



**NETAJI SUBHAS INSTITUTE  
OF TECHNOLOGY**

**DATABASE MANAGEMENT SYSTEM  
PROJECT**

**ATHENA  
BOOKSTORE**

**Submitted By**  
**Bhavya Bansal 2016UCO1523**  
**Aastha Agrawal 2016UCO1526**

# **CERTIFICATE**

This is to certify that **Bhavya Bansal(2016UCO1523)** and **Aastha Agrawal(2016UCO1526)**, students of 3<sup>rd</sup> semester B.E. (2016-2020) of Netaji Subhas Institute of Technology, Delhi have successfully completed the project **Bookstore Management System** in **MYSQL** using **nodejs** under the guidance of **Ms Swati Aggarwal**.

**Signature**  
**Ms Swati Aggarwal**

# **ACKNOWLEDGEMENT**

We take this opportunity to express our sincere gratitude to all those who helped us in various capacities in undertaking this project and devising the report.

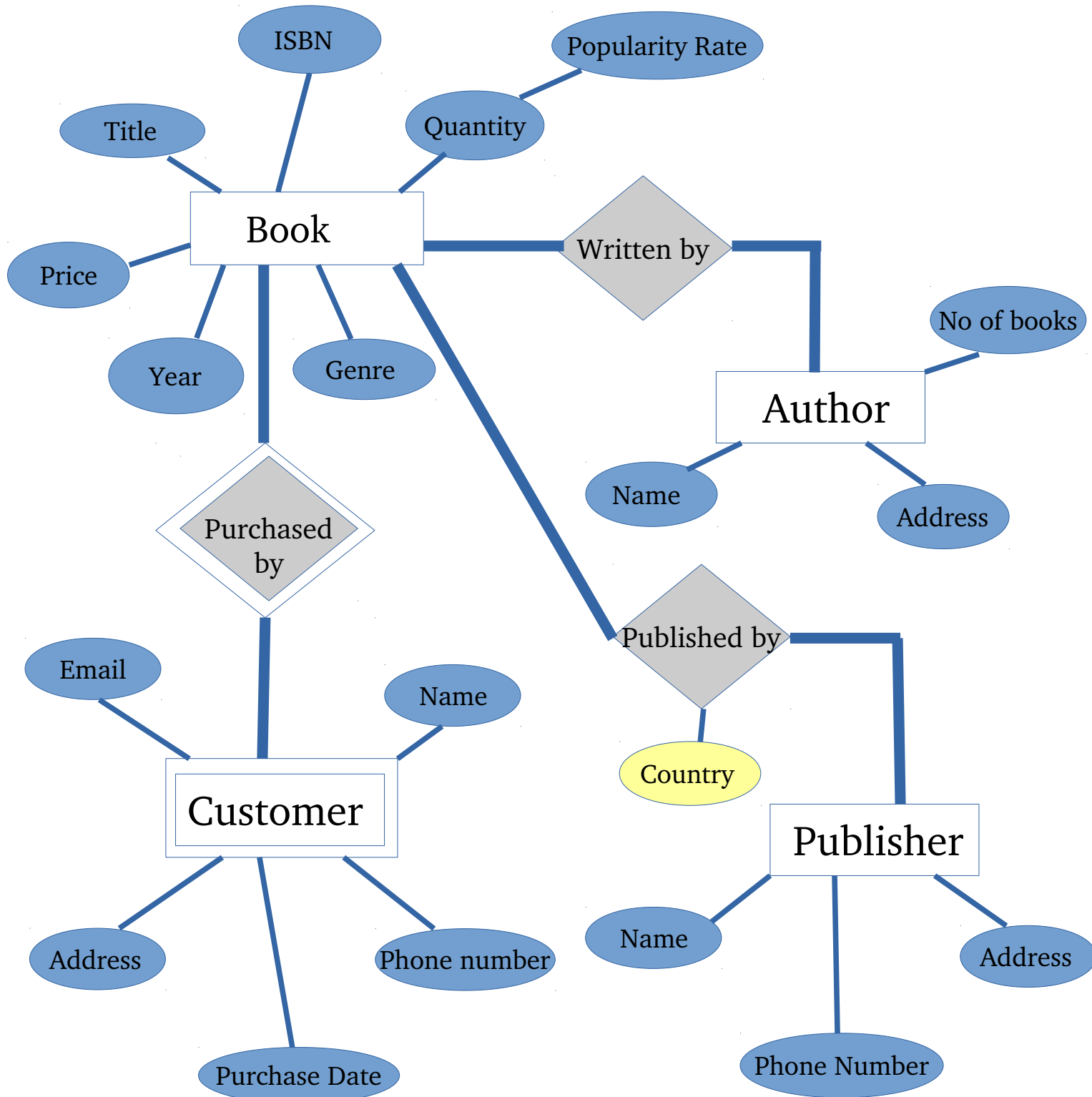
We are privileged to express our sense of gratitude to our respected teacher, Ms Swati Aggarwal, whose unparalleled knowledge was an immense support in completing the project.

