

RELATIONAL DATABASE MANAGEMENT SYSTEM

FOR A NETWORK

PROJECT WORK

p-1201

**SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE AWARD OF THE DEGREE**

**B.E-COMPUTER SCIENCE AND ENGINEERING,
BHARATHIAR UNIVERSITY, COIMBATORE**

PROJECT WORK DONE BY,

ARVIND, B. V., (0027K0159)

PRIYA, S., (0027K0192)

GUIDED BY,

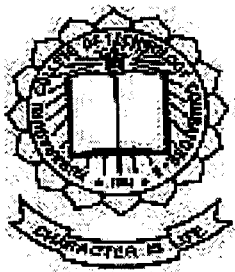
Prof. MUTHUKUMAR, A., M. Sc., M.C.A., M. Phil.

DEPARTMENT OF MATHEMATICS AND COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

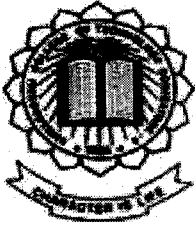
KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE- 641006



MARCH 2004

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE- 641006



CERTIFICATE OF PROJECT WORK

This is to certify that the project titled '**RELATIONAL DATABASE MANAGEMENT SYSTEM FOR A NETWORK**' is done by

ARVIND, B. V., 0027K0159

PRIYA, S., 0027K0192

and submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Science and Engineering of Bharathiar University, Coimbatore.

Head of the Department

(Dr. Thangasamy, S.)

DATE: 22.03.2004

Guide

(Professor. Muthukumar, A.)

DATE: 22.03.2004

Certified that the candidates were assessed by us in the project work viva-voce examination held on 23.03.2004

Internal Examiner

External Examiner

DECLARATION


We,

Arvind, B. V., 0027K0159

Priya, S., 0027k0192

Here-by, declare that the project titled 'Relational Database Management System For a Network' is done by us under the guidance of Professor. Muthukumar, A., and to the best of our knowledge, a similar work has not been submitted to the Bharathiar University or any other institution, for the fulfillment of the requirement of the course of study.

This report is submitted on the partial fulfillment of the requirements for the award of Degree of Bachelor of Computer Science and Engineering of Bharathiar University.


Arvind. B. V


Priya. S

Place: COIMBATORE

Date: 22.03.2004

Counter Signed by



Prof. Muthukumar, A., M.Sc., M.C.A., M.Phil.,

Acknowledgement

ACKNOWLEDGEMENT

We take immense pleasure to thank everybody who has supported us in the course of this project.

We thank our Professor and Head of the Department, Computer Science, Dr. Thangasamy, S., who made his valuable presence at the reviews and gave his thoughtful words.

To be grateful to our guide, Professor MuthuKumar, A., M. Sc., M.C.A., M. Phil., Department of Mathematics and Computer Applications, is indeed a great pleasure and duty. He has supported us throughout our project, has put up with our shortcomings and has given helpful corrections. He has taken keen interest in our project and is interested in still enhancing it with his knowledge; for this we are really thankful.

We greatly appreciate and thank our Project Coordinator, Ms. Chandrakala, D., M.E., Senior Lecturer, Department of Computer Science, who took great care to be organize all reviews and provide advice and guidance on all project details. We also thank Ms. Devaki, S., M.S., Assistant Professor, Department of Computer Science, who was also present at the review and has given valuable recommendations.

We sincerely thank the lab administrators, who have helped us many times.

Our heartiest thanks to our friends who have taken pains to clear our doubts and discuss details whenever we asked for.

Synopsis

SYNOPSIS

The project 'Relational Database Management System' involves creation of a database management system and using this to create database in a network. The database system finds application in fields where information storage and manipulation of the stored information are important.

In the existing database systems, there is a plethora of functional options available to the user. The reason for developing this project is to customize the database system and provide only the functionalities required to the user, thereby reducing the learning effort that a database user has to put.

A wide range of the basic functionalities that is required for the efficient usage of database is provided here. Many of the DML and DDL commands that are important to the user are supported. A great level of independence has been provided in implementing these different functions; so future enhancement can be done easily.

The user information is greatly safeguarded by maintaining a separate domain for storing information about each user. Password protection is provided to protect the system from unauthorized access. The actual burden of the system is borne by the server. The role of the client is reduced to an information supplier and information presenter.

Extensive testing has been done to prevent the system from different inconsistencies that may arise. The system is flexible and efficient, with room provided for future enhancement.

Contents

TABLE OF CONTENTS

1	Introduction	
1.1	Existing system	1
1.2	Proposed system	1
2	Project Specification	
2.1	Product definition	2
2.2	Project plan	3
3	Requirements Specification	
3.1	Introduction	4
3.2	General description	4
3.3	Specific requirements	5
4	Design Document	
4.I	System Design	8
4.I.1	Server system	8
4.I.2	Client system	19
4.I.3	Server administration	19
4.I.4	Data flow diagram	20
4.II	System Development	26
5	Product Testing	
5.1	Unit testing	27
5.2	Integration testing	29
6	Future Enhancements	30
7	Conclusion	31
8	References	32
9	Appendix	
9.1	Sample source code	33
9.2	Sample output	43

Introduction

1.INTRODUCTION

The system developed is a simulation of a Relational Database Management System (RDBMS) for handling data over a network.

The development of this system has been undertaken as more of a study than of creating a system that is commercially viable to applications. The system, however attempts to provide all the basic, yet, essential functionality of any typical RDBMS. The system while functioning over a network can serve multiple clients.

This system can be used to study the functioning of an RDBMS and can help in practicing simple queries. It has tremendously helped us, in-turn, to learn the techniques behind designing and constructing an RDBMS.

1.1 EXISTING SYSTEMS

While the existing systems are at a better status in terms of functionality and commercial usage, but at the cost of decades of continuous and collective efforts.

Our system, however, will continue to enhance its features and serve other purpose, besides the ones stated above.

A very detailed discussion of the system developed by us follows this introductory note.

Product Specification

2. PROJECT SPECIFICATION

2.1 PROJECT DEFINITION

The project aims to build a relational DBMS, with network handling capabilities. In the initial phase of the project, software to create database for a single user is done. This would include most of the DBMS operations supported by SQL. In the second phase, the network feature of the project is dealt. In this phase, the system is designed to accept multiple users, providing separate domain for users.

In the design of the database, care is taken so that the architecture of the database is flawless. Module independence is given the highest priority. The reason for focusing more on modularity is because the testing process can be carried in an ad-hoc manner. In implementing queries, the original proposal for CREATE query was to take into consideration the primary key and foreign key implementations. In the SELECT query, the proposal was to include the 'where' option of the query along with the selected field display implementation. The DML operations that were considered for implementation were INSERT, ALTER, and DROP, DELETE, TRUNCATE and DESC. The data types that the DBMS planned to support were number, date and character.

In the network part, the planned architecture was to have different domains for the users. The domain allotted to users were also made sure were unique. The table names created within the domain should contain unique names. In creation of new domain, if the user does not have an account, a new domain is created for him. The parsing of the queries are to be carried out in the server side, no processing is done on the client side.

2.2 PROJECT PLAN

In the implementation of database the information of the table is stored in the form of structure. The members are ordinary members and not structure pointers. So, these way details of all the tables can be stored in the database-using array of structures. In the main catalog file, list of all users who can access the database are available. When the user name and the password match, control is taken to the directory (creation of a domain involves creation of directory here) of the user.

All the user actions are recorded and stored only in the domain specified by the user. Once the user exits, control is brought out of his domain.

In creation of records, once the record is parsed for syntactical correctness, records got in as string is written into the file. Thus the records are stored in file in different rows. For the DML commands the file is read and stored in a structure and then altered according to the requirement.

All the user names are stored in a file along with password information. When the user types profile wrongly, new user option is displayed, which would enable a new user to create a session for DBMS. The removal of user is done at the discretion of the administrator. Once the user is destroyed, all the records stored by the user are also lost and the user has to create only a new session.

Requirements Specification

3.REQUIREMENTS SPECIFICATION

3.1 INTRODUCTION

3.1.1 PURPOSE

The SRS clarifies the users, about the technical intricacies, system design, functional requirements and other relevant issues pertaining to the development of **RELATIONAL DATABASE MANAGEMENT SYSTEM FOR A NETWORK.**

3.1.2 SCOPE

The software can be used in a network or in a single user environment. Though, the software developed does not currently support a few DML operations, provision have been made for future addition of the DML operations.

3.1.3 GLOSSARY

RDBMS - Relational Database Management System

SQL - Structured Query Language

DDL - Data Definition Language

DML - Data Manipulation Language

IP - Internet Protocol

3.2 GENERAL DESCRIPTION

3.2.1PRODUCT FUNCTIONS

When a user wants to create a database, the user is first created a domain to work in. Upon the creation of the domain, the user can use the database. The user queries are brought in from the client side and processed in the server. In the server side, the parser first scans the input for syntactic correctness, and then the appropriate functions are called. If format of the query is wrong, error is thrown to the client side. Before a user logs in user name and password verification are done. Thus the system is saved from the malicious user.

3.2.2 USER CHARACTERISTICS

The user of the database is required to know the ORACLE functioning and the SQL queries. Since the format of various DDL and DML commands have to be specified in the same format as it is done in SQL, a deep knowledge in SQL would facilitate the easy usage of the database. People with decent knowledge in ORACLE can easily understand the architecture of the database also. Also if the administrator is well versed in TCP/IP standards, it will be easier for him to follow the information sharing and session maintenance.

3.3 SPECIFIC REQUIREMENTS

3.3.1 FUNCTIONAL REQUIREMENTS

3.3.1.1 INTRODUCTION

The modules involved in development of the system are

- Database creation
- Network management

In the design of the database system, a parser is first designed, which is used for processing the query sent by the user. Next step in designing the database involves the handling of DML and DDL operations.

