TR-81-009

# Design and Implementation of

# A Relational Database Management System

This work was supported by National Science Council Grant

NSC70-0404-E001-01.



December 1981

Content

# Design and Implementation of A Relational Database Management System

Jyh-Sheng Ke, Ching-Liang Lin, Chiou-Feng Wang, Yuh Ling Hwang

Institute of Information Science, Academia Sinica, Taipei, R.O.C.

Yun Ying Ling, Cheng Chung Chu, Chiu Ying Ying

Wang Chen Cheng, Chih Chien Chen

Department of Computer Science, Tamkang University, Taiwan, R.O.C.

ABSTRACT

This report describes the idiosyncracy of the design and implementation of a general purpose relational model database management system.

## 1. Introduction

Starting from December 1980, we have been working for developing a distributed database system partially supported by National Science Council under the contract NSC70-0404-E001-01.

The project has been scheduled into two stages. The first stage was set from December 1, 1980 to November 30, 1981. The main function of this stage is to implement a relational database management system and to do theoretical research on communication protocol design. In parallel with the liberary automation project at Institute of Information Science, Academia Sinica, we have implemented a general purpose data entry system (GPDES) and a relational database management system (RDBMS) [1]. The integration of GPDES and RDBMS constructs an information management system. Fig.1 shows the system configuration. Implementation of GPDES and RDBMS was finished in May 1981. The preliminary design of RDBMS is not well-defined, and the performance of RDBMS is not as good as we expected. Therefore, we decided to redesign the RDBMS from then on. The report describes some idiosyncracies behind the design and implementation of RDBMS. For convinence, this report should be read in conjunction with the accompanied paper in [1].

## 2. DBMS Design Aspects

The major role of the database management system (DBMS) is to allow the data to be shared by a variety of users and to allow the users to deal the data in abstract terms, rather than as the computer stores the data. In database term, this is called "data independency". More specifically, the objective of the DBMS is to allow the user to specify only what must be done, with little or no attention on how to do it. Among other objectives of the DBMS are, data integrity (validation and consistence), concurrency control, security control,

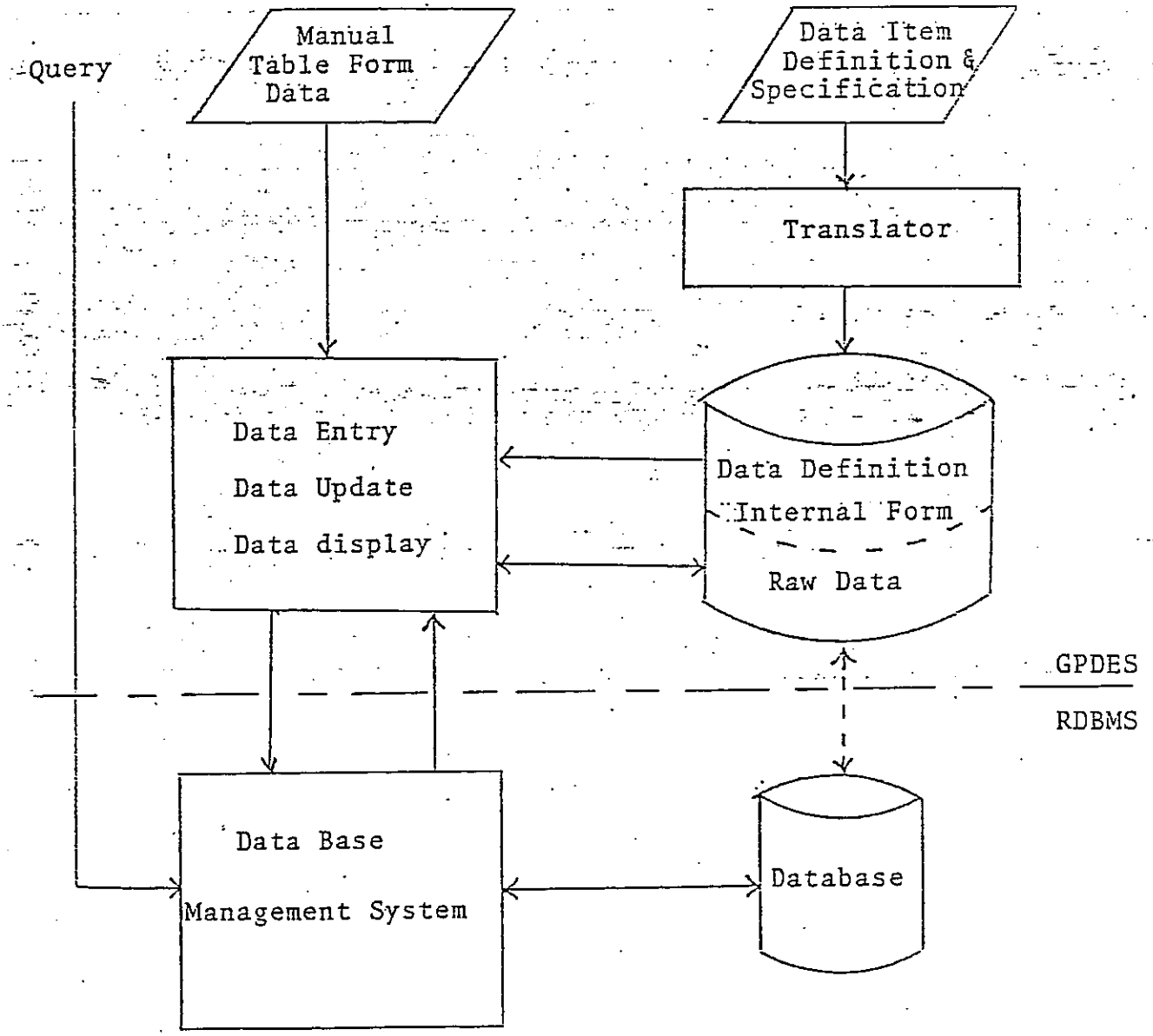



Figure 1. Information Management System Configuration

and recovery.

The first consideration of database design is the representation of data. A standard viewpoint regarding abstraction levels of databases is shown in Fig.2.

