A Report on

Procurement tracking system

Submitted by

Aparna PL (14IT132)

Neha B (14IT224)

Prerana K R (14IT231)

S S Karan (14IT252)

V Sem B.Tech (IT)

in partial fulfilment of the course

of

Database systems (IT301)

under the guidance of

Ms J R Shruti (Asst. Lecturer)

as a part of

B. Tech in Information Technology

At

Department of Information Technology

National Institute of Technology Karnataka, Surathkal.

November 15, 2016

CERTIFICATE

This is to certify that the project entitled **"Procurement tracking system"** is a bona fide work carried out as part of the course **Database systems (IT301)**, under my guidance by

1. Aparna PL 14IT132

2. Neha B 14IT224

3. Prerana K R 14IT231

4. S S Karan 14IT252

students of V Sem B.Tech (IT) at the Department of Information Technology, National Institute of Technology Karnataka, Surathkal, during the academic semester Jul-Dec 2016 in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Information Technology, at NITK Surathkal.

Place: Surathkal

Date: 15 Nov, 2016

Signature of the Instructor

Signature of Examiner

ABSTRACT

The aim of this project is to develop a Procurement tracking system for an organization. As there are number of activities involved in the procurement encompassing many departments, the status of Purchase Requisition changes dynamically. In order to expedite the priority purchase requisitions and to estimate the time of material receipt, an information system to track the dynamic status of Purchase Requisitions is essential. The present project aims at tracking the dynamic status of Purchase Requisitions in a procurement department.

ACKNOWLEDGEMENT

A successful and satisfactory completion of any significant task is the outcome of invaluable aggregate combination of different people in radial direction explicitly and implicitly. We would therefore take the opportunity to thank and express our gratitude to all those without whom the completion of our project would not be possible.

We extend our thanks to our project advisor and project co-ordinator, Ms J R Shruti, Asst. Lecturer, Department of Information Technology, NITK, Surathkal, for her constant help and support. We express our heartfelt gratitude to our HoD, Dr. Ram Mohan Reddy, Department of Information Technology, NITK, Surathkal, who has extended support, guidance and assistance for the successful completion of the project.

Aparna PL

Neha B

Prerana K R

S S Karan

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

The aim of this project is to develop tracking system for procurement activities in an organization. Procurement system helps in supply chain management function of organizations. Many organizations manage their ERP functions on SAP platform. SAP system helps in tracking purchasing functions from indent (requirement) generation, tendering, purchase order generation, inspection and receipt and usage. However, in any typical public procurement setup, where the system is not fully paperless due to audit/record requirements, there are many activities which are not captured in SAP system e.g., which purchase officer is dealing with the indent, when the file is sent to Technical department for evaluation, when the file is sent to finance department for concurrence etc. This limitation leads to opacity in procurement functions carried out in organization. The present system exactly addresses these gaps and fulfils the transparency requirement and also enhances efficiency.

Users of this system include an administrator, Procurement head, Bid dealing officers and service/maintenance/technical(SMT) groups (Mechanical, Instrumentation, HR, IT etc.). Procurement department receives indent (request for procurement) from various maintenance groups like mechanical, Instrumentation, civil, electrical, Safety etc. Maintenance users are required to login to view the state of their indent. The procurement head assigns the Purchase requisition (indent) to one of the dealing officers in procurement department to process the case depending on the current number of cases being handled by the DO (Dealing Officer). Dealing officer floats a tender for the PR and updates status of the bid in the database. At any point of time, SMT users will be able to check the status of their PR and would be notified in case of change of status. Subsequent activities like receipt of offers from vendors, file being sent to SMT group for technical evaluation, receipt after evaluation, status of approval for priced offer, concurrence of finance, placement of Purchase order is tracked in the system. This helps SMT groups to track the various activities being carried out by procurement department and will help in planning their maintenance functions.

1.1.1 ACTORS

People who interact with the database-

- Indentors
- Procurement Head
- Dealing Officers

1.1.2 SOME SAMPLE QUERIES

An indentor can-

- Add new Purchase requests
- View Status of existing requests for his department
- Update status of evaluation report (satisfactory/unsatisfactory)

Procurement Head can-

- View Requests made by user groups
- Assign PRs to Dealing officers based on the current number of PRs being handled by each of them

A DO can-

- View all PRs assigned to him
- Add tender
- Add extension dates if required
- Update status of a tender after each step of processing

1.2 PROJECT OBJECTIVES

Typical procurement cycle in a procurement department from receipt of Purchase Requisition to issue of Purchase Order in public sector typically takes 2-4 months depending on type of complexity of procurement and type of tender. As there are number of activities involved in the

procurement encompassing many departments, the status of Purchase Requisition changes dynamically In order to expedite the priority purchase requisitions and to estimate the time of material receipt, an information system to track the dynamic status of Purchase requisitions is essential. The present project aims at tracking the dynamic status of Purchase Requisitions in a procurement department. Users of this information system include Indenting department, Procurement department, Finance department and Top management of the company.

