**Database management systems 2**

End-term project “Library system”

Didar Auyesbay

Bakdaulet Shambilov

Dias Lesbay

Namazbek Bekzhanov

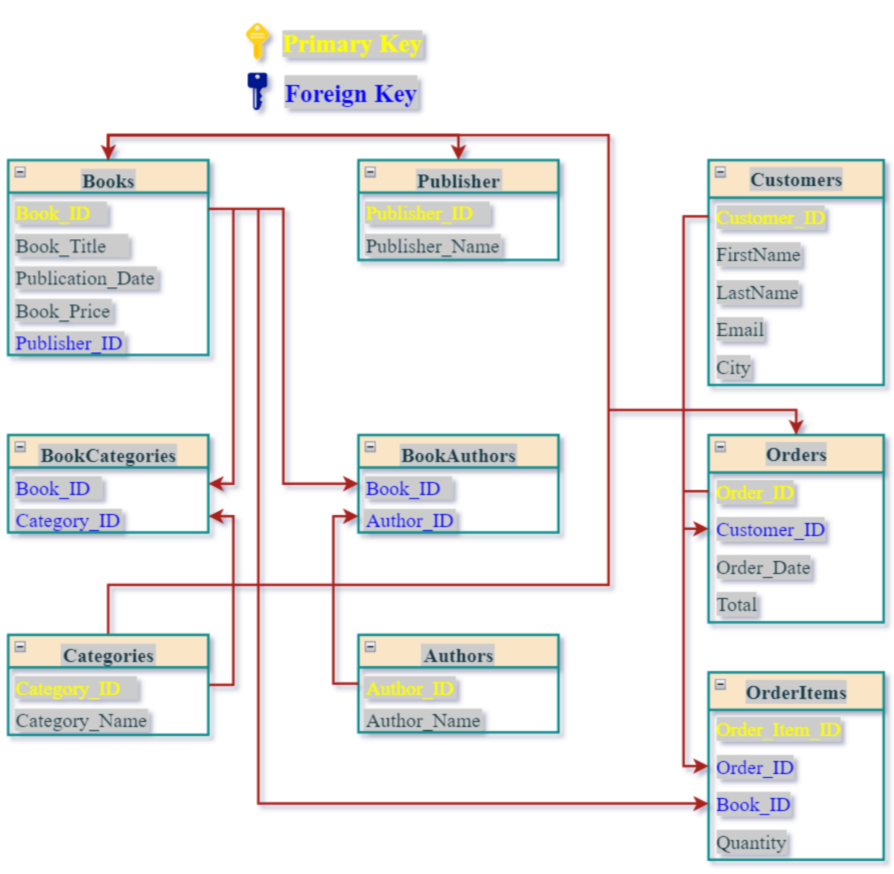
Dauken Aldan

# **Introduction to the system**

The library system project is a completely modern solution for library management with a lot of features such as cataloguing, book management, customer account administration and online ordering.

Our library system consists of all data that a real-life library would need and work with. This system provides all abilities that are said before. Our project is an essential tool for librarians, helping them to fasten library operations and provide better services to users.

# **ERD of the system**

  
Pic 1.1 ERD of the system

In this 3NF design, the data is organized so that each table contains data about a single entity or concept, and there are no repeating groups or transitive dependencies.

The *Books* table contains data about each book, including its title, publication date, ISBN, price, and the publisher’s ID. The *Authors* table contains information about each author, and the *Book* *Authors* table links books to their respective authors.

The *Publishers* table contains information about each publisher, and the *Books* table links each book to its publisher. The *Categories* table contains information about book categories, and the *Book* *Categories* table links books to their respective categories.

The *Orders* table contains information about each order, including the customer ID, order date, and total cost. The *Order* *Items* table contains information about each item ordered, including the book ID, quantity, and unit price. The *Order* *Items* table is linked to the *Orders* table using the Order ID foreign key.

With this design, the bookshop database management system can efficiently manage its inventory, track customer orders, and generate reports on book sales, authors, and publishers.

# **Explanation of why the structure follows normal forms**

1. *Books* Table:

* Each column in the *Books* table contains a single value (atomic).
* The primary key for this table is Book ID, which uniquely identifies each book.
* There are no transitive dependencies in this table.

1. *Authors* Table:

* Each column in the *Authors* table contains a single value (atomic).
* The primary key for this table is Author ID, which uniquely identifies each author.
* There are no transitive dependencies in this table.

1. *Book* *Authors* Table:

* This table represents a many-to-many relationship between books and authors, where each book can have multiple authors and each author can write multiple books.
* The primary key for this table is a combination of *Book ID* and *Author ID*, which uniquely identifies each record.
* There are no transitive dependencies in this table.

1. *Publishers* Table:

* Each column in the *Publishers* table contains a single value (atomic).
* The primary key for this table is *Publisher ID*, which uniquely identifies each publisher.
* There are no transitive dependencies in this table.

