

LAPORAN TUGAS AKHIR

PROGRAM FILM CATALOG DB

Disusun untuk Memenuhi Matakuliah Dasar Pemrograman Komputer
Dibimbing oleh Bapak Wahyu Sakti Gunawan Irianto

Oleh:

NURU SYAHRIRRAMADHAN

NIM 240533605851

S1 PTI '24 OFF C



UNIVERSITAS NEGERI MALANG
FAKULTAS TEKNIK
DEPERTEMEN TEKNIK ELEKTRO DAN INFORMATIKA
PRODI S1 PENDIDIKAN TEKNIK INFORMATIKA
APRIL 2025

PROGRAM FILM CATALOG DB

Dasar Teori

1. Selection Sort

Sorting adalah suatu proses pengurutan data yang sebelumnya disusun secara acak atau tidak teratur menjadi urut dan teratur menurut suatu aturan tertentu. Selection Sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya sampai elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisinya dan langsung ditukar. Selection Sort adalah algoritma pengurutan yang bekerja dengan cara membagi data menjadi dua bagian, bagian yang sudah terurut dan bagian yang belum terurut. Algoritma ini mencari elemen dengan nilai terkecil (atau terbesar) dari bagian yang belum terurut, kemudian menukarnya dengan elemen pertama pada bagian tersebut. Proses ini diulangi sampai seluruh data terurut.

2. Search Algorithm

Search Algorithm atau algoritma pencarian dibagi menjadi dua, yaitu Linear Search adalah metode pencarian sederhana yang memeriksa setiap elemen dalam array secara berurutan dari awal hingga akhir untuk menemukan elemen yang dicari. Dan Binary Search yang merupakan metode pencarian yang efisien untuk array yang sudah terurut. Algoritma ini bekerja dengan cara membagi dua bagian array dan membandingkan elemen tengah dengan nilai yang dicari, kemudian mempersempit pencarian ke bagian kiri atau kanan sesuai hasil perbandingan.

Kode Program C++

```
#include <iostream>

#include <iomanip>

#include <mysql_driver.h>

#include <mysql_connection.h>

#include <cppconn/prepared_statement.h>

#include <algorithm>

#include <vector>

#include <cctype>

#include <regex>

using namespace std;

sql::mysql::MySQL_Driver* driver;

sql::Connection* con;
```

```
// Fungsi untuk koneksi database

void connectToDatabase() {

    try {

        driver = sql::mysql::get_mysql_driver_instance();

        con = driver->connect("tcp://127.0.0.1:3306", "root", "12345");

        con->setSchema("tugas_akhir");

    }

    catch (sql::SQLException& e) {

        cerr << "SQL Error: " << e.what() << endl;

        exit(1);

    }

}
```

```
struct Movie {

    int id;

    string judul;

    string genre;

    float rating;

    string tanggal_rilis;

};
```

```
// Fungsi untuk validasi input angka

template <typename T>

T getValidNumberInput(const string& prompt) {

    T value;

    string input;

    while (true) {

        cout << prompt;

        getline(cin, input);
```

```

try {

    if (input.empty()) {

        throw invalid_argument("Input tidak boleh kosong");

    }

    regex pattern("^[-+]?[0-9]*\\.?[0-9]+$");

    if (!regex_match(input, pattern)) {

        throw invalid_argument("Input harus berupa angka");

    }

    if constexpr (is_same<T, int>::value) {

        value = stoi(input);

    }

    else if constexpr (is_same<T, float>::value) {

        value = stof(input);

    }

    return value;

}

catch (const exception& e) {

    cerr << "Error: " << e.what() << endl;

    cin.clear();

}

}

```

// Fungsi untuk validasi tanggal

```

string getValidDateInput() {

    string date;