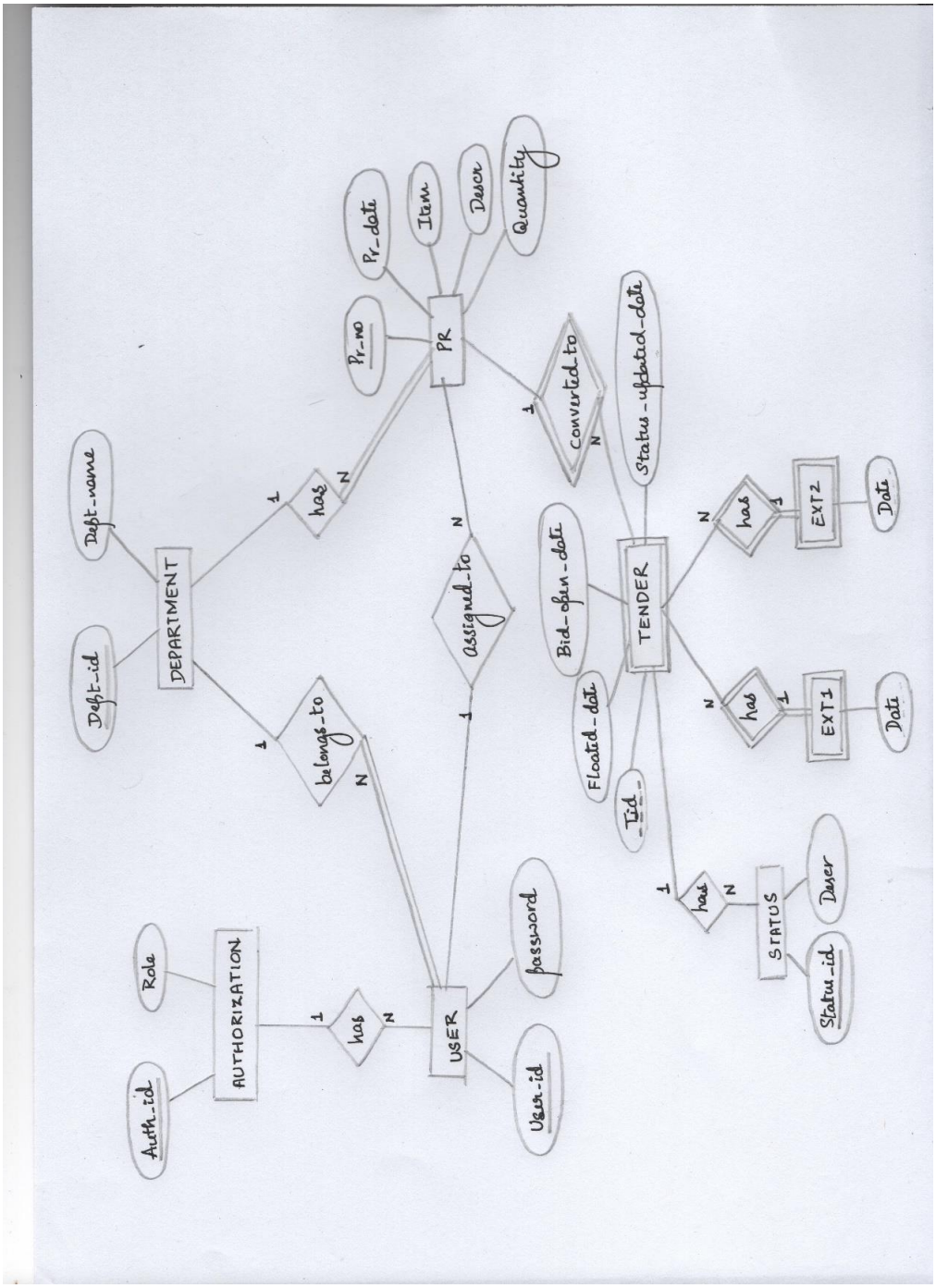
1.3 PROJECT SCOPE

Normally all public sector companies use some platform of ERP like SAP/JD Edwards etc. Major procurement milestones like PR generation date, Tender floating date and issue of Purchase order are available in ERP system. However, there are many inter and intra departmental activities like assigning Dealing officer for each PR, sending the offers for technical evaluations, receipt, sending the file for obtaining permission for opening of Price bids, receipt, obtaining approval for issuing Purchase order etc are still not tracked on ERP system. The project can be developed in existing ERP system itself so that the gaps in information can be bridged and a completely transparent procurement system can be made available to all users in the Organization. The scope of the project covers all activities which are carried out on papers outside the ERP system.

CHAPTER 2

SYSTEM DESIGN

2.1 LOGICAL DESIGN



2.2 SCHEMA DIAGRAM

The transformation from the entity-relationship model to the relational model is very straightforward. A feasible set of relational schemas is as follows.

User

User_id	Password	Dept_id	Auth_id

Dept

Dept_id	Dept_name

Authorization

Auth_id	Role

PR

PR_No	PR_date	Dept	Item	Descr	Quantity	DO

Tender

Tid	Pr_no	Floated_date	Bid_open_date	Status	Status_date

Status

Status_id	Descr

Ext1

T_id	Date

Ext2

T_id	Date

2.3 ADVANCED LOGICAL DESIGN

2.3.1 NORMALIZATION TECHNIQUES

Some queries might have to travel with more than one table based on foreign key by some kind of joining. If we have 100's of table then joining kind of operation will take a lot of time. So, it is undesirable. Alternative solution is keeping a universal or central table having all attributes together. It eases our information retrieval but there can be lots duplication or redundant values. Redundancy is repeated values in a same table, which leads to wastage of space, example, for 100 employees working for a same department, we have to repeat same department information for all 100 employees in a table. Even though we have huge storage capacity at lower costs these days, it results in anomalies.

Anomalies is a kind of inconsistent information and overhead. Database won't show any error and will simply accept the values. It can happen while insert, delete and updating information

1. Insert anomaly: while inserting employee information we might enter wrong department details, which will lead to anomaly.

2. Delete anomaly: there are 100 employees in a department "sales". If we delete all 100 employees in that department then we would lose department details too. There is no other way to extract department details of "sales".

3. Update anomaly: while updating a table we might enter some other details by mistake, which will lead to anomaly.

These are the overheads, when you combine all tables into a centralized one. So dividing the table is inevitable.

Idea is dividing large table into small tables having less number of attributes in such a way that your design reduces anomalies and subsequently degree of redundancy that are present in the table. This systematic procedure is called normalization.

Normalization is carried out having functional dependency and candidate keys in mind. It is a step-wise process that divides relations into several pieces until we eliminate redundancy and anomalies.

There are many steps to achieve this normalization.

1. 1st normal form

2. 2nd normal form

3. 3rd normal form

4. Boyce-codd normal form

What is key in general?

Given a value of an attribute, we must be able to uniquely identify all other attributes given in a table. That is, a complete row also called tuple or record.

A → BC

→This symbols denotes “determines” or “returns”

(group of attributes) → (group of attributes)

LHS is called key and RHS is called values.

Such a relation is called functional dependency (FD).

There are 3 different kinds of FD’s:

1. Trivial Functional Dependency (FD): What is got on RHS is already LHS

Example: A determines itself.

