Host Variables

```
  EXEC SQL BEGIN DECLARE SECTION;
    int     cno;
    varchar cname[31];
    varchar street[31];
    int     zip;
    char    phone[13];
  EXEC SQL END DECLARE SECTION;
```

The varchar data type is translated by Pro*C into the following:

```
  /* varchar cname[31]; */
  struct {
    unsigned short len;
    unsigned char arr[31];
  } cname;
```

Data Type Correspondence

Oracle Data Type	`C Data Type`
char	char
char(N)	char array[N+1]
varchar(N)	varchar array[N+1]
date	char array[10]
number(6)	int
number(10)	long int
number(6,2)	float

Using Host Variables

```
scanf("%d",&cno);
EXEC SQL SELECT cname
         INTO   :cname
         FROM   customers
         WHERE  cno = :cno;

scanf("%d%s%s%d%s",&cno,cname.arr,
                   street.arr,&zip,phone);
cname.len  = strlen(cname.arr);
street.len = strlen(street.arr);
EXEC SQL INSERT INTO customers
         VALUES (:cno,:cname,:street,:zip,:phone);
```

Indicator Variables

```
EXEC SQL BEGIN DECLARE SECTION;
struct {
    int     ono;
    int     cno;
    int     eno;
    char    received[12];
    char    shipped[12];
  } order_rec;
  struct {
    short     ono_ind;
    short     cno_ind;
    short     eno_ind;
    short     received_ind;
    short     shipped_ind;
  } order_rec_ind;
  int  onum;
EXEC SQL END DECLARE SECTION;
```

Reading null values

```
scanf("%d",&onum);
EXEC SQL SELECT *
         INTO   :order_rec INDICATOR :order_rec_ind
         FROM   orders
         WHERE  ono = :onum;
if (order_rec_ind.shipped_ind == -1)
  printf("SHIPPED is Null\n");
else
  printf("SHIPPED is not Null\n");
```

Writing null values

```
onum = 1021;
order_rec.shipped_ind  = -1;
EXEC SQL update orders
         set shipped = :order_rec.shipped INDICATOR
                       :order_rec.shipped_ind
         where ono = :onum;
```

SQL Communications Area (`sqlca`)

`sqlca.sqlcode`	Interpretation
0	SQL statement executed successfully
> 0	No more data present or values not found
< 0	Error occurred while executing SQL statement

To include the `sqlca` definition, use

```
EXEC SQL INCLUDE sqlca;
```

Error check

```
EXEC SQL SET TRANSACTION READ WRITE;
EXEC SQL INSERT INTO customers VALUES
         (custseq.nextval,:customer_rec.cname,
          :customer_rec.street,:customer_rec.zip,
          :customer_rec.phone);
if (sqlca.sqlcode < 0) {
  printf("\n\nCUSTOMER (%s) DID NOT GET ADDED\n",
         customer_rec.cname.arr);
  EXEC SQL ROLLBACK WORK;
  return;
}
EXEC SQL COMMIT;
```

Not found check

```
EXEC SQL SELECT zip, city
         INTO   :zipcode_rec
         FROM   ZIPCODES
         WHERE  zip = :customer_rec.zip;
if (sqlca.sqlcode > 0) {
  zipcode_rec.zip = customer_rec.zip;
  printf("Zip Code does not exists; Enter City: ");
  scanf("%s",zipcode_rec.city.arr);
  zipcode_rec.city.len =
       strlen(zipcode_rec.city.arr);
  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL INSERT INTO zipcodes (zip, city)
           VALUES (:zipcode_rec);
  EXEC SQL COMMIT;
}
```