In the network management system, a procedure to handle new client, to authenticate existing client, to create session for the client and to handle transactions between the client and server are taken care.

3.3.1.2 LIST OF INPUTS

The user is first prompted for the user name and password. If user gives incorrect information, the user is requested for the information again. Upon entering the information correctly, the user is displayed a command prompt.

After logging in, the user has to type the SQL queries .For the create function, the user has to specify the query in the usual format along with fields and constraints. When syntax is correct, a table is created by the DBMS as specified by the user, if

syntax is incorrect, the user is displayed the flaw in the statement and prompted for the command again.

In order to insert record into table, the user keys the insert query in. If the syntax of the query is correct, the new record is stored; otherwise an error is thrown. Before inserting, the constraints are checked.

To delete the table, the user types delete command followed by the table name. If the table contains no records the table is dropped, otherwise the user is asked to truncate the table before dropping it.

In altering the table, only the add option is supported, the user types the query along with the field to be updated. If the query is correct, both the metafile and the records already present have to be modified.

For truncating the table the user has to type the truncate query along with the table name. All records are deleted from the table, if the query is correct.

In order to log out, user types exit in the command prompt. This logs out the user and also changes the information about the user in the user detail file.

3.3.1.3 INFORMATION PROCESSING REQUIREMENTS

In order to allow the user to access the database, a record storing the user information (username, password and status) is required. If a new user logs in, the information about the user is updated in the record.

After the user is logged in, to execute create query successfully, the user name, table name and the constraints are specified properly. A catalog file is required to keep track of the tables created by the user. Using this catalog file the unique constraint is checked.

For the successful completion of other DML queries, the data structure created in create function should be proper. The data structure created in create query is altered and in cases destroyed (in drop query) upon the execution of DML commands.

3.3.2 DESIGN CONSTRAINTS

3.3.2.1 STANDARDS COMPLIANCE

The queries input by the user follow the SQL query format. In giving the DML queries, the format that is used in SQL, is used here as well. In the establishment of connection between the client and server, TCP standard is adhered. The communication between the clients and server take place through sockets. The creation of socket and communication between the client and server through socket take place as specified by TCP.

3.3.2.2 HARDWARE AND SOFTWARE REQUIREMENTS

The memory requirements for the software come to 20 KB. In the client side, it is sufficient to store the client program also. The processor speed is 455 M.Hz.

The software is written in C language. It is written in Linux 9.0 platform. In writing software some of the C functions unique to Linux have been made use of.

3.3.2.3 HARDWARE LIMITATIONS

The application is dependent more upon the, capacity of server. Most of the processing of the user queries and network administration is carried out in the server side, therefore client is reduced to the position of dummy, which is just required to get the user input and pass it to server for further processing.

(No other user interface is provided here, the user is required to use the database in the command prompt only)

3.3.2.4 USER INTERFACE

The user is displayed a command prompt as the interface on the client side. Message is displayed as text, on the client side. By making use of appropriate tool, a graphical interface can also be provided

Design Document

4. DESIGN DOCUMENT

4.1. SYSTEM DESIGN

The system has been structured in the following manner:

- ♦ Server system
- ♦ Client system
- ♦ Server administration

A detailed discussion of how each of these modules is designed, constructed and what function it carries out shortly follows.

4.1.1 SERVER SYSTEM

The server system is the core of these systems. It provides the database functionalities to the clients on the network. Hence this system is sub-structured into

- ♦ RDBMS core system
- ♦ Network functionalities

4.1.1.1 RDBMS CORE SYSTEM:

The RDBMS is that core software which receives the queries from the clients and interprets them and performs the requested function. This the RDBMS does in two stages, further splitting this sub-system's functionalities.

Stage I:

This chapter is called 'Query validation'. The query received from the client is checked for the syntax. This is done in order to ensure that enough details are provided for the requested operation to be performed and that these details are valid.

Stage II:

In this 'Query processing' chapter, the relevant operations for the previously validated query are performed. That is, operations like creating or altering a table, manipulating records in the table, etc are done as the case may be.

4.1.1.1.a QUERY VALIDATION

Queries Handled

The RDBMS designed can handle all the DDL and DML queries as specified in the SRS. The queries handled are in the simplest forms.

DDL queries

Create-it creates a table

Drop –it is used to drop i.e. purge a table from the database. This can be done only when the table has no records

Truncate-it can remove all records from the specified table

Alter-here the simplest option of adding a new field to the existing schema of a table is supported

DML queries

Insert-records are inserted into the table using this query

Select-in this query, the system supports the simplest form of extracting all the records from the specified table

Update-a particular column entry of all the records can be updated

Delete-this is used to delete all the records from the table. In the absence of the 'Where' option it functions like the truncate query.

The system supports only the simplest form of the DML, in that, the 'Where-form' of the queries is not supported. Hence the choice of selecting, updating or deleting a particular record is at present unavailable.

Query Grammar

The validation of the queries is performed considering the grammar specified by the designers of the system. But most of the queries adhere to the SQL-like formats. Given the supported functionalities of the system, the grammar defines the query structure as follows:

```
<QUERY>          ::= <COMMAND> <TERMINATOR>
<COMMAND>       ::= <DDL> | <DML>

<DDL>           ::= <DDL_CMD> <OBJECT> <NAME> |
                   <DDL_CMD> <OBJECT> <NAME> <DDL_SPECIFIC>
<DML>           ::= <DML_CMD> <DML_SPECIFIC>

<DDL_SPECIFIC>  ::= <CREATE> | <ALTER>
<DML_SPECIFIC> ::= <INSERT> | <SELECT> | <UPDATE> | <DELETE>

<CREATE>        ::= <OPEN> <NAME> <DATA_TYPE> <SIZE> <CLOSE> |
                   <OPEN> <NAME> <DATA_TYPE> <SIZE> <FIELD> <CLOSE>
<ALTER>         ::= <KEYWD_ADD> <OPEN> <NAME> <DATATYPE>
                   <SIZE> <CLOSE> |
                   <KEYWD_ADD> <OPEN> <NAME> <DATA_TYPE> <SIZE> <FIELD> <CLOSE>

<INSERT>        ::= <KEYWD_INTRO> <NAME>
                   <KEYWD_VALUES> <OPEN> <DATA> <CLOSE> |
                   <KEYWD_INTRO> <NAME>
                   <KEYWD_VALUES> <OPEN> <DATA> <VALUES> <CLOSE>

<SELECT>        ::= <KEYWD_STAR> <KEYWD_FROM> <NAME>
<UPDATE>        ::= <NAME> <KEYWD_SET> <NAME> <EQUALTO> <DATA>
<DELETE>        ::= <KEYWD_FROM> <NAME>
```

<FIELD>	::=<SEPARATOR><NAME> <DATA_TYPE> <SIZE> <FIELD>
<VALUES>	::=<SEPARATOR><DATA> <SEPARATOR><VALUES>
<DATA>	::=<STRING> <NUMBER>
<NAME>	::=<LETTER> <LETTER><DIGIT> <LETTER><SPL_CHAR> <LETTER><NAME>
<SIZE>	::=<OPEN><NUMBER><CLOSE> <OPEN><NUMBER><SEPARATOR><NUMBER><CLOSE>
<STRING>	::=<LETTER> <STRING>
<NUMBER>	::=<INTEGER> <FLOAT>
<FLOAT>	::=<INTEGER><DECIMAL><INTEGER>
<INTEGER>	::=<DIGIT> <DIGIT><INTEGER>
<OBJECT>	::=<table>
<DDL_CMD>	::=<create alter drop truncate>
<DML_CMD>	::=<insert update select delete>
<DATA_TYPE>	::=<char number>
<LETTER>	::=<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>
<DIGIT>	::=<0 1 2 3 4 5 6 7 8 9>
<KEYWD_ADD>	::=<add>
<KEYWD_FROM>	::=<from>
<KEYWD_VALUES>	::=<values>
<KEYWD_INTRO>	::=<into>
<KEYWD_STAR>	::=<*>
<SEPARATOR>	::=<,>
<TERMINATOR>	::=<.>
<OPEN>	::=<(>
<CLOSE>	::=<)>
<SPL_CHAR>	::=<_>

4.1.1.1.bQUERY PROCESSING

Once the validation of the queries is over the processing of the query continues if the query complies with the grammar. A detailed description of what steps go into the processing of each query follows.

CREATE COMMAND

The create command query is similar to the create query of the SQL. After the parsing of create query is done by the parser, a call is made to create procedure with user name, table name, field structure array and the number of fields. When parsing is done the table name and the field structure are created. These parameters are passed to the create function. The structure members are field name, field type, size and constraint. The size parameter is subdivided into size1 and size2. The size2 parameter is used for decimal numbers to hold the decimal part. The field name, table name are character pointers, size1 and size2 are integers and the constraint parameter is a character. The cons field holds either "o" or "p"(depending on whether the key is primary or not). For each field a structure member is dedicated, so in order to store all the fields an array of structures is created.

In assigning the structure members the fields given by the user are tokenized using the C function. Except for the last field all other fields are tokenized using the "," separator, for the last field ")" is used as the tokenizing character.

In create command first the command, table and name of table are extracted using blank space as tokenizing character. When extracting the fields, a variable is used to keep a count of the number of the fields present. In assigning the structure members of character data type, the size1 is the size specified by the user. The size2 parameter is assigned to 0, in the case of character data type, since this field is of no use for character data type. In case of a number, the size can be specified in two ways, first as a single parameter (whole number) and second with two parameters to hold both integer size and decimal size. After the data types are extracted the constraint value is extracted, if the following field is null, then it is taken as ordinary field, otherwise as a primary field.