A→A

A→ AB

AB→A

2. Non-trivial FD: given a value of an attribute get a unique value.

Example:

A → B

A → BC

AB → CD

3. Semi non-trivial FD: it is not deriving completely new information. It partially brings a record, that is, not the value of entire set of attributes.

Example:

AB → BC B is common on both sides

ACD → EFCD CD is common both sides

FD's are quite useful in identifying keys, identifying equivalences of FS's and finding minimal

FD set. There are rules applied on FD such as inference rules (reflexive, transitivity, augmentation, union, etc), closure.

Closure – how many attributes you are able to determine by one attribute. It is the base of entire normalization process. Using closure property we can determine candidate keys.

Candidate keys - are key or set of keys that identify all other attributes in a table. A table can have many candidate keys, but at any moment, only one candidate key can be as a primary key of a table. If there are n attributes then 2^n-1 candidate keys are possible except null.

Sometimes, candidate key with non-key attribute uniquely determine a table. Such key is called **superkey**.

Minimal super key or candidate key which has less no of attributes is called **primary key**, which uniquely identifies a tuple.

In FD's, LHS must be a key for every table. In BCNF, we have 0% redundancy in table. To achieve this, we go through series of normalization from 1st NF to BCNF. It is not mandatory to go from 1st NF to BCNF. But, it is a convention to follow this sequence.

Every normal form should be lossless, and FD preserved.

1st Normal Form: A relation is said to be in first normal form then it should satisfy the following[2]

- No multi-valued attribute
- No composite attribute
- identify primary key

Here, the relationship is converted to either relation or foreign key or merging relations.

Foreign key: giving primary key of one table as a reference to another table.

Outcome of 1st normalization:

- Primary key has been identified in each table using closure property (minimal super key)
- Composite attributes has been resolved
- Multi-valued attributes has been resolved

User

User_id	Password	Dept_id	Auth_id

Dept

Dept_id	Dept_name

Authorization

Auth_id	Role

PR

PR_No	PR_date	Dept	Item	Descr	Quantity	DO

Tender

Tid	Pr_no	Floated_date	Bid_open_date	Status	Status_updated_date	Extension dates

Status

Status_id	Descr

Here, extension dates are multivalued. Each tender can have at most 2 extension dates. Hence for this relation to satisfy 1NF, we split it as follows-

Ext1

T_id	Date

Ext2

T_id	Date

Also, as DO is a reference to user table, it should not be null. But it might be null before its assignment to a DO. Thus, we split it as-

PR

PR_No	PR_date	Dept	Item	Descr	Quantity

Assign

Pr_no	DO

2nd Normal Form:

(i) Repeating column values are taken out and maintained in a separate table. So that change can be done only once in the new table rather than all entries in the first table[2].

Rule is foreign key must be on the N side else again multi-value in a column will occur.

(ii) Identify prime attribute (part of candidate key that determines anything else), it is also called partial dependency, and eliminate it. Because, 2nd NF is based on Full Functional dependency (key should determine all other attributes in a table)

(iii) Use foreign key on many side

User

User_id	Password	Dept_id	Auth_id

Dept

Dept_id	Dept_name

Authorization

Auth_id	Role

PR

PR_No	PR_date	Dept	Item	Descr	Quantity

Assign

Pr_no	DO

Tender

Tid	Pr_no	Floated_date	Bid_open_date	Status	Status_updtaed_date

Status

Status_id	Descr

Ext1

T_id	Pr_no	Date

Ext2

T_id	Pr_no	Date

3rd Normal Form:

- Only columns with direct dependency of the primary key shall be in the entity.[2]
- No transitive dependencies: non-prime attributes transitively depending on the key. .

Example: A→B→C == A → C.

A → B

B is non-key attribute here

B→C suddenly becomes key attribute here. Because of this, we will get repeated values in a column. Therefore, it should be eliminated.

3rd NF should hold the condition that: if X → Y then,

- Either X is a super key or
- Y is a prime attribute

Following this condition will never allow transitive dependency.

User

User_id	Password	Dept_id	Auth_id

Dept

Dept_id	Dept_name

Authorization

Auth_id	Role

PR

PR_No	PR_date	Dept	Item	Descr	Quantity

Assign

Pr_no	DO

Tender

Tid	Pr_no	Floated_date	Bid_open_date	Status	Status_updtaed_date

Status

Status_id	Descr

Ext1

T_id	Date

Ext2

T_id	Date

Every candidate key in a table determines all other attributes and no non-key attributes determine any attributes in the tables.

2.3.2 GLOBAL SCHEMA

PR	
Attribute name	**Attribute size (in bytes)**
Pr_no	Int(4)
Pr_date	Date(3)
Dept	Int(4)
Item	Char(20)
Descr	Char(50)
Quantity	Int(4)

Tender	
Attribute name	**Attribute size (in bytes)**
T_id	Int(4)
Pr_no	Int(4)
Floated_date	Date(3)
Bid_open_date	Date(3)
Status	Int(4)
Status_update_date	Date(3)

Authorization	
Attribute name	**Attribute size (in bytes)**
Auth_id	Int(4)
Role	Char(20)

Department	
Attribute name	**Attribute size (in bytes)**
Dept_id	Int(4)
Dept_name	Char(20)

Assign	
Attribute name	**Attribute size (in bytes)**
Pr_no	Int(4)
DO	Char(20)

Ext2	
Attribute name	**Attribute size (in bytes)**
T_id	Int(4)
Pr_no	Int(4)
Date	Int(3)

Ext1	
Attribute name	**Attribute size (in bytes)**
T_id	Int(4)
Pr_no	Int(4)
Date	Int(3)

Status	
Attribute name	**Attribute size (in bytes)**
Status_id	Int(4)
Descr	Char(20)

User	
Attribute name	**Attribute size (in bytes)**
User_id	char(20)
Password	char(20)
Dept_id	Int(4)
Auth_id	Int(4)