Connecting to Oracle

```
EXEC SQL BEGIN DECLARE SECTION;
  varchar userid[10], password[15];
EXEC SQL END DECLARE SECTION;

printf("Enter your USERID: ");
scanf("%s", userid.arr);
userid.len = strlen(userid.arr);
printf("Enter your PASSWORD: ");
system("stty -echo");
scanf("%s", password.arr);
password.len = strlen(password.arr);
system("stty echo");
printf("\n");
EXEC SQL CONNECT :userid IDENTIFIED BY :password;
if (sqlca.sqlcode != 0) {
  printf("Connect Failed\n");
  exit(0);
}
```

Cursors

The syntax for cursor declaration is as follows:

```
EXEC SQL DECLARE <cur-name> CURSOR FOR
  <select-statement>
  [FOR {READ ONLY | UPDATE [OF <column-list>]}];
```

A cursor is opened using the following syntax:

```
EXEC SQL OPEN <cur-name>;
```

The `fetch` statement has the following syntax:

```
EXEC SQL FETCH <cur-name> INTO
         <host-var>, ... <host-var>;
```

The syntax of the `close` cursor statement is

```
EXEC SQL CLOSE <cur-name>;
```

```
void print_customers() {
EXEC SQL DECLARE customer_cur CURSOR FOR
         SELECT cno, cname, street, zip, phone
         FROM   customers;

EXEC SQL SET TRANSACTION READ ONLY;
EXEC SQL OPEN customer_cur;
EXEC SQL FETCH customer_cur INTO
         :customer_rec INDICATOR :customer_rec_ind;
while (sqlca.sqlcode == 0) {
 customer_rec.cname.arr[customer_rec.cname.len] = '\0';
 customer_rec.street.arr[customer_rec.street.len] = '\0';
 printf("%6d  %10s  %20s  %6d  %15s\n",
        customer_rec.cno,customer_rec.cname.arr,
        customer_rec.street.arr,customer_rec.zip,
        customer_rec.phone);
EXEC SQL FETCH customer_cur INTO
          :customer_rec INDICATOR :customer_rec_ind;
}
EXEC SQL CLOSE customer_cur;
EXEC SQL COMMIT;
}
```

Positioned deletes and updates

```
EXEC SQL DECLARE del_cur CURSOR FOR
    select *
    from   employees
    where  not exists
             (select 'a'
              from   orders
              where  orders.eno = employees.eno)
    FOR UPDATE;

  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL OPEN del_cur;
  EXEC SQL FETCH del_cur into :employee_rec;
  while (sqlca.sqlcode == 0) {
    EXEC SQL DELETE FROM EMPLOYEES
             WHERE CURRENT OF del_cur;
    EXEC SQL FETCH del_cur into :employee_rec;
  }
  EXEC SQL COMMIT RELEASE;
```

