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

1. Introduction ................................................................. 1
2. DBMS Design Aspects ..................................................... 1
3. Implementation of RDBMS ............................................... 9
   3.1 Relational Interface System ......................................... 10
   3.2 Physical Storage System ............................................. 11
Conclusion ............................................................................. 12
References ........................................................................... 12

APPENDIX I. DDL AND DML

APPENDIX II. PHYSICAL DATA BASE DESIGN
Design and Implementation of A Relational Database Management System

Jyh-Sheng Ke, Ching-Liang Lin, Chiu-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 library 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 idiosyncrasies 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.

![Diagram showing three levels of abstraction]

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 independence. 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:

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>P#</th>
<th>PNAME</th>
<th>D#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>WONG</td>
<td>P1</td>
<td>BOLT</td>
<td>D1</td>
<td>50</td>
</tr>
<tr>
<td>S1</td>
<td>WONG</td>
<td>P2</td>
<td>GEAR</td>
<td>D2</td>
<td>70</td>
</tr>
<tr>
<td>S2</td>
<td>CHEN</td>
<td>P2</td>
<td>GEAR</td>
<td>D1</td>
<td>60</td>
</tr>
<tr>
<td>S2</td>
<td>CHEN</td>
<td>P3</td>
<td>RING</td>
<td>D2</td>
<td>40</td>
</tr>
<tr>
<td>S3</td>
<td>HONG</td>
<td>P1</td>
<td>BOLT</td>
<td>D3</td>
<td>90</td>
</tr>
<tr>
<td>S3</td>
<td>HONG</td>
<td>P3</td>
<td>RING</td>
<td>D3</td>
<td>80</td>
</tr>
</tbody>
</table>

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.

\[ R, T: \text{ relation names} \]
\[ r, t: \text{ tuple variables ranging respectively over } R \text{ and } T \]
\[ c(r): \text{ condition over } r \]
\[ c(r, t): \text{ condition over } r \text{ and } t \text{ jointly} \]

**EXAMPLE RELATIONS**

\[
\begin{array}{cccc}
S & P & SP & P1 \\
S\# & SNAME & AGE & P\# & PNAME \\
\hline
S1 & WONG & 30 & P1 & GEAR \\
S2 & HONG & 35 & P2 & RING \\
S3 & CHEN & 40 & P3 & TIRE \\
\end{array}
\]

\[
\begin{array}{cccc}
SP & P\# & PNAME \\
S1 & P1 & P1 & GEAR \\
S1 & P2 & P3 & TIRE \\
S2 & P3 & P4 & BCLT \\
S3 & P1 & P5 & BELT \\
S3 & P2 \ \\
S3 & P3 \\
\end{array}
\]

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

\[ R[c] = \{ r | r \in R \land c(r) = 1 \} \]

**Example:** \( S[\text{AGE} > 30] \)

\[
\begin{array}{cccc}
S\# & SNAME & AGE \\
\hline
S2 & HONG & 35 \\
S3 & CHEN & 40 \\
\end{array}
\]

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

5
\[ R[L] = \{ r[L] \mid r \in R \} \]

Example: \( S[S\# \text{, AGE}] \)

\[
\begin{array}{ll}
S\# & \text{AGE} \\
--- & --- \\
S1 & 30 \\
S2 & 35 \\
S3 & 40 \\
\end{array}
\]

**Definition:** The JOIN of \( R \) and \( T \) is denoted by \( R[A \times B]\langle T \rangle \), \( A \) and \( B \) are subsets of attributes in \( R \) and \( T \) respectively, and defined as:

\[ R[A \times B] = \{ rt' \mid r \in R, t' \in T', r[A] = t[B] \}, \]

where \( T' \) is the projection of \( T \) which excludes \( B \) from \( T \).