1. *Categories* Table:

* Each column in the *Categories* table contains a single value (atomic).
* The primary key for this table is *Category ID*, which uniquely identifies each category.
* There are no transitive dependencies in this table.

1. *Book* *Categories* Table:

* This table represents a many-to-many relationship between books and categories, where each book can belong to multiple categories and each category can have multiple books.
* The primary key for this table is a combination of *Book ID* and *Category ID*, which uniquely identifies each record.
* There are no transitive dependencies in this table.

1. *Orders* Table:

* Each column in the *Orders* table contains a single value (atomic).
* The primary key for this table is *Order* *ID*, which uniquely identifies each order.
* There are no transitive dependencies in this table.

1. *Order* *Items* Table:

* This table represents a many-to-many relationship between orders and books, where each order can have multiple items and each item can belong to multiple orders.
* The primary key for this table is *Order* *Item* *ID*, which uniquely identifies each order item.
* There are no transitive dependencies in this table.

# **Explanation and coding part of each item from “Add the following”**

## **Procedure which does group by information**

CREATE OR REPLACE PROCEDURE GroupBooksByPublication\_Date IS

publication\_date Date;

num\_books INTEGER;

BEGIN

FOR r IN (

SELECT Publication\_Date, COUNT(\*) AS NumBooks

FROM Books

GROUP BY Publication\_Date

) LOOP

publication\_date := r.Publication\_Date;

num\_books := r.NumBooks;

DBMS\_OUTPUT.PUT\_LINE('Publication Date: ' publication\_date ', Number of Books: ' || num\_books);

END LOOP;

END;

BEGIN

GroupBooksByPublication\_Date;

END;

Explanation: a procedure that groups books by their publication date and prints out the total number of books published on each date.

## **Function which counts the number of records**

CREATE OR REPLACE FUNCTION GetNumBooks RETURN INTEGER IS

NumBooks INTEGER;

BEGIN

SELECT COUNT(\*) INTO NumBooks

FROM Books;

RETURN NumBooks;

END;

DECLARE

num\_records INteger;

BEGIN

num\_records := GetNumBooks();

dbms\_output.put\_line('Number of records: ' || num\_records);

END;

Explanation: Retrieves the number of records in a table named ‘Books’ and returns the result as an integer value

## **Procedure which uses SQL%ROWCOUNT to determine the number of rows affected**

CREATE OR REPLACE PROCEDURE UpdateBookPrice(BookID IN Number, NewPrice IN NUmber) AS

v\_count NUMBER;

BEGIN

UPDATE Books

SET book\_Price = NewPrice

WHERE Book\_ID = BookID;

v\_count := SQL%ROWCOUNT;

IF v\_count = 1 THEN

DBMS\_OUTPUT.PUT\_LINE('Book updated successfully.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Error updating book.');

END IF;

END;

BEGIN

UpdateBookPrice(102, 900000);

END;

select \* from books where book\_id = 102

Explanation: updates the price of a book in a database table named Books. The procedure takes two input parameters: BookID and NewPrice. It also includes a validation to check if the update was successful or not.

## **Add user-defined exception which disallows to enter title of item (e.g. book) to be less than 5 characters**

CREATE OR REPLACE PROCEDURE SearchBook(title IN VARCHAR2) IS

searchError exception;

price number;

pub\_date date;

BEGIN

IF LENGTH(title) < 5 THEN

RAISE searchError;

Else

select book\_price, publication\_date into price, pub\_date

from books

where book\_title = title;

dbms\_output.put\_line(' Title: ' title ' Price: ' price ' Publication date: ' || pub\_date);

END IF;

exception

when searchError then

dbms\_output.put\_line('The length of book would be greater than 5 characters');

when others then

dbms\_output.put\_line('Ондай кітап жоқ!');

END;

begin

SearchBook('ыгтгрытргфап');

end;

Explanation: searches for a book in the database table named Books by its title. If the book is found, the procedure retrieves the book's price and publication date from the database and displays them. If the length of the book title is less than five characters, a custom exception named searchError is raised.

## **Create a trigger before insert on any entity which will show the current number of rows in the table**

CREATE TRIGGER CounterBooks

BEFORE INSERT ON Books

FOR EACH ROW

DECLARE

count INTEGER;

BEGIN

SELECT COUNT(\*) INTO count FROM Books;

DBMS\_OUTPUT.PUT\_LINE('Кітаптар саны: ' || count);

END;

insert into Books(Book\_id, BOOK\_TITLE, PUBLICATION\_DATE, BOOK\_PRICE, PUBLISHER\_ID) values(202, 'Дәукенді қуған Аяжан', '03/03/2008', 200000, 17);

Explanation: Fires before a new row is inserted into the Books table. The trigger counts the number of rows in the Books table and displays the count