Mail Order Database Application

```
#include <stdio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0

typedef struct {
  int cno; varchar cname[31]; varchar street[31]; int zip; char phone[13];
} customer_record;
typedef struct {
  short cno_ind,cname_ind,street_ind,zip_ind,phone_ind;
} customer_indicator_record;
typedef struct { int zip; varchar city[31]; } zipcode_record;
typedef struct {
  int eno; varchar ename[31]; int zip; char hdate[12];
} employee_record;
typedef struct {
  short eno_ind,ename_ind,zip_ind,hdate_ind;
} employee_indicator_record;
```

```
typedef struct {
  int ono,cno,eno; char received[12],shipped[12];
} order_record;
typedef struct {
  short ono_ind,cno_ind,eno_ind,received_ind,shipped_ind;
} order_indicator_record;

EXEC SQL INCLUDE sqlca;

void print_menu();
void add_customer();
void print_customers();
void update_customer();
void process_order();
void remove_customer();
void delete_old_orders();
void print_invoice();
void prompt(char [],char []);
```

```
void main() {
  EXEC SQL BEGIN DECLARE SECTION;
    varchar userid[10], password[15];
  EXEC SQL END DECLARE SECTION;
  char ch; int  done=FALSE,loginok=FALSE,logintries=0;
  do {
    prompt("Enter your USERID: ",userid.arr);
    userid.len = strlen(userid.arr);
    printf("Enter your PASSWORD: ");
    system("stty -echo");
    scanf("%s", password.arr);getchar();
    password.len = strlen(password.arr);
    system("stty echo"); printf("\n");
    EXEC SQL CONNECT :userid IDENTIFIED BY :password;
    if (sqlca.sqlcode == 0) loginok = TRUE;
    else printf("Connect Failed\n");
    logintries++;
  } while ((!loginok) && (logintries <3));
  if ((logintries == 3) && (!loginok)) {
    printf("Too many tries at signing on!\n"); exit(0);
  }
```

```
  while (done == FALSE) {
    print_menu();
    printf("Type in your option: ");
    scanf("%s",&ch); getchar();
    switch (ch) {
      case '1': add_customer(); printf("\n"); break;
      case '2': print_customers(); printf("\n"); break;
      case '3': update_customer(); printf("\n"); break;
      case '4': process_order(); printf("\n"); break;
      case '5': remove_customer(); printf("\n"); break;
      case '6': delete_old_orders(); printf("\n"); break;
      case '7': print_invoice();
                printf("\nPress RETURN to continue");
                getchar(); printf("\n"); break;
      case 'q': case 'Q': done = TRUE; break;
      default:  printf("Type in option again\n"); break;
    }
  };
  EXEC SQL COMMIT RELEASE;
  exit(0);
}
```

```
void print_menu() {
  printf("****************************************\n");
  printf("<1> Add a new customer\n");
  printf("<2> Print all customers\n");
  printf("<3> Update customer information\n");
  printf("<4> Process a new order\n");
  printf("<5> Remove a customer\n");
  printf("<6> Delete old orders \n");
  printf("<7> Print invoice for a given order\n");
  printf("<q> Quit\n");
  printf("****************************************\n");
}
void prompt(char s[], char t[]) {
  char c; int i = 0;
  printf("%s",s);
  while ((c = getchar()) != '\n') {
    t[i] = c;
    i++;
  }
  t[i] = '\0';
}
```

Add Customer

```
void add_customer() {
  EXEC SQL BEGIN DECLARE SECTION;
    customer_record crec;
    zipcode_record  zrec;
  EXEC SQL END DECLARE SECTION;

  prompt("Customer Name: ",crec.cname.arr);
  crec.cname.len = strlen(crec.cname.arr);
  prompt("Street       : ",crec.street.arr);
  crec.street.len = strlen(crec.street.arr);
  printf("Zip Code     : ");
  scanf("%d",&crec.zip); getchar();
  prompt("Phone Number : ",crec.phone);

  EXEC SQL SELECT zip, city
           INTO   :zrec
           FROM   ZIPCODES
           WHERE  zip = :crec.zip;
```

```
  if (sqlca.sqlcode > 0) {
     zrec.zip = crec.zip;
     prompt("Zip not present; Enter City: ",zrec.city.arr);
     zrec.city.len = strlen(zrec.city.arr);
     EXEC SQL SET TRANSACTION READ WRITE;
     EXEC SQL INSERT INTO zipcodes (zip, city) VALUES (:zrec);
     EXEC SQL COMMIT;
  }

  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL INSERT INTO customers VALUES
    (custseq.nextval,:crec.cname,:crec.street,:crec.zip,:crec.phone);
  if (sqlca.sqlcode < 0) {
    printf("\n\nCUSTOMER (%s) DID NOT GET ADDED\n", crec.cname.arr);
    EXEC SQL ROLLBACK WORK;
    return;
  }
  EXEC SQL COMMIT;
}
```