```

```

regex datePattern("^\\d{4}-(0[1-9]|1[0-2])-(0[1-9]|12|[0-9]|3[01])$");

while (true) {

    cout << "Enter release date (YYYY-MM-DD): ";

    getline(cin, date);

    if (regex_match(date, datePattern)) {

        return date;

    }

    cerr << "Format tanggal salah! Gunakan format YYYY-MM-DD\n";

}

// Fungsi untuk validasi rating
float getValidRatingInput() {

    while (true) {

        float rating = getValidNumberInput<float>("Enter rating (0-10): ");

        if (rating >= 0.0f && rating <= 10.0f) {

            return rating;

        }

        cerr << "Rating harus antara 0 sampai 10\n";

    }

}

// Selection Sort by ID
void sortById(vector<Movie>& movies, bool ascending = true) {

    int n = movies.size();

    for (int i = 0; i < n - 1; i++) {

        int ext = i;

```

```

    for (int j = i + 1; j < n; j++) {

        if (ascending ? (movies[j].id < movies[ext].id) : (movies[j].id > movies[ext].id)) {

            ext = j;

        }

    }

    swap(movies[i], movies[ext]);

}

}

// Selection Sort by Title

void sortByTitle(vector<Movie>& movies, bool ascending = true) {

    int n = movies.size();

    for (int i = 0; i < n - 1; i++) {

        int ext = i;

        for (int j = i + 1; j < n; j++) {

            if (ascending ? (movies[j].judul < movies[ext].judul) : (movies[j].judul > movies[ext].judul)) {

                ext = j;

            }

        }

        swap(movies[i], movies[ext]);

    }

}

// Selection Sort by Rating

void sortByRating(vector<Movie>& movies, bool ascending = true) {

    int n = movies.size();

    for (int i = 0; i < n - 1; i++) {

        int ext = i;

        for (int j = i + 1; j < n; j++) {

            if (ascending ? (movies[j].rating < movies[ext].rating) : (movies[j].rating > movies[ext].rating)) {

```

```
        ext = j;
    }
}
swap(movies[i], movies[ext]);
}
}
```

// Binary Search by ID

```
Movie* binarySearchById(vector<Movie>& movies, int targetId) {
    int left = 0;
    int right = movies.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (movies[mid].id == targetId) {
            return &movies[mid];
        }

        if (movies[mid].id < targetId) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    return nullptr;
}
```

```
// Linear Search by Title (case insensitive)

Movie* searchByTitle(vector<Movie>& movies, const string& title) {

    for (auto& movie : movies) {

        string movieTitleLower = movie.judul;

        string searchTitleLower = title;

        transform(movieTitleLower.begin(), movieTitleLower.end(), movieTitleLower.begin(), ::tolower);

        transform(searchTitleLower.begin(), searchTitleLower.end(), searchTitleLower.begin(), ::tolower);

        if (movieTitleLower.find(searchTitleLower) != string::npos) {

            return &movie;

        }

    }

    return nullptr;

}
```

// Create

```
void addMovie(const Movie& movie) {

    try {

        sql::PreparedStatement* pstmt = con->prepareStatement(

            "INSERT INTO movies (judul, genre, rating, tanggal_rilis) VALUES (?, ?, ?, ?)");

        pstmt->setString(1, movie.judul);

        pstmt->setString(2, movie.genre);

        pstmt->setDouble(3, movie.rating);

        pstmt->setString(4, movie.tanggal_rilis);

        pstmt->execute();

        delete pstmt;

    }
```

```
catch (sql::SQLException& e) {

    cerr << "SQL Error: " << e.what() << endl;
```



```

    }
}

// Read all
vector<Movie> getAllMovies() {
    vector<Movie> movies;

    try {
        sql::Statement* stmt = con->createStatement();

        sql::ResultSet* res = stmt->executeQuery("SELECT * FROM movies");

        while (res->next()) {
            Movie movie;

            movie.id = res->getInt("id");

            movie.judul = res->getString("judul");

            movie.genre = res->getString("genre");

            movie.rating = res->getDouble("rating");

            movie.tanggal_rilis = res->getString("tanggal_rilis");

            movies.push_back(movie);
        }

        delete res;

        delete stmt;
    }

    catch (sql::SQLException& e) {
        cerr << "SQL Error: " << e.what() << endl;
    }

    return movies;
}