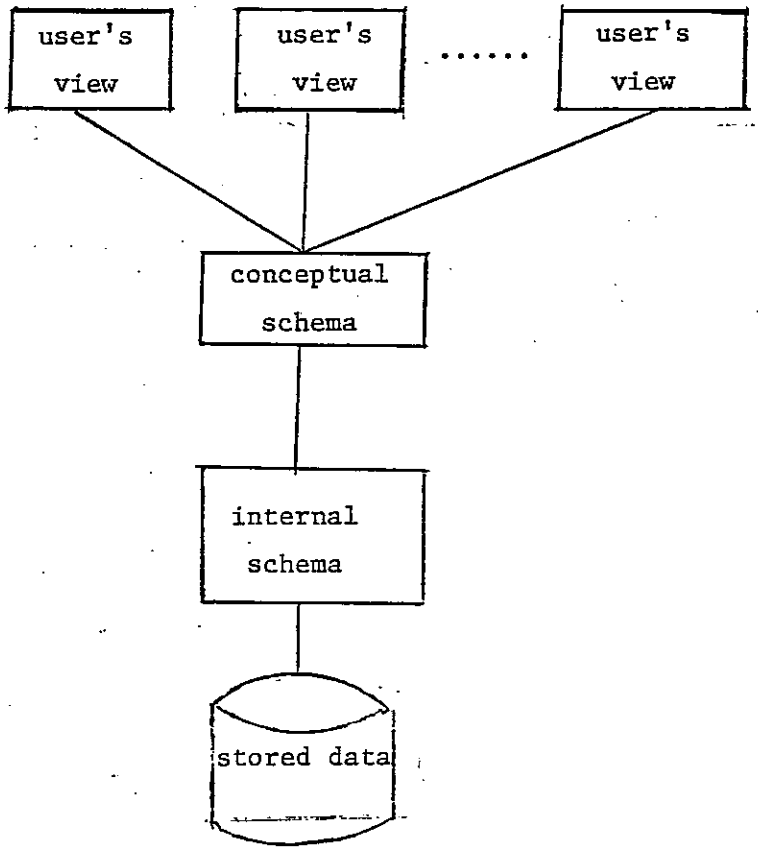



Fig.2 Three levels of abstraction

The highest level of abstraction (user's view) sees only part of the second level of abstraction(conceptual level). The difference between these two levels is limited. Both of these two levels represent data with abstract concepts and are not physically existing. The third level, internal schema, represents the data structure in which the data is physically stored.

Representation of the three levels of abstraction is called a data model (usually only emphasize on the conceptual level). A data model represents the underlying structure of the database. The most well-known data models are, relational model, hierarchical model, and network model. The main difference among these three models lies in the representation of the relationships between the data entities. The relational model has merits of simplicity and data independency. In the relational model database, the data entities and their relationships are represented with two-dimensional tables. Fig.3 shows an example of the relational table.

| S# | SNAME | P# | PANME | D# | QTY |
|---|---|---|---|---|---|
| S1 | WONG | P1 | BOLT | D1 | 50 |
| S1 | WONG | P2 | GEAR | D2 | 70 |
| S2 | CHEN | P2 | GEAR | D1 | 60 |
| S2 | CHEN | P3 | RING | D2 | 40 |
| S3 | HONG | P1 | BOLT | D3 | 90 |
| S3 | HONG | P3 | RING | D3 | 80 |

Fig.3. Relational table

To retrieve information from a relational database is essentially to do manipulation on a set of related tables. Manipulation of relational tables can be accomplished by using set-oriented operations, namely selection, projection, join, division, union, intersection, difference, and product. In the following we will first give the formal definitions of these relational operations, and then some examples will be used to demonstrate the expressive power

of these relational operations.

We will assume that the reader of this report is already familiar with the concept of relational database, and the following notations will be used throughout the rest of this report.

R, T: relation names

r, t: tuple variables ranging respectively over R and T

c(r): condition over r

c(r,t): condition over r and t jointly

EXAMPLE RELATIONS

S

| S# | SNAME | AGE |
|---|---|---|
| S1 | WONG | 30 |
| S2 | HONG | 35 |
| S3 | CHEN | 40 |

P

| P# | PNAME |
|---|---|
| P1 | GEAR |
| P2 | RING |
| P3 | TIRE |

SP

| S# | P# |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S2 | P3 |
| S3 | P1 |
| S3 | P2 |
| S3 | P3 |

P1

| P# | PNAME |
|---|---|
| P1 | GEAR |
| P3 | TIRE |
| P4 | BOLT |
| P5 | BELT |

DEFINITION: The RESTRICTION (or SELECTION) of R is denoted by R[c] and defined as:

$$R[c] = \{r \mid r \in R \wedge c(r) = 1\}$$

Example: S[AGE > 30]

| S# | SNAME | AGE |
|---|---|---|
| S2 | HONG | 35 |
| S3 | CHEN | 40 |

DEFINITION: The PROJECTION of R is denoted by R[L], L is a subset list of attributes in R, and defined as :

$R[L] = \{r[L] \mid r \in R\}$

Example: S[S#, AGE]

| S# | AGE |
|---|---|
| S1 | 30 |
| S2 | 35 |
| S3 | 40 |

DEFINITION: The JOIN of R and T is denoted by R[A*B]T, A and B are subsets of attributes in R and T respectively, and defined as :

$R[A*B] = \{rt' \mid r \in R, t' \in T', r[A] = t[B],$ where T' is the projection of T which excludes B from T$\}$

Example: SP[P# * P#]P

| S# | P# | PNAME |
|---|---|---|
| S1 | P1 | GEAR |
| S1 | P2 | RING |
| S2 | P3 | TIRE |
| S3 | P1 | GEAR |
| S3 | P2 | RING |
| S3 | P3 | TIRE |

DEFINITION: The DIVISION of R on attributes A by T on attributes B is denoted by R[A/B]T and defined as:

$R[A/B]T = \{r[\overline{A}] \mid r \in R \wedge \forall t \in T, \exists q \in R \text{ s.t. } r[\overline{A}] = q[\overline{A}] \wedge q[A] = t[B])\}$

where $\overline{A}$ is the attribute list of R complementary to A.

Example: SP[P# / P#]P

| S# |
|---|
| S3 |

DEFINITION: The PRODUCT of R and T is denoted by $R \times T$, and defined as:

$$R \times T = \{rt \mid r \in R \wedge t \in T\}$$

Example: $S \times P$

| S# | SNAME | AGE | P# | PNAME |
|---|---|---|---|---|
| S1 | WONG | 30 | P1 | GEAR |
| S1 | WONG | 30 | P2 | RING |
| S1 | WONG | 30 | P3 | TIRE |
| S2 | HONG | 35 | P1 | GEAR |
| S2 | HONG | 35 | P2 | RING |
| S2 | HONG | 35 | P3 | TIRE |
| S3 | CHEN | 40 | P1 | GEAR |
| S3 | CHEN | 40 | P2 | RING |
| S3 | CHEN | 40 | P3 | TIRE |

DEFINITION: The INTERSECTION of R and T is denoted by $R \cap T$ and defined as:

$$R \cap T = \{r \mid r \in R \wedge r \in T\}$$

Example: $P \cap P1$

| P# | PNAME |
|---|---|
| P1 | GEAR |
| P3 | TIRE |

DEFINITION: The DIFFERENCE of R and T is denoted by $R - T$ and defined as:

$$R - T = \{r \mid r \in R \wedge r \notin T\}$$

Example: $P - P1$

| P# | PNAME |
|---|---|
| P2 | RING |

DEFINITION: The UNION of R and T is denoted by $R \cup T$ and defined as:

$$R \cup T = \{r \mid r \in R \vee r \in T\}$$

Example: P ∪ P1

| P# | PNAME |
|---|---|
| P1 | GEAR |
| P2 | RING |
| P3 | TIRE |
| P4 | BOLT |
| P5 | BELT |

In the relational database, there is a set of base relations which represents the raw information of a problem domain in the real world. The combination of above relational operations possesses a powerful capability for deriving relations from the underlying base relations. The following examples demonstrate the expressive power of the above defined relational operations.