Update Customer

```
void update_customer() {
  EXEC SQL BEGIN DECLARE SECTION;
    customer_record crec;
    zipcode_record zrec;
    int cnum;
    varchar st[31];
    char ph[13], zzip[6];
  EXEC SQL END DECLARE SECTION;

  printf("Customer Number to be Updated: ");
  scanf("%d",&cnum);getchar();

  EXEC SQL SELECT *
           INTO   :crec
           FROM   CUSTOMERS
           WHERE  cno = :cnum;
  if (sqlca.sqlcode > 0) {
    printf("Customer (%d) does not exist\n",cnum);
    return;
  }
```

```
crec.street.arr[crec.street.len] = '\0';
printf("Current Street Value          : %s\n",crec.street.arr);
prompt("New Street (n<ENTER> for Same): ",st.arr);
if (strlen(st.arr) > 1) {
  strcpy(crec.street.arr,st.arr);
  crec.street.len = strlen(crec.street.arr);
}
printf("Current ZIP Value      : %d\n",crec.zip);
prompt("New ZIP (n<ENTER> for same): ",zzip);
if (strlen(zzip) > 1) {
  crec.zip = atoi(zzip);
  EXEC SQL SELECT zip, city INTO :zrec FROM ZIPCODES WHERE zip = :crec.zip;
  if (sqlca.sqlcode > 0) {
    zrec.zip = crec.zip;
    prompt("Zip not present; Enter City: ",zrec.city.arr);
    zrec.city.len = strlen(zrec.city.arr);
    EXEC SQL SET TRANSACTION READ WRITE;
    EXEC SQL INSERT INTO zipcodes (zip, city) VALUES (:zrec);
    EXEC SQL COMMIT;
  }
}
```

```
  printf("Current Phone Value: %s\n",crec.phone);
  prompt("New Phone (n<ENTER> for same): ",ph);
  if (strlen(ph) > 1) {
    strcpy(crec.phone,ph);
  }

  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL UPDATE customers
           SET street = :crec.street,
               zip    = :crec.zip,
               phone  = :crec.phone
           WHERE cno = :crec.cno;
  if (sqlca.sqlcode < 0) {
     printf("\n\nError on Update\n");
     EXEC SQL ROLLBACK WORK;
     return;
  }
  EXEC SQL COMMIT;
  printf("\nCustomer (%d) updated.\n",crec.cno);
}
```

Remove Customer

```
void remove_customer() {
  EXEC SQL BEGIN DECLARE SECTION;
    customer_record crec;
    int cnum,onum;
  EXEC SQL END DECLARE SECTION;

  printf("Customer Number to be deleted: ");
  scanf("%d",&cnum); getchar();

  EXEC SQL SELECT *
           INTO :crec
           FROM CUSTOMERS
           WHERE cno = :cnum;
  if (sqlca.sqlcode > 0) {
    printf("Customer (%d) does not exist\n",cnum);
    return;
  }
```

```
  EXEC SQL DECLARE del_cur CURSOR FOR
      SELECT ono FROM orders WHERE cno = :cnum;

  EXEC SQL SET TRANSACTION READ ONLY;
  EXEC SQL open del_cur;
  EXEC SQL fetch del_cur into :onum;
  if (sqlca.sqlcode == 0) {
      printf("Orders exist - cannot delete\n");
      EXEC SQL COMMIT;
      return;
  }
  EXEC SQL COMMIT;

  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL DELETE FROM customers
           WHERE cno = :cnum;
  printf("\nCustomer (%d) DELETED\n",cnum);
  EXEC SQL COMMIT;
}
```