DATA STRUCTURES CREATED DURING CREATE

After the parameters are passed to create function, two files are created. One is the metafile and the other is the file to store the records of the table. Appending the key word "meta" to the table name forms Metafile name and another file is opened with the name of the table itself.

If there is no catalog file created for the user, catalog file is also created during the process. Catalog file is used to store a list of tables that a particular user creates. In case if, user has already created a table, then in the subsequent table creations, only the table names have to be inserted into the catalog file.

Reading from the structure passed by the parser function creates the metafile. Using the count as index, fields from the structure are read and written into the metafile. The metafile serves as a repository of information for further reference. The contents of metafile can be changed only by alter command. The format verification of the create command and its parameters is carried out by the parser itself. (Refer fig 4.1.5)

ALTER COMMAND

The 'alter' query is processed by first verifying that the specified table exists (by verifying the catalog file). Then the new field to be added to the schema is compared against the existing schema (available in the metafile), so that no duplication occurs. If no duplication is found, the new field is added to the schema by updating the metafile for that table. Otherwise the alter operation fails.

Altering the table schema should provide for updating the existing table records with null values for the new field. Rewriting the entire table details with null value for the new field does this.

When the alter table function is called, another function to change the existing records of the table is also called. Initially the metafile is changed to accommodate the new field. Then by reading information from the records and the metafile, the existing records are suitably updated, i.e., NULL entries are added for the newly added column.

TRUNCATE COMMAND

In order to clear all the records of a table, the 'truncate' command can be used. If a table exists in the user's database, the contents of the file for the associated table is just deleted. The metafile and the catalog are left undisturbed. (Refer figure 4.1.1.7)

DROP COMMAND

The 'drop' command purges the specified table from the user's database if one such table is located. But for this the table should be devoid of records. If the table has records the user is notified about this and further processing of the drop query is suspended.

If the table is found to have no records, the file for containing the records is deleted. Also the system does away with the associated metafile for the table. The table name is also expelled from the catalog of the user's database. (Refer figure 4.1.1.8)

INSERT COMMAND

In inserting a tuple into a table, the format of the query is same as that of SQL. After the query is parsed, the values are passed to the insert function. The insert function takes table name and value string as its parameters. In the insert function first a structure is created to read the meta file content. A count of actual fields present in the metafile is first taken. It is then compared with the actual parameters given by the user.

INITIAL CHECKS DONE BEFORE INSERTING

The number of parameters, given by the user is first checked with number of parameters permitted. In case of failure to meet the requirement, error message is reported. If the parameters match, then the values are scanned to check for other constraints match. First the parameter extracted is checked to see whether it meets the length constraint. In order to do that, the constraint that is set on the length parameter of the particular field is got from the metafile. In case of character data type only the length of the field is checked. In case of numeric data type, the number will be available as a string initially, so the number is converted to integer using the available built in C function. The number is then checked for its length using a simple algorithm. The

algorithm involves dividing the number repeatedly by 10 until it yields a number greater than 10. The number of times divided gives the length of the number. If the number field includes a decimal digit also, then the mantissa part and decimal part are separated. A similar check is made to the decimal part also.

The data types supported in the database are character data type and number data type (it includes whole number as well as fraction). In case if the value of field exceeds the value than it is supposed to take, then an error is thrown. The available system can be modified slightly to accommodate other data types as well.

The second check that is carried out before inserting the value is the key constraint. The constraint that is supported by this system is primary alone. In implementing the primary constraint, first meta file is read. The contents of the meta file are placed in a structure. For each attribute a structure member is dedicated, containing attribute name, data type of attribute, size of the attribute (this includes the size1 and size2 parameters, for character type data the size2 field is always taken as 0, for integer it is either 0 or if there is a decimal part then it contains the length of the decimal part). Since different data types are involved in storing the contents of metafile, the use of structure is justified.

In implementing the constraint, after reading the contents of metafile, the constraint is stored in the structure member. For all non-primary keys the letter 'o' is used to identify that it. In inserting tuples into records, 3 cases are considered. In first case, the steps involved in carrying out the insert operation when the table contains no record, is addressed. In the second case the steps involved in carrying insert operation, when one of the fields is a primary key and the table contains at least more than one record in it are addressed. In the third case the steps involved in carrying insert operation, when none of the fields in the table are primary.

INSERT PROCEDURE

A separate variable is allocated to identify the primary key in the table. In case if the table does not contain any primary key, this variable is assigned an arbitrary number (in this case it is 777). In case the table contains primary key, the field number is stored in the variable.

When the table does not contain any record, the tuples are inserted into the file created by the create procedure. Only the length constraints are checked in this case. When inserting, the tuple a “#” symbol is associated with the last field to identify the end of the record. In this case, the record is not checked to find out whether it contains a primary key or not, since regardless of the case the record is going to be the first record to be inserted.

When the table contains a primary key constraint and there is more than one record present in the table already, then the tuple to be inserted is checked for all the key constraints before entering. The file containing the records is first opened, then the field corresponding to the primary key is extracted from the first record, and it is compared with the value the user has given. If the field values match then the unique constraint is violated, so an error is thrown indicating the error. In case if the field values are different, then the user input is compared with the subsequent records present in the table. If the record input by the user is a valid one then the record is appended to the table.

In the third case, when the table contains no primary key and if there is more than one record in the table then, the key constraint values are not required to be checked, so the records are inserted into the table directly. The records input by the user are simply appended in the table; the only check performed in this case is checking whether the length of the field is within the permitted value. The value of the variable that holds the field number of primary key in this case will be 777, so the constraints are not checked. Since only primary constraint is implemented here, all the other fields are considered as ordinary fields. Implementation of other constraints is also possible by including procedures for them.

When inserting values into table the fields are separated by blank space.

The client is updated with the result of the insert operation. (Refer figure 4.1.1.9)

SELECT COMMAND

For this query, the file, whose name is the same as that of the table and containing the records of this table, is opened. All the records from this file are fetched and given to the client as the records of the requested table. (Refer figure 4.1.1.10)

UPDATE COMMAND

Here, once the validation of the new entry is done, all the records of the table are updated i.e. the column entry for all the record is rewritten with the new value. Validation here implies verifying if the new value conforms to the data-type and size of that column. Consulting the metafile of that table does this.

The client is notified of the outcome of the query. (Refer figure 4.1.1.11)

DELETE COMMAND

As the 'where-form' of query is not supported, this command functions exactly like the 'truncate' query. (Refer figure 4.1.1.7)

4.1.1.2 NETWORK FUNCTIONALITIES

The server, as the name implies has to take care of requests from multiple clients. Hence it waits at a well-known port for clients to connect and establishes a session with an authenticated client.

4.1.1.2.a CONCURRENT-SERVER FEATURES

The server of this system has been built as a concurrent server –a server that handles multiple clients simultaneously. The client and server communicate by creating sockets and use TCP.

Once a client requests for a session, she is authenticated and then the server process 'forks' an independent child process to handle this client; while it still continues to 'listen' to other new client requests. The forked process handles any further requests from that client. The entire sequence of operations can be illustrated as follows.

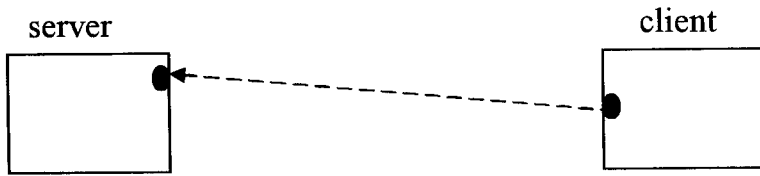


Fig. 4.1.1.2.a

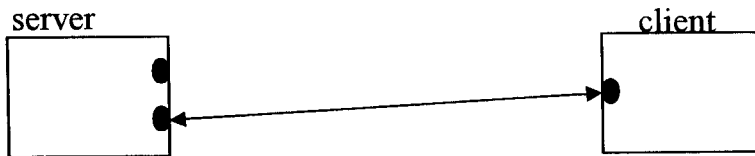


Fig. 4.1.1.2.b

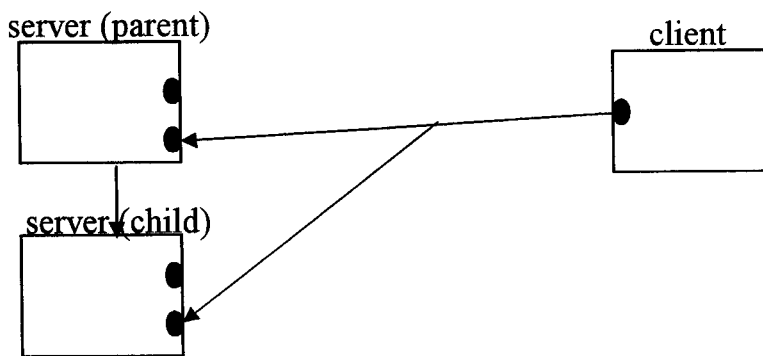


Fig. 4.1.1.2.c

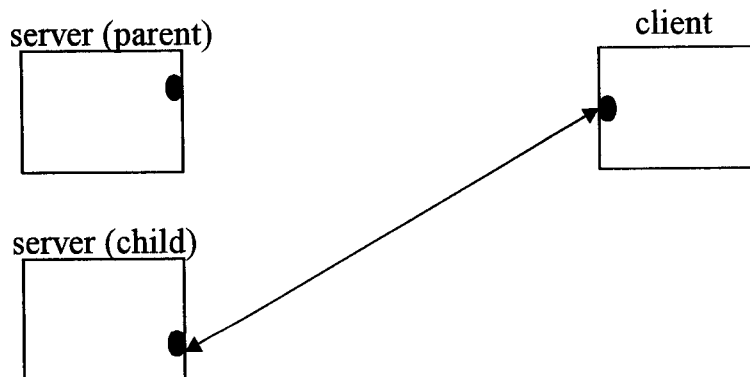


Fig. 4.1.1.2.d

Fig 4.1.1.2.a^[3] –status of client –server before call to accept

4.1.1.2.b^[3] –status of client –server after call to accept

4.1.1.2.c^[3] –status of client –server after fork returns

4.1.1.2.d^[3] –status of client –server after parent and child close appropriate sockets

4.1.2.CLIENT SYSTEM

The client system is the interface with the help of which the user will provide the queries to the server, in order to access the database. The client system is also responsible to provide the login prompt for the user on start-up. Hence - forth the client will only be involved in sending the queries to the server for processing and receiving the results form the server and display them to the client.

