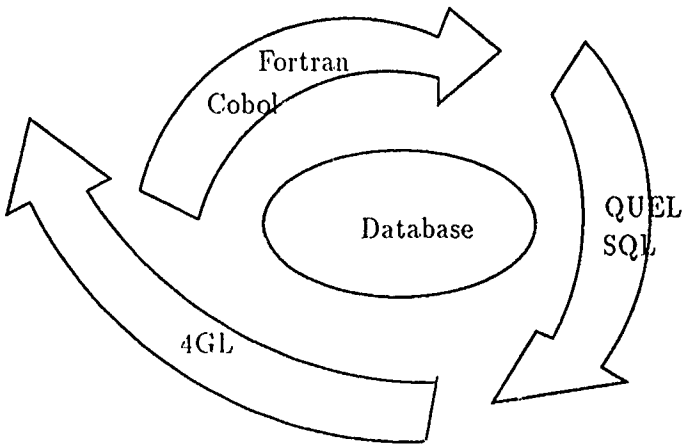


Figure 2.1. Progression to the 4GL (6:28)

The first commercial Database Management Systems (DBMS) were somewhat connected to languages like FORTRAN and COBOL. They were dependent upon these languages for retrieving records or manipulating the database files. In the next development cycle of DBMSs, query languages were used. These were more user friendly in the context that the users were able to retrieve data and manipulate files in the database by using more understandable constructs. These query languages were closer to natural-English. Soon, there were many of these languages on the market. In response to the need for an automated environment of query language commands, 4GLs were created. 4GLs were introduced to the arena of DBMS in two steps First, they were just an extra package

for the query-based languages. Later on, complete applications were built within one programming environment using 4GLs.

When building a query-based application, it may be difficult for the system architecture to anticipate all of the functionality and flexibility that will be required.

The best environment for a query-based application is where the 4GL is tightly integrated with the DBMS. In this case, queries interact with the database more quickly, directly, and completely. While the application evolves, the user can work with any kind of data that is related to the application. Tight integration means that the users don't have to learn about the new tools, because the 4GL provides all the capabilities. The 4GL is not only a complete programming environment for the new users, but also it provides a wide variety of programming capabilities for professional users (3).

In high-performance systems, where applications are more sophisticated and more complex, tight integration is an important asset. Small performance gains give the user a great deal of extra productivity.

2.3 4GL Facilities

No matter what the complexity of a 4GL product, the main idea is to accelerate the development of computer application systems by speeding up and simplifying the programming process. Moreover, the maintenance is expected to be easier than that of a standard coded program.

An ideal 4GL environment has its own tools such as: database creation tools, data input tools, application generator tools, and decision support tools. Figure 2.2 illustrates a complete 4GL environment.

The 4GL environment tools are centralized around the DBMS. Each supporting tool has its own internal tools. These are:

- Database Creation Tools
  - *An on-line data dictionary/directory* to support data element/field description, record and file/database definitions, and maintenance capabilities.

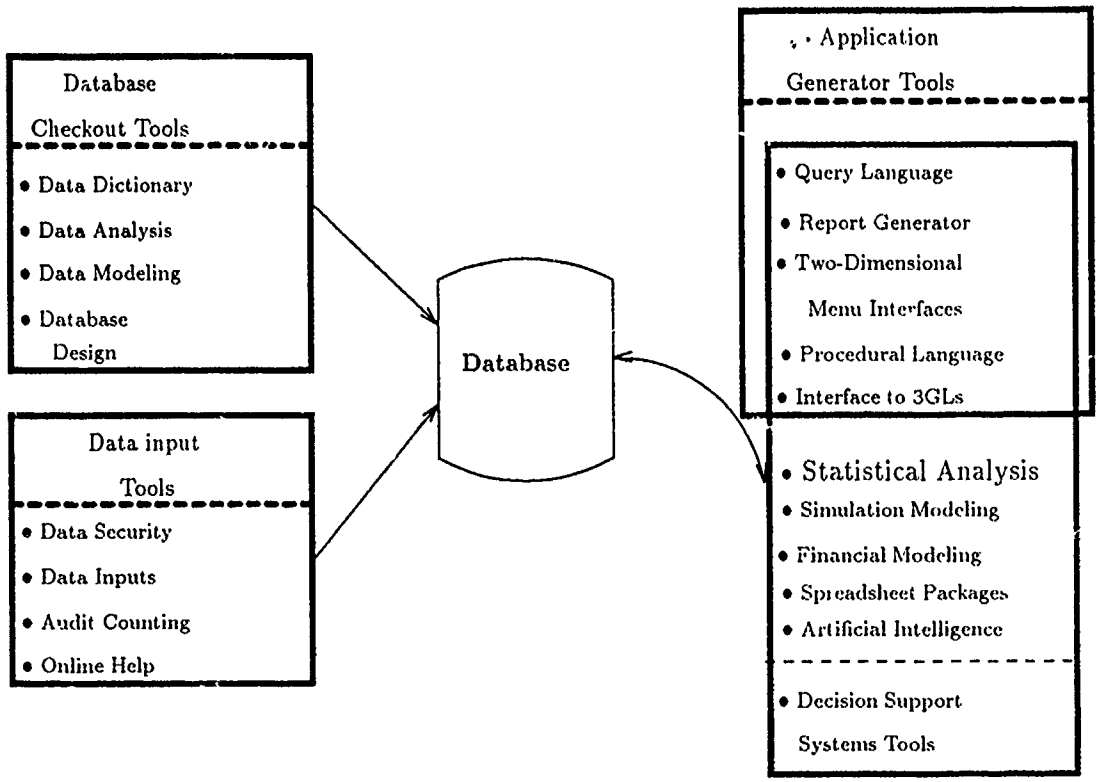


Figure 2.2. 4GL Facilities for Production Systems(6:65)

- *Data analysis tools* to monitor performance and to enhance and analyze user views of data.
- *Data modeling and database design tools* to speed up system building.

• Data Input Tools

- *Screen design and painting facilities*, the most important data input tools. The screen painter should create color blocks, reverse video, highlighting, and fields for user data-entry.
- *Data security and integrity controls* which should be invoked automatically.
- *Audit controls* which should be easy to use and automatic. Audit controls allow one to store information on who read, created, updated, or deleted data items.

- *On-line Help*, which the user should be able to invoke.

- **Application Generator Tools**

  - *A query language*, which is required either for simple queries that display a single record or for complex queries that display data that were projected, joined, or searched, with various conditions applying.

  - *A report generator facility*, which provides the user with simplified means of generating and formatting reports. The default values provided allow standard reports to be produced with a minimum of specifications. Custom reports can also be designed with the use of procedures. Often the report generator can perform simple arithmetic functions such as averaging, calculating percentages, and finding maxima and minima.

  - *A graphics generator*, which can adjust the formats, color, shading, scale, and labeling of the graphs and help create well-defined graphical outputs.

  - *A two-dimensional menu interface facility* for a dialogue with the user.

2.4 4GLs in Systems Development

In the development of information systems, I preferred to trace Gregory and Wojtkowski's methodology (6:131-147). They handled this concept in a very understandable style in their new book (6):

> The development of information systems is a complex issue. To deal with this complexity problem, a disciplined approach, the classical system development life cycle, is chosen.

The development of information systems follows a linear process, namely, the system development life cycle (SDLC). There are some applications that are highly structured. These have to follow a linear development life cycle which corresponds to *prespecified computing*. For these kinds of applications, the requirements have to be shaped well ahead of time. On the other hand, this approach has two weaknesses:

- It takes a long time (months, even years) to develop the system
- If the user expectations are not met, starting all over might be necessary, which costs a lot.

To handle these problems, the prototyping approach can be used in which 4GLs play a major role as development tools.

In its own environment, the prototyping cycle should have a development tool (4GL) to quickly build the prototypes using the data resources available. Also, an energetic user who is ready to tackle the system problems and a prototype builder are needed. Like every system development cycle, prototyping also has steps. In Figure 2.3, these cycles can be seen as separate steps.

- In Step 1, the main objective is to gather enough information to initialize the prototype. The user and the designer should discuss the process to provide a solid understanding. Then, the system is defined with a set of data elements in their local relationships. At the end, a cost analysis should be done.
- In Step 2, an interactive prototype of the application system is built. This prototype should meet the requirements set up in Step 1.
- The purpose of Step 3 is to refine the initial requirements by using the working prototype. This follows the question 'Is the user satisfied?' If the answer to this question is 'yes,' then the working prototype is improved for a final product.
- If the answer to the earlier question is 'no,' then Step 4 is realized. The working prototype is refined, revised, enhanced, and Step 3 is repeated. This is where the available fourth-generation technology is used to advantage.

2.5 Analyzing High-Performance 4GL/RDBMS Application Requirements

'High-performance' applications differ from the others in some natures. They are larger in size, complexity or scope. In order to make a healthy choice about the 4GL/RDBMS, a robust analysis is necessary (3).

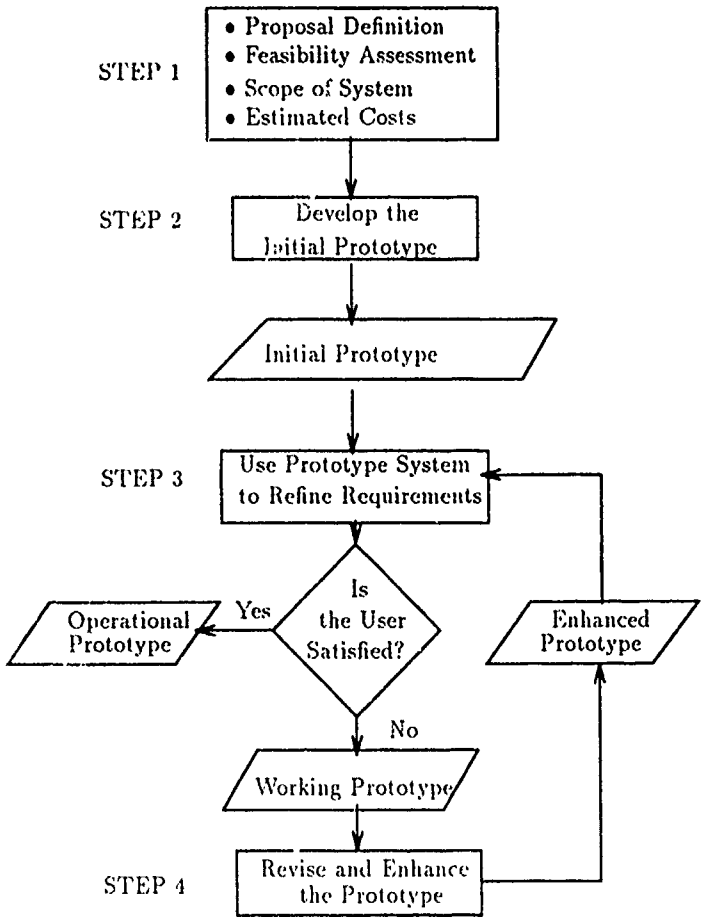


Figure 2.3. The Model of the Prototyping Cycle(6:65)

The analysis should, first, take the user/developer needs into consideration. Second, to make full use of the software throughout the lifecycle of the application development, software product evaluation is important for the selection of this software.

The TWX application is deemed to be a high-performance one. Today, most applications use both 4GL and an RDBMS. The 4GL and RDBMS, together, can be used to create different applications in different complexities.

The expectations from a specific 4GL for a high-performance application are (3):

- Developer ease-of-use: From the developer point of view, the 4GL should be easy to develop applications.
- End user ease-of-use: From the user point of view, the system created via the 4GL should be easy to use, consistent in the flow of operation.
- Quick in querying: The queries should be performed quickly by meeting needs for a comprehensive query traffic (the queries could be complicated).
- Comprehensive query needs: It should be capable of manipulating a loaded, complex database.
- Flexibility to quickly tailor operations: To be able to alter the operations for the specific needs of the application.
- Permits data manipulation: Like deleting, altering, inserting or updating the current values in the database.
- Requires tight integration of 4GL and RDBMS.

A needs analysis is necessary to make a good decision about the 4GL and the RDBMS. There are some points that should be considered. These are:

- 4GL/RDBMS Ratio

  From the items above, the last one is somewhat dependent on the prevailing circumstances. When a 4GL and a RDBMS are combined to be used together, the question of the ratio comes up: How much RDBMS, how much 4GL?

  4GL is used for its powerful commands to manipulate the data. On the other side, RDBMS is used for storage and retrieval capabilities. When the application is expected to include the new operations or the sophisticated data in the future, then, for the flexibility of the application, the 4GL must provide a wide range of reporting options (3).

  The 4GL must also provide an extensive set of commands and a development tool for the developer. These two are also necessary for the maintainers.

Another issue about the discussion of 'how much 4GL and how much RDBMS' is the expected capabilities of RDBMS. Data security, access by multiple users for multiple purposes, and integration with other software and interfaces all combine on the relative capabilities of the RDBMS (3).

- On-line Transaction Processing vs Queries

On-line transaction processing (OLTP) and query based systems are two main classes in today's 4GL/RDBMS applications (Figure 2.4). Essentially, most of the applications make use of the two classes.