Example: \( SP[\text{P}\# \times \text{P}\#]\langle \text{P}\# \rangle \)

\[
\begin{array}{llll}
\text{S}\# & \text{P}\# & \text{PNAME} \\
--- & --- & --- \\
S1 & P1 & \text{GEAR} \\
S1 & P2 & \text{RING} \\
S2 & P3 & \text{TIRE} \\
S3 & P1 & \text{GEAR} \\
S3 & P2 & \text{RING} \\
S3 & P3 & \text{TIRE} \\
\end{array}
\]

**Definition:** The DIVISION of \( R \) on attributes \( A \) by \( T \) on attributes \( B \) is denoted by \( R[A / B]\langle T \rangle \) and defined as:

\[ R[A / B]\langle T \rangle = \{ r[\overline{A}] \mid r \in R \land \forall t \in T, \exists q \in R \text{ s.t.} \]

\[ r[\overline{A}] = q[\overline{A}] \land q[A] = t[B] \}\}

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

Example: \( SP[\text{P}\# / \text{P}\#]\langle \text{P}\# \rangle \)

\[
\begin{array}{ll}
\text{S}\# \\
--- \\
S3 \\
\end{array}
\]

6
DEFINITION: The PRODUCT of R and T is denoted by \( R \times T \), and defined as:
\[
R \times T = \{ rt \mid r \in R \land t \in T \}
\]
Example: \( S \times P \)

<table>
<thead>
<tr>
<th>S#</th>
<th>SNAME</th>
<th>AGE</th>
<th>P#</th>
<th>PNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>WONG</td>
<td>30</td>
<td>P1</td>
<td>GEAR</td>
</tr>
<tr>
<td>S1</td>
<td>WONG</td>
<td>30</td>
<td>P2</td>
<td>RING</td>
</tr>
<tr>
<td>S1</td>
<td>WONG</td>
<td>30</td>
<td>P3</td>
<td>TIRE</td>
</tr>
<tr>
<td>S2</td>
<td>HONG</td>
<td>35</td>
<td>P1</td>
<td>GEAR</td>
</tr>
<tr>
<td>S2</td>
<td>HONG</td>
<td>35</td>
<td>P2</td>
<td>RING</td>
</tr>
<tr>
<td>S2</td>
<td>HONG</td>
<td>35</td>
<td>P3</td>
<td>TIRE</td>
</tr>
<tr>
<td>S3</td>
<td>CHEN</td>
<td>40</td>
<td>P1</td>
<td>GEAR</td>
</tr>
<tr>
<td>S3</td>
<td>CHEN</td>
<td>40</td>
<td>P2</td>
<td>RING</td>
</tr>
<tr>
<td>S3</td>
<td>CHEN</td>
<td>40</td>
<td>P3</td>
<td>TIRE</td>
</tr>
</tbody>
</table>

DEFINITION: The INTERSECTION of R and T is denoted by \( R \cap T \) and defined as:
\[
R \cap T = \{ r \mid r \in R \land r \in T \}
\]
Example: \( P \cap P1 \)

<table>
<thead>
<tr>
<th>P#</th>
<th>PNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>GEAR</td>
</tr>
<tr>
<td>P3</td>
<td>TIRE</td>
</tr>
</tbody>
</table>

DEFINITION: The DIFFERENCE of R and T is denoted by \( R - T \) and defined as:
\[
R - T = \{ r \mid r \in R \land r \notin T \}
\]
Example: \( P - P1 \)

<table>
<thead>
<tr>
<th>P#</th>
<th>PNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>RING</td>
</tr>
</tbody>
</table>

DEFINITION: The UNION of R and T is denoted by \( R \cup T \) and defined as:
\[
R \cup T = \{ r \mid r \in R \lor r \in T \}
\]
Example: $P \cup P_1$

\[
\begin{array}{l}
P\# & PNAME \\
P1 & GEAR \\
P2 & RING \\
P3 & TIRE \\
P4 & BOLT \\
P5 & BELT \\
\end{array}
\]

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:

\[
\begin{array}{l}
S(S\#, SNAME, AGE) \\
P(P\#, PNAME, COLOR) \\
D(D\#, MGR) \\
E(E\#, D\#, SALARY) \\
SPD(S\#, P\#, D\#, QTY) \\
\end{array}
\]

Example: Find the supplier who supplies $P_2$.

\[
\begin{align*}
R1 &= SPD[P\# = 'P2'] \\
R2 &= R1[S\#] \\
PRINT R2 & & \text{/* to print the answer */}
\end{align*}
\]

Example: Find the supplier who supplies red parts.

\[
\begin{align*}
R1 &= P[COLOR = 'red'] \\
R2 &= SPD[P\# * P\#]R1 \\
R3 &= R2[S\#] \\
PRINT R3 & & \text{/* to print the answer */}
\end{align*}
\]
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.