We take this opportunity also to thank our friends and contemporaries for their co-operation and compliance.

# **INTRODUCTION**

The System is designed to manage a Bookstore. It handles the various transactions that could occur in a bookstore from both admin and users perspective. It handles adding of new books to database, purchasing books from the store( having filters according to the popularity of the book, price of the book, books by specific author, etc). It automates the quantity of books left in the store after each purchase and also decides the popularity of the book by a specific formula. Achieving this objective is difficult using a manual system as the information is scattered, can be redundant and collecting relevant information may be very time consuming. All these problems are solved using this project. The overall functioning of the system, schemas and its attributes are mentioned in the following pages.

# ER MODEL



# RELATIONAL SCHEMA

## Original Relational Schema- from the ER Diagram

**customer**(isbn, email, name, address, purchase\_date, phone\_num)  
**book**(isbn, genre, title, price, year, quantity, popularity\_rate)  
**publisher**(name, phone\_num, address)  
**author**(name, address, no\_of\_books\_written)  
**written\_by**(isbn, author\_name, author\_address)  
**published\_by**(isbn, publisher\_name, country)

where :-

- customer\_phone\_num is **multivalued** attribute
- popularity is **derived** attribute

# NORMALIZATION

Normalization is a process that “improves” a database design by generating relations that are of higher normal forms.

## ♦ First Normal form

*We say a relation is in 1NF if all values stored in the relation are single-valued and atomic. 1NF places restrictions on the structure of relations. Values must be simple.*

In table customer, we have a multivalued attribute phone number hence it is not in 1NF form whereas the rest of the tables are in 1NF form since they have only single/atomic attributes.

To convert customers into 1NF form we break customer into two tables such that we get rid of multivalued attributes.

**customer(isbn, email, name, address, purchase\_date,  
phone\_num)**

can be broken down into

- ➔ **customer(isbn, email, name, address, purchase\_date)**
- ➔ **customer\_phonenum(isbn, phone\_num)**

**Now all tables are in 1NF form**


## ◆ Second Normal Form

*A relation is in 2NF if it is in 1NF, and every non-key attribute is fully dependent on each candidate key. (That is, we don't have any partial functional dependency.)*

In the table customer, name and address of customer depend only on email of customer( ie independent of other primary key) hence it is not in 2NF form. Whereas in other tables it is not so, so they are in 2NF form.

To convert customer into 2NF form:-  
**customer:**

ISBN	Email	Name	Address	Purchase_date



The diagram shows a horizontal line with three upward-pointing arrows. The first arrow is positioned under the 'Email' column, the second under the 'Name' column, and the third under the 'Address' column. This represents the functional dependencies Email → Name and Email → Address.

*The above relation has redundancies: the invoice date is repeated on each invoice line. We can improve the database by decomposing the relation into two relations:*

**customer(isbn, email, name, address, purchase\_date)**

can break down into:-

- ➔ **customer(email, name, address)**
- ➔ **book\_purchased(isbn, email, purchase\_date)**

**Now all tables are in 2NF form**



## ◆ Third Normal Form

*A relation in 3NF will not have any transitive dependencies of non-key attribute on a candidate key through another non-key attribute.*

In the table book, popularity rate is dependent on the quantity of books available in the store( ie if there are more books in the store then the popularity of the book is also more).

*If quantity < 10 :- popularity = 25*

*If quantity ≥ 10 and < 15 :- popularity = 50*

*If quantity ≥ 15 and < 20 :- popularity = 75*

*If quantity ≥ 20 :- popularity = 100*

ISBN	Title	Genre	Price	Year	Quantity	Popularity_rate
					↑	↓

So it is not in 3NF form whereas the others do not have derived attributes and are hence in 3NF form. To convert book table into 3NF form we do:-

*We correct the situation by decomposing the original relation into two 3NF relations.*

ISBN	Title	Genre	Price	Year	Quantity
------	-------	-------	-------	------	----------