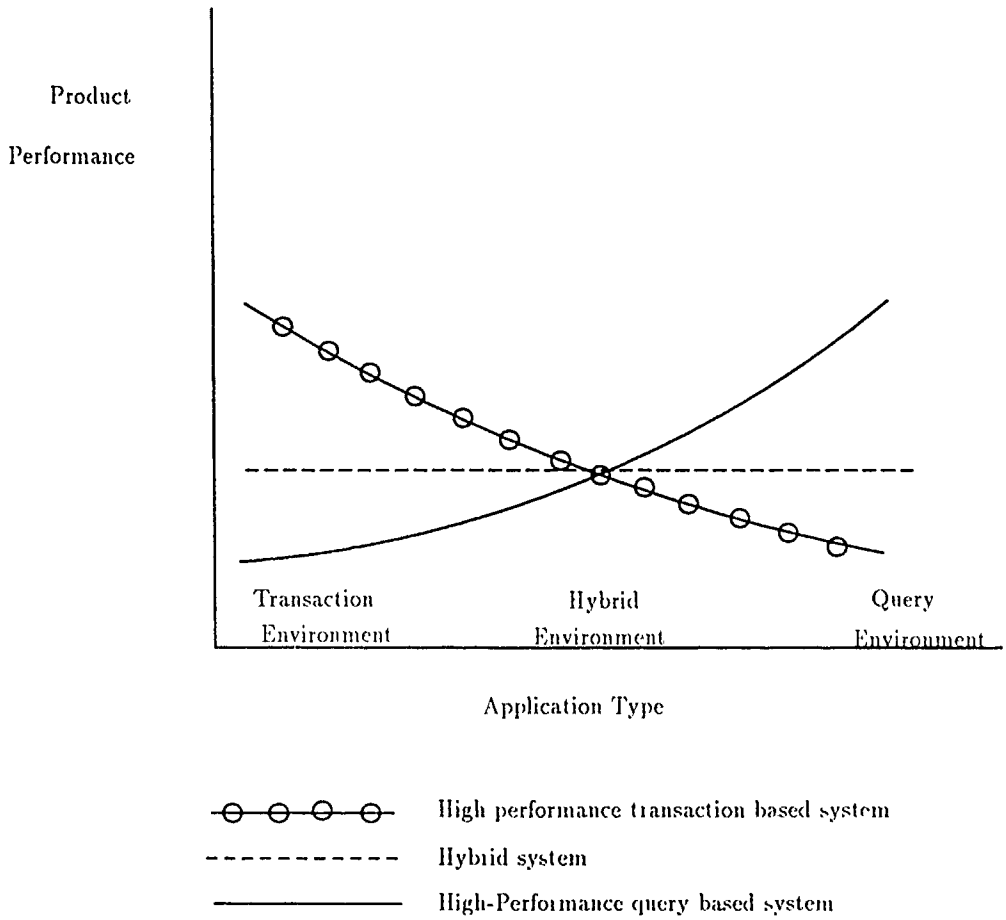


Figure 2.4. 4GL/RDBMS Performance

These two systems are, in effect, different in both design and operation. In OLTP's, also known as the transaction-based systems, the main functionality is that a small amount of data is updated very often. On the other side, query-based systems are optimized for dynamically selecting data that satisfies the user's requests and his complex query conditions.

A needs analysis has to be done before the purchase of the product. At the end of this analysis, the application will be classified whether it is a query-based or a transaction-based. This analysis, in turn, becomes the most important criterion for making a decision on the product evaluation. The TWX, the application which is in question for this thesis, is a hybrid one between these two classes. To make the distinction, one must look at the details of these two systems.

Transaction-Based Application Characteristics:

Transaction-based, or OLTP, systems are commonly used in high-volume disk read/write applications such as automatic bank teller machines and airline reservations. These systems require instantaneously updated information, in tasks such as:

- Updating huge databases quickly and accurately.
- Handling a lot of reads and writes which are not really complicated.
- Executing fixed commands that are not under the direction and control of the end user.

The functionality, itemized above, is easily accomplished by this kind of system. However, when the queries starts getting complicated, it takes quite some time to execute these in transaction-based system.

Query-Based Application Characteristics:

Since these systems are mainly involved in sophisticated querying, they are often used in the scientific area. Huge amounts of data must be retrieved, studied and manipulated. Flexibility and the speed of operations are the two important requirements. High performance query-based systems must:

- Navigate through the massive amount of data and provide complex operations to gather the data not just from one source but from many.
- Provide flexibility for applications future enhancements.
- Have a nice interface for the user, so that he learns the fundamentals of the system easily.
- Provide access to needed development tools including debugging.
- Integration of the system with the other environments such as PC or linker so well that the 4GL/RDBMS is the center of the entire system. This is specially desired in multi-user environments.

Performance Demands: By definition, high-performance 4GL/RDBMS applications demand extra performance from the CPU, from storage and software functionality. A complete analysis should be assessed about the high-performance software application. The following criteria are listed in a clean-cut way in order to clarify the application needs (6):

- Size of database: The number of records per dataset and the number of datasets per database.
- Data Dimensions: The number of characters to occupy a single field.
- Data Types: The data types that are allowed.
- Expected Retrieval Requirements: Like speed.
- Query Requirements: If queries are lengthy and complex, if they reference previous commands, if queries are executed by programmers or end users...
- Reporting: If simple/complex reports can be generated through from a screen-based environment without any additional commands, if these reports can be programmed and compiled into applications...
- Integration: If the users are familiar with the operating system, if the commands are not too different from the conventional ones, if integration with modules is required for processing, analysis or reporting.

- Application Growth: If needs change in the future, if the database can grow without affecting applications, if applications can grow regardless of rebuilding the database.

2.6 Selection Criteria For 4GLs

INGRES and ORACLE RDBMSs are joined with 4GLs to produce powerful applications. Choosing the right 4GL is important to achieve the maximum productivity.

The Fourth Generation Languages differ in options, efficiency, reliability, and cost. A well-planned, carefully executed selection improves the chances of finding the right software. In their book (6), Gregory and Wajtkowski explain the evaluation phase:

Evaluation: The primary objective of the evaluation is to identify the 'critical' factors which make the distinction between an appropriate and an inappropriate language. These factors are listed in Table 2.2.

Table 2.2. Selection Criteria for 4GLs(6)

USER NEEDS	APPLICATION NEEDS
User friendliness	
Menus and prompts	**Hardware & Operating Systems**
Integration across modules	
Default Report Formats	Mainframe, mini, macro compatibility
Help facility	
Clear error messages	Operating system compatibility
Full screen data entry and editing	Resource use: CPU, Memory, storage
Novice and expert mode (procedural and non-procedural mode)	**Communication Linkage**
Supports prototyping	Other databases
Clear and Helpful Documentation	Special-purpose software
Initial license, installation and annual costs	Other computers
	Costs and resource usage
Vendor Support	**Language**
User training	Procedural/nonprocedural
Applications consulting	Compiled/interpreted
Hotline and technical support	Customized menus, prompts, forms, warnings, errors messages and reports
Product updates	
User groups	Standard symbols and conventions
Number of installations	Common and user defined functions
Time-sharing access	
Efficiency and reliability	
Data	**Data Management**
Support cost	
Pricing Structure	Data dictionary
	Common DBMS
	Data types
	Simultaneous access
	Data security

The critical factors are actually extracted from the following set of groups:

User Needs: User needs should be met depending on the user's level of knowledge about the system. For a novice user, this is most critical. On-line help and built-in tutorial help the novice users find their way. If the user is an experienced one, 4GL must offer tools and have the power of a procedural language.

A 4GL should support prototyping as stated earlier in this chapter. This will enable the user to interact closely with the system during the design and testing stages of the application development cycle. A 4GL should produce documentation for the application, including specifications, documenting history of changes, and different versions of the application reports.

Application Needs: The language should not only satisfy the user's needs, but also satisfy the application needs:

- The language should have compatible code.
- The language should include some processing facilities depending on the intended application. These processing facilities can be programming loops, array processing, computational functions, type of databases and files...

Efficiency: The language's efficiency will be different from one operating system to another operating system. For this reason, the language's efficiency characteristics must apply to the appropriate operating system.

The language's efficiency, the number of users, the size and nature of the applications, and the desired response time determine the need for the resources.

Hardware requirements: The 4GL should be flexible enough to work under various hardware and/or software environments. The language should meet the needs of the environment, instead of forcing the environment to adjust to suit the language. Since there are many PC users, the language should allow the PC user to query and extract mainframe data, download the data to the PC, and manipulate the data locally with the standard PC tools or with a PC version of the mainframe language. Or he should be able to design the application and later upload it to the mainframe

2.7 *4GL Types*

Mainly, there are two types of 4GLs on the market:

1. **Application Programmer 4GL:**

   This type of software is used by the data processing departments of many business organizations. It is used to develop transaction processing systems or large databases in the mainframe environment; that is, it is used for company-wide systems.

   Most of the time, the installation and the use of tools associated with products in this category require a highly technical staff. Usually this software is very procedural and too complex for most end-user computing. Its use needs some extensive training. ORACLE is an example of this class. Some of the ORACLE products can be used by application programmers as well as by highly trained user-developers.

2. **General-Purpose 4GLs:**

   There are two categories of products in this group: those that comprise both DBMS and a 4GL and those that do not have a proprietary DBMS.

   DBMS/4GLs are often called the development/information center 4GLs. Some are used by professional programmers only; others can be used for departmental computing by the end users as well. INGRES is an example of this class.

   ORACLE and INGRES systems are explained in detail in the next chapter.

III. ORACLE's & INGRES's 4GLs

3.1 ORACLE's 4GL

ORACLE is a distributed relational database system created by ORACLE Corporation of Belmont, California. Versions of ORACLE exist for micro, mini, and mainframe systems. ORACLE's first commercial release was in 1977 and as of this writing, the product is at version 6.0 (19).

The type of 4GL that is being used by ORACLE is for application programmers. This is mainly used to develop transaction processing systems or large databases.

ORACLE evolved into a 4GL product that can run on a variety of mainframes, minis, and personal computers. It supports a large number of operating systems such as MS-DOS, UNIX, VM/SP, and VMS. The ORACLE system structure is shown in Figure 3.1.

Fourth Generation Environment (4GE): SQL and the relational database model help the application designer to create their work. But, SQL doesn't satisfy the application development requirements. It doesn't have the capability of formatting conditional procedures, and advanced data validation routines/functions which must be supplied by application tools put on top of SQL. On the other side, SQL provides the transition phase to a Fourth Generation Environment (4GE)(4:6-7,289).

Figure 3.1 also includes the complete 4GE tools in ORACLE. The major tools in this environment are explained below.

Application Development:

As David Pepin defined application in his book(15:353):

> An application is a set of functions working together to perform a specific task; described physically, it is a collection of programs, tools, and utilities that interact to perform a specific task.

In ORACLE, a large collection of tools and products allow application developers to create almost any type of application. One of three methods are used depending on the tools available and the type of functions which are needed by the application:

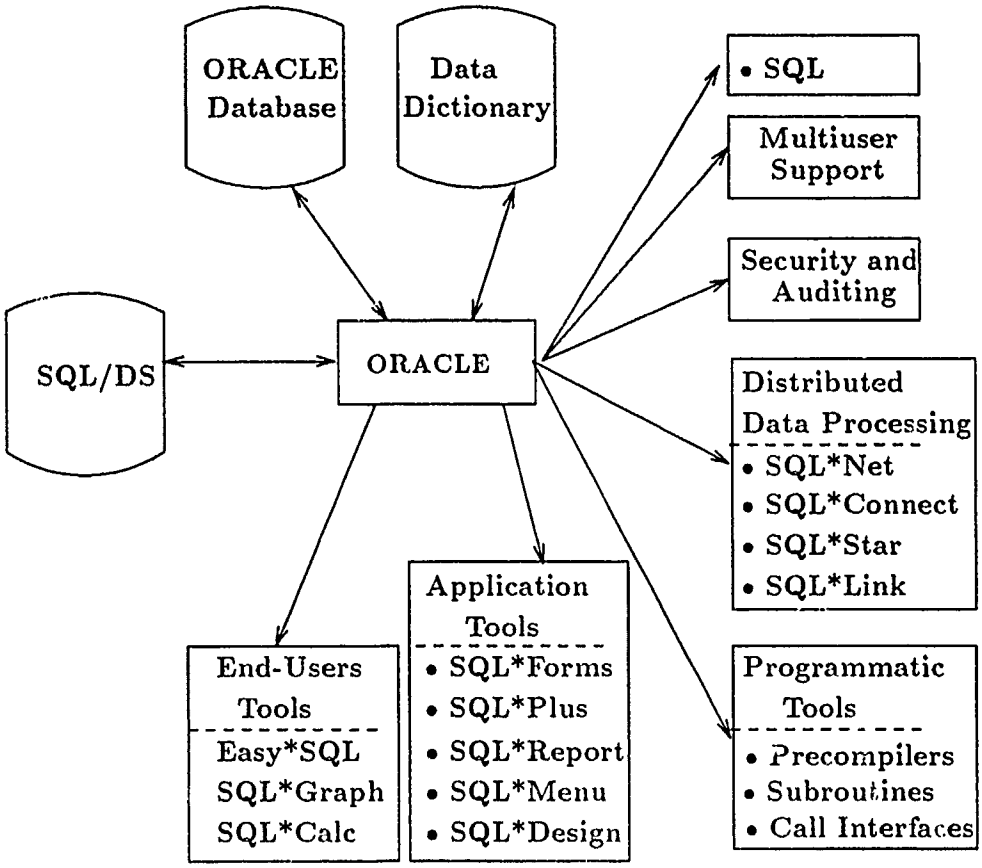


Figure 3.1. ORACLE Facilities(6:85)

- SQL*Plus
- A high level language
- SQL*Forms

Most of the applications exploit these three methods at the same time. But, several factors allow each of these products to be the basis for an entire application: each can create a shell that calls external modules of similar and dissimilar code from within the shell; each can interact directly with the user; and each can interact directly with the ORACLE database(15).