![Diagram showing architecture of 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 consistency.

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 database 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.

![Diagram of Page Structure of PSS](image)

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

DATA DEFINITION LANGUAGE (DDL)

Syntax:

```
/*
SCHEMA IS name

* /

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 

*/

REATION IS name

$USER IS <name, right> .......... 

$WITHIN directory filename FILES

(assume filename in accordance with
 the directory and the filename)

$PAGE SIZE blockno (initial pagesize 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.
Based on the syntax above, we give an example to design a conceptual relation.

-1-
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.

```
part
    ↓
requirement
    ↓
project
```

Each relation has attributes as shown below:
- **part**: (part_no, part_desc, qty_on_hand, recorder, unit_cost)
- **project**: (project_no, project_desc, manager, comp1_date)
- **reart**: (project_no, part_no, qty_read)

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_desc**: string character has length less than 20
- **qty_on_hand**: integer >0 and < 200 units kg
- **recorder**: string
- **unit_cost**: float

... etc.
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 record with the same key value"
  ON MODIFY CALL "MODIFY record with the same key value"
  ON OPEN CALL "Permission and concurrency control"
  ON CLOSE CALL ..... 

RELATION IS project
  WITHIN P1,P2(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> )`
   | `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-exp>`

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

16. `<enumerative-range>` ::= `( <enumerative-list> )`
7. <enumerative-list> ::= <string>
   | <string>, <enumerative-list>

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

11. <units> ::= <id>

12. <length> ::= <unsigned-integer>

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

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

18. <float> ::= <integer>, <unsigned-integer>
   | <integer>,
   | ., <unsigned-integer>

21. <relation-context> ::= <rel-exp>
   | <rel-exp> <relation-context>

22. <rel-exp> ::= <relation> <rel-exp> <field-exp> <rel-block> ; END
   | <relation> <field-exp> <rel-block> ; END
   | <relation> <field-exp> ; END

23. <relation> ::= RELATION IS <rel-name> ; WITHIN <file-exp>

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

26. <file-exp> ::= <file-name>
   | <file-name>, <file-exp>

27. <rel-exp> ::= $PAGESIZE <unsigned-integer>

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

29. <relation-set> ::= ; USER IS <user-exp>
      | ; RULEPART <key-exp>
      | <index-exp>
      | ; ACCESS USING <mode>
      | ; ON <check-exp>

30. <field-exp> ::= <field block>
   | <field block> <field-exp>

31. <field-exp> ::= <attribute-exp>
   | <attribute-exp> <field-exp>

32. <field-block> ::= <attribute>
       | <attribute> <attribute-block>

33. <attribute-exp> ::= <attribute>
   | <attribute> <attribute-block>

34. <attribute> ::= $ATTRIBUTE <role-name> DOMAIN <domain-name>

35. <attribute-block> ::= IS <sub-exp> RESULT OF <derived-name>
37. \texttt{< ac_exp >} := ACTUAL
    | VIRTUAL
38. \texttt{< op_exp >} := GET
    | STORE
    | MODIFY
    | DELETE
39. \texttt{< derived_psm_name >} := \texttt{< id >}
40. \texttt{< check_psm_name >} := \texttt{< id >}
41. \texttt{< role_name >} := \texttt{< id >}
42. \texttt{< domain_name >} := \texttt{< id >}
43. \texttt{< user_exp >} := \texttt{< user block >}
        | \texttt{< user block > < user_exp >}
44. \texttt{< user block >} := \texttt{< < user_name > < right >}}
45. \texttt{< user_name >} := \texttt{< id >}
46. \texttt{< right >} := R
        | W
        | RW
        | WR
47. \texttt{< key_exp >} := \texttt{< key_name >}
        | \texttt{< key_name > \texttt{< key_exp >}}
48. \texttt{< key_name >} := \texttt{< id >}
49. \texttt{< index_exp >} := \texttt{< index block >}
        | \texttt{< index block > < index_exp >}
50. \texttt{< index block >} := ; INDEX USING \texttt{< index_name >}
51. \texttt{< index_name >} := \texttt{< attribute_name >}
        | \texttt{< attribute_name > \texttt{< index_name >}}
52. \texttt{< attribute_name >} := \texttt{< id >}
53. \texttt{< mode >} := BAD
54. \texttt{< check_exp >} := \texttt{< semantic check >}