Let S, P, D, E, and SPD be the set of base relations in the relational database and defined as:

S(S#,SNAME,AGE)
P(P#,PNAME,COLOR)
D(D#,MGR)
E(E#,D#,SALARY)
SPD(S#,P#,D#,QTY)

Example: Find the supplier who supplies P2.

```
R1 = SPD[P# = 'P2']
R2 = R1[S#]
PRINT R2              /* to print the answer */
```

Example: Find the supplier who supplies red parts.

```
R1 = P[COLOR = 'red']
R2 = SPD[P# * P#]R1
R3 = R2[S#]
PRINT R3              /* to print the answer */
```

Example : Find the supplier who supplies all parts.

```
R1 = SPD[S#,P#]
R2 = R2[P# / P#]P
PRINT R2
```

Example: Find the employee who has salary greater than his manager.

```
R1 = D[MGR * E#]E
R2 = R1[D#,MGR,SALARY]
R3 = E[D# * D#]R2      /* should rename SALARY of R2 into Salary1 */
R4 = R3[SALARY > SALARY1]
R5 = R4[E#]
PRINT R5
```

Example: Find the manager who purchases parts from supplier WONG.

```
R1 = SPD[D# * D#]D
R2 = S[SNAME = 'WONG']
R3 = R1[S# * S3]R2
R4 = R3[MGR]
PRINT R4
```

3. Implementation of RDBMS

The implementation of RDBMS has been divided into two parts each of which is handled by a three-member team. The first part includes the definition of Data Definition Language (DDL) and Data Manipulation Language (DML). Compilers of both languages have also been implemented by the same people. The Data Definition Language can be used to define the database schemas which represent the information about data entities and the associations among data entities. The Data Manipulation Language can be used to specify relational operations over the stored data. The second part includes the definition of the physical storage structures and the algorithms for implementing relational operations. Concurrency control and indexing techniques have also been attacked carefully. The syntaxes of DDL and DML and the algorithms of relational

operations are shown in the appendix of this report. Fig.4 shows the architecture of the RDBMS.

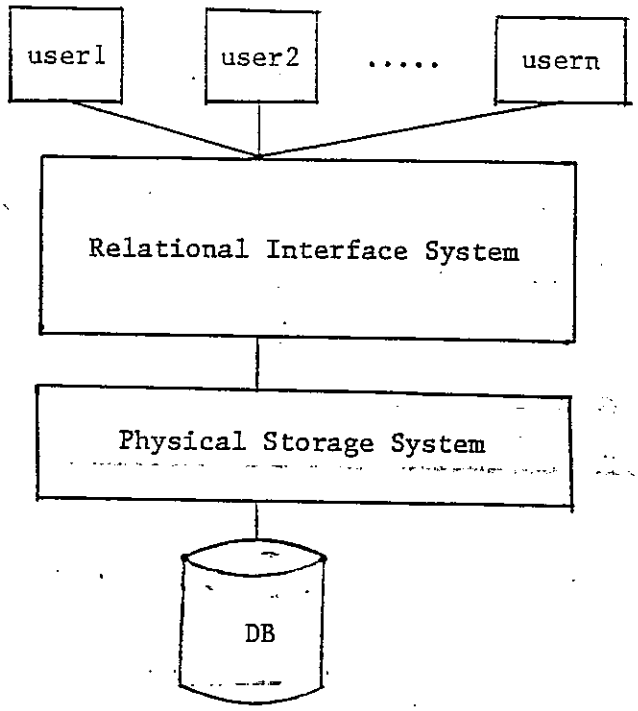



Fig.4. Architecture of RDBMS

The RDBMS includes two subsystems, namely Relational Interface System (RIS) and Physical storage System (PSS). The RIS Provides DDL, DML, authorization, integrity check, and support for alternative views of stored data. The PSS maintains indexes on selected attributes, manages disk space allocation, file access, storage buffers, locking, and transaction consistence.

3.1 Relational Interface System

The Relational Interface System provides high level, data independent facilities for data definition, manipulation, and retrieval. The data definition facilities of the RIS allow the data base administrator to define the data

at conceptual level as well as internal level. They also allow a variety of external views to be defined upon common underlying data to provide different users with different views. The external views of the common data also facilitate some authorization checks and privacy control. The data manipulation facilities of the RIS allow data to be inserted, deleted, modified, manipulated, and retrieved by using a set of high level relational operators which exempt the users from knowing the internal storage structure of the stored data. Detail of the DDL and DML facilities are illustrated in the appendix. Our next step is to implement a high level query language on the top of the existing DML facility. The query statements will be translated into a sequence of relational operations. Optimization of the relational operations program will also be included in this further research.

3.2 Physical Storage System

The Physical Storage System manages disk storage allocation, indexed file access. B-tree structure has been used for indexing secondary keys of each relation. The disk space contains a set of files each of which is constructed by a set of fixed-size pages. The data structure of each page is about the same as other database systems and is shown in Fig.5.

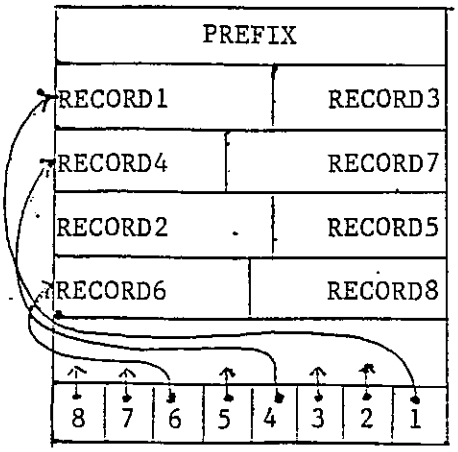



Fig.5. Page structure of PSS

For the purpose of saving space, we employ the concept of logical page number (what the relational operator sees) and physical page number (what the data is really stored). A page map table has been maintained to map logical page number into physical page number. Page identification is managed by hashing or clustering. The PSS also supports concurrency control by using locking techniques. Detail of the PSS organization will be published in further report.