An entire application done completely in SQL*Plus is rare but may be necessary when SQL*Forms or a high-level language ORACLE interface is not available. However,

SQL*Plus-based utilities are used in parts of many applications.

On the other side, there is a great advantage in developing an application based on SQL*Forms. Because, it can be transferred to a variety of mainframes, minicomputers, and microcomputers with no changes. Also, an entire application package can be created using SQL*Forms.

A cost-saving advantage of SQL*Forms is that a prototype can be created rapidly. In the time it takes to create a single form using a 3GL, the prototype of an entire application can be created. The prototype can also be used as part of the final product with some additional triggers (a trigger is a set of commands that are executed at, or triggered by, a certain event when a form is run)(15:378-405).

*3.1.0.1 SQL*Forms* SQL*Forms lets the application developer to build forms-based applications quickly for entering, querying, updating, and deleting data. Using simple menus and using the powerful screen painter, the application needs are specified. SQL*Forms then combines the instructions with information from ORACLE's data dictionary to generate the application.

SQL*Forms gives the ability to:

- insert data into the database by typing the data directly into the fields
- view, update, or delete several records on the screen at one time
- type query conditions directly into the fields to be queried.

For example, the following SQL Language statements will insert a set of data into the database:

```
INSERT INTO ORD
VALUES (610,'07-JAN-87','A'101,'08-JAN-87',101.4)
INSERT INTO ITEM
VALUES (610,1,100860,35,1,33)
```

SQL*Forms can display the data that is not in any database but is calculated on the basis of data stored in the database. One can generate data that records such things as the time, date, or operator associated with each change to the database.

Understanding the Basic of the Forms:

Below are the basic definitions that are needed to know to work successfully with SQL*Forms.

- Page: The part of the form that is seen on the screen. A form can have many pages.
- Block: Data and text that correspond to one table in a database.
- Base Table: The table on which a block is based.
- Record: Data from one row in a table.
- Field: A highlighted or underlined area on the screen that can display a value. The value usually corresponds to a value from a column in a database table.
- Single-Record Block: A block that can display only one record.
- Multi-Record Block: A block that can display more than one record.
- Value: An item of information in a field.

The forms in Figure 3.2 shows three blocks on the same page. Block, fields, and records are shown on the margin separately.

The steps which should be followed to accomplish the goals in building the application can be outlined as follows:

- Clarify the goals that are going to be achieved.
- Determine which tables will be used in the form.
- Determine the blocks a particular form will contain.
- Determine the order in which the blocks are going to be laid out in a particular form. The order of the blocks can be sequenced for the ease of the operator first. Then, it can be changed.

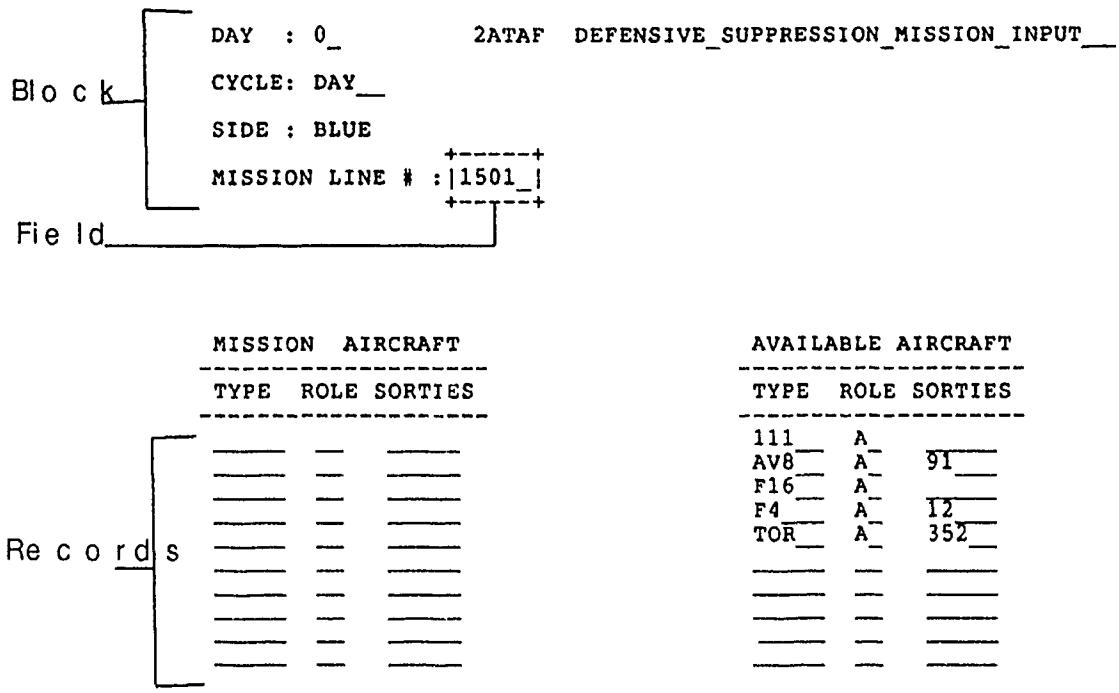


Figure 3.2. Forms Representation in ORACLE

- Use the Default Block capability to build a rough draft of the form.
- Modify and re-size the default fields and create new ones until all the fields that are needed are created.
- Enhance the form visually by adding text and highlighting certain parts of the form with boxes and lines.
- Add validation checks and supply default values and other fields criteria.
- Define triggers. A trigger is a set of commands that is executed at or 'triggered' by a certain event when a form is run.
- Test the form often–SQL*Forms is designed to permit convenient testing of a form in design phase. Diagnosing errors as soon as possible makes the form design easier.

*3.1.0.2 SQL*Forms Components* The components of SQL*Forms include:

- SQL*Forms (also called IAD), the Interactive Application Designer, which creates or modifies the form in the database. It is the main component which can call the others. It is also executed when CREATE or MODIFY is chosen from CHOOSE FROM window.
- SQL*Forms (Convert)(also called IAC), the interactive application converter, which converts a form between database and INP format. It is executed when GENERATE or LOAD is chosen from the CHOOSE FROM window.
- SQL*Forms (Generate)(also called IAG), the interactive application generator, reads an INP file and generates a FRM file and runs it. It is executed when RUN is chosen from the CHOOSE FROM window.

As illustrated in Figure 3.3, the designer can use components of SQL*Forms to convert from one format to another.

3.1.0.3 Triggers A *trigger* is a set of SQL*Forms commands peculiar to ORACLE that are executed at, or triggered by, a certain event when a form is run. They can be used to validate, assist, and enhance what the operator enters on the form.

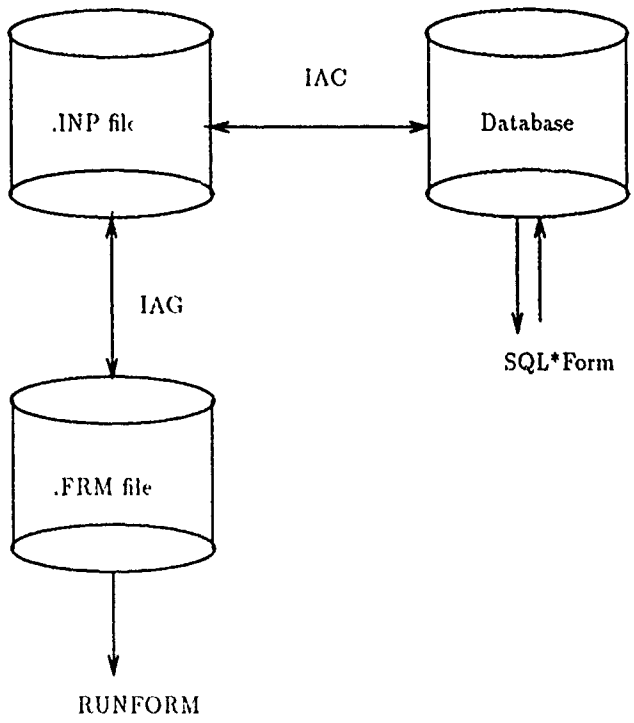


Figure 3.3. Form Formats And Components

Triggers hold the key to much of SQL*Forms' power and also much of its complexity. Although writing the triggers is not like conventional programming, it can be very complex. Trigger events can be associated with five kinds of events:

- *entry*- when the operator first runs a form or when the cursor enters a new block, record, or field
- *query*-before or after records are retrieved
- *change*–after the operator changes a value, or before or after inserted, updated, or deleted records are committed to the database
- *exit*–when the operator leaves a form or when the cursor leaves a block, record,or field
- *keystrokes*–when the operator presses a function key.

There is an other kind of trigger that is not triggered by a specific event at all. These are user-named triggers–common triggers or subtriggers that can be used or called from other triggers.

The Structured Query Language, SQL, is the key method working with data in a 4GE environment. In addition to the normal database operations, SQL commands in triggers are used to:

- place data in fields of a form
- perform calculations on data in a form
- reformat data in a form
- check whether data exists in the database
- compare data in fields of a form.

With one difference and two extensions, SQL commands in triggers are virtually the same syntax as SQL commands in other ORACLE products. The major syntax difference is that SQL commands in triggers do not end with a semicolon (;).

It is wise to test the SQL commands before they are placed in triggers. In general, any SQL command can be used in trigger. However, there are some warnings:

- 'SELECT' commands may only be used in a post-change trigger (a kind of trigger that is only activated when the value of a field is changed).
- Data modification commands (INSERT, UPDATE, and DELETE) may only be used in commit (pre- and post-insert, update, and delete) triggers.
- If INSERT, UPDATE, or DELETE commands are used to modify a base table for a block in the current form, it is *necessary* to update any data that might be on the form.

When the operator wants to delete, insert, or update a record from a block, that block has to belong to a table in the database. As mentioned before, each table should

correspond to a base-table in the database. This is necessary to make the changes to the database. If this wasn't the intention, the designer would not have to choose a base table for that block.

Although it might seem to be a problem, there is a solution to this. If, on the same page, there are several blocks together at the same time, and if the value of a field in one of these blocks which has a base-table in the database has been changed, another field in another block can be changed and committed too (the second block not having a base table).

*3.1.0.4 SQL*Plus* SQL*Plus is a fourth generation language (essentially SQL with additional features) designed to manage all interactions within ORACLE. It allows the operator to create, modify, and join database tables; control database access; create reports; and transfer data among ORACLE systems distributed on different computer systems.

ORACLE's version of the SQL language, SQL*Plus is the most functionally complete and powerful SQL in the market. SQL*Plus has outer join, hierarchical structuring, output formatting, minus/difference and date/time operators, as well as a formidable selection of row functions such as standard deviation, soundex, and null value replacement (6).

The SQL*Plus program can be used with the SQL database language and its procedural language extension, PL/SQL. The SQL database language allows the operator to store and retrieve data conveniently. Through SQL*Plus, one can:

- enter, edit, store, retrieve, and run SQL commands and PL/SQL blocks
- format, perform calculations on, store, and print query results in the form of reports
- list column definitions for any table
- access and copy data between SQL databases
- send messages to and accept responses from an end user

PL/SQL programs (called blocks) can also be used to manipulate data in the database. PL/SQL blocks begin with DECLARE, BEGIN, or a block name. SQL*Plus treats

PL/SQL blocks in the same manner as SQL commands, except that a semicolon (;) or a blank line does not terminate and execute a block. One can terminate PL/SQL blocks by entering a period (.) by itself on a new line. Below is an example of how a PL/SQL block looks:

```
SQL> DECLARE
  2      * NUMBER := 100;
  3  BEGIN
  4      FOR I IN 1..10 LOOP
  5          IF TRUNC (I/2) = I/2 THEN   --I is even
  6             INSERT INTO temp VALUES(I,X, 'I IS EVEN');
  7          ELSE
  8             INSERT INTO temp VALUES(I,X,'I IS ODD');
  9          END IF;
 10          X := X + 100;
 11     END LOOP;
 12  END;
 13  .
```

One major difference between the ORACLE and INGRES SQL is that ORACLE allows the use of variables to store table names. This enables the SQL 'from' statements to be loaded at runtime (for example after the team's affiliation is known in TWX). The primary advantage of using variables is that the size of the SQL code can be cut in half.

*3.1.0.5 SQL*ReportWriter* SQL*ReportWriter is a general purpose tool for developing and executing reports, specially designed for application developers who know the SQL language. With the SQL*ReportWriter, one can:

- combine multiple SQL statements in a single report to easily define complex relationships
- create ad hoc reports using a rich set of defaults

- performs complex calculations
- run reports interactively or in production environments with flexible runtime parameters
- fully customize all parts of the report definition.

In order to build reports successfully with SQL*ReportWriter, these steps should be followed: First, the **Action** choice is selected from the Main menu and **New** is selected from the pull-down menu.

The next step is to define one or more queries. Queries enable the user to specify the data he plans to use. He can access data from one or more tables residing in one or more databases. He can use multiple queries in a report, and he can create relationships between them. This is also a good time to enter a comment describing the purpose of the report, and at the same time to define the page size and margins.

Once the queries are defined, the user can use group settings to specify where groups of data from the queries should be placed in the report. One can think of groups as a tool to perform 'coarse' or overall placement of data in the report.