Process Order

```
void process_order() {
  EXEC SQL BEGIN DECLARE SECTION;
    customer_record crec;
    int eenum,cnum,pnum,qqty,ord_lev,qqoh;
  EXEC SQL END DECLARE SECTION;
  FILE *f1; char ch; int nparts;

  EXEC SQL SET TRANSACTION READ ONLY;
  do {
    printf("Employee Number: ");
    scanf("%d",&eenum); getchar();
    EXEC SQL SELECT eno
             INTO   :eenum
             FROM   employees
             WHERE  eno = :eenum;
    if (sqlca.sqlcode > 0)
      printf("Employee (%d) does not exist\n",eenum);
  } while (sqlca.sqlcode!=0);
  EXEC SQL COMMIT;
```

```
do {
  printf("New Customer (y or n)? ");
  scanf("%s",&ch); getchar();
} while ((ch != 'y') && (ch != 'Y') &&
         (ch != 'n') && (ch != 'N'));
if ((ch == 'y') || (ch == 'Y')) {
  add_customer();
  EXEC SQL SET TRANSACTION READ ONLY;
  EXEC SQL select custseq.currval
           into   :cnum
           from   dual;
  EXEC SQL COMMIT;
}
else {
  printf("Customer Number: ");
  scanf("%d",&cnum); getchar();
}
```

```
EXEC SQL SET TRANSACTION READ ONLY;
EXEC SQL SELECT *
         INTO  :crec
         FROM  customers
         WHERE cno = :cnum;
if (sqlca.sqlcode > 0){
  printf("Customer (%d) does not exist\n",cnum);
  EXEC SQL COMMIT;
  return;
}
EXEC SQL COMMIT;

EXEC SQL SET TRANSACTION READ WRITE;
EXEC SQL INSERT INTO orders (ono,cno,eno,received)
         VALUES (ordseq.nextval,:cnum,:eenum,sysdate);
if (sqlca.sqlcode != 0) {
  printf("Error while entering order\n");
  EXEC SQL ROLLBACK WORK;
  return;
}
nparts = 0;
```

```
  do {
    printf("Enter pno and quantity,(0,0)to quit:  ");
    scanf("%d%d",&pnum,&qqty); getchar();
    if (pnum != 0) {
      EXEC SQL SELECT qoh,olevel INTO :qqoh,:ord_lev FROM parts WHERE pno=:pnum;
      if (qqoh > qqty) {
        EXEC SQL INSERT INTO odetails VALUES (ordseq.currval,:pnum,:qqty);
        if (sqlca.sqlcode == 0) {
          nparts++;
          EXEC SQL UPDATE parts SET qoh = (qoh - :qqty) WHERE pno=:pnum;
          if (qqoh < ord_lev){
            EXEC SQL UPDATE parts SET qoh = 5*olevel WHERE pno=:pnum;
            f1 = fopen("restock.dat","a");
            fprintf(f1,"Replenish part (%d) by (%d)\n",pnum, 5*ord_lev - qqoh);
            fclose(f1);
          }
        } else  printf("Cannot add part (%d) to order\n", pnum);
     } else printf("Not enough quantity in stock for (%d)\n", pnum);
    }
  } while(pnum > 0);
  if (nparts > 0) EXEC SQL COMMIT;
  else EXEC SQL ROLLBACK WORK;
  printf("NEW ORDER PROCESSING COMPLETE\n");
}
```

Delete Old Orders

```
void delete_old_orders() {
  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL DELETE FROM ospecs
           WHERE ono in
                  (select ono
                   from orders
                   where shipped <  (sysdate - 5*365));
  EXEC SQL DELETE FROM orders
           WHERE shipped < (sysdate- 5*365);
  EXEC SQL COMMIT;
  printf("ORDERS SHIPPED 5 YEARS or EARLIER DELETED!\n");
}
```