        | \texttt{< concurrent check >}
        | \texttt{< privacy check >}
55. \texttt{< semantic check >} := \texttt{< op_exp > CALL < check_psm_name >}
56. \texttt{< concurrent check >} := \texttt{< condition_exp > CALL < check_psm_name >}
57. \texttt{< privacy check >} := PRIVACY CALL < check_psm_name >
58. \( \langle \text{condition-exp} \rangle := \text{OPEN} \\
\quad \text{CLOSE} \)

59. \( \langle \text{unsigned-integer} \rangle := \langle \text{digit} \rangle \\
\quad \text{|} \langle \text{digit} \rangle \langle \text{unsigned-integer} \rangle \)

60. \( \langle \text{digit} \rangle := 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \)

61. \( \langle \text{id} \rangle := \langle \text{letter} \rangle \\
\quad \text{|} \langle \text{letter} \rangle \langle \text{alph-num} \rangle \)

62. \( \langle \text{alph-num} \rangle := \langle \text{digit} \rangle \\
\quad \text{|} \langle \text{letter} \rangle \)

63. \( \langle \text{Letter} \rangle := \text{A} \mid \text{B} \mid \text{C} \mid \text{D} \mid \text{E} \mid \text{F} \mid \text{G} \mid \text{H} \mid \text{I} \mid \text{J} \mid \text{K} \mid \text{L} \mid \text{M} \mid \text{N} \\
\quad \text{|} \text{O} \mid \text{P} \mid \text{Q} \mid \text{R} \mid \text{S} \mid \text{T} \mid \text{U} \mid \text{V} \mid \text{W} \mid \text{X} \mid \text{Y} \mid \text{Z} \\
\quad \text{|} \text{a} \mid \text{b} \mid \text{c} \mid \text{d} \mid \text{e} \mid \text{f} \mid \text{g} \mid \text{h} \mid \text{i} \mid \text{j} \mid \text{k} \mid \text{l} \mid \text{m} \mid \text{n} \\
\quad \text{|} \text{o} \mid \text{p} \mid \text{q} \mid \text{r} \mid \text{s} \mid \text{t} \mid \text{u} \mid \text{v} \mid \text{w} \mid \text{x} \mid \text{y} \mid \text{z} \)
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: fixed length (balance tree)
content=>
{
    int freesrtr;%point to the head of free queue%
    int availr;%point to the head of avail queue%
    int rootrtr;%point to the root of balance tree%
    struct direct[];
}

(2) DOMAIN file: variable length (40 + variable)
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 kename[];
    struct ndx[];
}

(4) PROTECTION file:
content=>
(a) 6 blocks for ID directories
{
    char idname[15];
    int link;
    char unused[15];
}

(b) data area
{
    struct protect;
}

struct direct
{
    char name[15]; /*domain or relation name */
    char type; /*DOMAIND->D; RELATION->R */
    int offset; /*offset in domain or relation file */
    int totalen; /*max size record length */
    int rlink; /*balance tree right link */
    int llink; /*balance tree left link */
}
# dom_hd

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 range_type; /* 0: no range, 1: str type, 2: num type */
    int range_len; /* total length for range area */
    char unit[2];
    char dom_unused[30];
};

# rel_hd

struct rel_hd
{
    int fileno; /* total fileno used by relation */
    int page_size; /* page size for physical allocation */
    int pageso; /* max pages in one file */
    int attno; /* field no in this relation */
    int keyno; /* number of fields in key group */
    char access_mode; /* access mode */
    char indexno; /* No. of index tables established */
    int indexlen; /* total length of index message */
    char dbname[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 cdbname[15]; /* taskname for open call */
    char cdbname[15]; /* taskname for close call */
    char reltype; /* type is variable or fix length */
    char rel_unused[24];
};

# num

struct num
{
    int rtfrom; /* initial range start value */
    int rto; /* initial range ending value */
    int rstep; /* use '0' close range assign */
};

# str

struct str
{
    char value[3]; /* string value buffer area */
    char strrend; /* use '0' close */
};
```c
struct rang_num {
    int tolen;   /* total length of numeric value buffer area */
    struct num[]; /* numeric value buffer structure */
    char end;    /* use '0' close */
};

struct indx {
    int len;    /* length 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 protect[]; /* message buffer include relation_name ; right */
};

struct protect {
    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 actual or virtual call */
    char dname[15]; /* taskname for delete 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  rename
   WHERE <where clause> $condition to be met.