3.1.0.6 Programming Tools in ORACLE The programming interface allows application programmers to access ORACLE from within third-generation languages. This interface supports languages such as COBOL, C, BASIC, FORTRAN, Ada, PL/I, and Pascal.

SQL is a non-procedural language. That is most statements are executed independently of preceding or following statements. The non-procedural nature of SQL makes it a very easy language to learn and to use.

On the other hand, 3GL languages like C, COBOL, or FORTRAN are procedural. That is, most statements are executed depending on proceeding or following statements through such constructs as loops, and conditional control statements. The procedural nature of these languages makes them very flexible.

The ORACLE Call Interfaces, OCIs allows the user to write applications that take advantage of both the non-procedural capabilities of SQL and the procedural capabilities

- performs complex calculations
- run reports interactively or in production environments with flexible runtime parameters
- fully customize all parts of the report definition.

In order to build reports successfully with SQL*ReportWriter, these steps should be followed: First, the **Action** choice is selected from the Main menu and **New** is selected from the pull-down menu.

The next step is to define one or more queries. Queries enable the user to specify the data he plans to use. He can access data from one or more tables residing in one or more databases. He can use multiple queries in a report, and he can create relationships between them. This is also a good time to enter a comment describing the purpose of the report, and at the same time to define the page size and margins.

Once the queries are defined, the user can use group settings to specify where groups of data from the queries should be placed in the report. One can think of groups as a tool to perform 'coarse' or overall placement of data in the report.

3.1.0.6 Programming Tools in ORACLE The programming interface allows application programmers to access ORACLE from within third-generation languages. This interface supports languages such as COBOL, C, BASIC, FORTRAN, Ada, PL/I, and Pascal.

SQL is a non-procedural language. That is most statements are executed independently of preceding or following statements. The non-procedural nature of SQL makes it a very easy language to learn and to use.

On the other hand, 3GL languages like C, COBOL, or FORTRAN are procedural. That is, most statements are executed depending on proceeding or following statements through such constructs as loops, and conditional control statements. The procedural nature of these languages makes them very flexible.

The ORACLE Call Interfaces, OCIs allows the user to write applications that take advantage of both the non-procedural capabilities of SQL and the procedural capabilities

of a 3GL. These applications can be more powerful and flexible than applications written in either the host language or SQL alone.

The OCIs allow the user to communicate with ORACLE through a subroutine library supported for several high-level programming languages As Figure 3.4 shows, the user compiles and links an OCI program in the usual way.

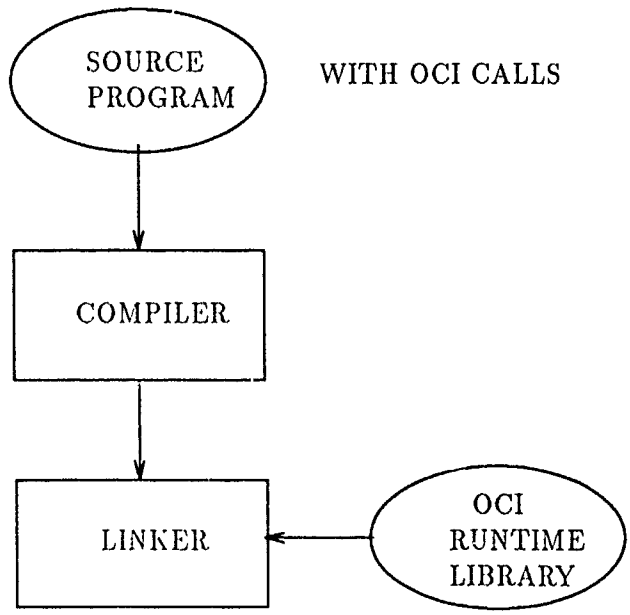


Figure 3.4. OCI Development Process
(13)

The OCIs support all SQL query, data manipulation, data definition, and data control facilities that are available interactively through SQL*Plus.

3.2 INGRES's 4GL

INGRES is a distributed relational database system offered by INGRES Corporation of Alameda, California. Versions of INGRES also exist for micro, mini, and mainframe systems. However, INGRES is written in the Assembly language of the target system, and is available on fewer machines and operating systems than ORACLE(19).

Being different from ORACLE's 4GL, INGRES uses general purpose 4GL. Under this, there are two categories: the ones that combine both the DBMS and 4GL and the ones that only have a 4GL. INGRES has both of them: A DBMS and a 4GL. In INGRES, 4GL is more visible to the application programmer/designer, so that, a separate block can be seen for INGRES 4GL. The advantages of 4GL in INGRES is more or less the same as the ones in ORACLE. The INGRES facilities can be seen in Figure 3.5.

3.2.1 Fourth Generation Environment (4GE) Primarily, the INGRES 4GL specifies the menu operations by controlling the user's movement among the frames and procedures of an application by forms (ABF) (Figure 3.6 illustrates the INGRES application components in 4GE). Besides the operations that manage the applications, the INGRES 4GL can combine some certain operations with each frame (a frame contains a form and a menu; it's the basic element in an ABF application) in order to access the database directly and to control the form that displays the data(17:3-15).

By using INGRES 4GL, users can(17:4-6):

- Access the database to retrieve , append, or update rows
- Manipulate forms by specifying initialization, defining field and key activations, and setting field attributes
- Perform calculations on items in the form, whether displayed on the screen or not
- Call other frames, INGRES 4GL procedures, INGRES modules, or the operating system
- Use hidden fields for calculation or data that the user does not need to see
- Carry out multi-row queries with submenus

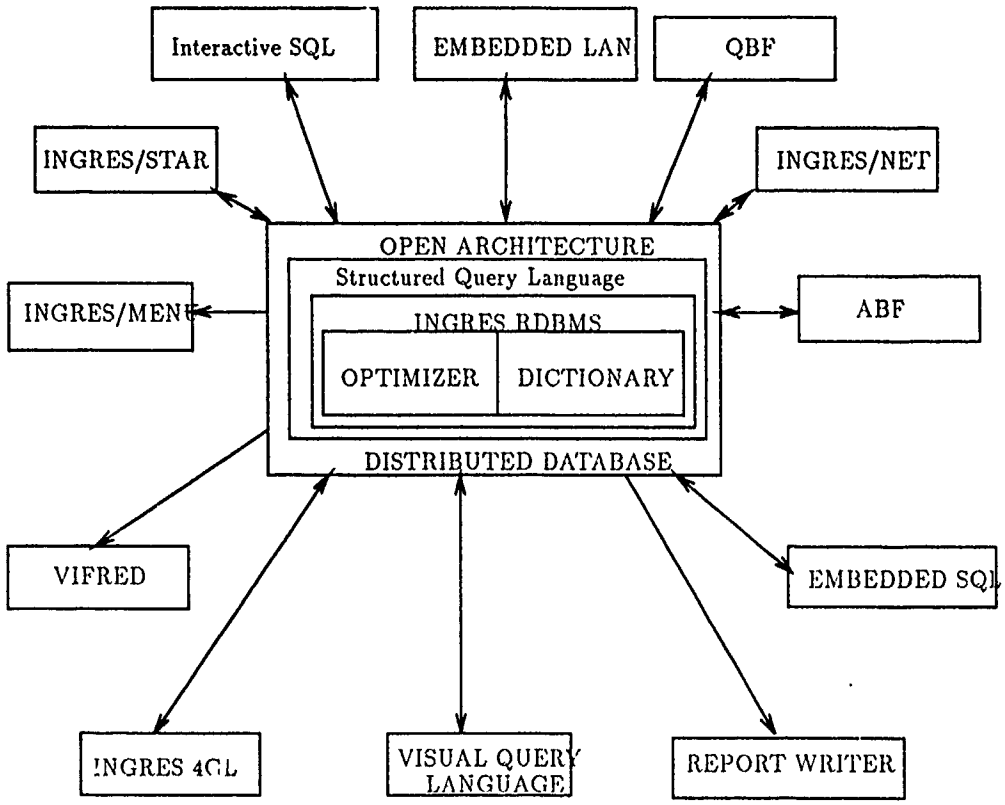


Figure 3.5. INGRES Facilities(19)

- Perform selective processing on the table fields

Application Development: The INGRES facility for creating customized, forms-based applications is *Applications-By-Forms* (ABF). An ABF uses standard INGRES forms and menus to access a database and perform a series of operations (such as queries, update and reports). Using ABF one can define, test, and run fully developed applications without having to use a conventional programming language.

Applications-By-Forms lets the user create an application without having to worry about the location and management of source files, object files, linkage programs, compilers, editors, and the other tools of conventional programming. ABF uses INGRES user interfaces such as the Visual-Forms-Editor, Query-By-Forms, and Report-By-Forms

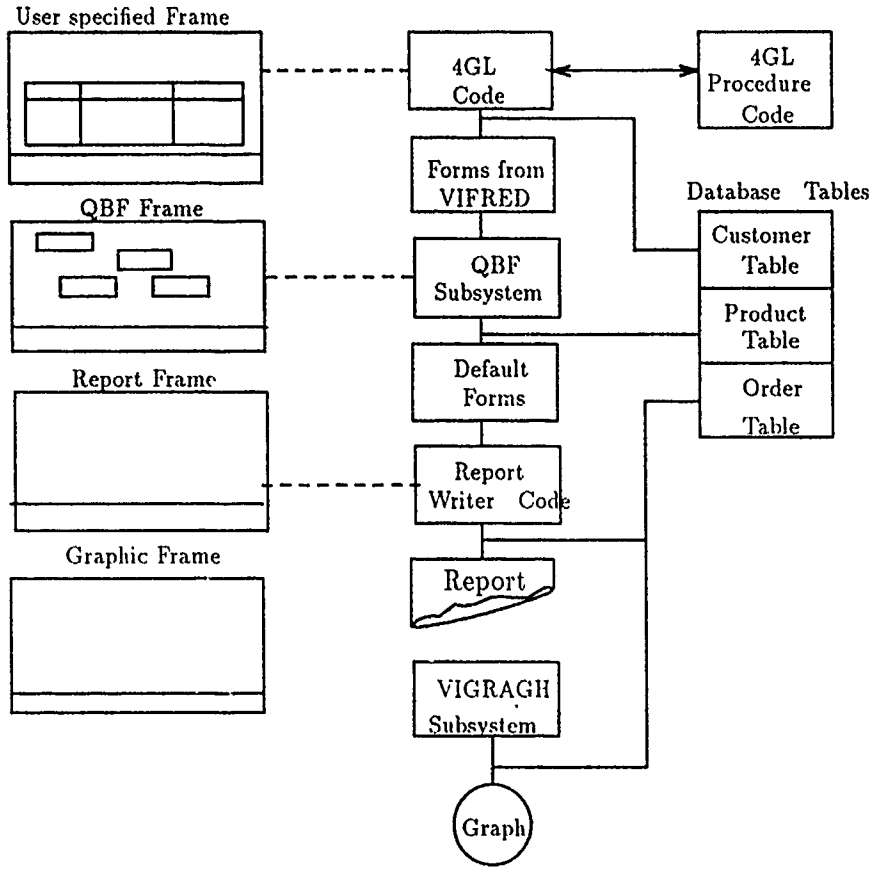


Figure 3.6. INGRES Application Components(18:11)

to create a sequence of forms and reports that will let the application users manipulate the data in a convenient way. The user of an INGRES application normally needs to access database table information on the frame, then lets the user chose the kind of data manipulation required.

In developing the application within ABF, the designer uses INGRES 4GL to specify the applications' general structure and to define any custom processing steps that the application uses. He designs the overall flow of an application in a series of consistent, easy-to-use menus. He can then fine-tune an application by indicating what is to happen when the application user chooses a menu operation.

The form is the input and output medium for a forms-based application, and a large part of any forms application involves operations that get data from and display data to the form. INGRES 4GL makes such interactions easy to specify and uses the form as an integral port of the 4GL specification.

In INGRES 4GL the designer can code statements based on either of the INGRES query languages, SQL or QUEL, to access and manipulate the database. Procedures accessed through the 4GL activations can be written in 4GL itself or can be based in standard programming languages such as C, Pascal, and others. For additional flexibility, these procedures can include embedded query language statements (Embedded SQL or EQUEL), providing easy access to the INGRES database at all levels of programming endeavor.

One of INGRES 4GL's major strengths is incorporating the power of ANSI Standard SQL and Embedded SQL into itself, thus greatly reducing the need to call separate embedded language procedures from ABF.

An important point to remember here is about the language to be used in developing the application. The designer can only use one language inside an application. He can not mix it with the other one.

3.2.2 Applications-By-Forms (ABF) ABF is the INGRES facility for creating customized, forms-based applications. An ABF application uses standard INGRES forms and menus to access a database and to perform a series of operations such as queries, updates, and reports. ABF uses user interfaces such as the Visual-Forms-Editor, Query-By-Forms, and Report-By-Forms to create a sequence of forms and reports.

ABF has several distinct advantages for application development:

- A code manager for all files related to an application
- A dynamic test environment
- The use of INGRES modules
- The use of INGRES 4GL for custom processing

- The need for fewer lines of code than in conventional programs

ABF automatically provides access to the system editor and to the Visual-Forms-Editor as the elements of the applications are created.

ABF supplies a test environment for the applications. The applications can be run and debugged before they are actually defined. ABF provides default actions if an undefined object is referenced. This environment allows the small pieces to be tested.