## Conclusion

In this report, we have described the design philosophy and implementation idiosyncracy of a relational database management system. The implementation of the RDBMS has not been completed yet. However, it is in good progress. The syntax of DDL and DML we have presented in this report is by no means complete or well-defined. We are continually changing this subject whenever we find the necessary.

## References


1. C. L. Lin and J. S. Ke etcl., "Design of a Universal Data Entry System", TR-81-005, Institute of information Science, Academia Sinica, Aug. 1981.
2. S. Atre, "DATA BASE — Structured Techniques for Design, Performance, and Management," Wiley Series, 1980.
3. C. J. Date, "An Introduction to Database Systems," 3rd ed., Addison-Wesley, 1981.
4. J. D. Ullman, "Principles of Database Systems," Computer Science Press, 1980.

# APPENDIX I. DDL AND DML

## DATA DEFINITION LANGUAGE (DDL)

Syntax:

```
SCHEMA IS name
/*      domain definition          */
DOMAIN IS name
$PIC(TRUE) IS F(9) [S(9), I(9), V(9),...,]
        LEFT(RIGHT) JUSTIFY,
$TYPE IS COMPUTATIONAL [DISPLAY, BIT, INDEX]
        FROM int1 TO int2 STEP BY int3
         [    (brown, red, ....)        ]
        IN units
$END
.  .

/*      relation definition        */
RELATION IS name
$USER IS <name, right> .......
$WITHIN directory fileno FILES
        ------(assign filename in accordins with
               the directory and the fileno)
$PAGE SIZE blockno  (initial pasesize 8 blocks)
$FIELD role_name DOMAIN name
 [ IS | ACTUAL | RESULT OF derived program
      | VIRTUAL |
 ]

 [ ON | GET      |
      | STORE    | CALL check-program
      | MODIFY   |
      | DELETE   |
 ]
$FIELD,....
$RULE PART name1,name2,...
$ACCESS USING VSAM, DAM ,...          (default DAM)
$INDEX USING name1,name2,...
$ON  | DELETE |
     | MODIFY | CALL semantic program
     | GET    |
     | STORE  |
$ON |OPEN| CALL concurrent program
    |CLOSE|
$ON PRIVACY CALL check_program
$END
```

Remarks:

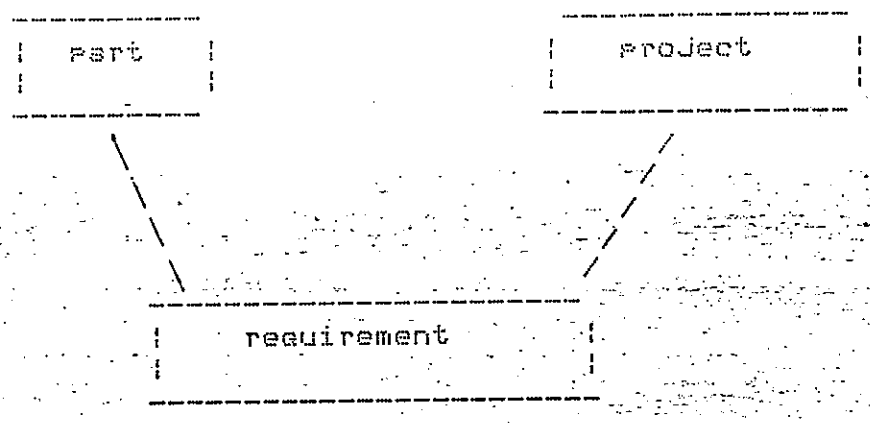
Name for meta-relation, i.e. relation created by relation operation, is default to be $user name[A-Z,a-z,0-9][A-Z].

Meta-relation name is run-time determined.

Basing on the syntax above, we give an example to design a conceptual relation.

Example 1:

We have three relations, namely, project,part,and requirement. The semantics is that project requires a specific quantity of parts to carry on.




Each relation has attributes as shown below:

part(part_no,part_descr,qty_on_hand,recorder,unit_cost)

project(project_no,project_descr,manager,compl_date)

requ(project_no,part_no,qty_requ)

also,

each attribute has a set of permissible values, i.e. its DOMAIN.

The formats and qualification of these attributes are:

```
part_no      integer
part_descr   string character has length less than 20
qty_on_hand  integer >0 and < 200 units ks
recorder     string
unit_cost    float
  :            :
  :            :
    e.t.c.
```

Using the proposed schema we can describe this problem domain as follows:

```
SCHEMA IS sale
;ON OPEN CALL "....."
;ON PRIVACY CALL ....GRANT delete e.t.c

DOMAIN IS part_no
;PICTURE IS I(6)
;TYPE IS display
DOMAIN IS part_descr
;PICTURE IS S(20)
;TYPE IS display

DOMAIN IS qty_on_hand
;PICTURE IS I(4)
;TYPE IS display
        FROM 0 TO 200 STEP BY 1
;
;
;


RELATION IS part
;WITHIN p1,p2(FROM 1 TO 1000,page size=256)
;ATTRIBUTE part_no DOMAIN part_no
;ATTRIBUTE part_descr DOMAIN part_descr
;ATTRIBUTE qty_on_hand DOMAIN qty_on_hand
;ATTRIBUTE unit_cost DOMAIN unit_cost
;RULE PART part_no .
;ACCESS USING VSAM
;INDEX USING qty_on_hand
;ON DELETE CALL "DELETE recrt with the same key value"
;ON MODIFY CALL "MODIFY recrt with the same key value"
;ON OPEN CALL "permission and concurrency control"
;ON CLOSE CALL ....


RELATION IS project
;WITHIN pr1,pr2(FROM 1 TO 1000,page size is 2048)
;ATTRIBUTE .....
; ....
;
;
;
```

BNF representation of DDL:

BNF of Domain and Relation Conceptual Schema:

```
1. < schema context > ::= < comment context >
                        | < domain context >
                        | < relation context >

2. < comment context > ::= < comment >
                         | < comment > < comment context >

3. < comment > ::= /* < string > */

4. < domain context > ::= < domain >
                        | < domain > < domain context >

5. < domain > ::= DOMAIN IS < domain-name > < domain block > $END
                | DOMAIN IS < domain-name > $END

6. < domain block > ::= $PICTURE IS < pic-exp >
                      | $TYPE IS < type-exp >

7. < pic-exp > ::= < pic-type >
                 | < pic-type > < just-exp >

8. < pic-type > ::= I( < length > )
                  | F( < length > : < length > )
                  | S( < length > )
                  | V( < length > )

9. < just-exp > ::= LEFT JUSTIFY
                  | RIGHT JUSTIFY

10. < type-exp > ::= < type >
                   | < type > < range-exp >

11. < type > ::= COMPUTATIONAL
               | DISPLAY

12. < range-exp > ::= < range >
                    | IN < units >
                    | < range > IN < units >

13. < range > ::= < numeric-range >
                | < enumerative-range >

14. < numeric-range > ::= < numeric-range-exp >
                        | < numeric-range-exp > < numeric-range >

15. < numeric-range-exp > ::= FROM < value > TO < value > STEP BY < value >
                            | FROM < value > TO < value >

16. < enumerative-range > ::= ( < enumerative-list > )
```

```
17. < enumerative-list > := < string >
                          | < string > , < enumerative-list >

18. < domain-name > := < id >

19. < units > := < id >

20. < lentth > := < unsigned-integer >

21. < value > := < integer >
               | < float >

22. < integer > := < unsigned-integer >
                 | ( + | - ) < unsigned-integer >

23. < float > := < integer > . < unsigned-integer >
               | < integer > .
               | . < unsigned-integer >

24. < relation context > := < rel_exp >
                          | < rel_exp > < relation context >

25. < rel_exp >:=< relation > < pas_exp > < field_exp > < rel-block> ; END
                 | < relation > < field_exp > < rel-block > ; END
                 | < relation > < field_exp > ; END

26. < relation > := RELATION IS < rel-name > ; WITHIN < file_exp >

27. < rel-name > := < id >

28. < file_exp > := < file-name >
                  | < file-name > , < file_exp >

29. < page_exp > := ; PAGESIZE < unsigned_integer >

30. < rel-block > := < relation set .
                   | < relation set > < rel-block >

31. < relation set > := ; USER IS < user_exp >
                      | ; RULEPART < key_exp >
                      |   < index_exp >
                      | ; ACCESS USING < mode >
                      | ; ON < check_exp >

32. < field_exp > := < field block >
                   | < field block > < field_exp >

33. < field block > := < attribute_exp >
                     | < attribute_exp > < field block >

34. < attribute_exp > := < attribute >
                       | < attribute > < attribute block >

35. < attribute > := ; ATTRIBUTE < rels_name > DOMAIN < domain_name >

36. < attribute block > := ; IS < se_exp > RESULT OF < dest rel_name_exp
```

```
                    | ; ON < op_exp > CALL < check_psm_name >

37. < ac_exp > ::= ACTURAL
                 | VIRTUAL

38. < op_exp > ::= GET
                 | STORE
                 | MODIFY
                 | DELETE

39. < derived_psm_name > ::= < id >

40. < check_psm_name > ::= < id >

41. < role_name > ::= < id >

42. < domain_name > ::= < id >

43. < user_exp > ::= < user block >
                   | < user block > < user_exp >

44. < user block > ::= < < user_name > , < right > >

45. < user_name > ::= < id >

46. < right > ::= R
                | W
                | RW
                | WR

47. < key_exp > ::= < key_name >
                  | < key_name > , < key_exp >

48. < key_name > ::= < id >

49. < index_exp > ::= < index block >
                    | < index block > < index_exp >

50. < index block > ::= ; INDEX USING < index_name >

51. < index_name > ::= < attribute_name >
                     | < attribute_name > , < index_name >

52. < attribute_name > ::= < id >

53. < mode > ::= DAM

54. < check_exp > ::= < semantic check >
                    | < concurrent check >
                    | < privacy check >

55. < semantic check > ::= < op_exp > CALL < check_psm_name >

56. < concurrent check > ::= < condition_exp > CALL < check_psm_name >

57  < privacy check > ::= PRIVACY CALL < check_psm_name >
```

58. < condition_exp > := OPEN
                        | CLOSE

59. < unsigned_integer > := < digit >
                          | < digit > < unsigned_integer >

60. < digit > := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

61. < id > := < letter >
            | < letter > < alph_num >

62. < alph_num > := < digit >
                  | < letter >

63. < letter > := A | B | C | D | E | F | G | H | I | J | K | L | M |
                  | O | P | Q | R | S | T | U | V | W | X | Y | Z |
                s | b | c | d | e | f | s | h | i | J | k | l | m |
                  | o | p | q | r | s | t | u | v | w | x | y | z |

The information of the relational schema will be kept in four dictionary files:

(1) DIRECTORY file :

keep all domain names or relation names information by using balance tree data structure .

(2) DOMAIN file:

keep all information of each domain

(3) RELATION file:

keep all information of each relation

(4) PROTECTION file:

keep information of relations which can be accessed by each user ID

## FILE STRUCTURE OF SCHEMA:

(1) DIRECTORY file: fix length ( balance tree )

content=>

```
{
int freeptr;/*point to the head of free quene*/
int avail;/*point to the head of avail quene*/
int rootptr;/*point to the root of balance tree*/
struct direct[];
}
```

(2) DOMAIN file: variable length (40 + varaible)

each record content=>

```
{
struct dom_hd;
struct rans_str;
struct rans_num;
}
```

(3) RELATION file: variable length (130 + variable)

each record content=>

```
{
struct rel_hd;
struct filename[];
struct attribute[];
struct keyname[];
struct ndx[];
}
```

(4) PROTECTION file:

content=>

(a) 6 blocks for ID directory

```
{
        char idname[15];
        int  link;
        char unused[15];
}
```