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

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

semantics:
   modify those tuples in rename 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  rename

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

Basically, the syntax and operation is derived from Codd’s relational algebra. Syntax 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

Example:

```sql
JOIN <Dno> 
FROM DEF,EMP 
INTO Person <Dno> 
WHERE salary < 1000
```
detail syntax */

1) 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:
   relnm= \{ [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 relnm 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 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, 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.

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

] semantics:
relnm = \{ t1 | [t1.attr. list] is in rel1, for all [attr. list]
in rel2 and <other condition> \}

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

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

] 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 adding 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,

- string match composed of:
  - exact match—match whole string
  - exact word match—match word by word.
    - word is defined to be a group of strings with space separated.
  - partial string match—match string by string

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

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 \in S as:

\[ \begin{align*}
P & \equiv [IN, OUT] \\
\forall x \in S, \exists y \in S, y &= x \quad \text{true} \\
\forall Y \in S, \forall x \in S, Y! &= x \quad \text{true} \\
& \text{(5) [attr, (AND, OR, , ,)] attr,..., [IN, OUT] {1,2,...,discrete list} or} \\
& \text{1..2--continuous range} \\
& \text{(6) attr = 'word' or '?string'} --- \text{partial match} \\
& \text{(7) attr IN {'word' or '?string'}} \end{align*} \]
representation of DML:

1. \(<\text{where clause}\> ::= \text{WHERE} \<\text{condition}\>
2. \(<\text{condition}\> ::= \<\text{logical and-exp}\>
   \| \<\text{logical and-exp}\> \<\text{or-operator}\> \<\text{condition}\>
3. \(<\text{logical and-exp}\> ::= \<\text{relational exp}\>
   \| \<\text{relational exp}\> \<\text{and-operator}\> \<\text{logical and-exp}\>
4. \(<\text{relational exp}\> ::= \<\text{attribute}\> \<\text{relational-operator}\> \<\text{operand}\>
   \| \<\text{attribute}\> \<\text{set-operator}\> \<\text{discrete-operand}\>
   \| \<\text{set-attribute}\> \<\text{relational-operator}\> \<\text{operand}\>
   \| ( \<\text{condition}\> )
5. \(<\text{and-operator}\> ::= \text{AND} \)
6. \(<\text{or-operator}\> ::= \text{OR} \)
7. \(<\text{relational-operator}\> ::= \<\text{==}\> | \<\text{!=}\> | \<\text{>}\> | \<\text{>=}\> | \<\text{<}\> | \<\text{<=}\>
8. \(<\text{set-operator}\> ::= \text{IN} | \text{OUT}
9. \(<\text{operand}\> ::= \<\text{attribute}\>
   \| \<\text{constant}\>
10. \(<\text{constant}\> ::= \<\text{integer}\>
    \| \<\text{float}\>
    \| \<\text{char}\>
    \| \<\text{strings}\>
11. \(<\text{set-attribute}\> ::= [ \<\text{attr-list}\> ]
12. \(<\text{attr-list}\> ::= \<\text{attribute}\>
    \| \<\text{attribute}\> \<\text{set-attri-operator}\> \<\text{attr-list}\>
13. \(<\text{set-attri-operator}\> ::= \<\text{and-operator}\>
    \| \<\text{or-operator}\>
14. \(<\text{discrete-operand}\> ::= ( \<\text{discrete-list}\> )
15. \(<\text{discrete-list}\> ::= \<\text{discrete-constant}\>
    \| \<\text{discrete-constant}\> , \<\text{discrete-list}\>
16. \(<\text{discrete-constant}\> ::= \<\text{constant}\>
    \| \<\text{subrange}\>
17. \(<\text{integer}\> ::= \<\text{unsigned integer}\>
    \| \<\text{sign}\> \<\text{unsigned integer}\>
18. \(<\text{unsigned integer}\> ::= \<\text{digit}\>
    \| \<\text{digit}\> \<\text{unsigned integer}\>
19. \(<\text{float}\> ::= \<\text{integer}\> , \<\text{unsigned integer}\>
    \| \<\text{integer}\>