By using the readily available blocks, new applications can be created easily. The user can also use queries, forms, and reports by linking them together. He can also incorporate INGRES modules such as Query-By-Forms, making INGRES tools directly available to the application developer(17).

An ABF application may include these components:

- Frames: The basic operational units of an ABF application. The end user interacts with the application through forms and menus defined within the frame structure
- Procedures: Separate modules of INGRES 4GL or a host language code that perform specific operations
- Tables: Database tables containing data on which the application operates
- Reports: Data formatted for the display or printing
- Graphs: Data presented in a visual, graphics format.

Frames: Central to each ABF application is a collection of related units called *frames*. An application is composed of frames. Frames let the user manipulate the information in a database. A frame visually consists of a form, which can be considered the equivalent of a form on a piece of paper, and a menu of operations through which the user can query the database, run reports, use INGRES user interfaces, run operating system programs, and perform other tasks.

There are four types of frames:

- User-specified frames: Custom frames created by the application developer

- QBF frames: Frames that perform database queries using Query-By-Forms
- Report frames: Frames that display or print reports
- Graph frames: Frames that display graphs

For each user-specified frame, the designer must create an INGRES 4GL specification defining each menu operation along with statements that determine what happens when the user chooses each operation. In an INGRES 4GL specification, the user can start up other frames, run INGRES user interfaces, run external applications or system programs, display his own help files, and performs specific data manipulations. He uses the system editor to create the INGRES 4GL file, which Applications-By-Forms stores for him in the directory he chooses for source code files. Being different from ORACLE, since there is no concept as 'base table' for each 'block', the update, insert, and delete statements can be used anywhere in the frame.

Whenever the designer includes a QBF frame in an application, he is specifying that the application use Query-By-Forms to access the database. When defining a QBF frame, the designer might specify or create the table of JoinDef (join definition) and the form with which to run QBF. For queries involving one table only, the Join Definition phase is optional. The designer can also specify command line flags to be used in the call on QBF. When the frame is activated, QBF begins executing in the appropriate fashion based on the form, table, or JoinDef, and flags specified. In designing the application, the designer can use a QBF default form as is, or he can enhance it with the Visual-Forms-Editor. QBF frames are particularly useful for operations that interact directly with the database such as adding new rows to a table or retrieving data from a series of tables.

A *report* frame consists of a report and a menu for running it. The frame may include a form on which the user can enter one or more values used by the report at run time. The designer can create the form using the form editor. No 4GL code is necessary for a report frame.

Graphs help present data in a clear, visually striking way. A graph is similar to a report except that the data are displayed in a bar chart, a pie chart, or a plot.

Procedures: A user-specified frame may require specific operations that differ from the capabilities provided by existing INGRES user interfaces. For such operations, the application can call a procedure written in INGRES 4GL or in programming languages such as C or FORTRAN. A *procedure* is an INGRES 4GL or other host language routine that is declared in a procedure definition. A procedure can be called by frames or other procedures within an application. Note that the reverse is not true; that is, a procedure cannot call a frame.

A procedure, like a subroutine, can execute frequently used sets of statements and then return data to the calling frame when it has finished. Procedures are often used for frequently performed calculations or other processing. An example:

```
procedure addtax (cost = float8,
    taxrate = float4) =
    begin
       cost = cost + (cost * taxrate);
       return cost;
    end;
```

The procedure above, 'addtax', performs a frequently used tax calculation, then returns a result to the calling frame.

3.2.3 Query-By-Forms (QBF) QBF is a visually oriented, forms-driven interface to INGRES. QBF allows both new and experienced users to access tables in INGRES databases and perform routine data retrieval. The users may retrieve and modify data from database tables without needing to learn the commands of QUEL or SQL, the INGRES query languages.

The tasks the designer can perform using QBF are divided into two phases: Join Definition and Query Execution. Within the Join Definition phase, he might create the objects that QBF uses to retrieve and manipulate data in the database. In Query Execution phase, he performs the retrieval and manipulation of data using the objects created previously (or by default).

QBF uses forms on the terminal screen in two distinct ways. In the Query Execution phase, a form is used to append, retrieve of modify data in the database tables. In the Join Definition phase, a forms interface enables the user to identify and define relationships between tables to be accessed.

From QBF, the user can access other features of INGRES that are forms-based parts. Two of these features include different data input modes and an extensive help facility for using QBF.

A companion product of QBF in the INGRES line of forms management systems is INGRES/FORMS Visual-Forms-Editor (VIFRED). VIFRED allows the user to modify forms in the following ways:

- change the appearance of a form to reflect the application more clearly.
- change the attributes of the fields to protect their contents or to display their data in a manner more consistent with their meaning.
- specify range checks and cross-field checks to maintain the maximum possible integrity for the data in the form.

When QBF is invoked for some task, QBF generally provides a default form for the work. However, because VIFRED can be used to edit a form, the user can instruct QBF to provide the edited form instead of the default.

3.2.4 Visual-Forms-Editor VIFRED is a visually oriented, menu-based editor designed to edit forms. It is used to edit the layout of system- and user-defined forms. With VIFRED, the user can change the appearance of the forms that Query-By-Forms (QBF) uses, or he can create his own forms for use in an application program. VIFRED edits, redefines or creates forms for the terminal screen. On the other hand, QBF is concerned with data manipulation. QBF queries and updates actual data values stored in an INGRES database. The user can neither use QBF to edit a form, nor VIFRED to edit actual data. A sample VIFRED is shown in Figure 3.7 as it is seen on the screen.

The menus in VIFRED display the commands. Moreover, because VIFRED is visually oriented, the form is always on the screen during the time it is being edited. The

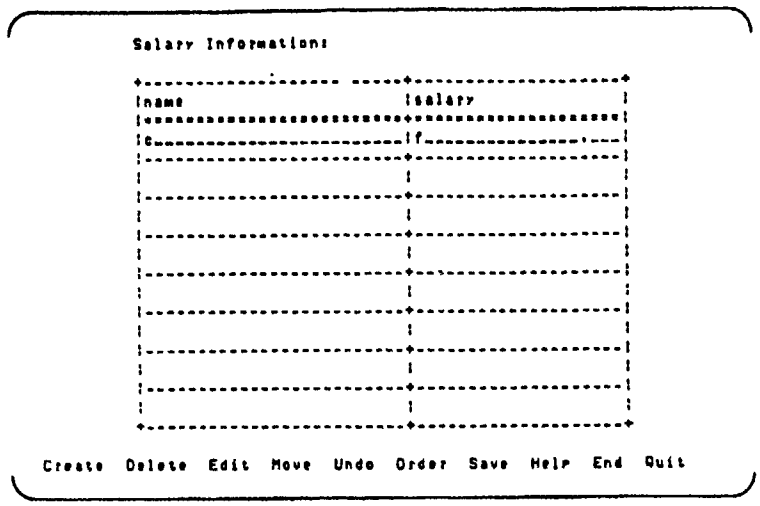


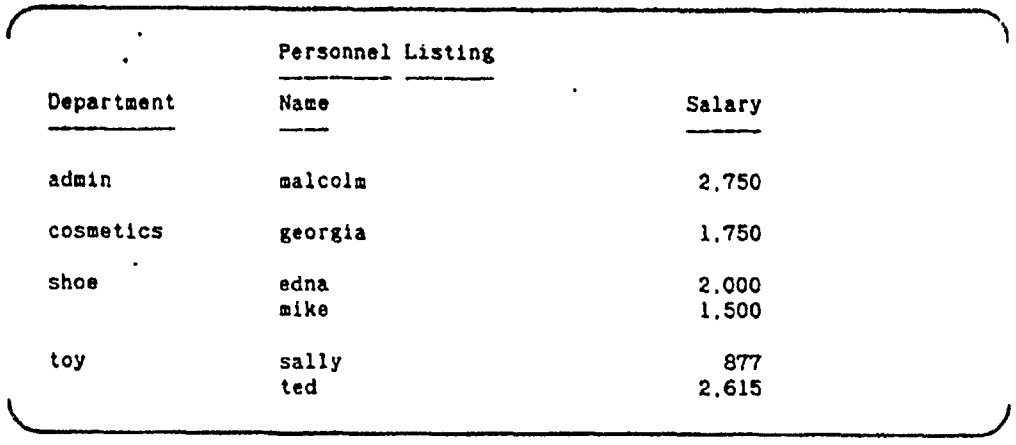
Figure 3.7. Visual Forms Editor in INGRES

form can be used for display or manipulation of data. However, the quality of a form is greatly dependent on the designer. How the form is designed and implemented can make the difference between an efficient and wasteful data processing. A properly designed form engages the user, making the terminal screen a more human environment. This can optimize the work done by that user. Therefore, energy devoted to the definition of forms in a computer system has a significant effect on the overall utility of the system.

3.2.5 Report-Writer The INGRES report writing facility allows the user to create highly formatted listings of data from his database in a flexible manner. The Report-Writer can be used to create organized summaries of data for inclusion in other documents and to produce regular listings of data for management and production needs, as well as for ad hoc purposes.

Capabilities of the Report-Writer include the following:

- Use of QUEL or SQL, INGRES's powerful query languages to specify the data to be used in the report
- Features similar to word-processing capabilities, such as centering, justification and automatic pagination



Personnel Listing

Department	Name	Salary
admin	malcolm	2,750
cosmetics	georgia	1,750
shoe	edna	2,000
	mike	1,500
toy	sally	877
	ted	2,615

Figure 3.8. Reports-By-Forms in INGRES

- Complete flexibility in specifying how the report will look
- Powerful and automatic aggregation capabilities over changes in value of the data columns, as well as over pages or the entire report
- Storage of report specifications within the database, so that the user can easily run a report. He can also specify parameters to stored reports to vary the data to be reported.

Reports-By-Forms is flexible and easy to use, and provides tools adequate to the task of customizing most default report definitions (see Figure 3.8). The designer (or the user) can also use RBF to define the report formatting commands on which to base a more complex report, and subsequently write out a text file containing those commands. Then he could use a text editor to further enhance the report definition and finally use **report** to compile the changed report into database.

The ability to produce reports is a basic and essential feature of a computer system. A report can be considered the organization of data for orderly output in print or on a terminal screen. When reports are written in INGRES, data are retrieved from a database, sorted formatted according to pre-established specification and written to a file listed directly to an output device. A report combines the two basic elements: data and

specifications for output. Both the data to be reported and the output specifications can be defined by the users or generated as system defaults, both within the context of special report-writing software.

3.2.6 Programming Tools in INGRES In INGRES, embedded SQL is an embedding of the SQL database into standard procedural programming languages, such as C or Fortran, known as the host languages. Embedded SQL is compact and powerful, giving the user access to a full range of INGRES database and forms-control functions within an application. It includes all the standard SQL commands that within an application. It includes all the standard SQL commands that are available in interactive, forms-based applications with or without database access.

Embedded SQL gives the user full power of the INGRES *Forms Run-Time System* (FRS) in order to create forms-based applications. Using forms the user can create with the Visual-Forms-Editor, he can move data to and from forms, he can switch forms from one port of an application to the next, and he can specify the operations he can, as a user, perform on the data displayed in the form.

3.3 The Comparison

In the low level, there are some differences between the ORACLE and INGRES 4GLs. A comparison of the canonical features will show these. This comparison is done by looking at some of the major characteristics that are peculiar to a Fourth Generation Language.

- ENVIRONMENT: ORACLE and INGRES have both two canonical modes to work with; interactive and procedural. In ORACLE, the interactive mode is used via SQL*Forms. The designer has a great amount of flexibility in this almost completely interactive facility. ORACLE allows the designer to edit .INP files in the 4GL code. This procedural application design is harder and takes a longer time than the interactive mode for a designer.

  INGRES's interactive QUEL (SQL) is an interface that enables the designer to manipulate data in database using QUEL (SQL). As it is in ORACLE, the application can be edited in 4GL in a procedural way as well.

- PROCEDURAL LANGUAGE: Procedural control commands, which perform various processing tasks such as controlling the flow of the execution or handling processing error conditions represent integral parts of both languages. Statements in a procedure can be classified as:

  1. Directive
  2. Command
  3. Procedural control

  In ORACLE, these statements are either in SQL*Forms, appearing as the menu items, or they are within the triggers. Some of the examples:

```
CALL        form
CALLINPUT
EXIT
GOBLK       block
```

```
GOFLD      [block.]field
MENU
```

In INGRES, these statements can take place in the ABF as defaults or they can be typed in the source code. Some INGRES statements are:

```
CALL
CALLFRAME
CALLPROC
EXIT
NEXT
IF..THEN..ELSE
```

- DATA DICTIONARY INTEGRATION: When a database is created, INGRES sets up the system catalogs, tables that hold information about that particular database as the designer or the user works with it. These system catalogs store specifications for the tables, indexes, forms, reports, and queries associated with that database.

  In ORACLE, the data dictionary is a set of tables to be used as a read-only reference guide about the database. For example, it will tell

  - the usernames of ORACLE users
  - rights and privileges they have been granted
  - names of database objects (tables, views, indexes, clusters, synonyms, and sequences)
  - information about primary and foreign keys
  - default values for columns
  - constraints applied to a table
  - how much space has been allocated for, and is currently used by, the objects belonging to a database user

- DATA TYPES: In ORACLE, from the list of SQL*Forms field types, the data type for any field can be determined. These are:

  - CHAR fields may contain any combination of displayable characters, including letters, digits, blank spaces, punctuation, and special characters.
  - ALPHA fields may contain any combination of letters, either upper- or lower-case.
  - TIME fields may contain a time of day in the format HH24:MM:SS.
  - NUMBER fields may contain any number, with or without a sign, or decimal point, or scientific notation.
  - INT fields may contain any integer–a number without a decimal point.
  - MONEY fields may contain a number representing a sum of money.
  - RNUMBER, RINT, and RMONEY fields are right-aligned, instead of left-aligned.
  - DATE fields may contain a date in the format DD:MM:YY.
  - JDATE (Julian Date) fields may contain a date in the format MM/DD/YY.
  - EDATE (European date) fields may contain a date in the format DD:MM:YY.