The client software is the one that initiates the connection with the server. If the request is accepted, the client will continue to send requests and receive results from the server through the socket it created for itself (as illustrated in figures above).

4.1.3.SERVER ADMINISTRATION

Only these users are allowed access to the database. Even these users are required to prove his authority to use the database, every time he wants to use the database.

AUTHENTICATION

A list of all the users of the database is maintained in a file called 'users'. The status of the user, i.e. whether or not the user is logged on to the system, is also maintained. Any user, who has to use the database, is authenticated by the database system. For this, every user is required to provide a username and password. This username and password are verified against the actual ones and if found matching, the user is permitted to access database. (Refer figure 4.1.1.2)

4.1.4. DATA FLOW DIAGRAM

LEVEL I

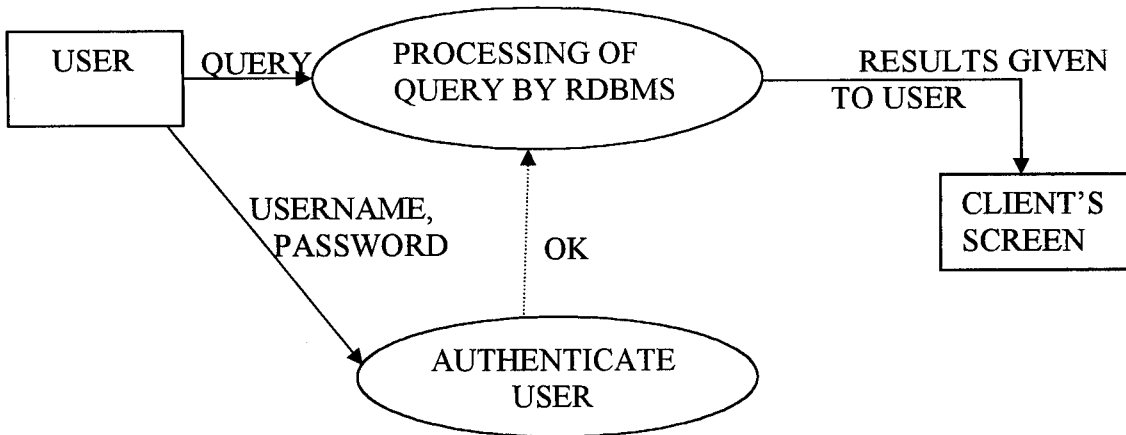


Fig. 4.1.1.1

LEVEL II

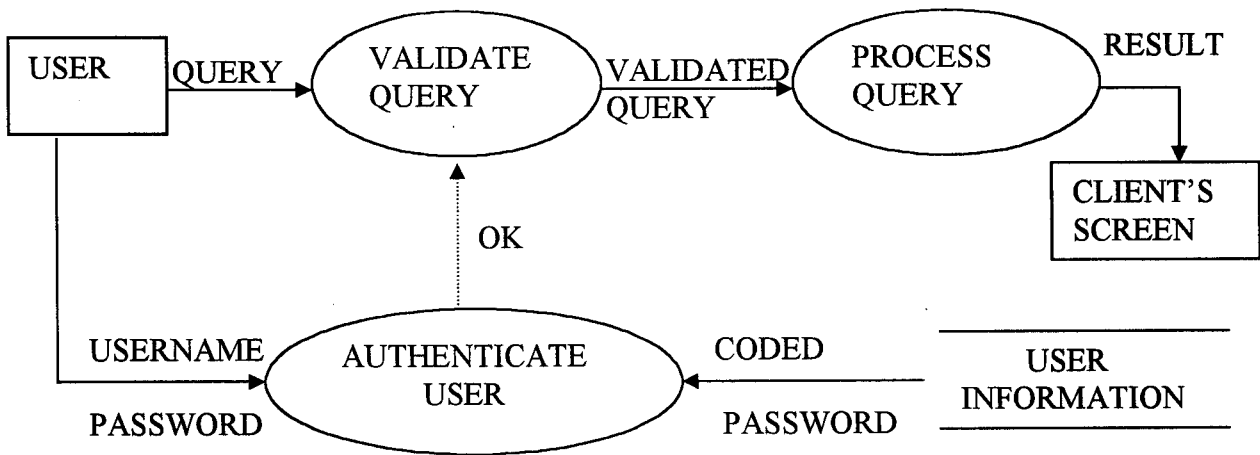


Fig. 4.1.1.2

LEVEL III

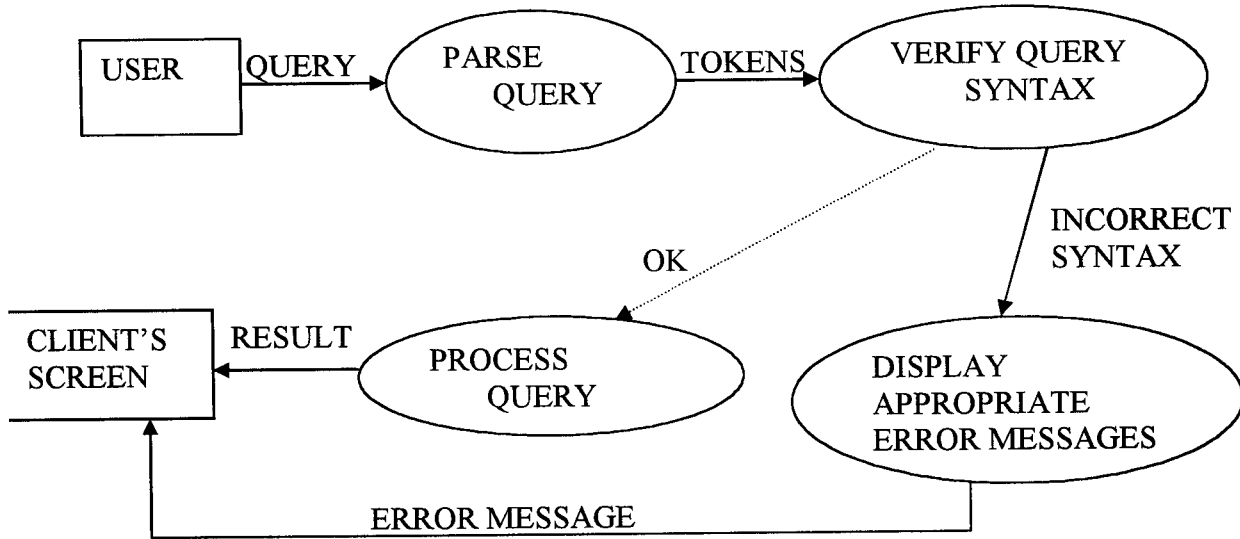


Fig. 4.I.1.3

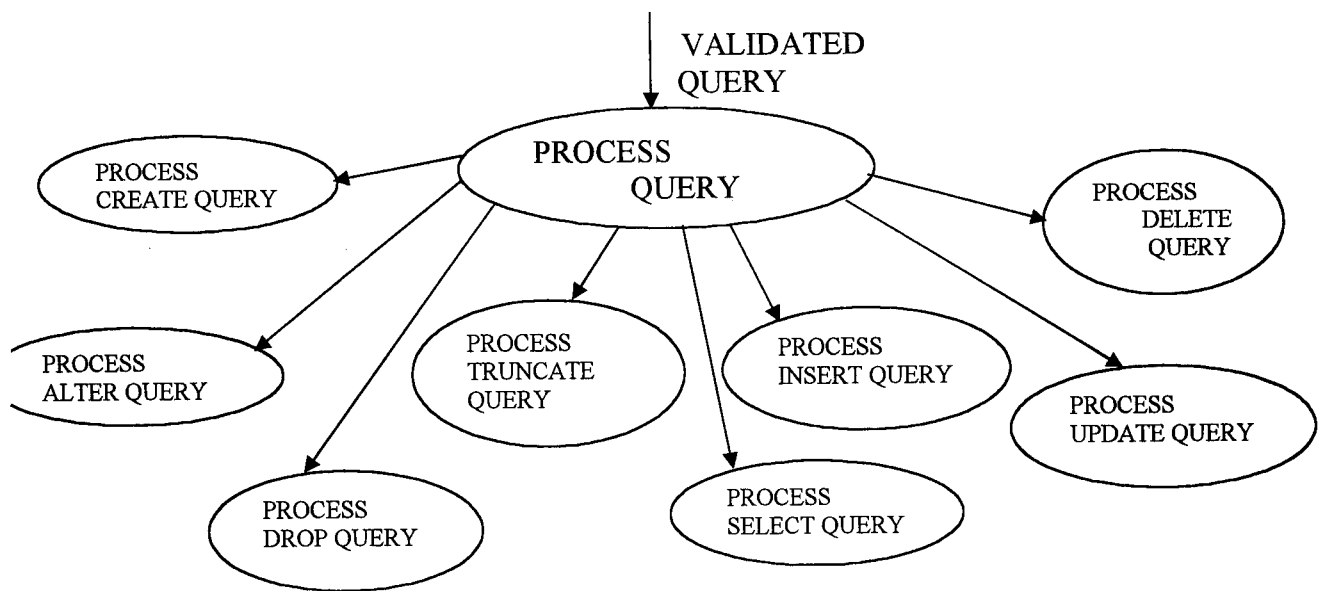


Fig. 4.I.1.4. EXPANSION OF 'PROCESS QUERY' NODE

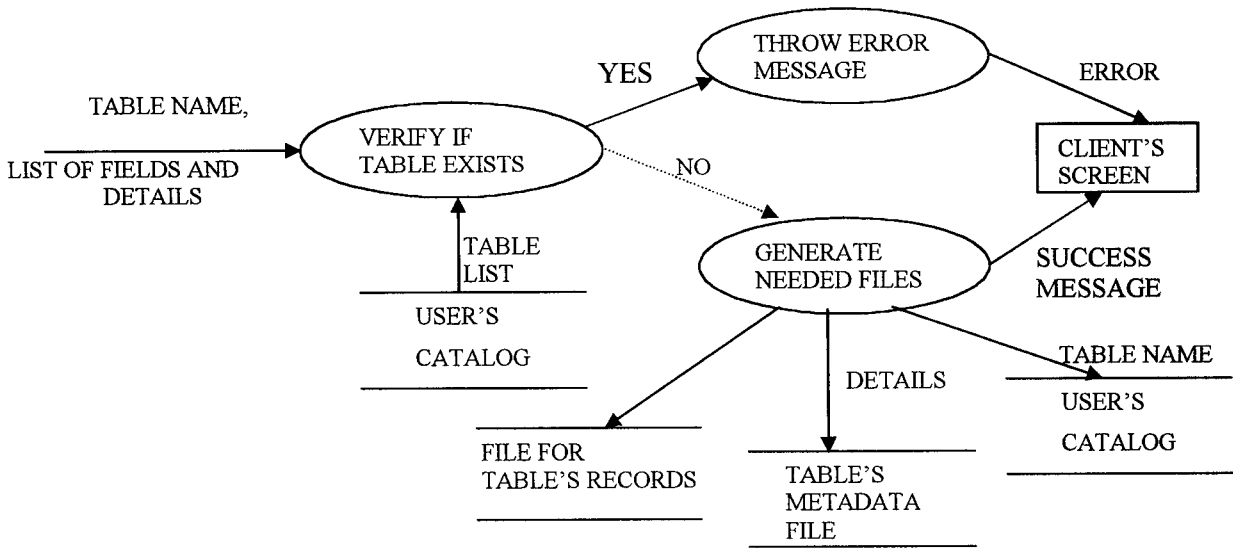


Fig. 4.I.1.5. EXPANSION OF 'PROCESS CREATE QUERY' BUBBLE

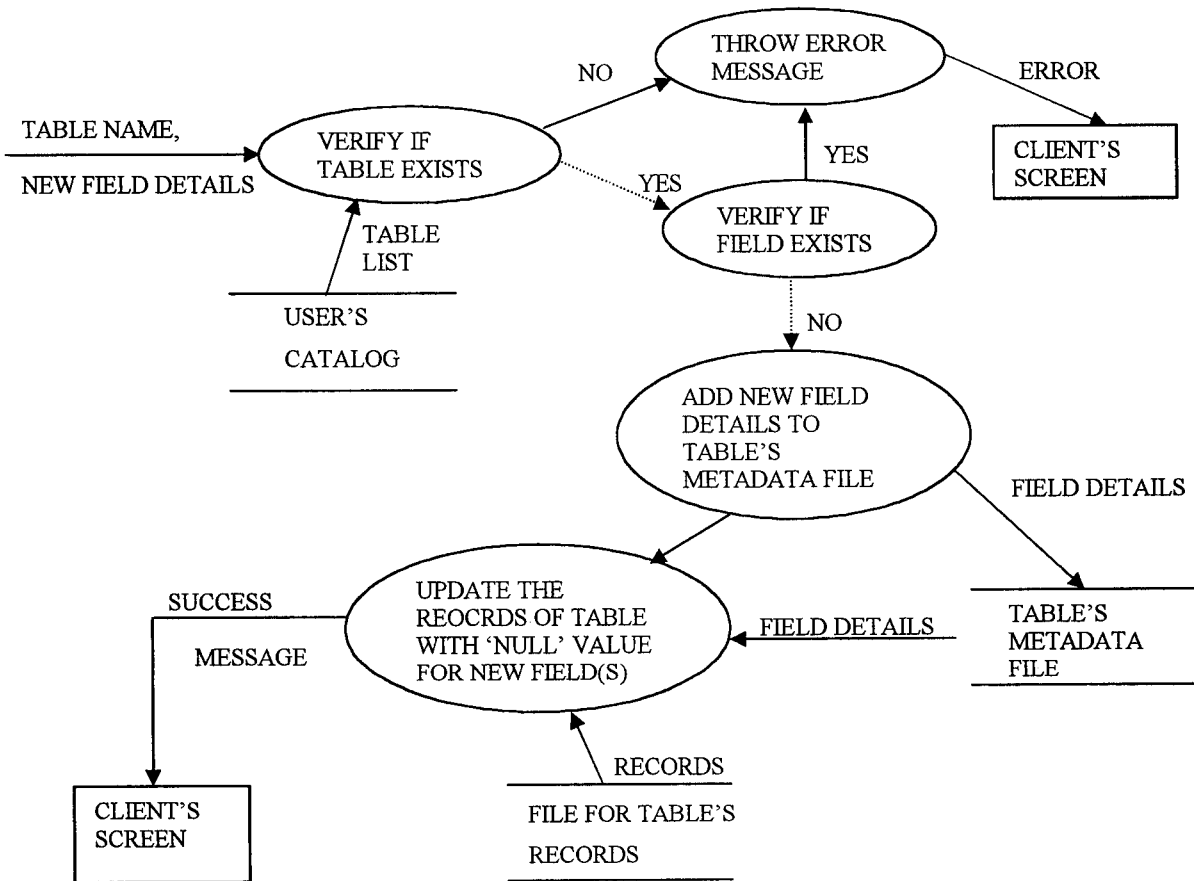