Quantity	Popularity_rate
----------	-----------------

**book(isbn, genre, title, price, year, quantity, popularity\_rate)**  
we divide it into:-

➔ **book(isbn, genre, title, price, year, quantity)**

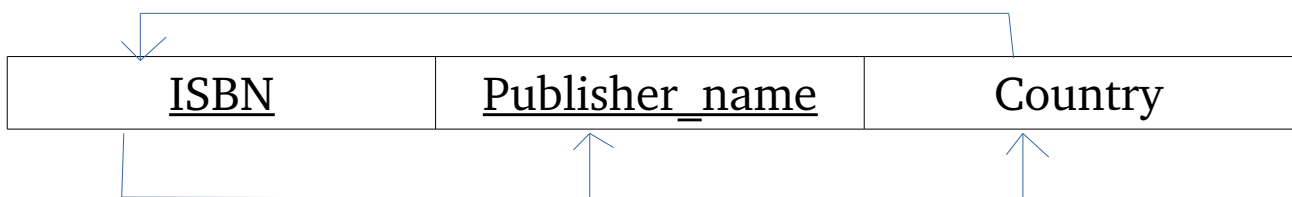
➔ **popularity(quantity, popularity\_rate)**

**Now all tables are in 3NF form**

## ◆BCNF Form

*a relation is in BCNF if it is in 1NF and if every determinant is a candidate key.*

In the table publisher, one of the primary key is dependent on non key attribute hence it is not in BCNF form whereas in other tables this is not the case so they are in BCNF form.



To convert it into BCNF form we break the tables into two tables like:-

<u>Publisher_name</u>	<u>Country</u>
-----------------------	----------------

ISBN	<u>Country</u>
------	----------------

**published\_by**(isbn, publisher\_name, country)

can be divided into:-

- **published\_in**(publisher\_name, country)
- **isbncode**(isbn, country)

**Now all tables are in BCNF form**

# Relational Schema after Normalization

- ➔ `book(isbn, genre, title, price, year, quantity)`
- ➔ `popularity(quantity, popularity_rate)`
- ➔ `customer(email, name, address)`
- ➔ `book_purchased(isbn, email, purchase_date)`
- ➔ `customer_phonenum(isbn, phone_num)`
- ➔ `author(name, address, no_of_books_written)`
- ➔ `written_by(isbn, author_name, author_address)`
- ➔ `publisher(name, phone_num, address)`
- ➔ `published_in(publisher_name, country)`
- ➔ `isbncode(isbn, country)`

## Example:-

```
mysql> describe book;
```

Field	Type	Null	Key	Default	Extra
ISBN	int(11)	NO	PRI	NULL	
genre	varchar(45)	NO		NULL	
title	varchar(45)	NO		NULL	
price	int(11)	YES		NULL	
year	date	NO		NULL	
quantity	int(11)	NO	MUL	NULL	

6 rows in set (0.04 sec)

```
mysql> describe customer;
```

Field	Type	Null	Key	Default	Extra
email	varchar(25)	NO	PRI	NULL	
name	varchar(45)	NO		NULL	
address	varchar(45)	NO		NULL	

3 rows in set (0.64 sec)

# CONSTRAINTS

- **PRIMARY KEY:**

Primary key is used to denote the set of attributes that is chosen by the database designer as the principal means of identifying tuples.

ATTRIBUTE	PRIMARY KEY OF
Name, address	author
ISBN	book
ISBN	book_purchased
Email	customer
phone_num	customer_phonenum
name	publisher
publisher_name	published_in

- **NOT NULL**

The not null constraint on an attribute specifies that the null value is not allowed for that attribute; in other words, the constraint excludes the null value from the domain of that attribute.

RELATIONS	ATTRIBUTES DECLARED TO BE NOT NULL
author	Name, address
book	Isbn, genre, title, year, quantity
Book_purchased	-
customer	Name, address
customer_phonenum	email
isbncode	Country, isbncode
Popularity	Quantity, popularity_rate
published_in	country
publisher	phone_num
written_by	Issbn, author_name, author_address

## • FOREIGN KEY

The foreign key specification says that the values of attributes ( $Ak_1, Ak_2, \dots, Ak_n$ ) for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation  $s$ .