```

```

// Update

```

```

void updateMovie(const Movie& movie) {

    try {

        sql::PreparedStatement* pstmt = con->prepareStatement(

            "UPDATE movies SET judul=?, genre=?, rating=?, tanggal_rilis=? WHERE id=?");

        pstmt->setString(1, movie.judul);

        pstmt->setString(2, movie.genre);

        pstmt->setDouble(3, movie.rating);

        pstmt->setString(4, movie.tanggal_rilis);

        pstmt->setInt(5, movie.id);

        pstmt->execute();

        delete pstmt;

    }

    catch (sql::SQLException& e) {

        cerr << "SQL Error: " << e.what() << endl;

    }

}

```

// Delete

```

void deleteMovie(int id) {

    try {

        sql::PreparedStatement* pstmt = con->prepareStatement(

            "DELETE FROM movies WHERE id=?");

        pstmt->setInt(1, id);

        pstmt->execute();

        delete pstmt;

    }

    catch (sql::SQLException& e) {

        cerr << "SQL Error: " << e.what() << endl;

    }

}

```

```

void displayMenu() {

    cout << "\nMovie Catalog Database\n";

    cout << "1. View All Movies\n";

    cout << "2. Add Movie\n";

    cout << "3. Search Movie\n";

    cout << "4. Update Movie\n";

    cout << "5. Delete Movie\n";

    cout << "6. Exit\n";

    cout << "Choose option: ";

}

```

```

void displayMovies(const vector<Movie>& movies) {

    cout << "\nID\tJudul\t\tGenre\t\t\t Rating\t\tTanggal Rilis\n";

    cout << "-----\n";

    for (const auto& movie : movies) {

        cout << left

            << setw(4) << movie.id

            << setw(22) << (movie.judul.length() > 20 ? movie.judul.substr(0, 18) + "..." : movie.judul)

            << setw(30) << (movie.genre.length() > 28 ? movie.genre.substr(0, 27) + "..." : movie.genre)

            << right

            << setw(8) << movie.rating

            << setw(15) << movie.tanggal_rilis

            << endl;

    }

}

```

```

void searchMenu() {

    cout << "\nSearch Options:\n";

    cout << "1. Search by ID\n";

}

```

```
    cout << "2. Search by Title\n";

    cout << "Choose option: ";

}

void sortMenu() {

    cout << "\nSort Options:\n";

    cout << "1. Sort by ID (ascending)\n";

    cout << "2. Sort by ID (descending)\n";

    cout << "3. Sort by Title (A-Z)\n";

    cout << "4. Sort by Title (Z-A)\n";

    cout << "5. Sort by Rating (highest first)\n";

    cout << "6. Sort by Rating (lowest first)\n";

    cout << "Choose option: ";

}

int main() {

    connectToDatabase();

    int choice;

    do {

        displayMenu();

        choice = getValidNumberInput<int>("");

        cin.ignore();

        switch (choice) {

            case 1: { // View All Movies

                auto movies = getAllMovies();

                int sortChoice;

                sortMenu();
```

```
sortChoice = getValidNumberInput<int>("");  
cin.ignore();
```

```
switch (sortChoice) {  
    case 1: sortById(movies); break;  
    case 2: sortById(movies, false); break;  
    case 3: sortByTitle(movies); break;  
    case 4: sortByTitle(movies, false); break;  
    case 5: sortByRating(movies, false); break;  
    case 6: sortByRating(movies); break;  
    default: cout << "Invalid choice!\n";  
}
```

```
displayMovies(movies);
```

```
break;
```

```
}
```

```
case 2: { // Add Movie
```

```
    Movie movie;
```

```
    cout << "Enter title: ";
```

```
    getline(cin, movie.judul);
```

```
    cout << "Enter genre: ";
```

```
    getline(cin, movie.genre);
```

```
    movie.rating = getValidRatingInput();
```

```
    movie.tanggal_rilis = getValidDateInput();
```

```
addMovie(movie);
```

```
cout << "Movie added successfully!\n";
```

```
break;
```

```

}

case 3: { // Search Movie

    searchMenu();

    int searchChoice = getValidNumberInput<int>("");

    cin.ignore();


    auto movies = getAllMovies();

    sortById(movies); // Ensure sorted for binary search


    if (searchChoice == 1) {

        int id = getValidNumberInput<int>("Enter movie ID: ");


        Movie* found = binarySearchById(movies, id);

        if (found) {

            displayMovies({ *found });

        }

        else {

            cout << "Movie not found!\n";

        }

    }

    else if (searchChoice == 2) {

        string title;

        cout << "Enter movie title: ";

        getline(cin, title);


        Movie* found = searchByTitle(movies, title);

        if (found) {

            displayMovies({ *found });

        }

        else {