Fig. 4.I.1.6. EXPANSION OF 'PROCESS ALTER QUERY' BUBBLE

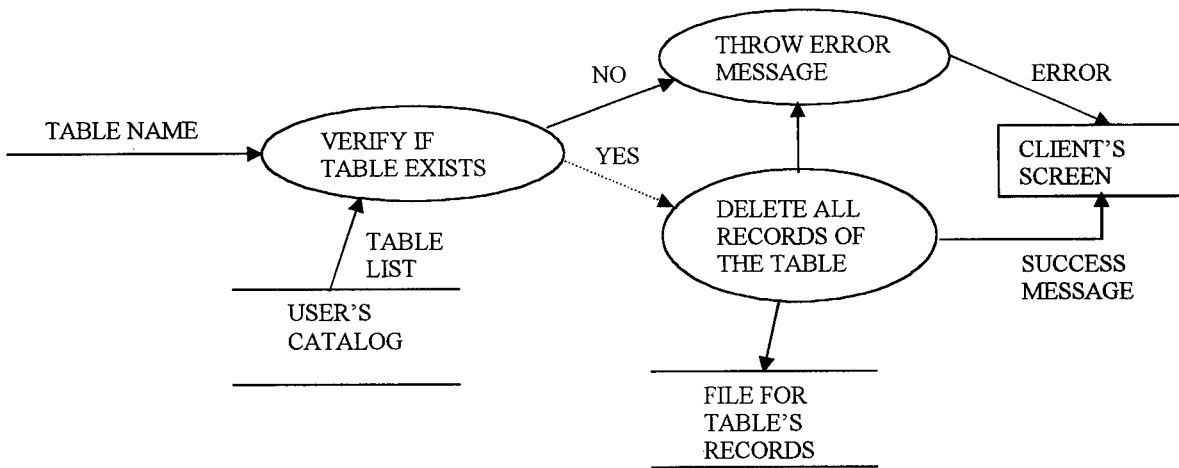


Fig. 4.I.1.7. EXPANSION OF 'PROCESS TRUNCATE QUERY' BUBBLE AND 'PROCESS DELETE QUERY' BUBBLE

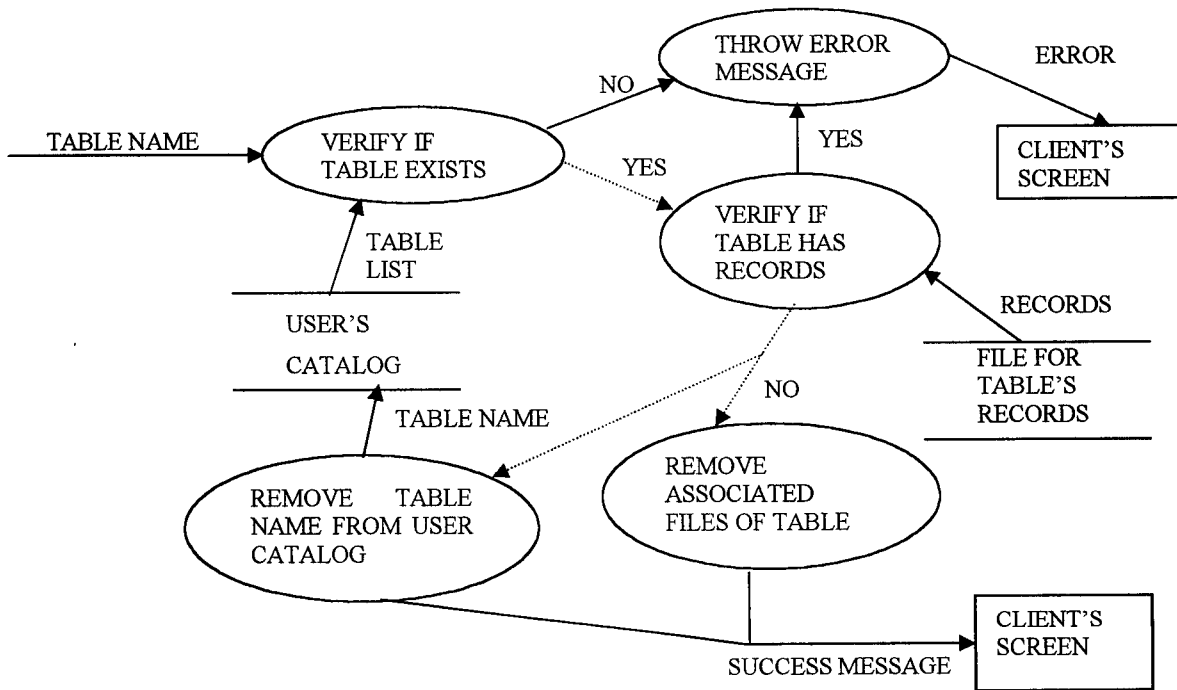


Fig. 4.I.1.8. EXPANSION OF 'PROCESS DROP QUERY' BUBBLE

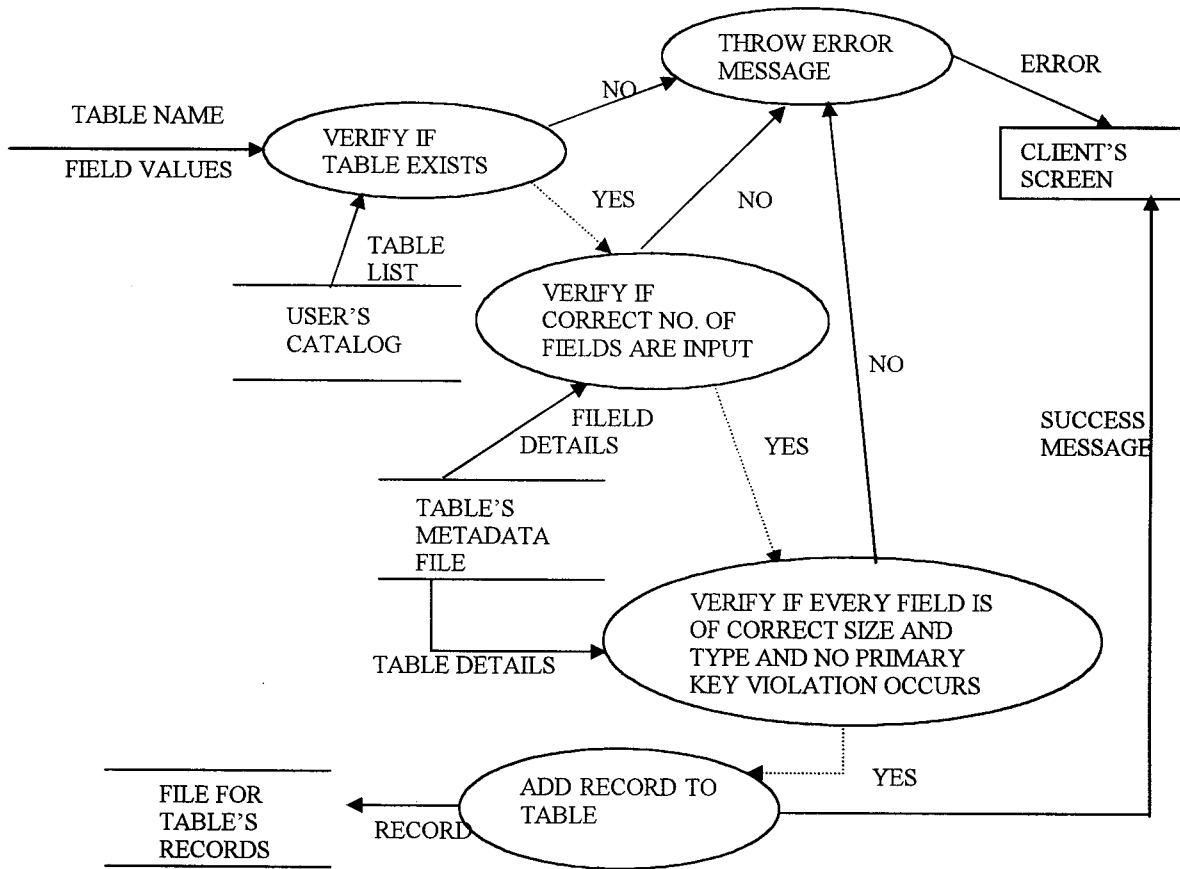


Fig. 4.I.1.9. EXPANSION OF 'PROCESS INSERT QUERY' BUBBLE

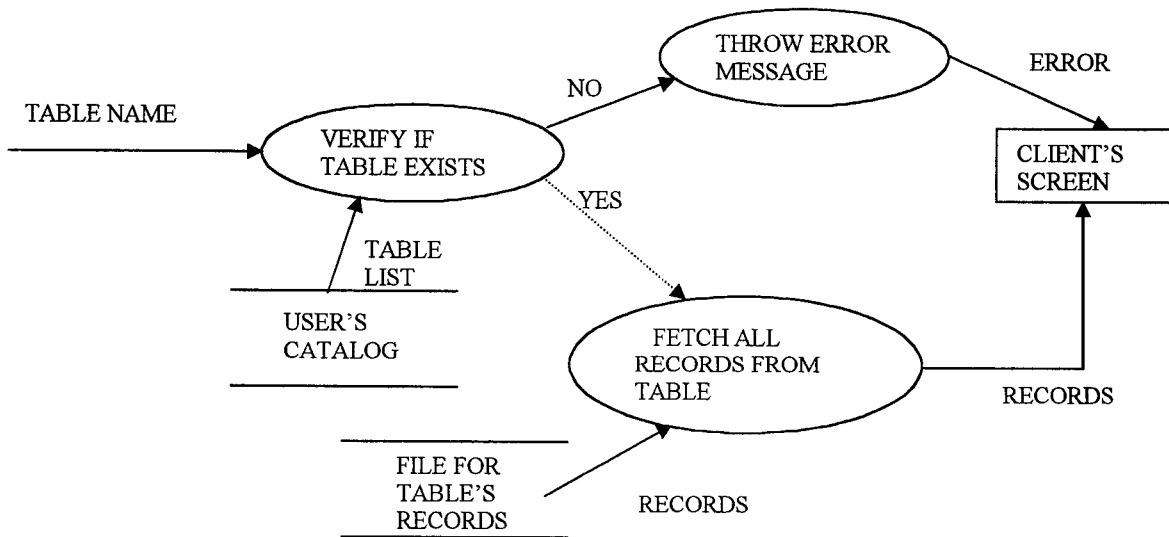


Fig. 4.I.1.10. EXPANSION OF 'PROCESS SELECT QUERY' BUBBLE

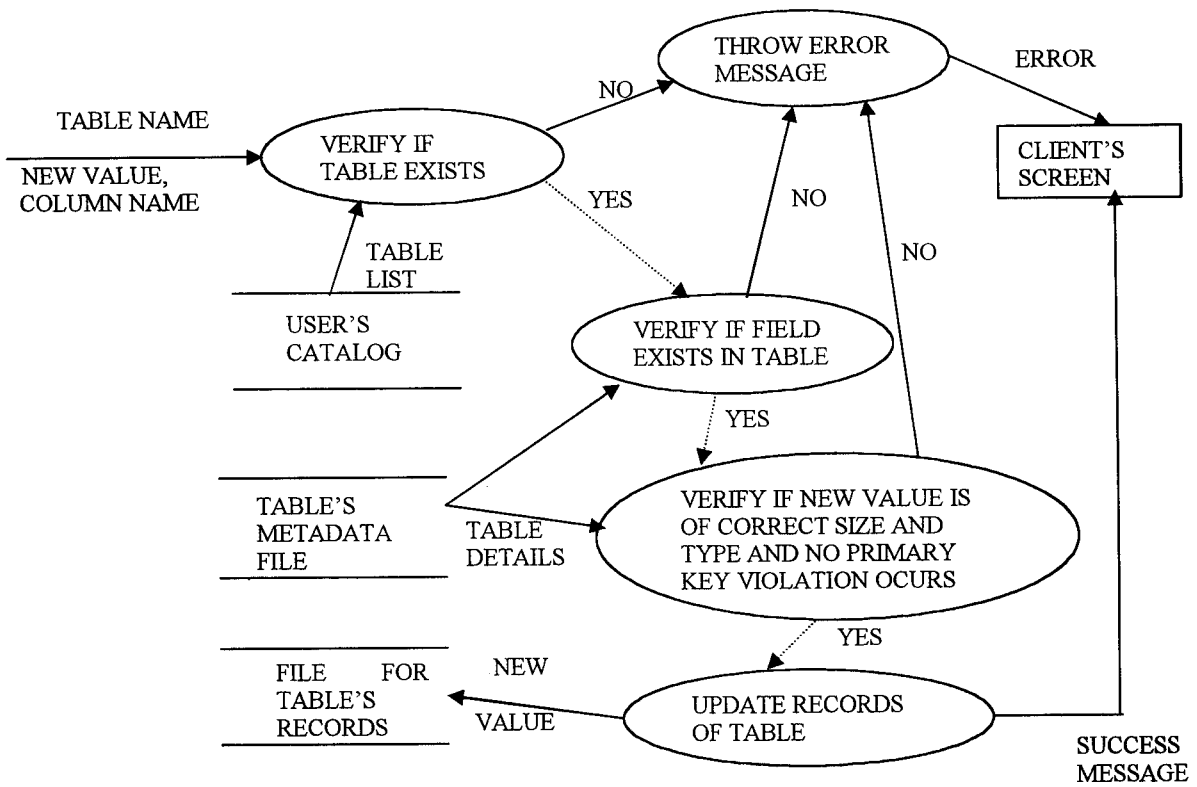


Fig. 4.I.1.11. EXPANSION OF 'PROCESS UPDATE QUERY' BUBBLE

4.II.SYSTEM DEVELOPMENT

This section details how the various subsystems which make-up the entire system were developed.