  In INGRES, the data types are:

  - CHAR(N) is the fixed-length character string including only the printable characters.
  - VCHAR(N) is the variable-length character string including all the characters except the NULL.
  - VARCHAR(N) is the variable-length character string including all the characters.
  - F(N) is the floating point number.
  - FLOAT is the same as f(n).
  - I(N) is the integer number with a length of two bytes.

- INTEGER is the integer number with a length of four bytes.
- DATE data type columns holds absolute dates, absolute types, or time intervals.
- MONEY data type contains decimal currency data. INGRES provides great flexibility with regard to the money data type. Using environment variables or a set command, the local currency conventions can be adopted.

The data types for a 4GL and how these are implemented in ORACLE and INGRES are summarized in Table 3.1

Table 3.1. Canonical Data Types

Canonical data type	ORACLE	INGRES
character	CHAR ALPHA	CHAR(N) VCHAR(N) VARCHAR(N)
integer	INT	I(N) INTEGER
float	NUMBER	F(N) FLOAT
date	DATE JDATE EDATE	DATE
money	MONEY	MONEY

- DATA ENTRY: In ORACLE, data can be entered in the field, block, or form level. The INSERT statement inserts one or more rows into the table. The row that is recently entered becomes the last row in the table, if there is more than one row in the table.

  In INGRES, rows can be called with numbers. For example:

```
INSERTROW ROWNAME(4)
```

  opens up a new row immediately following row (4) in the table field.

  In ORACLE, inserts can be made to which are defined as the base tables in the block levels. In INGRES, there is no limitation.

DELETE and UPDATE statements works the same way in both database systems. INGRES has a different function called *validate*. This function performs validation checks on simple fields or table fields as they are defined with the Visual-Forms-Editor. If it is used without a parameter, the validate function performs a validation on every field in the current form.

In ORACLE, this is transparent to the user, and done by SQL*Forms. The values which are entered into the fields are immediately checked during the run-time.

- CREATING THE ENTRY FORM: In ORACLE, the application starts from a form called the entry form. To run the application, the name of this form is typed in the operating system prompt. INGRES does this via a frame called topframe.

- CREATING A DEFAULT SCREEN: ORACLE creates a default screen in the block level. Selecting the default item from the BLOCK window and specifying the default table name is enough to see the fields of that table.

  In INGRES, the designer can do this in ABF in two steps. While in QBF, VIFRED should be activated and from the 'create operations', 'table default operation' is chosen. INGRES has a unique and readily available function in the QBF level. This function is called 'Jointable' which makes the joining of two or more tables easier, when this is necessary.

- TESTING: In ORACLE, testing is realized in the same way as running an application. Additionally, there are two debugging utilities. The first one simply works by selecting the 'debug' operation in the form level. The second one is a pre-defined key. When this key is hit, after running into an error message, it gives detailed information about the error.

  INGRES applications can be tested in the run-time, as well. In both systems, it is better to run (test) the application on the system level. This will decrease the amount of overhead imposed by SQL*Forms and ABF, and gives a quick response to the user.

- QUERIES: In ORACLE, queries can be executed either by the pre-defined key or it can be typed into a trigger statement. When the trigger statement goes off, the query is executed. Complex queries must be typed in INGRES in the 4GL code itself.

  In ORACLE triggers seem to be an advantage. Instead of using a pre-defined key, writing SQL statements inside the 4GL code, or even instead of typing the 4GL code itself, triggers can be used, saving the application developer a lot of time.

  In INGRES, application flows with the INGRES-defined or user-defined functions.

The canonical features of a 4GL are satisfied in both database systems. However, the way these features were satisfied is not the same. Below, in Table 3.2, the canonical features of a 4GL are listed. The check marks indicate that that feature is satisfied. As seen, both ORACLE and INGRES satisfy these features. These are very general and apply to most applications. After the needs for the application are specified, the satisfying 4GL product can be used.

Table 3.2. Canonical Features of 4GLs(1)

Feature	ORACLE	INGRES
Simple queries	√	√
Simple queries and updates	√	√
Complex queries and updates	√	√
Database creation	√	√
Intelligent database operations	√	√
Generation of data-entry screens for key-entry operations	√	√
A procedural language	√	√
Spreadsheet manipulation	√	√
Multidimensional matrix manipulation		
Report generation	√	√
Report manipulation	√	√
Graphics manipulation	√	√
Decision support for What-if questions	√	√
Mathematical analysis tools		
Financial analysis tools		
Text manipulation	√	√
Designed for on-line operations	√	√
Easy debugging	√	√

IV. Building The Application In ORACLE

This chapter describes the process of building database application using the SQL*Forms by ORACLE. The Theater War Exercise is an implemented application in INGRES RDBMS. In this thesis effort the same application is implemented in ORACLE.

4.1 Approach

In designing the TWX application, the prototyping approach was used.

The word 'prototype' literally means 'first of the type.' The approach is to create a prototype for experimentation. The emphasis is on determining the adequacy of a proposed solution before investing in a large final system. The following general activities occurred in this design:

- Preliminary logical design of the database.

  The database that supports the TWX was already created. The relationships between the entities were realized. So, there was little to do in designing the logical database except the transfer of the raw data from the old system to the new one. The raw data which was kept in database tables was transported to ORACLE.

- Construction of the generation modules.

  In the construction of the modules, the logical flow of the program was the key to the design. To have a better idea about this flow, the application in the old system was observed and the game, the Theater War Exercise, was played. In this phase, the modules were distinguished.

  In Figure 4.1, these modules are shown. Each rectangle and the cloud represent a module that are dependent on another. The hard-lined rectangles represent the modules that are completed, whereas the dash-lined rectangles represent the ones that are to be completed. The cloud is the Land Simulation module which originally is not implemented in the INGRES RDBMS.

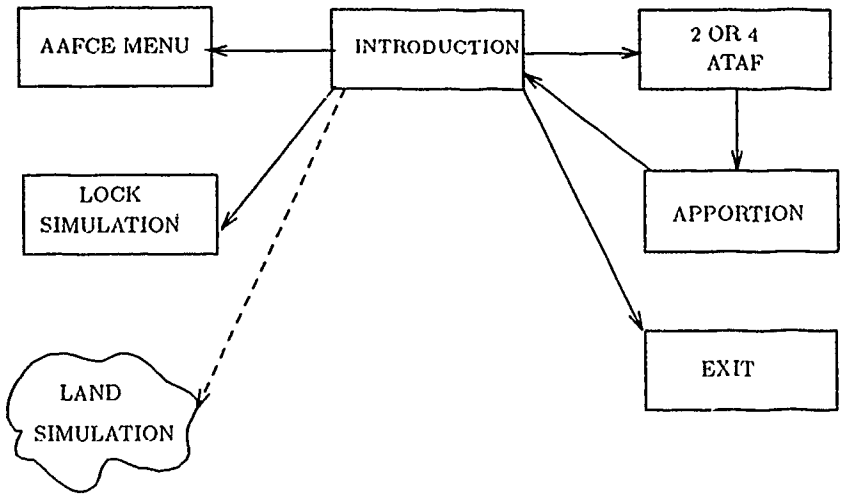


Figure 4.1. The TWX Application in ORACLE

- The necessary changes to the database.

  In Chapter III, a detailed discussion is given about the ORACLE and INGRES 4GLs. There is a major difference between these two RDBMSs. This is assumed to be a big disadvantage for ORACLE when designing an application in SQL*Forms. However, later on this turned out to be an advantage.

  In ORACLE SQL*Forms, in the block level, in order to update, delete, or insert data to the database, a base-table has to be chosen. Otherwise, the data manipulations can not be accomplished. This brought up a problem. If a base-table has to be chosen for each block, and , the source code practically had to be doubled, because the structure of the tables for the users BLUE and RED are identical but the data is different.

  At this point, an other feature of ORACLE was used. ORACLE allows the designer to define tables as variables, thus, cutting the source code down in half.

  In order to assign variables, some changes were made to the database itself. Since the tables were created in the database under the name 'OPS$TWX', The database was connected under this name. Then two user names were created and they were given access to the database. This is accomplished by this command which was given after the SQL prompt in ORACLE's SQL*Plus:

```
SQL> GRANT CONNECT, RESOURCE TO USER BLUE
          IDENTIFIED BY PASSWORD;
SQL> Grant succeeded

SQL> GRANT CONNECT, RESOURCE TO USER RED
          IDENTIFIED BY PASSWORD;
SQL> Grant succeeded
```

  At this time, these two new users were given access only to connect to the system.

  A user must have CONNECT privilege in order to access data in an ORACLE database. Every user with CONNECT is identified by both an ORACLE username

and a password. ORACLE usernames must be distinct within a database, regardless of their passwords. A user with only CONNECT privilege may (14):

- access the ORACLE database
- query (look at) other users' data (SELECT from tables and views), if SELECT access has been granted to the user or to the public
- perform data manipulation operations (INSERT, UPDATE, DELETE) on other users' tables, if the appropriate access has been granted
- create views, synonyms, and database links
- perform table or user exports.

If a user has both RESOURCE and CONNECT system privileges, then he has all the privileges associated with the CONNECT privilege and in addition he may create database objects, such as tables, indexes, clusters, and sequences. He may also enable or disable the auditing of his objects and grant to or revoke (opposite of grant) from other users privileges on his objects (10:16-18).

The tables in the OPS$TWX account are related to both users BLUE and RED. Some of the tables are being used in common by both users. All the tables in the database (125 tables), were given access to the related users one by one. If the table was related to user BLUE, it's given access to the BLUE user by this command in SQL*Plus while connected as OPS$TWX:

```
SQL> GRANT ALL ON TABLE TO BLUE;
```

There are certain options coming from the word GRANT in the above statement. If SELECT is used then the granted user can select, view and query tables from the specified table. If UPDATE is used, then the user can update the table which is granted. In case of a DELETE, user can delete the data. If the option ALL is used, the user has the options that are explained above.

The tables that specifically belong to the user BLUE were granted to that user with all the privileges. Because, the user BLUE should be able to manipulate the data

that belongs to him. The same steps were taken for the user RED. Eventually, it was seen that some of the tables are being used commonly. In these cases, both users were granted with the same privileges. So, in the application both users will be able to read the data in these common tables.

In some phases of the application, the user BLUE needs to check some tables that are granted to the user RED. The same necessity occurs for the RED user. For these tables, a SELECT permission is given to each side, thus, allowing them to view the data but not to change it.

ORACLE allows the user to call the tables in other names by creating synonyms for them. Synonyms are the names assigned to a table or a view that may then be used more conveniently to refer it. If the user has access to another user's tables, as in this case, he might create synonyms for it and refer to it by the synonym alone, without entering the user's name as a qualifier, which is somewhat cumbersome (11:5-64,5-65).

The command to create a synonym is:

```
SQL> CREATE SYNONYM [user.]synonym FOR [user.]table;
```

Synonyms were created for every table that is granted to users BLUE and RED by connecting to SQL*Plus in their names and passwords. The tables that are used only by BLUE are identical to the ones that are used by RED. The number of fields and the field names were also identical. This became an advantage in creating synonyms for the tables in BLUE and RED accounts. Since the tables were identical the same synonyms were created for the tables. In the design phase, this became a real advantage. In the block level (in SQL*Forms) only one table name could be used for each block. Since the SQL*Forms recognizes the user when he is connected to the forms, it makes the necessary changes to the appropriate tables.

Obviously, the same synonyms were used for the common tables. Such as WEATHER, USER_ID, or, TERMINAL_CON. Creating the synonyms provides both the data independence and location transparency: synonyms permit application to function

without modification regardless of which user owns the table or view and regardless of which database holds the table or view (11).

After these changes to the database, the application could be designed.

4.2 Designing The Application

In this section, instead of explaining the design phase of the TWX application in detail, step by step, a general methodology to design 'any' application is followed. The examples are especially chosen from the TWX application. Since only examples are given, this section doesn't cover the TWX application as a whole. Given the information above and below, readers of this thesis who are eager to learn more about the application are encouraged to see the application themselves. Not only can they see the application in the design phase, but also they can run the application and become a part of it.

For the time being, the application is running on a Sun386i which is located in the Wargaming Lab in AFIT. It can be run on any Sun station by remotely logging on to Moss, the current system. The forms can be run only in the shell environment. If the window size is not big enough and if the run command is not given in the shell, the system will be locked.

The main unit in building the application is a form. An application consists of one or more forms. It's a good idea to break down the application into as many forms as possible. This has some advantages. It decreases the form saving time and the form generating time.

On the other side, if there are separate logical blocks, each of these blocks can be designed as different forms in modules as it is in an Object Oriented Language. This improves the understandability of the code itself. In addition, if the application is to be modified or maintained later, these 'modules' become really handy. If one of the forms were to be modified, this would not effect the others.

The CHOOSE FORM window serves as a main menu in ORACLE's SQL*Forms. Here the name of the form that is to be worked on is entered (Figure 4.2).