```

```

        cout << "Movie not found!\n";

    }

}

else {

    cout << "Invalid choice!\n";

}

break;

}

case 4: { // Update Movie

    int id = getValidNumberInput<int>("Enter movie ID to update: ");

    cin.ignore();

    auto movies = getAllMovies();

    Movie* found = binarySearchById(movies, id);

    if (found) {

        Movie updated = *found;

        cout << "Current title: " << updated.judul << "\nNew title (press enter to keep): ";

        string newTitle;

        getline(cin, newTitle);

        if (!newTitle.empty()) updated.judul = newTitle;

        cout << "Current genre: " << updated.genre << "\nNew genre (press enter to keep): ";

        string newGenre;

        getline(cin, newGenre);

        if (!newGenre.empty()) updated.genre = newGenre;

        cout << "Current rating: " << updated.rating << "\nNew rating (press enter to keep): ";

        string ratingInput;

        getline(cin, ratingInput);

```

```

if (!ratingInput.empty()) {
    try {
        updated.rating = stof(ratingInput);

        if (updated.rating < 0 || updated.rating > 10) {
            throw out_of_range("Rating harus antara 0-10");
        }
    }
    catch (...) {
        cerr << "Input rating tidak valid, menggunakan nilai sebelumnya\n";
    }
}

cout << "Current release date: " << updated.tanggal_rilis << "\nNew release date (press enter to keep):
";

string newDate;
getline(cin, newDate);

if (!newDate.empty()) {
    regex datePattern("^\\d{4}-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])$");

    if (regex_match(newDate, datePattern)) {
        updated.tanggal_rilis = newDate;
    }
    else {
        cerr << "Format tanggal salah, menggunakan nilai sebelumnya\n";
    }
}

}

updateMovie(updated);

cout << "Movie updated successfully!\n";
}

else {

