

2. .

An Application Development Comparison INGRES vs. Sybase

Strategic Marketing

July 1988

Author: David Kellogg Contributors: Aaron Zornes, Shelby Thornton

MARCOM doc # MBW-098S-001

Intended Audience

This document is intended to provide technical staff with an example of application development in INGRES and Sybase. This document is moderately technical and an understanding of Embedded SQL, INGRES/4GL, and the C programming language is beneficial, but not required.

This document is intended for internal use only. It may be shown to clients or prospective clients, but should not be distributed to them.

Objective

This document provides a brief overview of today's application development technologies and presents sample programs which display the power of INGRES's application development tools. This document also demonstrates how cumbersome call-level interfaces are relative to more modern programming methods. This document is not intended as a general comparison of INGRES and Sybase. Neither is it intended as a full scale comparison of the two systems' application development tools.

Introduction

Fourth generation languages (4GLs) like INGRES/4GL currently offer the highest application development productivity seen in data processing. Regularly, INGRES customers report productivity gains of 500, 600, and sometimes 1000 percent after switching from traditional third generation languages (3GLs), like FORTRAN, COBOL, or C.

Less productive than 4GLs, yet still a great improvement over 3GLs, are embedded programming interfaces where standard SQL statements are embedded into 3GL programs. In Embedded SOL (ESQL), SQL statements are prefaced by the words "EXEC SQL" in an application's source code, and a pre-processor is used to convert these SQL statements into subroutine calls. In ESQL, all file input/output (I/O) is done using SQL commands rather than the host's native file system. Using SQL to perform file I/O is both easier and more portable than using the host file For example, coding an application using VAX RMS system. (Record Management Services) is not trivial, and the resultant application is not portable to a UNIX-based system. In addition, host file systems lack many benefits, such as physical data independence, which are offered by the relational model.

Standard embedded SQL (ESQL) provides only the capability to manipulate data in a database. It does not provide any facility for screen painting or forms control. INGRES extends ESQL by adding the power of a Forms Runtime System (FRS) to the standard Embedded SQL interface. Just as regular ESQL statements are identified by the keywords "EXEC SQL," forms runtime statements are prefaced by "EXEC FRS." The combination of ESQL with an embedded forms system is found only in INGRES, and it adds greatly to programmer productivity. Because the same FRS is used in all INGRES interfaces, all of INGRES (and all user-written applications written in 3GL, 4GL or both) have a common "look and feel."

Systems which lack a pre-processor often offer call-level interfaces instead. Rather than pre-processing SQL, users of call-level interfaces must make calls directly into vendor-supplied subroutine libraries. Using a call-level interface causes program code to expand, increases both development and maintenance costs, and raises the probability of coding errors during development. In general, call-level interfaces yield the lowest productivity gains.

The next section of this paper presents source code for a simple banking program. The examples clearly show the differences among the three previously mentioned environments: 4GL, embedded pre-processor, and call-level.

INGRES/4GL Code, Bank Deposit or Withdrawal

```
initialize = { cur bal = integer4, rows = integer4; }
"Enter" = {
        begin transaction;
        form := select cur bal = balance from account
                where act id = act;
        inquire ingres( rows = rowcount );
        if rows != 1 then
                callproc badaccount();
        endif;
        update account set balance = (cur bal - dbt) where act id = act;
        update teller set cash = (cash - dbt) where tlr_id = tlr;
        update branch set total > (total - dbt) where brn_id = brn;
        insert into history (account, branch, teller, debit, time)
                values (act, brn, tlr, dbt, "now");
        end transaction;
}
"Exit" = { exit; }
```

Notes

1

INGRES/4GL is tightly integrated with the INGRES Forms Runtime System and thus the above code performs more than just the simple banking transaction. The above code creates two screen menu items, "Enter" and "Exit"; takes the desired account, teller, branch, and debit from the screen; and then executes the simple banking transaction when the user selects the "Enter" menu item or FRS key.