Below, the different functions in the FORM level are listed:

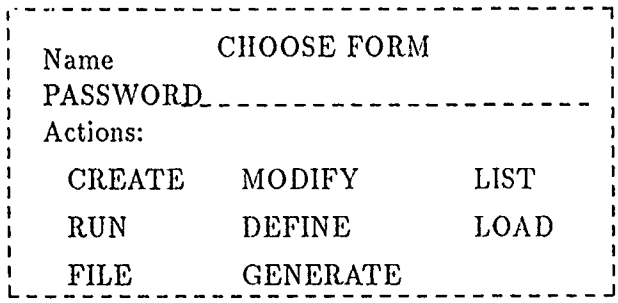


Figure 4.2. Forms Design in SQL*Forms

- < *CREATE* > creates a form.
- < *MODIFY* > lets the designer make changes to an existing form.
- < *LIST* > provides a list of existing forms that can be worked with.
- < *RUN* > runs a form for the operator to use or for the designer to test.
- < *DEFINE* > can change the form name or title and the validation unit. Also form triggers can be triggered.
- < *LOAD* > loads an .INP file into memory. To preserve the new form, it should be saved, and then it will appear in the list of forms.
- < *FILE* > can save or discard changes to a form, create a copy of the form under a new name, rename the current form, or delete the form.
- < *GENERATE* > converts the form definitions for the current form in working memory into a file that can be RUN. A form can not be run until it is generated. Any time a form has been modified, it must be regenerated to incorporate the changes. Otherwise, the older version would be running.

4.2.1 Creating a Form After the whole application is divided into modules in the design phase, one of the them is chosen to be the entry module. Each module corresponds to a form. Namely PASSWORD is the entry module that is designed as a form in the

TWX application. Now the whole application can be run by starting from this entry form. In Figure 4.2, the form PASSWORD can be seen.

To create the entry form (or any form), the name of the form must be typed into the NAME item in the CHOOSE FORM window and the CREATE item is chosen. Soon the CHOOSE BLOCK window can be seen (Figure 4.3).

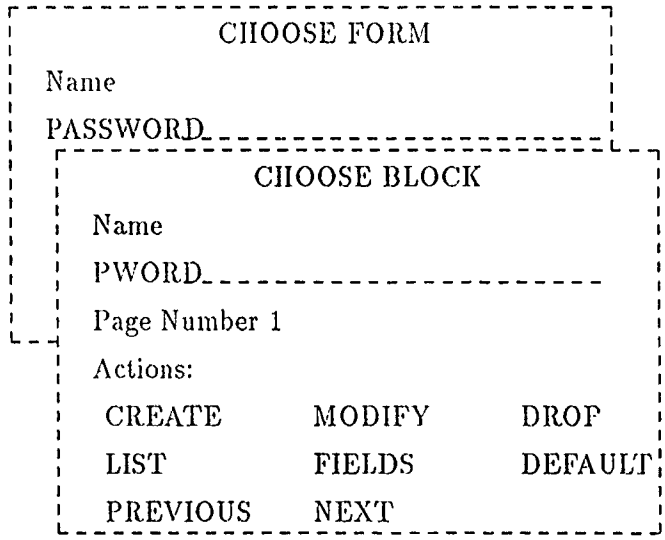


Figure 4.3. Block Design in SQL*Forms

4.2.2 Creating Blocks A form consists of one or more blocks. Blocks are the building units of each form. Each corresponds to one database table called the base-table. However, if the block won't be updated, deleted, or inserted into, it may not have to correspond to a database table. An example of this is a block that has only text in it.

Some important items and their functions in the CHOOSE BLOCK window are:

- < *CREATE* > lets the designer create a new block and puts him in the screen painter. Screen painter is the part of SQL*Forms where the custom forms are created and modified. In the screen painter blocks and fields are defined and modified and triggers are specified.

- < *MODIFY* > lets the designer make the changes to an existing block. If an existing block name and the correct page number for that block appear in the window, selecting MODIFY displays the block in the screen painter.
- < *DROP* > deletes the block that is chosen.
- < *DEFAULT* > creates the blocks with default settings. This is a quick and easy way to construct a form.

As shown in Figure 4.3, the PWORD block under the form PASSWORD is not created by the defaults, because, there is no base-table supporting this block. Even though the fields in this block are coming from three different tables in the database, no base-table is chosen. Reason being that there is no update, delete, or insert to and from any of these tables in the database.

If all the fields were coming from a table in the database then that particular table could be chosen as the base-table for that block. In this case, block can be created by using DEFAULT item. Besides, using the TABLE subitem in this DEFAULT window, one can pick the right table. SQL*Forms can let the designer view all the tables in the database. All the designer have to is to highlight the right table and hit the return key. By using the COLUMN subitem, one can pick the desired fields from a table in the database.

If the designer wants to create a multi-record block–one that can be displayed as several records from a field in the database at once, he can do this in the block level. In Figure 4.4, SPECIFY BLOCK OPTIONS window includes the parameters that could be changed. Although the screen painter displays only one row from each field, when the form is run, the specified number of rows will be seen on the screen during the runtime.

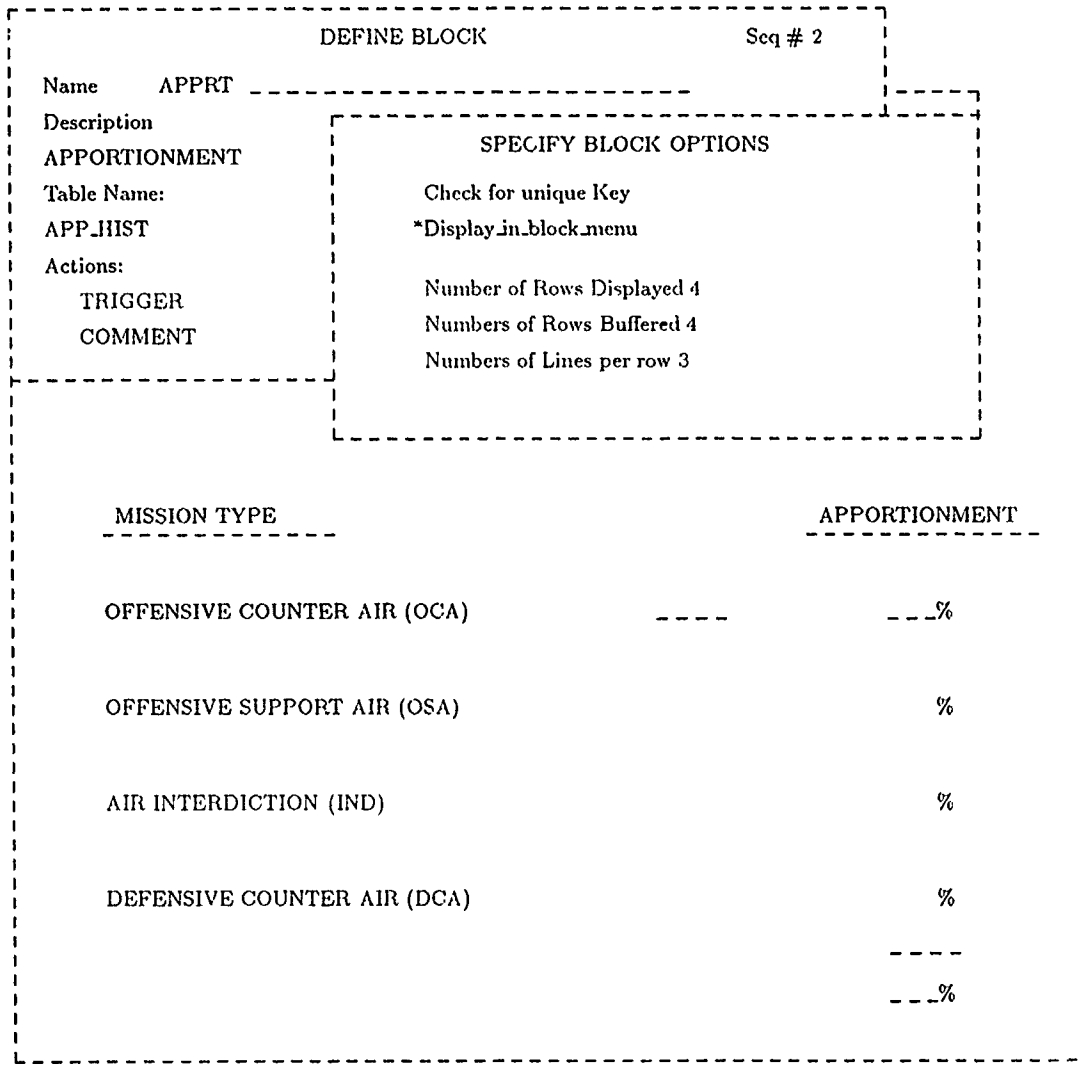


Figure 4.4. The APPORTIONMENT Block in the Design Phase

If any of the fields on the block on which multiple records are shown, does not belong to the base table chosen for that block, then, what will be seen on the screen during the runtime will be a clutter. One of the blocks (APPORTIONMENT) can be seen in Figure 4.4 as it is in the screen painter. Again in Figure 4.5, the same block is seen during the runtime with the multi-records.

SEMINAR : 1

SIDE : BLUE

CONFLICT DAY : 0

PROJECTED APPORTIONMENT FORM

MISSION TYPE		APPORTIONMENT
DEFENSIVE COUNTER AIR (DCA)	DCA	25 %
OFFENSIVE COUNTER AIR (OCA)	OCA	25 %
AIR INTERDICTION (IND)	IND	25 %
OFFENSIVE SUPPORT AIR (OSA)	OSA	25 %
		100 %

Figure 4.5. The APPORTIONMENT Block During Runtime

As explained earlier, a page is practically the same as a screen. On a page, there can be more than one block. The multiple tables might be needed to be updated, deleted, or inserted by viewing them altogether. As a matter of fact, the APPORTIONMENT block in Figure 4.5, consists of two blocks. In one block, the multi-records need to be seen bringing the necessity of a single base-table for that block. On the other side, the second block carries the crucial information for the user. Thus, the two blocks have to be on the same page simultaneously.

4.2.3 Saving, Running, and Testing the Form The TWX application is a multi-form environment. The first form, PASSWORD, is the entry form. This form has two blocks. Until this form (thus any other form) is saved, the changes made remain in working memory, but not in permanent storage. When the form is saved, all of the created form definitions are permanently stored in a database. To run the form, these form definitions must be converted into a file that can be run by SQL*Forms. This process is called 'generating' the form.

In order to save the file, the designer has to be in the FILE window. Here, he can save the current form (SAVE). He can discard the changes made to the current form (DISCARD). He can save a copy of the current form under an other name (SAVE AS). He can rename it (RENAME) or he can permanently delete the form from the database (DROP).

There are couple of points that can save the designer some minutes, or even hours. SQL*Forms doesn't let the designer go out of the form before he saves the changes he made to the form. Sometimes, he does not even have to make any changes to the form. SQL*Forms assumes that there has been a modification (when there is not) when the MODIFY key is hit. This precaution leaves no doubt about securing the current form against accidental mishaps. However, the designer might drop the current form accidentally. In this case, he can rename the form under a different name and later he can load the original form back again, thus, saving the previous one.

If a block is accidentally dropped (the same as deleted), in the form level, the current form can be discarded. What happens is the latest changes to that form are ignored. So,

the form can be reopened in the previous version of it.

After the form is created, the database tables can be viewed. All the columns in the table can be accessed through the form. If one or more columns from a table are not viewed by the form, then these can not be altered during the runtime.

4.2.4 Modifying the Form Through SQL*Forms, new fields can be created. These new fields don't necessarily have to belong to a table in the database. A drawn field can be moved to any place on the screen together with all its attributes. However, if a whole block is being moved inside a form, or from one form to an other one, then the block level attributes are lost. Although every field in that block keeps its attributes and triggers, block level 'where, order-by' clauses, block level triggers, multiple-record specifications turn into default values.

If a block is going to be moved, this can be done in the '.INP' file. But, the designer must use caution when modifying this file. Since when the forms are loaded by their .INP files, any minor change in the syntax can cause not to be able load the form both in the design-time and in the runtime. In SQL*Forms, the debugging tools do not work, if the form can not be loaded. Lots of precious time can be wasted after modifying the .INP file.

While in a block, the attributes of the field can be changed, modified, or redefined. To set the attributes or to change them, while on a field, simply the DEFINE FIELD window is chosen. In this window, the important attributes that could be set are:

- TRIGGER: Displays additional windows that are used to define triggers.
- ATTRIBUTES: Displays the SPECIFY ATTRIBUTES window, where the designer determines various attributes of the current field.
- VALIDATION: Displays the SPECIFY VALIDATION window, where the field length, query length, and criteria that validate what the operator enters are explained.
- COLUMNS: Displays the LIST COLUMNS window, which shows the name of each column in the base table for the current block.

From the TWX application, many examples can be given for the above field attributes.

In the PASSWORD block, after the user enters the seminar number, a post-field trigger checks the user input and validates it. If the seminar number is not the same one as the activated seminar number in the database, a message on the screen warns the user and forces him to enter a valid one.

In the same block, the SIDE field which consists of only one character rejects two digit insertions by validating the input. In this block, when the password is typed, it can not be seen. This is done by setting the attribute echo to off in the ATTRIBUTE WINDOW.