First the implementation of the RDBMS core subsystem was undertaken. Here, the subsystem that does query validation was the one to be developed first. Once the query grammar was decided upon, after accounting for the features to be supported, this system was developed in such a way that it performs the syntax analysis of the queries.

Next the implementation of the query- processing phase was done. For this, separate procedures were written for each of the command supported, as described in the previous section.

To follow was the integration of these two modules. Given a query the system was now able to analyze the syntax the query and take further steps towards processing it.

Then the development of the client and sever systems was taken-up. First simple server and client software were created. Once the message transfer between the server and client was found to be successful, this subsystem was further enhanced and integrated with RDBMS subsystem. Now server was able to handle client request in the form of query and give suitable response to the client.

Then the procedures to allocate separate space to every client, so that he gets access to only his database, were completed. Adding procedures for client-authorization furthered the system.

The system was in a perfectly working condition, with all the modules complete.

Product Testing

5.PRODUCT TESTING

5.1. UNIT TESTING

This is the process of testing individual modules for their functional efficiency, performance. This testing is performed prior to integration of the unit into a larger system. It allows the unit to be tested in greater detail than will be possible than when the unit is integrated into an evolving software product. All the modules of this system have undergone intensive testing.

5.1.a. QUERY VALIDATION MODULE

In this module several tests have been done to ensure that the query validation is error free.

- ❖ The query is first analyzed to find if it has one of the allowable commands namely – create, alter, truncate, drop, select, insert, update, delete. Else error is thrown saying it is an unknown command.
- ❖ The query at every stage is verified to see if it is complete, before any further analysis is done; else error is thrown saying incomplete query
- ❖ Since the system supports the two data-types namely character and number, the create query will test if any other invalid data-type is specified
- ❖ The table names and column names are allowed to be of maximum 16 character wide; these names can contain letters, digits and '_'; but the first character is always a letter. Procedures have been implemented successfully to notify any violations of the above-mentioned cases and they have been tested too.
- ❖ Extensive numbers of queries have been analyzed for syntax, in order to ensure that this sub-subsystem functions properly.

5.1.b. QUERY PROCESSING

The query processing stage implements the actual functions of the DBMS and hence has undergone many test procedures. The tests undertaken for the implementation of each of the command is listed.

- ❖ Tests have ensured that the procedures verify the existence of the specified table in the user's area before performing any operation; in the case of 'create' command, it has been ensured that only unique table names will be accepted.
- ❖ The number data-type supports both whole numbers and floating point numbers; the procedures test that, during insertion of records into the table, these numbers are well below the specified range; else they are rounded off
- ❖ Tests have ascertained that the specification or non-specification of the primary key constraint on a table's column does affect its manipulation
- ❖ It is checked that the user gives the exact number of field values when inserting a record into a table; else error is thrown on the inadequacy or excess number of arguments
- ❖ Notification of any primary key violation has been ensured
- ❖ Procedures have been made to ensure that the table to be dropped contains no records and that the user is notified about the same when it does; also tests have guaranteed that the associated files are removed
- ❖ In the procedures for update, tests have been performed, to make sure that the update operation on the given column of a table does not violate the primary key constraint

5.1.c. MODULE ON NETWORK FUNCTIONALITIES

- ❖ All errors arising out of improper reception or dispatch of messages- both requests and results- have been handled at the client and server side
- ❖ Cases of client being unable to connect to server, or errors in server accepting the client call have been notified

5.1.d. AUTHENTICATION

This module has been extensively tested in order to ensure that the authentication of clients works correctly

- ❖ Whether a user is able to log on to the server, when another client is already logged in the same name, has been verified; this is of vital importance since cases of concurrency control then comes into picture and is currently unavailable for the system
- ❖ Attempts of invalid password usage are successfully notified to client
- ❖ Tests have been done to make sure that the status of a client, whether or not he is currently logged-in, is successfully kept track of; whenever a user who is currently maintaining a session exits, his status entry is updated;

5.2 INTEGRATION TESTING

The modules in a subsystem communicate with each other through well-defined interfaces. The primary purpose of subsystem testing is to verify the operation of the interfaces between the modules in the subsystem.

- ❖ The integration of the query validation and query processing modules was done successfully and the tests were done to ensure that the interfaces between these two modules worked well; this was done for every possible query and made sure that error handling was still available
- ❖ The integration of these two modules with the networking functions was most importantly tested repeatedly for any errors
- ❖ The values of global variables were tested at many points to see if they contained the desired values

Future Enhancements

6.FUTURE ENHANCEMENTS

The designed system is in its novice stage and can be greatly enhanced to provide useful functionalities.

- As put-in before, the facility to manipulate individual records will improve the functionality of the system multi-fold
- Indexing features can be introduced to introduce and improve searching and sorting functions on the records
- Constraints like 'check', 'not-null' can be supported
- Implementation of the data storage methods can be improved to facilitate access and search procedures
- Support for mathematical functions like sum, average on columns can be brought in

Conclusion

7.CONCLUSION

The database system built is a successful simulation of a typical RDBMS. As described throughout the document, this database system supports the basic, yet the essential requirements of any RDBMS. The implementation of the DDL queries and the DML queries was successfully done.

As stated in the previous section, implementing procedures for handling individual records of a table will enhance this system to a great extent and transform it into a commercially viable system.

This RDBMS system, with aforesaid improvements, can then be exploited in building database of small and medium size. As database has become indispensable in any field of operation, this system might be of much higher value, when, handling reasonable amount of data with much simpler but vital procedures comes into consideration.

The system development has also furthered our interests in this field and served the sole purpose for which the project was undertaken.

References

8.REFERENCES

- [1] Garcia-Molina, H., Ullman, J., Widom, J., '*Database System Implementation*', Prentice Hall International Editions, 2000
- [2] Elmasri, R., Navathe, S., '*Fundamentals of Database Systems*', Pearson Education Asia, 2002 Indian Reprint
- [3] Stevens, R., '*UNIX Network Programming*', Volume I-*Networking APIs: Sockets And XTI*, Prentice Hall of India, 1998
- [4] Comer, D., Stevens, D., '*Internetworking With TCP/IP*', Volume III- *Client-Server Programming And Applications*, Pearson Education Asia, 2001.

Appendix

9. APPENDIX

9.1. SAMPLE SOURCE CODE

```
/*module that implements the following queries---create, alter, drop, truncate, update and
*delete */
```

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

```
int i;
FILE *fcat,*fnewtab,*ftabdet,*ftemp;
char str[20],meta[20]="meta";

/*    updates all rows inthe table    */
int update(char *uname,char *tname,char *cname,char * value)
{
FILE *fmeta,*ftemp,*ftable;

int tab,cnt=0,p_keyno=-1;
int para_cnt,i=0,valid_size;
float num,temp;
int integer,decimal, strlen,ch_len;
int size1,size2,column_no,flag=0;

char *tablename;
char *fld1,meta[20],*intsize,fieldsub[40];
char *colname,*coltype,*s1,*s2,*constraint,*str;

colname=(char *)calloc(1,30);
coltype=(char *)calloc(1,30);
s1=(char *)calloc(1,30);
s2=(char *)calloc(1,30);
constraint=(char *)calloc(1,30);
str=(char *)calloc(1,30);

printf("In update function\n\n");

tab=locate(uname,tname);
if(tab==-1)
{
    printf("Table does not exist\n");
    //exit(1);
    return(201);
}

tablename=(char *)calloc(1,30);
```

```

strcpy(tabname, "meta");
printf("%s\n", tabname);
strcat(tabname, tname);
printf("tabname:%s\n", tabname);

fmeta=fopen(tabname, "r"); /* opening meta file for the table*/
if(fmeta==NULL)
{
    printf("File open error\n");
    return(1000);
}
printf("File opened\n");
while((fscanf(fmeta, "%s", colname)!=EOF))
{
    printf("%s\n", colname);
    if(strcasecmp(colname, cname)==0)
    {
        fscanf(fmeta, "%s", coltype);
        printf("col type%s\n", coltype);

        fscanf(fmeta, "%s", s1);
        size1=atoi(s1);
        printf("size1%d\n", size1);

        fscanf(fmeta, "%s", s2);
        size2=atoi(s2);
        printf("size2%d\n", size2);

        fscanf(fmeta, "%s", constraint);
        printf("cons val is %s\n", constraint);
        if(strcmp(constraint, "p")==0)
        {
            p_keyno=cnt;
            printf("updating column of all records of the table violates primary
key constraint\n");
            return(224);
        }
        column_no=cnt;
        flag=1;
    }
    else
    {
        bzero(str, sizeof(str));
        fgets(str, 100, fmeta);
    }
}