```



```

        cout << "Movie not found!\n";

    }

    break;

}

case 5: { // Delete Movie

    int id = getValidNumberInput<int>("Enter movie ID to delete: ");

    deleteMovie(id);

    cout << "Movie deleted successfully!\n";

    break;

}

case 6:

    cout << "Exiting...\n";

    break;

default:

    cout << "Invalid choice!\n";

}

} while (choice != 6);

delete con;

return 0;

}

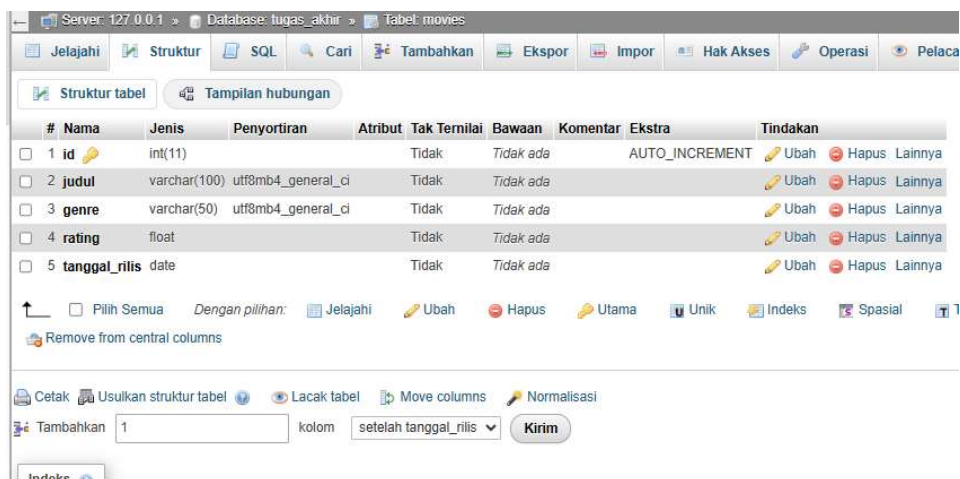
```

Database MySQL

MySQL adalah sistem manajemen basis data (DBMS) open-source yang digunakan untuk menyimpan, mengelola, dan mengakses data secara terstruktur menggunakan bahasa SQL (Structured Query Language). MySQL banyak digunakan dalam aplikasi web dan software karena ringan, cepat, dan mudah diintegrasikan dengan berbagai bahasa pemrograman seperti PHP, Python, Java, dan C++.

MySQL Connector/C++ adalah sebuah driver resmi yang disediakan oleh MySQL untuk memungkinkan aplikasi C++ terhubung dan berinteraksi dengan server database MySQL. Connector ini menyediakan Application Programming Interface (API) berorientasi objek, sehingga pengembang dapat melakukan operasi database seperti koneksi, eksekusi query, pengambilan data, dan transaksi menggunakan sintaks dan paradigma C++ modern. Library ini memungkinkan programmer untuk melakukan berbagai operasi database seperti membuka koneksi ke server MySQL, menjalankan perintah SQL (query), mengambil data dari database (SELECT), menambah, mengubah, atau menghapus data (INSERT, UPDATE, DELETE) dan menangani hasil query menggunakan objek C++

1. Struktur Database



#	Nama	Jenis	Penyortiran	Atribut	Tak Ternilai	Bawaan	Komentar	Ekstra	Tindakan
1	id	int(11)		Tidak	Tidak ada			AUTO_INCREMENT	Ubah Hapus Lainnya
2	judul	varchar(100)	utf8mb4_general_ci	Tidak	Tidak ada				Ubah Hapus Lainnya
3	genre	varchar(50)	utf8mb4_general_ci	Tidak	Tidak ada				Ubah Hapus Lainnya
4	rating	float		Tidak	Tidak ada				Ubah Hapus Lainnya
5	tanggal_rilis	date		Tidak	Tidak ada				Ubah Hapus Lainnya

2. Kamus Data movies

Kolom	Jenis	Tak Ternilai	Bawaan	Tautan ke	Komentar	Media type
id (Utama)	int(11)	Tidak				
judul	varchar(100)	Tidak				
genre	varchar(50)	Tidak				
rating	float	Tidak				
tanggal_rilis	date	Tidak				

Indeks

Nama kunci	Jenis	Unik	Dipadatkan	Kolom	Kardinalitas	Penyortiran	Tak Ternilai	Komentar
PRIMARY	BTREE	Ya	Tidak	id	6	A	Tidak	

Output Program

```
C:\ D:\TOM\Training Ground\C++ CODE\OOP\yhjh\x64\Release\yhjh.exe

Movie Catalog Database
1. View All Movies
2. Add Movie
3. Search Movie
4. Update Movie
5. Delete Movie
6. Exit
Choose option:
```