2.4 QUERIES

2.4.1 CREATION OF TABLES

```
CREATE TABLE `assign` (
  `Pr_no` int(11) NOT NULL,
  `DO` varchar(20) NOT NULL
)
CREATE TABLE `authorisation` (
  `Auth_id` int(11) NOT NULL,
  `Role` varchar(20) NOT NULL
)
```

```
CREATE TABLE `department` (
  `Dept_name` varchar(20) NOT NULL,
  `Dept_id` int(11) NOT NULL
)
CREATE TABLE `ext1` (
  `T_id` int(11) NOT NULL,
  `Date` date NOT NULL
)
CREATE TABLE `ext2` (
  `T_id` int(11) NOT NULL,
  `Date` date NOT NULL)
CREATE TABLE `pr` (
  `Pr_no` int(11) NOT NULL,
  `Pr_date` date NOT NULL,
  `Dept` int(11) NOT NULL,
  `Item` varchar(20) NOT NULL,
  `Descr` varchar(50) NOT NULL,
  `Quantity` int(11) NOT NULL
)
CREATE TABLE `status` (
  `Status_id` int(11) NOT NULL,
  `Descr` text NOT NULL
)
```

```
CREATE TABLE `tender` (
  `T_id` int(11) NOT NULL,
  `Floated_date` date NOT NULL,
  `Bid_open_date` date NOT NULL,
  `Pr_no` int(11) NOT NULL,
  `Status` int(11) NOT NULL,
  `Status_updated_date` date NOT NULL
)
CREATE TABLE `user` (
  `User_id` varchar(20) NOT NULL,
  `Dept_id` int(11) NOT NULL,
  `Auth_id` int(11) NOT NULL,
  `password` varchar(20) NOT NULL
)
```

Adding primary keys

- ALTER TABLE `assign` ADD PRIMARY KEY (`Pr_no`), ADD KEY `DO` (`DO`);
- ALTER TABLE `authorisation` ADD PRIMARY KEY (`Auth_id`);
- ALTER TABLE `department` ADD PRIMARY KEY (`Dept_id`);
- ALTER TABLE `ext1` ADD PRIMARY KEY (`T_id, Pr_no`);
- ALTER TABLE `ext2` ADD PRIMARY KEY (`T_id, Pr_no`);
- ALTER TABLE `pr` ADD PRIMARY KEY (`Pr_no`), ADD KEY `Dept` (`Dept`);
- ALTER TABLE `status` ADD PRIMARY KEY (`Status_id`);
- ALTER TABLE `tender` ADD PRIMARY KEY (`T_id`,`Pr_no`), ADD KEY `Pr_no` (`Status`), ADD KEY `Pr_no_2` (`Pr_no`);

- ALTER TABLE `user` ADD PRIMARY KEY (`User_id`), ADD KEY `Dept_id` (`Dept_id`,`Auth_id`), ADD KEY `Auth_id` (`Auth_id`);

Adding foreign key constraints

- ALTER TABLE `assign` ADD CONSTRAINT `assign_ibfk_1` FOREIGN KEY (`DO`) REFERENCES `user` (`User_id`), ADD CONSTRAINT `assign_ibfk_2` FOREIGN KEY (`Pr_no`) REFERENCES `pr` (`Pr_no`);
- ALTER TABLE `ext1` ADD CONSTRAINT `ext1_ibfk_1` FOREIGN KEY (`T_id`) REFERENCES `tender` (`T_id`) ON DELETE CASCADE;
- ALTER TABLE `ext2` ADD CONSTRAINT `ext2_ibfk_1` FOREIGN KEY (`T_id`) REFERENCES `tender` (`T_id`) ON DELETE CASCADE;
- ALTER TABLE `pr` ADD CONSTRAINT `pr_ibfk_1` FOREIGN KEY (`Dept`) REFERENCES `department` (`Dept_id`);
- ALTER TABLE `tender` ADD CONSTRAINT `tender_ibfk_2` FOREIGN KEY (`Status`) REFERENCES `status` (`Status_id`), ADD CONSTRAINT `tender_ibfk_3` FOREIGN KEY (`Pr_no`) REFERENCES `pr` (`Pr_no`);
- ALTER TABLE `user` ADD CONSTRAINT `user_ibfk_1` FOREIGN KEY (`Dept_id`) REFERENCES `department` (`Dept_id`), ADD CONSTRAINT `user_ibfk_2` FOREIGN KEY (`Auth_id`) REFERENCES `authorisation` (`Auth_id`);

Inserting into assign table

```
INSERT INTO `assign` (`Pr_no`, `DO`) VALUES
(14, 'DO1'),
(15, 'DO1'),
(17, 'DO2'),
(19, 'DO2'),
(16, 'DO3'),
(18, 'DO4');
```

Inserting into authorization table

```
INSERT INTO `authorisation` (`Auth_id`, `Role`) VALUES
(1, 'Admin'),
(2, 'Procurement Head'),
(3, 'Dealing Officer'),
(4, 'Indenter');
```

Inserting into department table

```
INSERT INTO `department` (`Dname`, `Dept_id`) VALUES
('Procurement', 0),
('IT', 1),
('Finance', 2),
('Electrical', 3),
('Mechanical', 4);
```

Inserting extension dates

```
INSERT INTO `ext1` (`T_id`, `Ext_date1`) VALUES
(1, '2016-10-06'),
(2, '2016-12-30'), (3, '2016-12-30');
INSERT INTO `ext2` (`T_id`, `Ext_date2`) VALUES
(1, '2016-12-30'),
(2, '2016-12-30');
```

Inserting into pr table

```
INSERT INTO `pr` (`Pr_no`, `Pr_date`, `Dept`, `Item`, `Descr`, `Quantity`) VALUES
(14, '2016-11-14', 2, 'Laptop', '1Tb hard disk, windows 10, 8gb ram intel core i5', 10),
(15, '2016-11-14', 2, 'Furniture', 'Black office chairs, Teakwood tables', 20),
(16, '2016-11-14', 1, 'Printer', 'Printers', 3),
(17, '2016-11-14', 1, 'Fans ', 'Ceiling fans', 20),
(18, '2016-11-14', 3, 'AC', 'ac', 2),
(19, '2016-11-14', 2, 'Hard disk', '2TB hard disk', 10);
```