FOREIGN KEY	REFERENCING RELATION	REFERENCED RELATION
PlaceName	Accommodation	Place
PlaceName	SportsVenue	Place
MemID	Sportsperson	Member
MemID	Official	Member
MemID	Coach	Sportsperson
PlaceName	Member	Accommodation
PlaceName	Event	SportsVenue
MemID	TravelTo	Member
PlaceName	TravelTo	SportsVenue
Winner	Event	Sportsperson
MemID	Manage	Official
EventName	Manage	Event
MemID	Participate	Sportsperson
EventName	Participate	Event

# TRIGGERS

## ➤ After insert trigger in book:

*By this trigger we want that as and when a book is added in the database then corresponding to its quantity its popularity is decided.*

```
USE `bookstore`;
```

```
DELIMITER $$
```

```
DROP TRIGGER IF EXISTS bookstore.book_AFTER_INSERT$  
$
```

```
USE `bookstore` $$
```

```
CREATE DEFINER=`root`@`localhost` TRIGGER  
`bookstore`.`book_AFTER_INSERT` AFTER INSERT ON  
`book` FOR EACH ROW
```

```
begin
```

```
    if new.quantity < 10 THEN
```

```
        insert into popularity values(new.quantity,25);
```

```
    end if;
```

```
    if new.quantity >= 10 and new.quantity < 15 THEN
```

```
        insert into popularity values(new.quantity,50);
```

```
    end if;
```

```
    if new.quantity >= 15 and new.quantity < 20 THEN
```

```
        insert into popularity values(new.quantity,75);
```

```
    end if;
```

```
    if new.quantity >= 20 THEN
```

```
        insert into popularity values(new.quantity,100);
```

```
    end if;
```

```
END$$
```

```
DELIMITER ;
```

## ➤ Update trigger on book

*By this trigger we ensure that whenever the quantity is reduced in the database ie a book is purchased and the shopkeeper doesn't buy a new book it means that its popularity is decreased hence it automatically decreases the popularity according to the formula used.*

```
USE `bookstore` ;
DELIMITER $$
DROP TRIGGER IF EXISTS
bookstore.book_AFTER_UPDATE$$
USE `bookstore` $$
CREATE DEFINER=`root`@`localhost` TRIGGER
`bookstore`.`book_AFTER_UPDATE` AFTER UPDATE ON
`book` FOR EACH ROW
BEGIN
if new.quantity <10 THEN
    update popularity set popularity.quantity =25;
end if;
if new.quantity >=10 and new.quantity<15 THEN
    update popularity set popularity.quantity =50;
end if;
if new.quantity >=15 and new.quantity<20 THEN
    update popularity set popularity.quantity =75;
end if;
if new.quantity >=20 THEN
    update popularity set popularity.quantity =100;
end if;
END$$
DELIMITER ;
```

### ➤ After insert trigger on book\_purchased

*By this trigger we ensure that whenever a book is purchased by a customer its quantity decreases in the database so that there is no inconsistency in data.*

```
USE `bookstore` ;
```

```
DELIMITER $$
```

```
DROP TRIGGER IF EXISTS
```

```
bookstore.book_purchased_AFTER_INSERT$$
```

```
USE `bookstore` $$
```

```
CREATE DEFINER=`root`@`localhost` TRIGGER
```

```
`bookstore`.`book_purchased_AFTER_INSERT` AFTER
```

```
INSERT ON `book_purchased` FOR EACH ROW
```

```
BEGIN
```

```
    update book set
```

```
    book.quantity=quantity-1
```

```
    where book.ISBN=new.ISBN;
```

```
    update popularity set
```

```
    popularity.quantity= (select quantity from book where  
book.ISBN=new.ISBN);
```

```
END$$
```

```
DELIMITER ;
```



# Some Queries

## Creation of table book:-

```
CREATE TABLE `bookstore`.`book` (  
  `ISBN` INT NOT NULL,  
  `genre` VARCHAR(45) NOT NULL,  
  `title` VARCHAR(45) NOT NULL,  
  `price` INT NULL,  
  `year` DATE NOT NULL,  
  `quantity` INT NOT NULL,  
  PRIMARY KEY (`ISBN`));
```

## Creation of table book\_purchased:-

```
CREATE TABLE `bookstore`.`book_purchased` (  
  `ISBN` INT NOT NULL,  
  `email` VARCHAR(25) NOT NULL,  
  `purchase_date` DATE NULL,  
  PRIMARY KEY (`ISBN`),  
  INDEX `email_idx` (`email` ASC),  
  CONSTRAINT `ISBN`  
    FOREIGN KEY (`ISBN`)  
    REFERENCES `bookstore`.`book` (`ISBN`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION,  
  CONSTRAINT `email`  
    FOREIGN KEY (`email`)  
    REFERENCES `bookstore`.`customer` (`email`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION);
```

**Insertion into book table:-**

```
INSERT INTO `bookstore`.`book` (`ISBN`, `genre`, `title`,  
`price`, `year`, `quantity`) VALUES ('300', 'deception point',  
'mystery', '700', '20150304', '21');
```