Gambar 1.1 Jendela Utama

```
Choose option: 1

Sort Options:
1. Sort by ID (ascending)
2. Sort by ID (descending)
3. Sort by Title (A-Z)
4. Sort by Title (Z-A)
5. Sort by Rating (highest first)
6. Sort by Rating (lowest first)
Choose option:
```

Gambar 1.2 Jendela View All Movie

```
ID      Judul      Genre      Rating      Tanggal Rilis
-----
6  Doraemon Movie  adventure, shounen, slice o...  7  2016-06-15
1  Dragon nest     action, fantasy, romance      8  2014-07-31
3  Home alone     comedy, adventure             7.7 1991-08-16
7  Inside Out     animation, drama, fantasy     8.1 2015-06-19
4  Interstellar   adventure, mystery, sains     8.7 2014-11-06
5  MY Life        slice of life, comedy         5.1 2006-09-29

Movie Catalog Database
1. View All Movies
2. Add Movie
3. Search Movie
4. Update Movie
5. Delete Movie
6. Exit
Choose option:
```

Gambar 1.3 Opsi sorting by title (A-Z)

```
ID      Judul      Genre      Rating      Tanggal Rilis
-----
4  Interstellar   adventure, mystery, sains     8.7 2014-11-06
7  Inside Out     animation, drama, fantasy     8.1 2015-06-19
1  Dragon nest     action, fantasy, romance      8  2014-07-31
3  Home alone     comedy, adventure             7.7 1991-08-16
6  Doraemon Movie  adventure, shounen, slice o...  7  2016-06-15
5  MY Life        slice of life, comedy         5.1 2006-09-29

Movie Catalog Database
1. View All Movies
2. Add Movie
3. Search Movie
4. Update Movie
5. Delete Movie
6. Exit
Choose option:
```

Gambar 1.4 Opsi sorting by rating (mulai terbesar)

```

Movie Catalog Database
1. View All Movies
2. Add Movie
3. Search Movie
4. Update Movie
5. Delete Movie
6. Exit
Choose option: 2

Enter title: Psychology
Enter genre: misteri, adult, action
Enter rating (0-10): 7.3
Enter release date (YYYY-MM-DD): 2001-09-13
Movie added successfully!

```

Gambar 1.5 Opsi Add Movie (Insert to database)

```

Search Options:
1. Search by ID
2. Search by Title
Choose option: 1

Enter movie ID: 3

```

ID	Judul	Genre	Rating	Tanggal Rilis
3	Home alone	comedy, advanture	7.7	1991-08-16

Gambar 1.6 Opsi search Movie berdasarkan ID

```

Choose option: 3

Search Options:
1. Search by ID
2. Search by Title
Choose option: 2

Enter movie title: Psychology

```

ID	Judul	Genre	Rating	Tanggal Rilis
8	Psychology	mysteri, adult, action	7.3	2001-09-13

Gambar 1.7 Opsi search Movie berdasarkan Judul

```

Choose option: 4

Enter movie ID to update: 8

Current title: Psychology
New title (press enter to keep): Psychology Adam Smith
Current genre: misteri, adult, action
New genre (press enter to keep): misteri, adult
Current rating: 7.3
New rating (press enter to keep): 7.3
Current release date: 2001-09-13
New release date (press enter to keep): 2001-09-13
Movie updated successfully!

```

Gambar 1.8 Opsi Update Movie

```

Movie Catalog Database
1. View All Movies
2. Add Movie
3. Search Movie
4. Update Movie
5. Delete Movie
6. Exit
Choose option: 5

Enter movie ID to delete: 1
Movie deleted successfully!