Inserting into status table

```
INSERT INTO `status` (`Status_id`, `Descr`) VALUES
(1, 'Pending'),
(2, 'PR Approved'),
(3, 'Tender Floated'),
(4, 'Bid Opening Date Sent'),
(5, 'Sent for Evaluation'),
(6, 'Received after Evaluation'),
(7, 'Feasible'),
(8, 'Sent to Finance for Approval'),
(9, 'Approval from Finance Received'),
(10, 'Approval for Award Sent'),
(11, 'Approval for Award Received'),
(12, 'Purchase Order');
```

Inserting into tender table

```
INSERT INTO `tender` (`T_id`, `Floated_date`, `Bid_open_date`, `Pr_no`, `Status`, `Status_updated_date`) VALUES
(1, '2016-11-14', '2016-11-26', 14, 4, '2016-11-14'),
(1, '2016-11-14', '2016-11-28', 15, 3, '2016-11-14'),
(2, '2016-11-14', '2016-11-30', 14, 3, '2016-11-14'),
(3, '2016-11-14', '2012-12-01', 14, 12, '2016-11-14');
```

Inserting into user table

```
INSERT INTO `user` (`User_id`, `Dept_id`, `Auth_id`, `password`) VALUES
('DO1', 0, 3, 'password'),
('E01', 2, 4, 'password'),
('E15', 2, 4, 'password'),
('PH01', 0, 2, 'password');
```

Queries to be executed

1. Get role of the currently logged in user

SELECT Role FROM user JOIN authorisation ON user.Auth_id=authorisation.Auth_id WHERE user.Auth_id=5;

2. Get status Description of tender for given Pr_no, T_id.

SELECT Descr FROM status WHERE Status_id = (SELECT Status_id FROM tender WHERE Pr_no=5 AND T_id=3);

3. Get total number of indents handled by each DO

"SELECT assign.DO,count(*) as count FROM pr join assign on pr.Pr_no=assign.Pr_no group by DO order by count ");

4. Update status of the current tender

UPDATE tender SET Status_id=5 WHERE T_id=4 AND Pr_no=5;

5. Get all tender details to be displayed to indentor

SELECT Pr_no,T_id, status. Descr, Status_updated_date FROM tender JOIN status ON tender.Status=status.Status_id WHERE Pr_no=5 AND T_id=2;

6. Assign all PRs which have not yet been assigned

SELECT Pr_no from pr where Pr_no NOT IN (SELECT Pr_no FROM assign);

2.4.2 VIEWS

CREATE view pr_view AS SELECT pr.Pr_no, pr.Pr_date, pr.Dept, pr.Item, pr.Descr, pr.Quantity, assign.DO FROM pr JOIN assign ON pr.Pr_no=assign.Pr_no;

2.4.3 PROCEDURES

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost`

PROCEDURE `pr_view`

(IN `flag` INT(11))

begin if (flag<10) then   select * from `pr` where (`pr`.`Pr_no` in (select `tender`.`Pr_no` from `tender` where (`tender`.`Status` <> 12)) or (not(`pr`.`Pr_no` in (select `tender`.`Pr_no` from `tender`)))) ;

else select * from `pr` where (`pr`.`Pr_no` in (select `tender`.`Pr_no` from `tender` where (`tender`.`Status` = 12)));

end if;

end$$

DELIMITER ;
```

2.4.4 TRIGGERS

```
DELIMITER $$

CREATE TRIGGER `status` BEFORE UPDATE ON `tender` FOR EACH ROW begin IF NEW.Status <> OLD.Status THEN  SET NEW.Status_updated_date =(select CURDATE()); END IF; END