(b) data area

```
{
        struct protect;
}
```

```
struct  direct
        {
        char name[15] ; /*domain or relation name */
        char type     ; /* DOMAIN-> D , RELATION->R */
        int  offset   ; /* offset in domain or relation file */
        int  totlen   ; /* message record length */
        int  rlink    ; /* balance tree right link */
        int  llink    ; /* balance tree left link */
```

```
        int  status    ; /* condition used ( in relation only ) */
        int  tag       ; /* balance factor */
        char unused[4];
        }


struct    dom_hd
        {
        char pic_typ    ; /* for pic 4 bits ,for type 4 bits */
        int  len1      ; /* pic len */
        int  len2      ; /* second len for pic F() only */
        char rang_typ   ; /* 0:no range, 1:str type, 2:num type */
        int  rang_len   ; /* total length for range area */
        char unit[2]    ;
        char dom_unused[30]     ;
        }


struct  rel_hd
        {
        int     fileno  ;/*total fileno used by relation*/
        int     passiz  ;/*page size for physical allocate*/
        int     pagno   ;/*max pagno in one file*/
        int     attno   ;/*field no in this relation*/
        int     keyno   ;/*number of fields in key group*/
        char    accmod  ;/*access mode*/
        int     indxno  ;/*No. of index tables established*/
        int     indxlen ;/*total length of index message*/
        char    sname[15];/*taskname for set call*/
        char    sname[15];/*taskname for store call*/
        char    mname[15];/*taskname for modify call*/
        char    dname[15];/*taskname for delete call*/
        char    oname[15];/*taskname for open call*/
        char    cname[15];/*taskname for close call*/
        char    reltype ;/*type is variable or fix length*/
        char    rel_unused[24];
        }


struct  num
        {
        int rfrom;      /*initial range start value*/
        int rto;        /*initial range ending value*/
        int rster;
        char rend;      /*use '0' close range assign*/
        }


struct  str
        {
        char value[];   /*string value buffer area*/
        char strend;    /*use '0' close*/
```

```
        }

struct  rans_num
        {
        int tolen;      /*total lensth of numeric value buffer area*/
        struct  num[];  /*numeric value buffer structure*/
        char end;       /*use '0' close*/
        }

struct  ndx
        {
        int     len     ;       /*lensth of index area */
        int     attr_no ;
                 /*No. of attribute which used in this index area*/
        struct  attr_name[];
                /* name of attribute which used in this index area*/
        char    filename[15];
                /* indexfilename which used by this index area*/
        }

struct  attr_name
        {
        char    name[15];
        }

struct  protect
        {
        struct  rprotect[];
                /*message buffer include relation_name ; right*/
        }

struct  rprotect
        {
        char    rname[15];      /*relation_name which be protected*/
        char    right   ;       /*protect status*/
        }

struct  attribut
        {
        char    field_name[15];
        char    dom_name[15];
        char    avtype  ;/* A for actual; V for virtual*/
        char    avname[15];/*taskname for acturel or virtual call*/
        char    gname[15];/*taskname for get call*/
        char    sname[15];/*taskname for store call*/
        char    mname[15];/*taskname for modify call*/
        char    dname[15];/*taskname for delete call*/
        }
```

```
struct   filename
         {
         char     name[15];
         }

struct   keyname
         {
         char     name[15];
         }
```

Storage operations:

(1)DELETE
(2)MODIFY
(3)GET
(4)STORE

/*  detail syntax  */

(1)DELETE
  FROM  relname
    WHERE <where clause> :condition to be met.

  semantics:
    delete tuple or tuples from relname which meet
    the where-clause condition.

(2)MODIFY
  FROM  relname
  SET(WITH)  (new value list with assignment syntax)
  WHERE  ......

  semantics:
    modify those tuples in relname which meet the
    where-clause condition by replacing the old
    values with the value specified in SET-clause.

(3)STORE
  FROM  (data list with comma separated)
    $PIPE  load via pipe file
    $FILE filename ...load via data file
  INTO relname

  semantics:
    store(load) tuple(s) to relation relname from list
    directly or, pipe file or unix file massively.

nformation retrieval:

asically, the syntax and operation is derived from CODD's
relational alegrbra.
yntax of this language is composed of six parts:

1)Command name

2)Join clause

3)From clause

4)Into clause

5)Project clause

6)Where clause

xample:

```
Join     <Dno,D#>
FROM     DEP,EMP
INTO     Person  <Dno>
WHERE    salary < 1000
```

detail syntax */

)Retrieve
FROM rel1
INTO rel2 [ < attr. list >]
[ WHERE
]

semantics:
retrieve tuples from rel1, and store into rel2.

constraints:
name in attr. list must be in rel1, but the ordering
is not preserved.

2)Union
FROM rel1,rel2,....,reln
INTO relnm [ < attr. list >]
[ WHERE
]

semantics:
relname=<[t] | [t] is in rel1 or rel2....and <other
condition> >

constraints:
all domains in rel1,rel2, .. must be One-one corresponding.
the role name of relname is default to be the same as rel1<
first relation >.
name in attr. list must be in rel1.

(3)Intersect
FROM rel1,rel2,...,reln
INTO relnm [ < attr. list >]
[ WHERE

]

semantics:
relnm= < [t] | [t] is in rel1 and rel2 ...and <other codition>>

constraints:
all domains in rel1,rel2,...must be one-to-one corresponding.

the role name of relnm is default to be the same as rel1.
attr. list must be in rel1, but ordering is not preserved.

(4)Difference
FROM rel1,rel2
INTO relnm [ < attr. list >]
[ WHERE

]

semantics:
relnm= < [t] |[t] is in rel1,but not in rel2 ,and <other
condition > >

constraints:
all domains in rel1 rel2 must be one-to-one corresponding.

the role name of relnm is default to be the same as rel1.
attr. list must be in rel1.

5)Division  < attr. list ; attr. list >
FROM  rel1,rel2
INTO  relnm  [ < attr. list > ].
[ WHERE

]

semantics:
relnm= {[t] |[t,attr. list] is in rel1, for all [attr. list]
in rel2 and <other condition> }

constraints:
attr. list ; attr. list is in rel1 and rel2 separately,
domain of them must be the same.
role name in relnm is a subset set of rel1 - attr. list.
attr. list must in relnm.

6)Join  [ <attr. list ; attr. list > ]
FROM  rel1,rel2
INTO  relnm  [ < attr. list > ]
[ WHERE

]

semantics:
.if <attr. ...>,..> is not specified, operation is just a
cartesian product.
.if <attr... ; ...  >is specified, this set of attributes is a
Joint attribute set, and is eliminated to one attribute after
operation.
.this Join operation includes =, >, or other types of Join
operation.
.relnm={[t1,t2] | [t1] is in rel1 ,and [t2] is in rel2,if [t1],
[t2] do not contain Joint field,otherwise,Joint field is pro-
Jected into one }
constraints:
attr. list is a Joint-attr. list.
name conflict is resolved by addins a prefix- $ to the
attribute name in the second relation.
role name of relnm is a closure of rel1 and rel2.

```
/
RE CLAUSE:
/
RE-CLAUSE contains a predicate calculus to determine true or false.

ring match composed of --
)exact match-- match whole string
)exact word match-- match word by word.
      .word is defined to be a group of strins with space separated.
)partial string match-- match string by string

mber match composed of --
)exact match
)range match-- discrete or continuous range

*detail syntax of WHERE clause   */

ERE
)attr [<,>,==,!=,<=,>=] attr
)attr [  "    "      ] [int,float,char,string]
)[attr,...(AND,OR, ,,;,) ] (<,>,==,!=,<=,>=) [int,float,char,string]
)attr [IN,OUT] { 1,2,3, ..discrete list } or
                {1,,2,,.. --continuous list for [int,float,char,string]}
      attr ==1 or attr ==2---or relation .

      Let P be a comparison operator, and X a variable or constant,
  S is a set of int , float , char or string rePresented bY
  discrete or Partition, we define X P S as:

  P==[IN,OUT]

  1,P==IN
      there eXists a Y in S,such that y==x is true
          2,P==OUT
              for all Y in S,such that Y!=X is true

      (5)[attr,(AND,OR, , ,;)attr,..] [IN,OUT] {1,2,..discrete list} or
                                                  1,,2--continuous range
      (6)attr == '$word' or '?string'----partial match
      (7)attr IN {'$word' or '?string'  }
```

representation of DML:

1. < where clause > := WHERE < condition >

2. < condition > := < logical and-exp >
| < logical and-exp > < or-operator > < condition >

3. < logical and-exp > := < relational exp >
| < relational exp > < and-operator > < logical and-exp >

4. < relational exp > := < attribute > < relational-operator > <operand>
| < attribute > <set-operator > < discrete-operand >
| < set-attribute > < relational-operator > < operand >
| ( < condition > )

5. < and-operator > := AND | ,

6. < or-operator > := OR | ;

7. < relational-operator > := == | != | > | >= | < | <=

8. < set-operator > := IN | OUT

9. < operand > := < attribute >
| < constant >

10. < constant > := < integer >
| < float >
| < char >
| < string >

11. < set-attribute > := [ < attr-list > ]

12. < attr-list > := < attribute >
| < attribute > < set-attri-operator > < attr-list >

13. < set-attri-operator > := < and-operator >
| < or-operator >

14. < discrete-operand > := { < discrete-list > }

15. < discrete-list > := < discrete-constant >
| < discrete-constant > , < discrete-list >

16. < discrete-constant > := < constant >
| < subrange >

17. < integer > := < unsigned integer >
| < sign > < unsigned integer >

18. < unsigned integer > := < digit >
| < digit > < unsigned integer >

19. < float > := < integer > . < unsigned integer >
| < integer > .

```
                | < sign > . < unsigned integer >
                | . < unsigned integer >

. < sign > ::= + | -

. < char > ::= ' < letter > '

. < string > ::= ' < word > '
                | '$ < word > '
                | '? < word > '

. < word > ::= < letter > < letter >
             | < letter > < word >

. < subrange > ::= < subrange value > .. < subrange value >

. < subrange value > ::= < integer >
                       | < float >
                       | < char >

. < attribute > ::= < leading letter >
                  | < leading letter > < following word >

. < leading letter > ::= < alph char > | $

. < following word > ::= < letter >
                       | < word >

. < letter > ::= < digit >
               | < alph char >
               | < special char >

. < digit > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

. < alph char > ::= A | B | C | D | E | F | G | H | I | J | K | L | M
                  | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
                  | a | b | c | d | e | f | g | h | i | j | k | l | m
                  | n | o | p | q | r | s | t | u | v | w | x | y | z

. < special char > ::= # | % | - | . | :
```

External schema:

The external schema is in general a sub_set of the conceptual schema.
We use it to define the view of each user, and it is derived from the relations in the conceputual schema.
It provides the protection of each unauthorized user, the data control and the transaction control.
It must fulfil:

(1)data type can be different from conceptual schema
(2)units can be different from conceptual schema
(3)data can be virtual

```
/*        syntax of external schema          */

EXTERNAL SCHEMA IS name
;ON ¦ OPEN ¦
;  ¦ CLOSE¦ CALL check_program
;ON PRIVACY CALL check_program
;END

DOMAIN IS name
;PICTURE IS ...
;TYPE IS .....
 <the syntax is the same as conceptual schema....>
;MAPPING To role_name of relation
;END

RELATION IS name
;ATTRIBUTE name DOMAIN name
;...
;MAPPING <relation,schema>,....
;DERIVED FROM <relational algebra>
;IS ¦ ACTUAL ¦
   ¦ VIRTUAL ¦
;ON ¦ OPEN ¦
   ¦ CLOSE¦ CALL check_program
;END
```

# APPENDIX II. PHYSICAL DATA BASE DESIGN

# Physical Data Base Design

## Physical storage structure :

To define the storage structure of the physical files.

### (A) Disk Storage Structure

A.1

```
struct Record_hd {
int  offset; /* the address of record's data info */
int  prefix; /* the length of this record's prefix */
int  recordsiz; /* the total size of this record */
                }
```

A.2

```
struct stors_map {
int  logi_pno; /* the result value of hashing */
int  phys_pno; /* the page no. of physical allocate */
int  freespace; /* indicate  page remains how much space */
                }
```

(B) Core Storage Structure

B.1

```
struct rel_head {
char  relname[15]; /* relation name */
char  right; /* the right(Read,Write,R/W) of this relation */
int   diroffset; /* relation's address in relation file */
char  Rrefcnt; /* indicates the status of relation */
int   fileno; /* total file no. used by relation */
int   pagesize; /* the size of one page ,unit is block */
char  rstat; /* relation status */
int   pageno; /* number of page in one file */
int   attrno; /* number of attribute in one relation */
int   keyno; /* the key contains no. of attribute */
char  accmode; /* access mode */
int   indxno; /* how many index table established */
int   indxlen; /* total length of index message */
int   RcallPros; /* bit indicates Call check_pros or not */
char  *RcallPros; /* pointer point to call program name */
char  *fileptr; /* pointer point to file name */
char  *keylink; /* pointer to the first attri name of key */
struct attribute *Rattptr; /* pointer to attri struct */
struct index *indxfptr; /* er point to index structure */
struct page *pageptr; /* pointer to page structure */
                    }
```

B.2

```
struct attribut {
char  attname[15]; /* attribute name */
```

```
int   klink; /* the link list of key */
int   Acallpros; /* bit indicates Call program or not */
                 /* include semantic,concurrent and check */
char  *Acallptr; /* pointer point to call program name */
struct domain *domptr; /* pointer to domain structure */
}
```

B.3

```
struct domain {
char  domname[15]; /* domain name */
char  Drefcnt; /* indicate the status of domain */
char  pictype; /* domain's picture and type */
int   len1; /* picture length */
int   len2; /* the 2nd length of pic only for floating type */
char  rangtype; /*the type of domain's range */
int   ranglen; /* the length of domain's range */
char  unit[2]; /* the unit of domain */
char  *address;/* the address of domain's var part info. */
}
```