```

Gambar 1.9 Opsi Delete Movie

ID	Judul	Genre	Rating	Tanggal Rilis
3	Home alone	comedy, adventure	7.7	1991-08-16
4	Interstellar	adventure, mystery, sains	8.7	2014-11-06
5	MY Life	slice of life, comedy	5.1	2006-09-29
6	Doraemon Movie	adventure, shounen, slice o...	7	2016-06-15
7	Inside Out	animation, drama, fantasy	8.1	2015-06-19
8	Psychology Adam Sm...	mysteri, adult	7.3	2001-09-13

Gambar 1.10 Data movies saat ini

Deskripsi Program

1. Alur Program

Program ini merupakan aplikasi katalog film berbasis C++ yang terhubung dengan database MySQL. Saat dijalankan, program pertama-tama melakukan koneksi ke database MySQL lokal dengan skema bernama “tugas_akhir”. Setelah koneksi berhasil, akan ditampilkan menu utama yang berisi beberapa pilihan, seperti melihat semua film, menambahkan film baru, mencari film, mengubah data film, menghapus film, dan keluar dari program.

Jika pengguna memilih untuk melihat semua film, maka data akan diambil dari database dan ditampilkan dalam bentuk tabel. Sebelum ditampilkan, pengguna diberi pilihan untuk mengurutkan data berdasarkan ID, judul, atau rating secara naik (ascending) maupun turun (descending). Jika pengguna memilih menambahkan film, mereka akan diminta untuk mengisi judul, genre, rating (antara 0–10), dan tanggal rilis dengan format YYYY-MM-DD. Program akan memvalidasi semua input tersebut agar sesuai format yang diharapkan. Setelah data dinyatakan valid, maka akan disimpan ke dalam database.

Untuk mencari film, pengguna dapat memilih ingin mencari berdasarkan ID atau judul. Pencarian berdasarkan ID dilakukan dengan metode binary search setelah data diurutkan berdasarkan ID, sedangkan pencarian berdasarkan judul dilakukan dengan metode linear search yang tidak membedakan huruf besar dan kecil. Jika pengguna memilih mengubah data film, mereka akan diminta untuk memasukkan ID film yang ingin diperbarui. Jika ID ditemukan, pengguna dapat mengganti data film tersebut dengan informasi baru. Sedangkan untuk menghapus film, pengguna cukup memasukkan ID film yang ingin dihapus, dan data tersebut akan dihapus dari database jika ID tersebut valid. Program akan terus berjalan dalam sebuah loop hingga pengguna memilih opsi keluar (Exit).

2. Struktur Program

Pada bagian awal program, terdapat:

- 2.1 Include library standar C++, seperti iostream, iomanip, dan string.
- 2.2 Library MySQL (mysql.h) untuk menangani koneksi dan query ke database.
- 2.3 Inisialisasi objek MYSQL, MYSQL_ROW, dan MYSQL_RES untuk keperluan koneksi dan pengolahan data dari MySQL.

Kemudian terdapat fungsi untuk validasi dan utilitas seperti:

- 2.4 isValidDate() → Mengecek apakah format tanggal yang dimasukkan valid (YYYY-MM-DD).
- 2.5 toLower() → Mengubah string menjadi huruf kecil semua, berguna untuk pencarian yang tidak case-sensitive.

2.6 validateRating() dan validateDate() → Memastikan input user sesuai ketentuan (misalnya rating antara 0–10, tanggal valid).

Dan terdapat fungsi lain seperti:

2.7 lihatFilm() → Menampilkan semua film dari database, dengan fitur pengurutan berdasarkan kolom tertentu dan arah sort.

2.8 tambahFilm() → Menambahkan entri film baru ke database dengan validasi input.

2.9 ubahFilm() → Memperbarui data film berdasarkan ID.

2.10 hapusFilm() → Menghapus data film dari database berdasarkan ID.

2.11 koneksiDatabase() → Membuat koneksi awal ke server MySQL dan memilih database.

2.12 tampilkanMenu() → Menampilkan opsi menu ke user dan menerima input

2.13 pilihan.input() → Fungsi pembantu untuk mengambil input string dari user.