In the next two examples, the simple banking transaction is implemented without the implicit form handling that is native in the INGRES/4GL. The next examples implement only database startup, transaction execution, and database shutdown.

ſ

.

٠

.

```
main()
       EXEC SQL begin declare section;
        int cur bal, act, brn, tlr, dbt;
        EXEC SQL end declare section;
        EXEC SQL connect database;
        EXEC SQL whenever sqlerror stop;
        /* get information about withdrawal */
        get_values(&act, &tlr, &brn, &dbt);
        EXEC SQL begin transaction;
        EXEC SQL select cur bal = balance from account
                where act id = :act;
        EXEC SQL update account set balance = :cur_bal - :dbt
                where act_id = :act;
        EXEC SQL update teller set cash = cash - :dbt
                where tlr_id = :tlr;
        EXEC SQL update branch set total = total - :dbt
                where brn id = :brn;
        EXEC SQL insert into history (account, branch, teller,
                debit, time)
                values (act, brn, tlr, dbt, "now");
        EXEC SQL end transaction;
        EXEC SQL disconnect;
```

}

.

Sybase Call-Level Code

```
<stdio.h>
# include
                <sybfront.h>
# include
                <sybdb.h>
# include
# include
                "app.h"
main(argc, argv)
int
        argc;
char
        *argv[];
ſ
        DBPROCESS
                         *dbproc;
        LOGINREC
                         *login;
        DBINT
                        1 account;
        DBINT
                        delta;
        DBINT
                         teller;
        DBINT
                        1 branch;
        DBINT
                        acc_balance;
        DBINT
                        tran num;
        DBINT
                        acc_tid;
        long
                        cnt;
/* Login to the database */
login = dblogin();
DBSETLPWD(login, "");
DBSETLUSER(login, "sa");
if ((dbproc = dbopen(login, (char *)NULL)) == NULL)
ſ
        printf("Can't connect with server\n");
        ex.t(1);
}
/* get the right database */
dbuse(dbproc, "database");
/* start the transaction */
dbfreebuf(dbproc);
dbcmd(dbproc, "begin transaction");
dbsqlexec(dbproc);
/* get the record to update */
dbfreebuf(dbproc);
dbfcmd(dbproc, "execute get_acc %d", l_account);
dbsqlexec(dbproc);
if (dbresults(dbproc) != SUCCEED)
{
        printf("Couldn't retrieve account %d\n", 1 account);
```

1

3

```
dbclose(dbproc);
       dbexit();
       exit(2);
}
dbbind(dbproc, 1, INTBIND, 0, sacc_balance);
dbbind(dbproc, 2, INTBIND, 0, &tran_num);
dbnextrow(dbproc);
acc balance += delta;
while (dbresults(dbproc) != NO_MORE_RESULTS);
/* update accounts */
dbfreebuf(dbproc);
dbfcmd(dbproc, "execute rep_account %d %d",
       acc_balance, tran_num);
dbsqlexec(dbproc);
while (dbresults(dbproc) != NO MORE RESULTS);
/* update cash */
dbfreebuf(dbproc);
dbfcmd(dbproc, "execute rep_cash %d %d",
       teller, delta);
dbsqlexec(dbproc);
while (dbresults(dbproc) != NO_MORE_RESULTS);
/* update branch */
dbfreebuf(dbproc);
dbfcmd(dbproc, "execute rep branch %d %d",
       l_branch, delta);
dbsqlexec(dbproc);
while (dbresults(dbproc) != NO MORE_RESULTS);
/* append to history */
dbfreebuf(dbproc);
l_account, tran num, delta, teller, l_branch,
       "09/20/87", 0);
dbsqlexec(dbproc);
while (dbresults(dbproc) != NO_MORE_RESULTS);
/* end the transaction */
dbfreebuf(dbproc);
dbcmd(dbproc, "end transaction");
dbsqlexec(dbproc);
while (dbresults(dbproc) != NO_MORE_RESULTS);
/* close the database */
dbclose(dbproc);
dbexit();
exit(0);
```

}