4.2.5 Triggers The application TWX is driven through SQL*Forms triggers. Triggers are activations placed strategically throughout the form. When certain inputs or events occur, trigger goes off. When this happens, the contents of the trigger–special rules and instructions written by the designer–are executed. SQL*Forms offers a wide range of built-in logic capability. With triggers, the designer can do even more. Triggers give the designer the power to take the shell of a form and craft it into a sophisticated application (12).

The capabilities of triggers and related examples from the application are listed below.

- Triggers can validate data entry in several ways.

  In the MISSION_INPUT block which is in the APPLICATION form, the field mission_number has a trigger that validates the data. When the line number is entered, this is compared to the one in the database. If this is not a valid line number then the trigger goes off and forces the user either to enter a valid number or to exit the application.

- Triggers can protect the database from operator errors, such as the entry of duplicate records or the deletion of vital records.

In PRIM_AC block (primary aircraft) which is in the APPLICATION form, there are four triggers in the block level. One of them prevents the user from deleting the primary aircraft entries while there are other aircraft entries from other kind, thus, protecting the application from accidental or intentional data manipulation.

- Triggers can limit operator access to specified forms.

  While in the APPORTIONMENT form, after the user enters the percentages, the entry-exit flag is set to one. So, when he tries to come back to the APPORTIONMENT form, he cannot enter the same form. He can only use this block once.

- They can display related field data by performing table look-ups.

  In the APPORTIONMENT form, there are two related blocks. One of them is the SORTIES_AVAILABLE and the other one is the PRIMARY_AIRCRAFT blocks. In the first one, the user can view the available data which shows the available aircraft that can be assigned. By looking at these records, the user can assign the aircraft in the PRIMARY_AIRCRAFT block. As a matter of fact, while the user assigns the aircraft, the total number decreases in the other block making immediate updates.

- They can compare values between fields in the form.

  In the APPORTIONMENT form and in the same-named block, the user first enters the apportionments (percentages) for the aircraft before he begins the game. Later, during the game, according to what he has played, he can compare the played-values with the predicted numbers.

- They can calculate field values and display the results of field calculations in different fields.

  It is quite easy to calculate the numbers in rows and put the result in a field. If this is going to be done for the columns in the runtime and dynamically, it will be quite complicated to come up with the total. Yet, this was the case in this application.

  In the APPORTIONMENT form, a summation of the numbers was necessary during the runtime. Such that, when the user inputs four percentages the total should be reflected on the screen instantly and dynamically. This screen is shown in Figure 4.5.

In ORACLE SQL*Forms, in order to calculate the total, a tutorial paper that was specially prepared for this purpose became very useful. (9).

In this calculation, five different triggers were used. First, the old values and the old sum were copied to an other field that wasn't seen on the screen before coming to the block. It was designed in such a way that while the user enters the different percentages (in tens), the old value is erased and the new one is either subtracted or added to the old sum depending on that the new value. So, the total is kept updated every time a new value is entered.

As a rule in the application, the summation of these four values should add up to one hundred, because these four numbers are percentages. Furthermore, an other trigger doesn't let the user go out of the block unless the summation is not equal to one hundred.

- Triggers can enforce block coordination during insert, update, delete, or query operations.

  In SQL*Forms, via triggers, all the insert, update, delete, or query operations can be fired and committed to the database. This can be done in the block or form level depending on the trigger that is being used.

  In the APPLICATION form, there is a screen where nine blocks are on the screen altogether. This screen can be seen in Figure 4.6. Four of these blocks are for the user to enter the values. After the user is done with entering to all of these fields, he simply hits the commit key, and in the form level the commit-key trigger fires and commits all the changes that are made. If there are no changes made, then the trigger does not fire.

2ATAF OFFENSIVE_COUNTER_AIR_MISSION_INPUT

DAY. 0

CYCLE: DAY

SIDE. BLUE

MISSION LINE #: 1004

TARGET # 34

PRIMARY AIRCRAFT

TYPE	ROLE	SORTIES

ATTACK SORTIES AVAILABLE

TYPE	ROLE	SORTIES
F4	A	12
F16	A	117
NF5	A	162
TOR	A	352

ESCORT AIRCRAFT

TYPE	ROLE	SORTIES

ESC SORTIES AVAILABLE

TYPE	ROLE	SORTIES
F4	A	12
F16	A	117
F15	D	257

DSUP AIRCRAFT

TYPE	ROLE	SORTIES

DSUP SORTIES AVAILABLE

TYPE	ROLE	SORTIES

ECM AIRCRAFT

TYPE	ROLE	SORTIES

ECM SORTIES AVAILABLE

TYPE	ROLE	SORTIES

Figure 4.6. A Multi-Block Screen (Page)

- Triggers expand the functionality of function keys.

  As stated earlier, one of the trigger types is key-defined triggers. There are twenty six key triggers in SQL*Forms. Some of them and their functions are as follows:

```
<KEY-CLRBLK> : clears the specified block
<KEY-CLRFRM> : clears the specified form
<KEY-CLRREC> : clears the record
<KEY-DELREC> : deletes the record
<KEY-EXEQRY> : executes a defined query
<KEY-EXIT>   : exits the current block or the form and either goes to
               the operating system prompt or to the form (or block)
               it is called.
<KEY-PRVREC> : goes to the previous record
<KEY-NXTREC> : goes to the next record
```

  Key triggers are most often used to disable keys or perform complex or multiple functions with a single key strokes.

4.2.6 Validating Data With SQL Statement With the additional SQL statements, greater capabilities can be added to the application.

Validating data by displaying related data: In the logistic movement form (LOG_MOV), some logistic movements are done by the user. He selects the airfields that the movement is going to be done from and to. There is a post-change trigger in both of these fields. In this trigger, the following SQL statement takes place:

```
SELECT 'X'
FROM    DUAL, BL_FIELD
WHERE  :LOG_MOVE.FIELD1 = BL_FIELD.FIELD
```

This trigger forces the user to enter valid airfields. In case of an invalid airfield number, a failure message can be typed as well. Thus, warning the user about the mishap.

```
There is no such field, please reenter!
```

The DUAL table is an unconventional use of SQL. It is a device exploited by SQL*Forms to do one thing–to set up a true or false test on field values (or system values, such as the system date) in a form.

A trigger that uses the DUAL table isn't interested in retrieving anything from the database. The DUAL table trigger only wants to know whether the WHERE CLAUSE in the SQL statement is true of false; in other words, a 'yes' or 'no' answer.

Consider the LOG_MOVE trigger. If the statement in the WHERE clause happens to be true, then the SELECT statement will fetch the value 'X' from a dummy table called DUAL. If the statement is false, then the trigger fails without fetching the value 'X'.

DUAL is an actual database table–the smallest conceivable table-supplied by ORACLE. As shown below, DUAL is a one row, one column table that contains a single constant value, 'X'.

```
DUMMY
-----
X
```

V. *Conclusions and Recommendations*

5.1 Overview

This chapter gives a summary of the solution to the problem that I stated in the introduction section and draws conclusions about it. Recommendations are presented for further courses of actions of this study.

5.2 Summary of Research

The goal of this thesis effort was to substitute the ORACLE RDBMS for INGRES RDBMS in this application, namely, *the Theater War Exercise.* The relational data was easily transferred. Then, the 4GL applications were built in ORACLE by using SQL*Forms utility. There were four benefits from this effort. The first two of them were directly related to the application and very necessary to draw conclusions about the results. The last two were possible side benefits. These are:

1. Determine a baseline for 4GL comparison in general
2. Compare and contrast the ORACLE and INGRES 4GLs
3. Development of a canonical set of 4GL functions that can be used to compare other systems
4. Development of automated methods to transfer INGRES 4GL to ORACLE 4GL

The work accomplished in this thesis effort may be summarized as follows:

- A detailed explanation of the 4GLs including answers to these questions:
  - What are they?
  - How were they evolved and why were they needed?
  - What can be done with them and how can they be used in the database arena?
  - What are the differences between some of the 4GLs and what criteria are used to bring the differences up?

- A detailed study of the ORACLE and INGRES 4GLs including the tools they use to build applications
- A comparison between the two 4GLs
- Building the application in ORACLE by using SQL*Forms

5.3 Conclusions

There were two areas in this effort when I undertook the study that were closely related to each other. First, a deep understanding of the 4GLs was necessary through a dense research. Then, the first theoretical knowledge should have been combined with the practicality of the application creation.

When I accomplished these, I achieved the first two goals of the effort; compare and contrast the two 4GLs and the building TWX application in ORACLE RDBMS.

Between the two 4GLs, ORACLE seemed easier for building either custom-made or default forms and applications. Again ORACLE is superior by allowing the designer to assign variables to database tables by saving disk space. ORACLE is very fast in the new system compared to INGRES in the old system.

The application built by ORACLE's SQL*Forms as far as its contents is very much like its counterpart INGRES. More than half the application was redesigned successfully.

5.4 Recommendations

Because of the time constraints during this study, I was more concerned about the main goals. I certainly spent most of my time tackling the problems I encountered in the course of this study. It was surprising for me to find some of the solutions to the problems in the design issue at the very last moments.

The development of a canonical set of 4GL functions can be derived after closely examining the application in the runtime and going through the source code in the mean time. The source code can be parsed depending on the functionality. Then, discrete 4GL functions can be derived. These functions may be used in the evaluation of other 4GLs.

Any project whose goal is some kind of application design by using 4GLs may have side benefits by exploring these 4GL functions. This kind of study leads to the understanding of the real capabilities of the other 4GLs.

Because of ORACLE SQL*Forms' almost totally automated design phase, there was hardly any motivation to develop the automated methods to transfer INGRES 4GL to ORACLE 4GL. Yet, this is a necessity and should be implemented in another study. Once this process is automated, the transfer of the applications will be quicker. Although this might seem quite accomplishable, the fact that the both 4GLs are running on different systems make the job harder.

Although this thesis effort is quite successful in many ways, the application creation in ORACLE has not been finished yet. I strongly recommend it be completed in another thesis effort to evaluate both systems in the best way possible.

Bibliography

1. Martin Arben. *Fourth Generation Languages.* Prentice-Hall, Englewool Cliffs, N.J., 1986.
2. Michael S. Brooks. Develop a new database and support software for the theater war exercise. Master's thesis, AFIT/GCS/ENG/87, School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, Oh, 1987.
3. CompuServe Data Technologies. *The 1990 Guide to High-Performance 4GL/RDBMS Applications*, part 2 edition, 1990.
4. Daniel J. Cronin. *Mastering ORACLE.* Hayden Books, 4300 West 62nd Street, Indianapolis, IN 46268, 1989.
5. Peter J. Gordon. A graphical player interface to the theater war exercise. Master's thesis, AFIT/GCS/ENG/89D-5, School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, Oh, 1989.
6. W. Gregory and W. Wojtkowski. *Applications Software Programming with Fourth-Generation Languages.* Boyd & Fraser Publishing Company, Boston, 1990.
7. Mark S. Kross. Developing new user interfaces for the theater war exercise. Master's thesis, AFIT/GCS/ENG/87-19, School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, Oh, 1987.
8. J. Martin. *Application Development Without Programmers.* Prentice-Hall, Englewool Cliffs, N.J., 1982.
9. Keith Morrison. Totaling columns in sql*forms. *Advanced InfoStructures*, 1, 1989.
10. Oracle Corporation. *Oracle Database Administrator Guide*, 6.0 edition, November,1988.
11. Oracle Corporation. *SQL Language Reference Manual*, 6.0 edition, November,1988.
12. Oracle Corporation. *SQL*Forms Designers Reference*, 6.0 edition, November,1988.
13. Oracle Corporation. *Oracle Call Interfaces (OCIs)*, 6.0 edition, September,1989.
14. Oracle Corporotion. *Oracle RDBMS Database Administrator's Guide*, 6.0 edition, November,1988.
15. David Pepin. *ORACLE Programmer's Guide.* QUE Corporation, 11711 N. COLLEGE Ave., Carmel, IN 46032, 1989.
16. Darrell A. Quick. Developing map-based graphics for the theater war exercise. Master's thesis, AFIT/GCS/ENG/88D-16, School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, Oh, 1988.
17. Relational Technology. *4GL Application Development Guide*, 1987.
18. Relational Technology. *Ingres ABF/4GL*, 6.0 edition, 1988.
19. Jonathan Sayles. *SQL Spoken Here.* QED Information Sciences, Wellesley, Mess., 1989.

20. K. R. Wilcox. Extending the user interface for the theater war exercise. Master's thesis, AFIT/GCS/ENG/88D-24, School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, Oh, 1988.

Vita

First Lieutenant Adnan Altunisik was born on April 28, 1965, in Ankara, Turkey. Upon graduation from Kuleli Military High School, he entered the Turkish Air Force Academy in Istanbul, Turkey. In August 1987, he received his Bachelor of Science degree in Electrical Engineering as well as his commission as a Second Lieutenant in the Turkish Air Force. Between September 1987 and May 1988, he attended the Turkish Air Force Communications School in Izmir, Turkey. His first duty assignment in May 1988 was as a Communications Officer in the Air Force Training Headquarters in Izmir, Turkey. In September 1988, he was assigned to the Air Force Techical Schools as the Commander of Electronic Maintenance Unit. He entered the School of Engineering, Air Force Institute of Technology in May 1989. After graduation in March (1991), First Lieutenant Altunisik will be assigned to the Turkish Air Force Headquarters, Ankara, Turkey.

Permanent address: Abdulhak Hamit Cad.
#851/A
Mamak, Ankara 06470
TURKEY