3. Koneksi database

Sebelum program bisa mengakses atau memanipulasi data film, hal pertama yang dilakukan adalah terhubung ke database MySQL. Program menggunakan library mysql.h dari MySQL Connector C++.

3.1 Inisialisasi koneksi menggunakan mysql_init().

3.2 Membuka koneksi ke server MySQL dengan mysql_real_connect() dan harus memasukkan parameter seperti:

3.2.1 host (biasanya localhost jika dijalankan di komputer sendiri),

3.2.2 username (contoh: root),

3.2.3 password (bisa dikosongkan jika belum diatur),

3.2.4 nama database (misal: db_film).

3.3 Jika koneksi berhasil, maka program akan menampilkan pesan berhasil dan melanjutkan ke menu utama.

3.4 Jika gagal, maka akan muncul pesan error, dan program akan berhenti.

4. Implementasi algoritma sorting dan searching

4.1 Algoritma Sorting

Program ini menggunakan **Selection Sort** untuk mengurutkan daftar film berdasarkan kolom tertentu yang dipilih user, misalnya berdasarkan *judul*, *genre*, *tanggal rilis*, atau *rating*. Proses ini terjadi saat user memilih opsi untuk melihat semua film, dan ingin melihatnya dalam urutan tertentu (naik atau turun).

```
void sortById(vector<Movie>& movies, bool ascending = true) {
    int n = movies.size();
    for (int i = 0; i < n - 1; i++) {
        int ext = i;
        for (int j = i + 1; j < n; j++) {
            if (ascending ? (movies[j].id < movies[ext].id) :
(movies[j].id > movies[ext].id)) {
                ext = j;
            }
        }
    }
}
```

```

        swap(movies[i], movies[ext]);
    }
}

```

4.2 Binary dan Linear Search

Binary Search hanya bisa dilakukan pada data yang sudah terurut. Cara kerjanya seperti berikut:

- 4.2.1 Pertama mengambil elemen tengah.
- 4.2.2 Jika sama dengan data yang dicari maka proses selesai.
- 4.2.3 Jika data yang dicari lebih kecil, fokus ke separuh kiri.
- 4.2.4 Jika lebih besar, fokus ke separuh kanan.
- 4.2.5 Dan ulangi proses ini sampai ketemu atau data tidak ditemukan.

Di program ini Binary Search digunakan untuk mencari **judul film** setelah daftar difilter dan diurutkan. Implementasinya kira-kira seperti ini:

```

Movie* binarySearchById(vector<Movie>& movies, int targetId) {
    int left = 0;
    int right = movies.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (movies[mid].id == targetId) {
            return &movies[mid];
        }

        if (movies[mid].id < targetId) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    return nullptr;
}

```

Berikutnya yaitu linear Search yang merupakan metode paling dasar, dengan mengecek setiap elemen satu per satu dari awal sampai akhir. Jika cocok dengan yang dicari, langsung kembalikan posisi. Linear Search cocok digunakan untuk data **belum terurut** atau pencarian cepat tanpa pengurutan.

Di Program Ini digunakan ketika user ingin mencari film dengan judul tertentu atau ketika data tidak diurutkan.

```

Movie* searchByTitle(vector<Movie>& movies, const string& title) {
    for (auto& movie : movies) {
        string movieTitleLower = movie.judul;
        string searchTitleLower = title;
    }
}

```

```
        transform(movieTitleLower.begin(), movieTitleLower.end(),
movieTitleLower.begin(), ::tolower);
        transform(searchTitleLower.begin(),
searchTitleLower.end(), searchTitleLower.begin(), ::tolower);
```

```
        if (movieTitleLower.find(searchTitleLower) !=
string::npos) {
            return &movie;
        }
        return nullptr;
    }
```


Daftar Pustaka

Weiss, M.A. (2014). *Data Structures and Algorithm Analysis in C++* (4th ed.). Pearson.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.

Deitel, H.M., & Deitel, P.J. (2012). *C++ How to Program* (8th ed.). Pearson Education.