Print Invoice

```
void print_invoice() {
  EXEC SQL BEGIN DECLARE SECTION;
    int zzip,cnum,eenum,onum,pnum,qqty;
    varchar st[31],eename[31],ccname[31],ccity[31],ppname[31];
    char ph[13];
    float sum,pprice;
    order_record orec;
    order_indicator_record orecind;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL DECLARE od_cur CURSOR for
    SELECT parts.pno, pname, qty, price
    FROM   odetails, parts
    WHERe  odetails.ono = :onum and odetails.pno = parts.pno;

  printf("Order Number: ");
  scanf("%d",&onum); getchar();
```

```
EXEC SQL SET TRANSACTION READ ONLY;
EXEC SQL SELECT *
         INTO   :orec INDICATOR :orecind
         FROM   orders
         WHERE  ono = :onum;
if (sqlca.sqlcode == 0) {
  EXEC SQL SELECT cno,cname,street,city,customers.zip, phone
           INTO   :cnum,:ccname,:st,:ccity,:zzip,:ph
           FROM   customers, zipcodes
           WHERE  cno = :orec.cno and customers.zip = zipcodes.zip;
  ccname.arr[ccname.len] = '\0';
  st.arr[st.len] = '\0';
  ccity.arr[ccity.len] = '\0';
  printf("*************************************************************\n");
  printf("Customer: %s \t Customer Number: %d \n", ccname.arr, cnum);
  printf("Street  : %s \n", st.arr);
  printf("City    : %s \n", ccity.arr);
  printf("ZIP     : %d \n",zzip);
  printf("Phone   : %s \n", ph);
  printf("-------------------------------------------------------------\n");
```

```
EXEC SQL SELECT eno, ename
         INTO   :eenum, :eename
         FROM   employees
         WHERE  eno = :orec.eno;
eename.arr[eename.len] = '\0';
printf("Order No: %d \n",orec.ono);
printf("Taken By: %s (%d)\n",eename.arr, eenum);
printf("Received On: %s\n",orec.received);
printf("Shipped  On: %s\n\n",orec.shipped);

EXEC SQL OPEN od_cur;
EXEC SQL FETCH od_cur
         INTO :pnum, :ppname, :qqty, :pprice;
printf("Part No.              ");
printf("Part Name    Quan.   Price     Ext\n");
printf("-------------------------------------------------------\n");
```

```
    sum = 0.0;
    while (sqlca.sqlcode == 0) {
      ppname.arr[ppname.len] = '\0';
      printf("%8d%25s%7d%10.2f%10.2f\n",pnum,
        ppname.arr, qqty, pprice, qqty*pprice);
      sum = sum + (qqty*pprice);
      EXEC SQL FETCH od_cur
               INTO :pnum, :ppname, :qqty, :pprice;
     }
     EXEC SQL CLOSE od_cur;
     printf("-------------------------------------------------------------\n");
     printf("                                    ");
     printf("TOTAL:           %10.2f\n",sum);
     printf("*************************************************************\n");
     EXEC SQL COMMIT;
  }
}
```

Recursive Queries

Consider a simple relational table **emps**:

emps

EID	MGRID
Smith	Jones
Blake	Jones
Brown	Smith
Green	Smith
White	Brown
Adams	White

and the recursive query on the **emps** table:

```
query(X) :- emps(X,'Jones');
query(X) :- emps(X,Y), query(Y).
```

```
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;
  int  eid, a;
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca;

main() {
  int newrowadded;
/* Cursor for emps at next level (Initial answers) */
  exec sql declare c1 cursor for select EID from emps where MGRID = :EID;
/* query(X) if emps(X,Y) and query(Y) */
  exec sql declare c2 cursor for select EID from emps,query where MGRID = A;
/* Cursor to print the answers */
  exec sql declare c3 cursor for select A from query;

  exec sql create table query(A integer not null, primary key (A));
```

```
/*Get initial answers using Cursor c1*/

  printf("Type in employee id:");
  scanf("%d",&eid);

  exec sql open c1;
  exec sql fetch c1 into :a;
  while (sqlca.sqlcode == 0) {
    exec sql insert into query values (:a);
    exec sql fetch c1 into :a;
  }
  exec sql close c1;
  exec sql commit work;
```

```
/* repeat loop of algorithm */
   do {
     newrowadded = FALSE;
     exec sql open c2;
     exec sql fetch c2 into :a;
     while (sqlca.sqlcode == 0) {
       exec sql insert into query values (:a);
       if (sqlca.sqlcode == 0)
         newrowadded = TRUE;
       exec sql fetch c2 into :a;
     }
     exec sql close c2;
    } while (newrowadded);
  exec sql commit work;
```

```
/*Print results from query table*/

  printf("Answer is\n");
  exec sql open c3;
  exec sql fetch c3 into :a;
  while (sqlca.sqlcode == 0) {
    printf("%d\n",a);
    exec sql fetch c3 into :a;
  }
  exec sql close c3;
  exec sql commit work;

  exec sql drop table query;
  exec sql commit work;
}/*end of main*/
```