\[ \text{\textless sign\textgreater} \; , \; \text{\textless unsigned integer\textgreater} \\
\quad \text{\textless unsigned integer\textgreater} \\
\quad \text{\textless char\textgreater} \; \text{\textless letter\textgreater} \; ' \\
\quad \text{\textless string\textgreater} \; \text{\textless word\textgreater} \; ' \\
\quad \text{\textless word\textgreater} \; \text{\textless letter\textgreater} \; \text{\textless letter\textgreater} \\
\quad \text{\textless letter\textgreater} \; \text{\textless word\textgreater} \\
\quad \text{\textless subrange\textgreater} \; \text{\textless subrange value\textgreater} \; \text{\textless subrange value\textgreater} \\
\text{\textless subrange value\textgreater} \; \text{\textless integer\textgreater} \\
\quad \text{\textless float\textgreater} \\
\quad \text{\textless char\textgreater} \\
\text{\textless attribute\textgreater} \; \text{\textless leading letter\textgreater} \\
\quad \text{\textless leading letter\textgreater} \; \text{\textless following word\textgreater} \\
\text{\textless leading letter\textgreater} \; \text{\textless alph char\textgreater} \; \text{"$"} \\
\text{\textless following word\textgreater} \; \text{\textless letter\textgreater} \\
\quad \text{\textless word\textgreater} \\
\text{\textless letter\textgreater} \; \text{\textless digit\textgreater} \\
\quad \text{\textless alph char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\text{\textless digit\textgreater} \; \text{\textless digit\textgreater} \; \text{\textless digit\textgreater} \\
\text{\textless alph char\textgreater} \; \text{\textless alph char\textgreater} \\
\quad \text{\textless alph char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\quad \text{\textless special char\textgreater} \\
\; -$
External schema:

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

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 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 storagemap {
    int logierno; /* the result value of hashing */
    int physerno; /* the page no. of physical allocate */
    int freespac; /* indicate page remains how much space */
}
(B) Core Storage Structure

E.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 Kreftnt; /* indicates the status of relation */
    int fileno; /* total file no. used by relation */
    int pagesize; /* the size of one page, unit is block */
    char rstatus; /* relation status */
    int pagesize; /* 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 Rcallpro; /* bit indicates Call check_pro or not */
    char *Rcallpro; /* 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 *attrptr; /* pointer to attri struct */
    struct index *indxptr; /* or point to index structure */
    struct page *pageptr; /* pointer to page structure */
}

E.2

struct attribut {

    char atname[15]; /* attribute name */
int klink; /* the link list of key */
int Acallpred; /* 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 domain[15]; /* domain name */
char Drefcnt; /* indicate the status of domain */
char pctype; /* domain's picture and type */
int len1; /* picture length */
int len2; /* the 2nd length of pic only for floating type */
char ranstype; /* the type of domain's range */
int randlen; /* the length of domain's range */
char unit[21]; /* the unit of domain */
char *address; /* the address of domain's var part info */
}

B.4
struct page {
int phypno; /* 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 *status; /* page status */
}
B.5

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

B.6

```c
struct index_file {
    char indexfilename[15]; /* index file name */
    char *idxptr; /* pointer to the 1st attri pointer of idx */
};
```

B.7

```c
struct Istattr {
    char *Idxattrptr; /* pointer to the attribute name */
};
```

B.8

```c
struct whereattr {
    char *relationptr; /* pointer to relation name in string buf */
    int nVarLengthAttri; /* the no. of var length attri in one relat */
    char *domainptr; /* pointer point to domain structure */
};
```

B.9

```c
struct type {
```
int type; /* indicate command's type */
char *strstr; /* pointer point to string buffer */
struct whereattr *wherestr; /* 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 separated)
$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, go 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 tuples from R1

-6-
(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
(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, attribute, domain table
    a. Set refcnt of domain table to -1
    b. Set refcnt of rel_head, page, attribute, file_descpt table

(3). Data manipulation

(a). Parsing
   1. Set 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 tmp link (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. create 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_descrt table
(a), One record contains 18 bytes.
(b), It reserves 10 file_descrt 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 dua loc _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 correspondences.
   (f). insert
       f.1 check page table
if no enough space then delete decision by refont
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 index_file, last_ptr, 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 allocate space to joint field and where clause
   (b). insert information to allocated space