B.4

```
struct page {
int   physpno; /* the physical page no. in core */
char  Prefcnt; /* indicate the status of page */
int   pagesize; /* the size of page physical allocate */
int   list; /* The link list of page no. in same file */
int   rel_id; /* directly pointer to rel_head */
int   petat; /* page status */
```

```
}
```

B.5

```
struct file_descrt {
char  ofilename[15]; /* the name of any one file */
char  Frefcnt; /* indicate the status of file */
int   fid;
char  ftype;
char  fmode;
                    }
```

B.6

```
struct indx_file {
char  indxfilnm[15]; /* index file name */
char  *idxptr; /*pointer to the 1st attri pointer of indx*/
                    }
```

B.7

```
struct Iattptr {
char  *Idxattptr; /* pointer to the attribute name */
                    }
```

B.8

```
struct whereattr {

char  *relptr; /* pointer to relation name in string buf */
int   Avarno; /* the no. of var length attri in one relat */
char  *domptr; /* printer point to domain structure */
                    }
```

B.9

```
struct type {
```

```
int   type; /* indicate command's type */
char  *strptr; /* pointer point to strins buffer */
struct whereattr *wherptr; /* pointer to whereattr struct */
                    }
```

```
(1)DELETE
   FROM          relname
   WHERE ....where clause ;condition to be met.

(2)MODIFY
   FROM          relname
   SET(WITH)     (new value list with assignment syntax)
   WHERE  ......

(3)STORE
   FROM  (data list with comma seperated)
         $PIPE  load via pipe file
         $FILE filename ...load via data file
   INTO relname

(4)Retrieve
   FROM          rel1
   INTO          rel2     [  < attr. list >]
   [  WHERE ...  ]

(5)Union
   FROM          rel1,rel2,....,reln
   INTO          relnm    [   < attr. list >]
   [  WHERE ...  ]

procedure:
         (a)Get tuple from R1
         (b)Store tuple to R3
         (c)if R1 not empty,so to (a)
         (d)Get tuple from R2
         (e)store tuple to R3


(6)Intersect
   FROM          rel1,rel2,...,reln
   INTO          relnm    [ <  attr. list  >]
   [ WHERE ...   ]

procedure:

         (a)Get tuple from R1
         (b)Get tuple from R2 with "condition is R1 tuple"
         (c)Store tuple to R3

(7)Difference
   FROM          rel1,rel2
   INTO          relnm    [ < attr. list  >]
   [  WHERE ...  ]


procedure:

         (a)Get tuple from R1
```

```
        (b)Get tuple from R2 with "condition is R1 tuple"
        (c)Store unmatched tuple

(8)Division     < attr. list ; attr. list >
   FROM         rel1,rel2
   INTO         relnm   [ < attr. list > ]
   [ WHERE ...  ]

Procedure:
        (a)Get


(9)Join         [ <attr. list ; attr. list > ]
   FROM         rel1,rel2
   INTO         relnm   [ < attr. list > ]
   [ WHERE ...  ]


Procedure:
        (a)Get tuple from R1
        (b)Get tuple from R2
        (c)Resolve condition
        (d)concatenate two tuples
        (e)Store new tuple to new relation
```

## Data Base Procedure

(1). Login

(a). open protection file

1. reference file_descpt table /* call open_file() */

(b). read protection file into core /* directory of ID */

(c). check user's name match or not

if match keep link of directory of ID

else can not login

/* link is a pointer point to rprotect structure */
/* rprotect keep all relations can be used */

(2). Clear and set rel_head ,attribut,domain table

a. set refcnt of domain table to -1

b. set refcnt of rel_head,page,attribut,file_descpt table

(3). Data manipulation

(a). Parsing

1. get command from terminal or file

2. syntax check

3. convert where clause into polish notation form

4. generate CMD table, type table and a string buffer

(b). Execute command

1. Initialize executing relations

/* initialize rel_head,attribute, and index table */

a. search in rel_head table /* call search() */

if search success

then

1. increase found relation's refcnt

2. decrease others in rel_head table

else

b. search in temporary relation file

if success then

b.1 seek tmplink (in temporary directory)

b.2 load temporary relation information

else

c. search in base relation file

if unsuccess then no such relation

else

c.1 seek link

c.2 read rprotect struct

c.3 check relation name match or not

c.4 if not match

then this user can not use such relation

d. load base relation information

e. check attribute in such relation or not

f. establish whereattr table

2. optimize polish notation form

3. crest new relation (temporary)

a. store temp relat and domain name into temp dir file

b. store relat, dom inf. into temp relat & dom file

4. match command name

if match then execute command task

else no such command

5. command processing

## Table Maintenance

(1). rel_head table

(a). One rel_head contains 48 bytes.

(b). It reserves 10 rel_head.
(Total occupies 10*48 = 480 bytes)

(c). This table is global.

(d). Insert

d.1 Search in rel_head table

d.2 If search success then call initialization function

else

if rel_head table no space

then delete one rel_head decision by refcnt

a. insert rel_head to rel_head table

b. call initialization function

(2). attribut table

(a). Dynamic allocates (attrno * 22) bytes

(b). This table is local.

(c). Max 2 - 3 attribut table in core.

(3). file_descpt table

(a). One record contains 18 bytes.

(b). It reserves 10 file_descpt records.
(Total occupies 18*10 = 180 bytes)

(c). This table is global.

(d). search

if success then

1. increase matched file's refcnt

2. decrease others in this table

3. return fd

else

1. delete decision by refcnt

2. open file which needed

3. keep this file's information

(4). domain table

(a). One domain record contains 28 bytes.

(b). It reserve 30 domain records.
(Total occupies 28*30 = 840 bytes)

(c). This table is global.

(d). insert

d.1 search in domain table

d.2 if search unsuccess

then check refcnt

if refcnt <= 0 then insert it

else

1. allocate another dom table

2. keep its location in dyaloc_hd

3. insert it

return required domain location in domain table

(e). Domain's variable part is dynamic allocated.

(5). page table

(a). One page record contains 7 bytes.

(b). It reserves 16 page records.
( Total occupies 7*16 =112 bytes )

(c). It reserves 16 blocks used for data page.

(d). This table is global.

(e). Its operation is one to one coresponderces.

(f). insert

f.1 check page table

if no enoush space then delete decision by refent

insert it

Initialization

(1). Initialize attribut table

(a). Search domain table and keep domain's location

(b). if attribute has G,S,M,D call

then

1. dynamic allocate space

2. insert call program name

(2). Initialize rel_head table

(a). dynamic allocate indx_file,lattptr,file name

attribut table and call program name

(b). insert information to allocated space

(c). set up key and index link

(3). Initialize type table

(a). dynamic alloc space to Joint field and where clause

(b). insert information to allocated space