Dynamic SQL - Execute Immediate

```
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;
  char sql_stmt[256];
  varchar userid[10], password[15];
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca;

void main() {
 strcpy(username.arr,"book");
 username.len = strlen(username.arr);
 strcpy(password.arr,"book");
 password.len = strlen(password.arr);
 EXEC SQL CONNECT :username IDENTIFIED BY :password;
 strcpy(sql_stmt,"update employees set hdate=sysdate where eno = 1001");
 EXEC SQL SET TRANSACTION READ WRITE;
 EXEC SQL EXECUTE IMMEDIATE :sql_stmt;
 EXEC SQL COMMIT RELEASE;
 exit(0);
}
```

Prepare, execute using

```
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;
  char sql_stmt[256];
  int num;
  varchar userid[10], password[15];
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca;

void main() {
  strcpy(username.arr,"book");
  username.len = strlen(username.arr);
  strcpy(password.arr,"book");
  password.len = strlen(password.arr);
  EXEC SQL CONNECT :username IDENTIFIED BY :password;
  strcpy(sql_stmt,"update employees set hdate=sysdate where eno = :n");
  EXEC SQL SET TRANSACTION READ WRITE;
  EXEC SQL PREPARE s FROM :sql_stmt;
```

```
  do {
    printf("Enter eno to update (0 to stop):>");
    scanf("%d",&num);
    if (num > 0) {
      EXEC SQL EXECUTE s USING :num;
      EXEC SQL COMMIT;
    }
  while (num > 0);

  EXEC SQL COMMIT RELEASE;
  exit(0);
}
```

Dynamic select- `sqlda` structure

```
struct sqlda {
  long       N; /* Maximum number of columns handled by this sqlda   */
  char     **V; /* Pointer to array of pointers to column values     */
  long      *L; /* Pointer to array of lengths of column values      */
  short     *T; /* Pointer to array of data types of columns         */
  short    **I; /* Pointer to array of pointers to indicator values  */
  long       F; /* Actual Number of columns found by DESCRIBE        */
  char     **S; /* Pointer to array of pointers to column names      */
  short     *M; /* Pointer to array of max lengths of column names   */
  short     *C; /* Pointer to array of actual lengths of column names*/
  char     **X; /* Pointer to array of pointers to indicator variable names */
  short     *Y; /* Pointer to array of max lengths of indicator variable names  */
  short     *Z; /* Pointer to array of actual lengths of indicator variable names */
};
```