$$

DELIMITER ;
```

CHAPTER 3

PHYSICAL DESIGN

3.1 ASSUMPTIONS

Number of tuples in each relation are shown in Table 3.1

Relation	Tuple
assign	1000
authorization	4
department	5
Ext1	2000
Ext2	1500
pr	1000
status	10
tender	3000
user	20

Table 3.1: Assumed number of tuples in each relation

3.2 STORAGE REQUIREMENTS: DISK PARAMETERS

- Avg Seek Time
- Rotational Delay(Latency time)
- Block Transfer Time
- Block pointer size
- Block Size

Following are the assumptions which are considered for storage requirements:

- Fixed length records are considered for all relations.
- The delimiter for each field is length of the field
- Total number of records in respective relations (provided in below table).

- Block size is 1024 bytes.
- Record doesn't span over multiple blocks (this can be achieved by taking floor function during calculating number of records per block to restrict single record doesn't span over blocks).
- Block pointer(Bp) size is 4 bytes
- Average Seek Time(S) is 20 ms irrespective of any site.
- Average Disk rotation time (Latency) Time (L) is 10 ms irrespective of any site.
- Block transfer rate (Tr) is 0.5 ms irrespective of any site.
- **Blocking factor= ceil(Block size / Record size in bytes)**
- **# no of blocks = ceil(# of records/ Blocking factor)**

Table 3.1 shows the above calculated values.

Relation	**# of records**	**Record size in bytes**	**Blocking factor**	**# no. of blocks**
user	20	48	22	1
department	5	24	43	1
authorization	4	24	43	1
pr	1000	85	13	77
assign	1000	24	43	24
tender	3000	21	49	62
status	12	24	43	1
ext1	2000	11	94	22
ext2	1500	11	94	16

Table 3.2: Record size, Blocking Factor, number of blocks for each relation

3.2.1 SPANNED/UNSPANNED RECORDS

Example:

Block size=512 bytes

Size of each record

1. user-48 Bytes=Unspanned (because, Block size is 512 bytes)

2. department-24 Bytes=Unspanned

3. authorisation-24 Bytes=Unspanned

4. pr-85 Bytes=Unspanned

5. assign-24 Bytes=Unspanned

6. tender-21 Bytes=Unspanned

7. status-24 Bytes=Unspanned

8. ext1-11 Bytes=Unspanned

9. ext2-11 Bytes=Unspanned

Spanned means occupying more than one physical block. Un-spanned means occupying only one physical block

3.3 ACCESS METHODS

Considering the assumption we can calculate easily the size of single record (tuple) of every relation with the help of Schema. The above table gives the number of records in each relation, size of each record, blocking factor for a particular block of that relation and number of blocks required to store entire relation.

Having records on secondary storage, if you want to access them faster, then you need indexing. If a database is frequently queried and it is too large then it is supposed to have index to increase performance [1]. There are various indexes used in databases. Here, we consider the following indexing scheme: Primary Index, Clustered Index and Secondary index. Based on the query, we decide what type of indexing file. Table 3.3 lists the indexing type for each relation.

Relation	Indexing type	Indexing attribute(s)	Is a key?
User	primary	User_id	yes
department	primary	Dept_id	yes