```

```

        cnt++;
    }
    if(flag==0) /* such a column has not been found in the table */
    {
        printf("Column not found\n"); return(225);
    }

printf("the p_key no is %d\n",p_keyno);
fclose(fmeta);

printf("VALUE:%s\n",value);
if(strcmp(coltype,"char")==0)
{
    ch_len=strlen(value);
    if(ch_len>size1)
    {
        printf("field %d longer",i+1);
        return(221);
    }

    //printf("this is test\n");
}
else if(strcmp(coltype,"number")==0)
{
    valid_size=is_a_number(value);
    if(is_a_number(value)<0)
    {
        printf("invalid number\n");
        return(222);
    }
    num=atof(value);
    integer=num;
    temp=num-integer;
    intsize=strtok(value,".");
    //printf("integer is %d\n",integer);
    //printf("%d\n",(10^(fldptr[i].size1-fldptr[i].size2)));
    if(strlen(intsize)>(size1-size2))
    {
        printf("integer part too long\n");
        return(221);
    }
    if(temp!=0)
    {
        intsize=strtok(value,".");

```

```

        value=value+(strlen(intsize)+1);

        if(strlen(value)>size2)
        {
            printf("decimal part is large\n");
            return(221);
        }
    }
}

ftemp=fopen("temp","w");
if(ftemp==NULL)
{
    printf("Error in file open\n"); exit(1);
}

ftable=fopen(tname,"r");
if(ftable==NULL)
{
    printf("table records---file open error\n");
    exit(1);
}

fld1=(char *)calloc(1,1000);
bzero(fld1,sizeof(fld1));
bzero(str,sizeof(str));
printf("GOING TO UPDATE RECORDS\n");
printf("TOTAL NO OF FIELDS IN TABLE:%d\n",cnt);

if(fscanf(ftable,"%s#",fld1)!=EOF)
{
    printf("RECORD:%s\n",fld1);
    do
    {
        for(i=0;i<cnt;i++)
        {
            bzero(str,sizeof(str));
            str=strtok(fld1,",");
            printf("FIELD:%s\n",str);
            fld1+=strlen(str)+1;
            /*if((i==column_no)&& (cnt==1))
                fprintf(ftable,"%s#",value);
            */
        }
    }
}

```

```

        if( (i==column_no) && (cnt!=(i+1)))
        {
            fprintf(ftemp,"%s,",value);
            continue;
        }
        if((i==column_no)&& (cnt==(i+1)))
        {
            fprintf(ftemp,"%s#\n",value);
            continue;
        }
        if((i+1)!=cnt)
            fprintf(ftemp,"%s",str);
        else
            fprintf(ftemp,"%s\n",str);
    }
    bzero(fld1,sizeof(fld1));
}
while(fscanf(ftable,"%s#",fld1)!=EOF);
}

else
{
    printf("Table has no records\n");return(226);
}

fclose(ftemp);
fclose(ftable);

remove(tname);
rename("temp",tname);
return(322);

}

```

```

/*    function
 *    name:        alter_add
 *    handles:     altering schema, ie. adding new fields to the schema
 *    parameters:
 *                uname---username
 *                tname---table name
 *                *newflds--array of structure containing
 *                description of the new fields
 *                cnt---a count of the number of fields added
 *
 */

```

```

int alter_add(char *uname, char *tname, struct fldesc *newflds, int cnt)
{

```

```

    int tab, i;
    char *tabname;
    printf("In alter function\n\n");

```

```

    tab=locate(uname, tname);
    if(tab==-1)
    {
        printf("Table does not exist\n");
        return(201);
    }

```

```

    tabname=(char *)calloc( sizeof(char), ( strlen(tname)+4 ) );
    strcpy(tabname, "meta");
    printf("%s\n", tabname);
    strcat(tabname, tname);
    printf("tabname: %s\n", tabname);

```

```

    ftemp=fopen(tabname, "a");
    if(ftemp==NULL)
    {
        printf("File open error\n");
        exit(1);
    }

```

```

        printf("File opened\n");

```

```

    for(i=0; i<=cnt; i++)
    {
        printf("count: %d\n\n", i);
    }

```



```

        fprintf(stdin,"%s %s %d %d
%c\n",newflds[i].colname,newflds[i].coltype,newflds[i].size1,newflds[i].size2,newflds[i].cons);
        fprintf(ftemp,"%s %s %d %d
%c\n",newflds[i].colname,newflds[i].coltype,newflds[i].size1,newflds[i].size2,newflds[i].cons);
    }
    fprintf(ftemp,"%s", "\n");
    fclose(ftemp);

return(312);
}

/*      truncate fn. Handled */
int truncat(char * uname,char *tname)
{

int tab;
tab=locate(uname,tname);
if(tab==-1)
    {
        printf("Table does not exist\n");
        return(201);
    }
ftemp=fopen(tname,"w");
fclose(ftemp);
printf("Table truncated\n");
return(313);

}

/*      this is to verify if a table exists in the user catalog      */
int locate(char *uname,char *tname)
{
int val;
fcatt=fopen("catalog","r+");
if(fcatt==NULL)
    {
        printf("error in file open");
        exit(1);
    }
while(fscanf(fcatt,"%s",str)!=EOF)
{
    //printf("%d\n",ftell(fcatt));
    if(strcasecmp(str,tname)==0)
    {
        val=ftell(fcatt)-strlen(str)-1;
        if(val==-1)

```

```

        val=0;
        fclose(fcat);
        return (val);
    }
}
fclose(fcat);
return -1;
}

/* implements drop command */
int drop(char *uname, char *tname)
{
    int rt;
    char s1[20], strtemp[20];
    rt=locate(uname, tname);
    if(rt==-1)
    {
        printf("table doesnt exist\n");
        return(201);
    }

    /* verify if table has records before dropping it */
    ftemp=fopen(tname, "r");
    if( fscanf(ftemp, "%s", strtemp)!=EOF)
    {
        fclose(ftemp);
        printf("Table contains records\n");
        return(212);
    }
    fclose(ftemp);

    /* removing catalog entry for the table */
    ftemp=fopen("temp", "w");
    fcat=fopen("catalog", "r+");
    printf("%d\n", rt);

    while(ftell(fcat)!=rt)
    {
        fscanf(fcat, "%s", s1);
        fprintf(ftemp, "%s\n", s1);
    }
    printf("%d\n", fseek(fcat, strlen(tname)+1, SEEK_CUR));
    while(fscanf(fcat, "%s", s1)!=EOF)
    {
        fprintf(ftemp, "%s\n", s1);
    }
}

```

```

    }
    fclose(ftemp);
    fclose(fcat);
    rename("temp","catalog");

    /*    removing metafile    */
    remove( strcat(meta,tname));

    /*    removing file for table    */
    remove(tname);

    return(314);
}

/*    creating a table and associated files is done here    */
int create(char *uname,char *tname,struct fldesc *g,int cnt)
{

int k=0;

fcat=fopen("catalog","a+");
if( fcat==NULL)
{
    printf("\nError in file open\n");
    exit(1);
}
rewind(fcat);
bzero(str,sizeof(str));
while( (fscanf(fcat,"%s",str))!=EOF)
{
    if( (strcasecmp(str,tname))==0)
    {
        printf("\nTable already exists");
        return(210);
    }
    bzero(str,sizeof(str));
}
fclose(fcat);

/*    verify that the fields in the table are unique    */
for(i=0;i<=cnt;i++)
{
    for(k=i+1;k<=cnt;k++)
        if( strcasecmp(g[i].colname,g[k].colname)==0)
            return(211);
}
}

```

```

fcat=fopen("catalog","a");
if(fcat==NULL)
{
    printf("Error in file open:catalog\n");
    exit(1);
}
fprintf(fcat,"%s\n",tname);
fclose(fcat);

fnewtab=fopen(tname,"w+");
if(fnewtab==NULL)
{
    printf("Error in file open:tname\n");
    exit(1);
}
fclose(fnewtab);

bzero(meta,sizeof(meta));
strcpy(meta,"meta");
strcat(meta,tname);
ftabdet=fopen(meta,"w");
if(ftabdet==NULL)
{
    printf("\nError in file open:metafile\n");
    exit(1);
}
//fprintf(ftabdet,"%s\n",tname);

for(i=0;i<=cnt;i++)
{
    fprintf(ftabdet,"%s %s %d %d
%c\n",g[i].colname,g[i].coltype,g[i].size1,g[i].size2,g[i].cons);
}
fclose(ftabdet);

return(311);
}

```

9.2 SAMPLE QUERIES

LOGIN PROMPT FOR USER:

User name : abcde

Password :

Output:

Successfully logged in

(assuming correct login the command prompt is displayed)

CREATE COMMAND:

Command:

```
Create table school(name char(10) primary, age number(3));
```

Output:

Table created

Command:

```
Create table school
```

Output:

Incomplete query

Command:

```
create table school(name char(10));
```

output:

Table already exists

Command:

```
Create table 12led(name char(10));
```

Output:

Invalid table name

Command:

```
Create table student(325ads char(10));
```

Output:

Invalid column name

Command :

```
Create table hostel(name char());
```

Output:

Invalid column size

INSERT COMMAND

Command:

```
Insert into school values(abcd,15);
```

Output:

1 row inserted

Command:

```
Insert into school values(abcd,15)
```

Output:

Command not properly terminated.

Command:

```
Insert into school values(abdcd);
```

Output:

Number of fields input do not match that in table

Command:

```
Insert into school values(
```

Output:

Query incomplete

Command:

Truncate table

Output:

Query incomplete

LOGGING OUT

Command:

Exit

Output:

INCORRECT LOGIN

User name :abcd

Pass word:

Output:

Invalid password (if pass word has been typed wrongly)

CREATION OF NEW USER

User name: efgh

Pass word:

Output:

User name not found

User name:efgh

Pass word:

Output:

New user created