Dynamic select - Example

```
#include <stdio.h>
#include <string.h>

#define MAX_ITEMS         40 /* max number of columns*/
#define MAX_VNAME_LEN     30 /* max length for column names*/
#define MAX_INAME_LEN     30 /* max length of indicator names*/

EXEC SQL BEGIN DECLARE SECTION;
  varchar username[20];
  varchar password[20];
  char    stmt[256];
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca;
EXEC SQL INCLUDE sqlda;
SQLDA *da;
extern SQLDA *sqlald();
extern void sqlnul();
int process_select_list();
```

```
main() {
  int i;
  strcpy(username.arr,"book"); username.len = strlen(username.arr);
  strcpy(password.arr,"book"); password.len = strlen(password.arr);
  EXEC SQL CONNECT :username IDENTIFIED BY :password;
  /* Allocate memory for the SQLDA and pointers to indicators and data. */
  da = sqlald (MAX_ITEMS, MAX_VNAME_LEN, MAX_INAME_LEN);
  for (i = 0; i < MAX_ITEMS; i++) {
      da->I[i] = (short *) malloc(sizeof(short));
      da->V[i] = (char *) malloc(1);
  }
  strcpy(stmt,"select eno,ename,hdate from employees where eno>=1");
  EXEC SQL PREPARE S FROM :stmt;
  process_select();
  /* Free space */
  for (i = 0; i < MAX_ITEMS; i++) {
      if (da->V[i] != (char *) 0) free(da->V[i]);
      free(da->I[i]);
  }
  sqlclu(da); EXEC SQL COMMIT WORK RELEASE; exit(0);
}
```

```
void process_select(void) {
  int i, null_ok, precision, scale;
  EXEC SQL DECLARE C CURSOR FOR S;
  EXEC SQL OPEN C USING DESCRIPTOR da;
  /* The DESCRIBE function returns their names, datatypes,
     lengths (including precision and scale), and NULL/NOT NULL statuses. */
  EXEC SQL DESCRIBE SELECT LIST FOR S INTO da;
  /* Set the maximum number of array elements in the descriptor to number found. */
  da->N = da->F;
  /* Allocate storage for each column.  */
  for (i = 0; i < da->F; i++) {
    /* Turn off high-order bit of datatype */
    sqlnul (&(da->T[i]), &(da->T[i]), &null_ok);
    switch (da->T[i]) {
      case  1 : break; /* Char data type */
      case  2 : /* Number data type */ sqlprc (&(da->L[i]), &precision, &scale);
        if (precision == 0) precision = 40;
        if (scale > 0) da->L[i] = sizeof(float); else da->L[i] = sizeof(int);
        break;
      case 12 : /* DATE datatype */ da->L[i] = 9; break;
    }
```

```
  /* Allocate space for column values.  sqlald() reserves a pointer location for
     V[i] but does not allocate the full space for the pointer.  */
   if (da->T[i] != 2) da->V[i] = (char *) realloc(da->V[i],da->L[i] + 1);
   else da->V[i] = (char *) realloc(da->V[i],da->L[i]);

  /* Print column headings, right-justifying number column headings. */
  if (da->T[i] == 2)
     if (scale > 0) printf ("%.*s ", da->L[i]+3, da->S[i]);
     else printf ("%.*s ", da->L[i], da->S[i]);
  else printf ("%-.*s ", da->L[i], da->S[i]);
  /* Coerce ALL datatypes except NUMBER to character. */
  if (da->T[i] != 2) da->T[i] = 1;

  /* Coerce datatypes of NUMBERs to float or int depending on the scale. */
  if (da->T[i] == 2)
   if (scale > 0) da->T[i] = 4;  /* float */
    else da->T[i] = 3;  /* int */
}
printf ("\n\n");
```

```
  /* FETCH each row selected and print the column values. */
  EXEC SQL WHENEVER NOT FOUND GOTO end_select_loop;
  for (;;) {
    EXEC SQL FETCH C USING DESCRIPTOR da;
    for (i = 0; i < da->F; i++) {
      if (*da->I[i] < 0)
          if (da->T[i] == 4) printf ("%-*c ",(int)da->L[i]+3, ' ');
          else printf ("%-*c ",(int)da->L[i], ' ');
      else
          if (da->T[i] == 3)     /* int datatype */
            printf ("%*d ", (int)da->L[i], *(int *)da->V[i]);
          else if (da->T[i] == 4)     /* float datatype */
            printf ("%*.2f ", (int)da->L[i], *(float *)da->V[i]);
          else                        /* character string */
            printf ("%-*.*s ", (int)da->L[i], (int)da->L[i], da->V[i]);
    }
    printf ("\n");
  }
  end_select_loop: EXEC SQL CLOSE C;
                   return;
}
```