authorization	primary	Auth_id	yes
pr	primary	Pr_no	yes
assign	primary	Pr_no	yes
tender	primary	T_id, Pr_no	yes
status	primary	Status_id	yes
Ext1	primary	T_id, Pr_no	yes
Ext2	primary	T_id, Pr_no	yes

Table 3.3: Indexing type for each relation

The following table Table 3.4 explains what the disk block access time to extract particular record for all the relations is.

Relation	# of records	# no of data blocks	Index size per record	# of index records per block	# no of index blocks	# no of block access without indexing	# no of block access with indexing
user	20	1	4+4	128	1	1	1
department	5	1	4+4	128	1	1	1
authorization	4	1	4+4	128	1	1	1
pr	1000	77	4+4	128	8	77	4
assign	1000	24	4+4	128	8	24	4
tender	3000	62	4+4+4	86	35	62	7
status	12	1	4+4	128	1	1	1
Ext1	2000	22	4+4+4	86	24	22	6
Ext2	1500	16	4+4+4	86	18	16	6

Table 3.4: Number of disk block access with and without indexing

# of index records per block= Block size / Index size per record

 # no of index blocks =ceil(# of records / # of index records per block)

 # no of block access without indexing = # no of data blocks

number of block accesses with indexing = ceil [log(# no of index blocks)] + 1

Indexing the data file definitely reduces the number of block accesses needed to find particular record from the data file. The complete statistics is showed in above table.

3.4 TIMINGS

Disk access time = Avg seek time + latency time + block transfer time

= 20 + 10 + 0.5

= 30.5 ms

Therefore, to access one random block and transfer it, the time is 30.5ms. If the blocks are consecutive seek time and latency time are not included. Also, there can be overhead delay and queuing delay.

3.5 SYSTEM SPECIFICATIONS

Query:

- ➔ SELECT Table_NAME "proknap",
- ➔ data_length "table data_length in Bytes",
- ➔ index_length "table index_length in Bytes",
- ➔ data_free "Free Space in Bytes"
- ➔ FROM information_schema.TABLES where Table_schema="proknap";

Results obtained foe the above query are shown in Fig. 3.1.

proknap	table data_length in Bytes	table index_length in Bytes	Free Space in Bytes
assign	16384	16384	0
authorisation	16384	0	0
department	16384	0	0
ext1	16384	16384	0
ext2	16384	16384	0
pr	16384	16384	0
pr_view	*NULL*	*NULL*	*NULL*
status	16384	0	0
tender	16384	32768	0
user	16384	32768	0

Fig 3.1 Table data length and index length for all tables

3.6 QUERY COSTS

Assumptions: T - number of blocks in tender table = 100

P- number of blocks in pr table = 100

A - number of blocks in the assign table = 50

E - number of blocks in ext1 table = 50

1. Simple query

SELECT * FROM tender;

Time taken: 0.0013s

2. SELECT * FROM `tender` t, pr i where t.Pr_no=i.Pr_no

Time taken: 0.0184s

Block nested loop join:

for each block T of t do begin

for each block P of p do begin

    for each tuple t in T do begin

    for each tuple p in P do begin

        Check if (t,p) satisfy the condition t.pr_no = p.pr_no;

        if they do add t,p to the result

        end if

    end

    end

end

end

Worst case cost: T + (T*P) = 100 + (100*100) = 10100.

Best case cost: T + P = 100 + 100 = 200

Index block nested loop join:

For each T record, cost of disk access for P is 1 for hash index and 3 on an average for BTree.

Cost: I + (I * Cost of search for R)

3. SELECT t.T_id, p.Pr_no, a.do FROM tender l, pr p, assign a WHERE t.pr_no=p.pr_no AND p.pr_no=a.pr_no;

Time taken: 0.0022 sec

Block nested loop:

for each block T of t do begin

for each block P of p do begin

for each block A of a do begin

    for each tuple t in T do begin

    for each tuple p in P do begin

    for each tuple a in A do begin

        Check if (l,i,r) satisfy the condition

        if they do add l.i.r to the result

        end if

    end

    end

    end

end

end

end

Cost (Worst Case): T + (T * P) + (T * P * A) = 100+(100*100)+(100*100*50)

I/Os: 510100

Cost (Best Case): L+ I + R = 100+100+50

I/Os: 250

4. SELECT * from assign a, pr p WHERE a.Pr_no = p.Pr_no;

Time taken: 0.0004 sec

Nested loop join:

for each tuple a in A do

for each tuple p in P do

if a.Pr_id == p.Pr_id

then add a,p into result

end if

end

end

Worst case cost: na * bp + ba = number of records(tuples)of A* number of blocks of P+ number of blocks of A

Best case cost: A + P = 50 + 100 = 150

Index nested loop join:

For each record A, cost of disk access for L is 1 for hash index and about 3 for BTree.

Cost: A + (A * blocking factor * Cost of search in L)

5. Creating Indexes

SQL executed: **SELECT Descr FROM status;**

Time taken Before Creating Index:0.0013 sec

Index Created: **CREATE index index1 ON status(Descr);**

Time taken After Creating Index index1:0.0011 sec

Indexing uses BTree : The index leaf nodes are stored in arbitrary order. Their position on the disk does not correspond to the logical position according to the indexing order. Therefore to find entry among the arbitrarily stored index leaf nodes we use **'Balanced Tree'**.

CHAPTER 4

IMPLEMENTATION AND RESULTS

Procurement tracking system was successfully implemented with an interactive user interface using MySQL, phpmyadmin for backend and HTML, CSS, PHP for web interface.Some screenshots of the frontend and backend are shown in the Fig 4.1, Fig. 4.2, Fig 4.3, Fig 4.4, Fig 4.5.

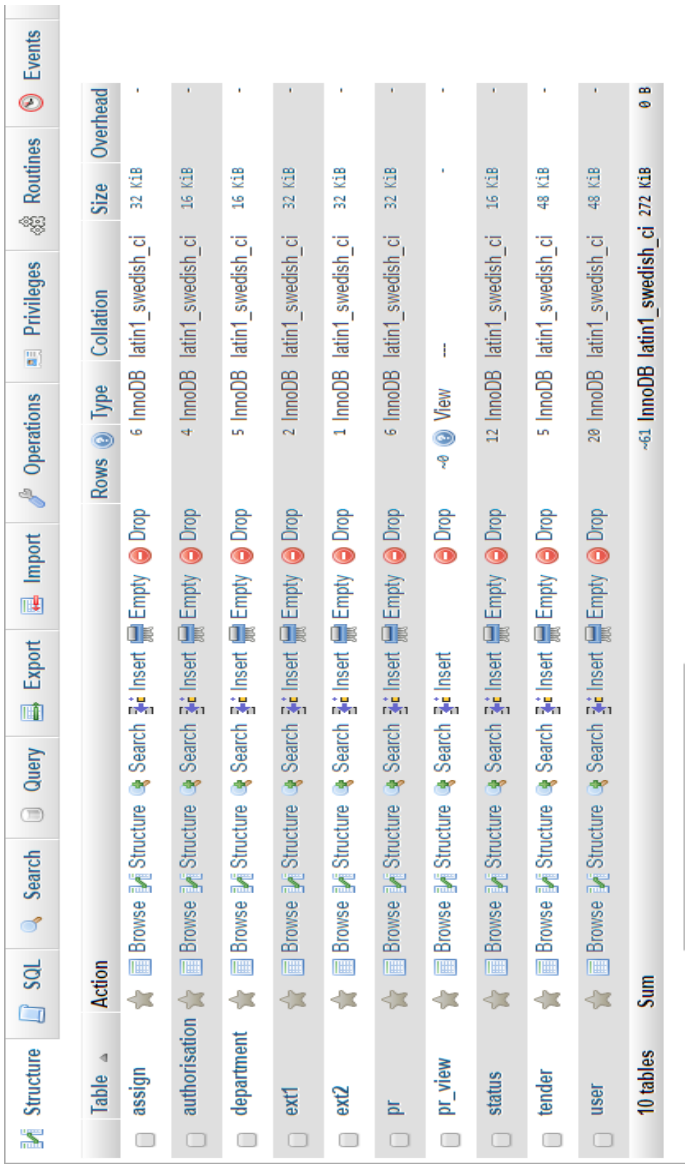


Fig 4.1:Screenshot of database

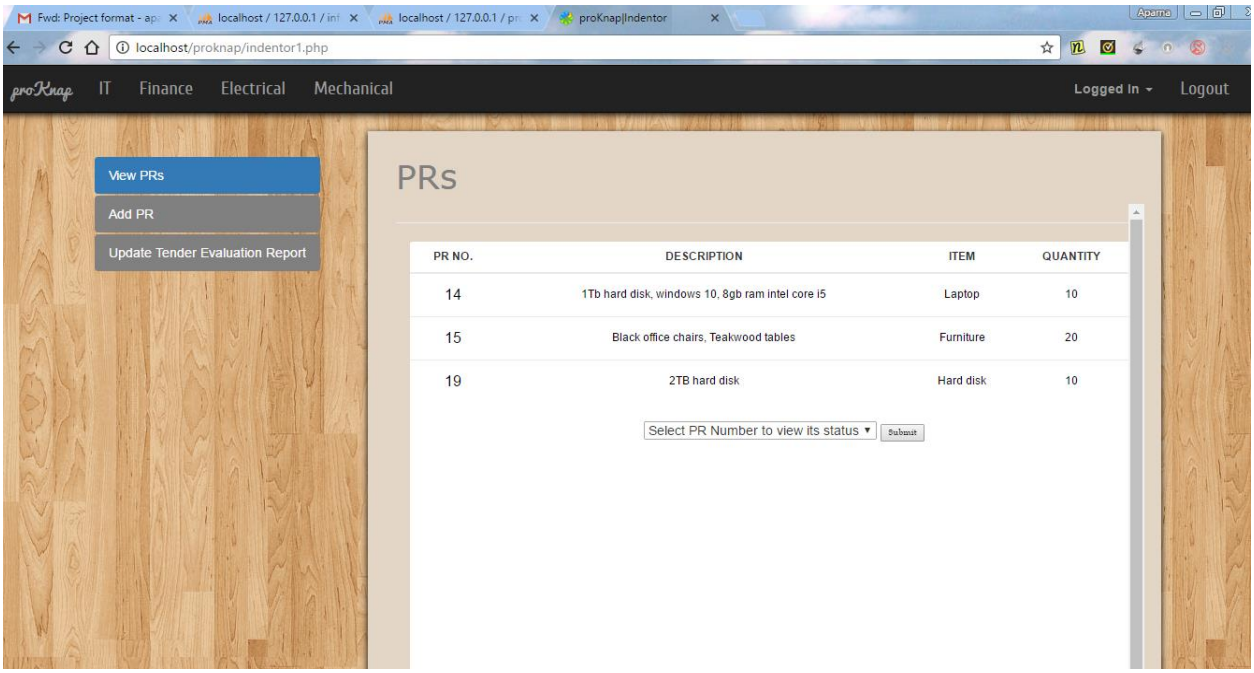


Figure 4.2: Indentor's page

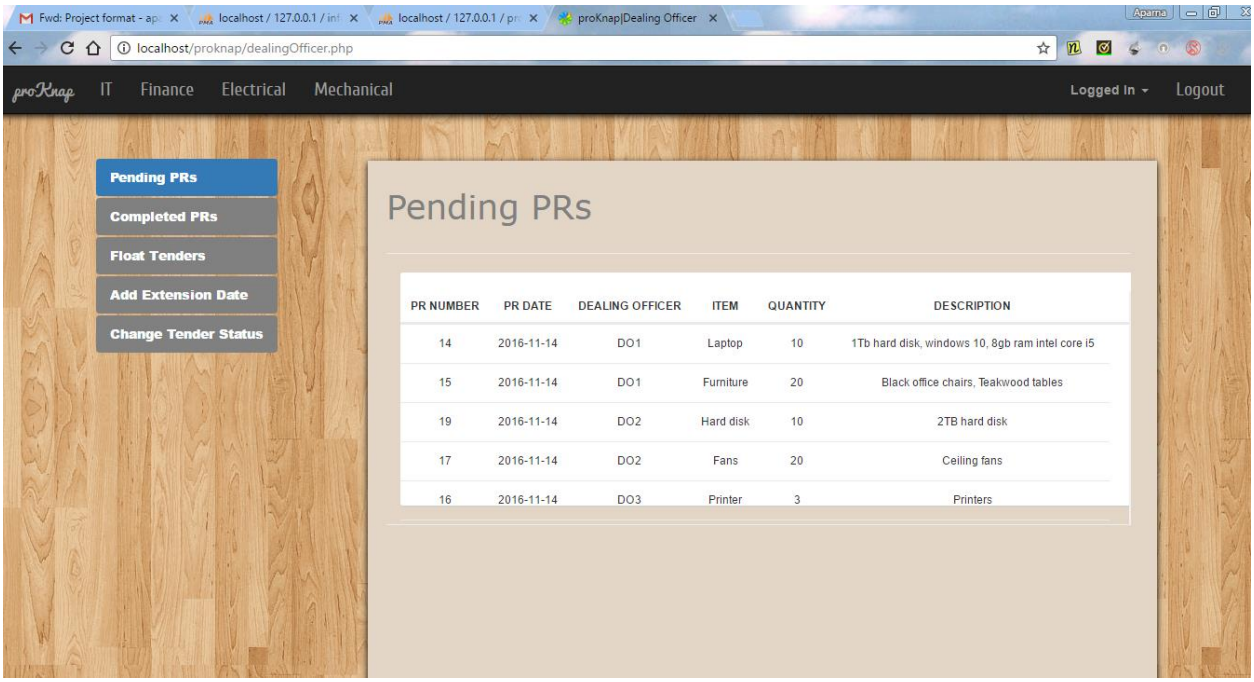


Figure 4.3: Dealing officer's page

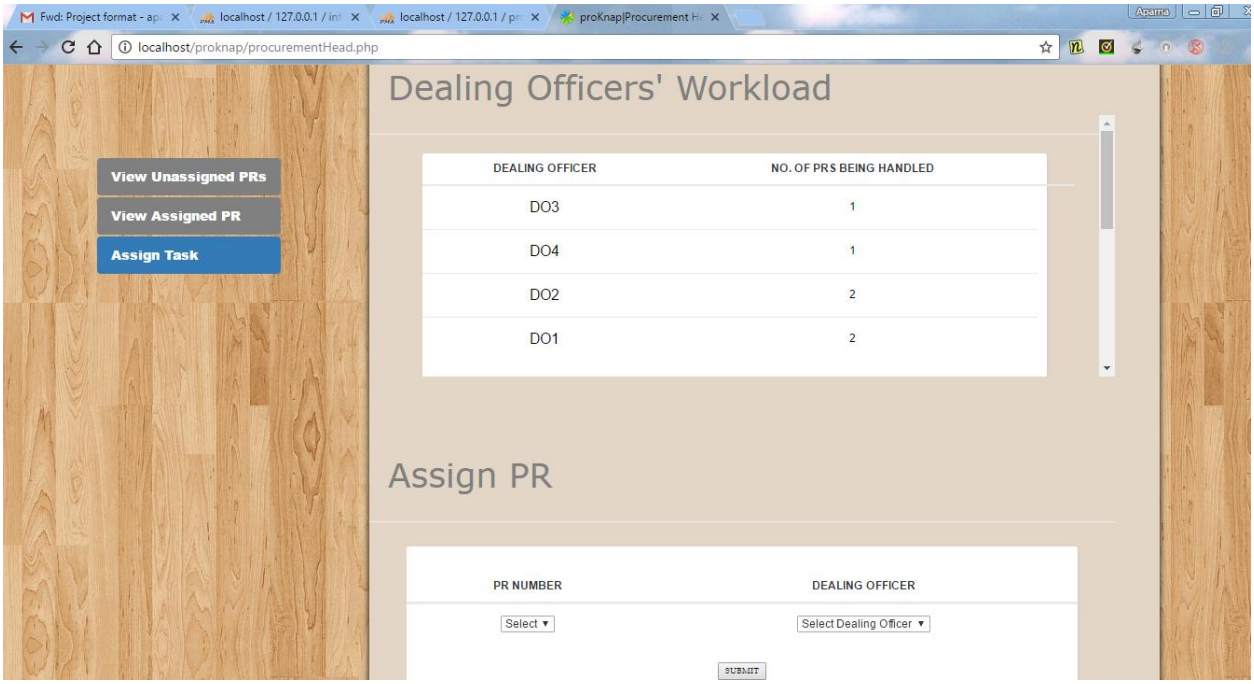


Figure 4.4: Procurement head's page

CONCLUSION

Procurement tracking system for an organization was successfully implemented. A conceptual schema was prepared after the detailed understanding of the current working of the system. It was then converted to a relational schema. To remove anomalies, the relational schema was normalized upto 3NF. Several queries were executed on the database and query processing time was calculated and times taken before indexing and after indexing were compared. This system is ready to be put into use by any organization to automate the task of tracking their procurements.

REFERENCES

[1] http://dev.mysql.com/doc/refman/5.7/en/

[2] Elmasri, Ramez. *Fundamentals of database systems*. Pearson Education India, 2008.