TELEVISION MANAGEMENT SYSTEM

Done by: Nila, Krishna Shree, Samyukta

PROBLEM STATEMENT

- Managing television channel data manually can lead to difficulty in tracking shows, scheduling, TRP ratings, costs, cast associations, marketing data, and upcoming replacements.
  This database automates analytics, maintains constraints, and improves decision-making for broadcasting companies.

ABSTRACT

1. **Database:**
    - **PostgreSQL** used to store structured data on channels, shows, genres, cast, episodes, advertisements, marketing, and schedules.
2. **Backend:**
    - **Flask** framework handles communication with the database, executes SQL queries, applies business logic, and processes user requests.
3. **Frontend:**
    - **HTML** for webpage structure.
    - **CSS** for styling and layout.

- **JavaScript** for interactivity, form validation, and dynamic updates.

4. **Workflow:**

   - User actions (e.g., add show, view TRP analytics) → request sent to Flask backend → Flask interacts with PostgreSQL → results returned and displayed on the web interface.

.

Objectives

Project Objectives:

1. **Purpose:**
    - Addresses challenges in managing large volumes of broadcasting data.
2. **Function:**
    - Centralized platform to organize shows, TRP ratings, costs, cast, ads, and schedules.
3. **Analytics:**
    - Provides TRP performance, budgeting insights, and scheduling optimization.

4. **Benefits:**
    - Enhances operational efficiency and supports better decision-making.
5. **Technical Scope:**
    - Demonstrates database concepts, full-stack development, and real-time data interaction.

.

Advantages

1. **Data Accuracy:**
    - Ensures consistent storage using constraints, triggers, and validation.
2. **Automation:**
    - Automates updates like show counts and TRP limits.
3. **Analytics:**
    - Provides insights on viewer engagement and production costs for better planning.
4. **Interface:**

- Built with HTML, CSS, JavaScript (frontend) and Flask (backend) for easy use.

5. **Benefits:**

   - Reduces manual work, prevents data loss, enables quick retrieval, and offers scalability for future needs.

ENTITIES

- Channel
- Genre
- Show
- CastAndCrew
- Episode
- Advertisement
- Marketing
- UpcomingShow

SCHEMA DIAGRAM

- ---------- Channel ----------
- channel_id (PK)
- channel_name
- satellite
- num_of_shows
- location

- ---------- Genre ----------
- genre_id (PK)
- genre_name

SCHEMA DIAGRAM

- ---------- Show ----------
- show_id (PK)
- channel_id (FK)
- genre_id (FK)
- title
- air_time
- director_id (FK to CastAndCrew)

- CastAndCrew
---------- ----------
- castcrew_id (PK)
- name
- role_type (Actor/Director/Producer/Writer)
- show_id (FK)
- role_description

SCHEMA DIAGRAM

- ---------- Marketing ----------
- marketing_id (PK)
- channel_id (FK)
- show_id (FK)
- trp
- cost_of_production

- ---------- Advertisement ----------
- ad_id (PK)
- channel_id (FK)
- company_name
- num_of_ads
- duration

SCHEMA DIAGRAM

- ---------- Episode ----------
- episode_id (PK)
- show_id (FK)
- season_number
- episode_number
- duration
- air_date

- ---------- UpcomingShow ----------
- upcoming_id (PK)
- channel_id (FK)
- title
- replacement_show_id (FK)
- director_id (FK)

Use of Triggers

- **Purpose:**
  - Maintain automatic data integrity and enforce business rules.
- **Functions Implemented:**
  - Prevent TRP values from exceeding 10.
  - Auto-update the number of shows in a channel on add/delete.
  - Store deleted show records for audit tracking.
- **Benefits:**
  - Keeps data consistent, error-free, and secure.
  - Reduces manual work by automating background checks.

Use of Joins

- Combine data from multiple related tables.
- Needed due to normalization across Channel, Show, Genre, Cast, Marketing, and Episodes.
- Used for TRP by genre, director analysis, episode duration, and cost comparison.
- Enables complex analytics and meaningful reporting

Use of Stored Procedures

- Perform complex and repetitive tasks efficiently at the database level.
- Reduce code duplication and improve performance.
- Handle tasks like replacing low-TRP shows, validating data, and generating weekly reports.
- Ensure faster execution, better security, and consistent business rule enforcement.

Frontend and Backend Technologies

Frontend – HTML and CSS Why HTML and CSS?

- HTML (HyperText Markup Language) provides the **structure** and **content** of web pages.
- CSS (Cascading Style Sheets) controls the **appearance**, **layout**, and **visual design**.
- Together, they make the interface **intuitive**, **clean**, and **responsive**.

Uses and Advantages:

- Ensures **user-friendly interaction** for database operations like viewing shows or TRP analytics.
- CSS improves **readability and design consistency** across pages.
- Easy to **integrate with Flask templates (Jinja2)** for dynamic data display.
- **Lightweight and fast-loading**, ensuring quick response times.
- Compatible with **all browsers and devices**, enhancing accessibility.

Backend – Flask Framework

- Why Flask?
- - Lightweight Python web framework that is easy to learn and deploy.
- - Simple yet powerful structure for backend logic.
- - Portable and scalable for both academic and real-world use.

- Uses and Advantages:
- - Acts as a bridge between frontend and PostgreSQL database.
- - Handles requests, executes SQL queries, and returns results dynamically.
- - Supports RESTful APIs for efficient data interaction.
- - Modular, secure, and easy to maintain.

- Integration Summary:
- - HTML/CSS displays input forms and results.
- - Flask processes requests and interacts with the database.
- - Together, they ensure seamless communication and real-time updates

Normalization

Normalization is the process of organizing data in a database to **reduce redundancy** and **improve data integrity**. It divides large tables into smaller ones and establishes relationships between them.

1NF – First Normal Form (Atomic Values)

1. Each column must contain **atomic (indivisible)** values.
2. Each record must be **unique**, identified by a **primary key**.
3. **No repeating groups or arrays** are allowed in any row.
4. Ensures that every field contains **only one value**.

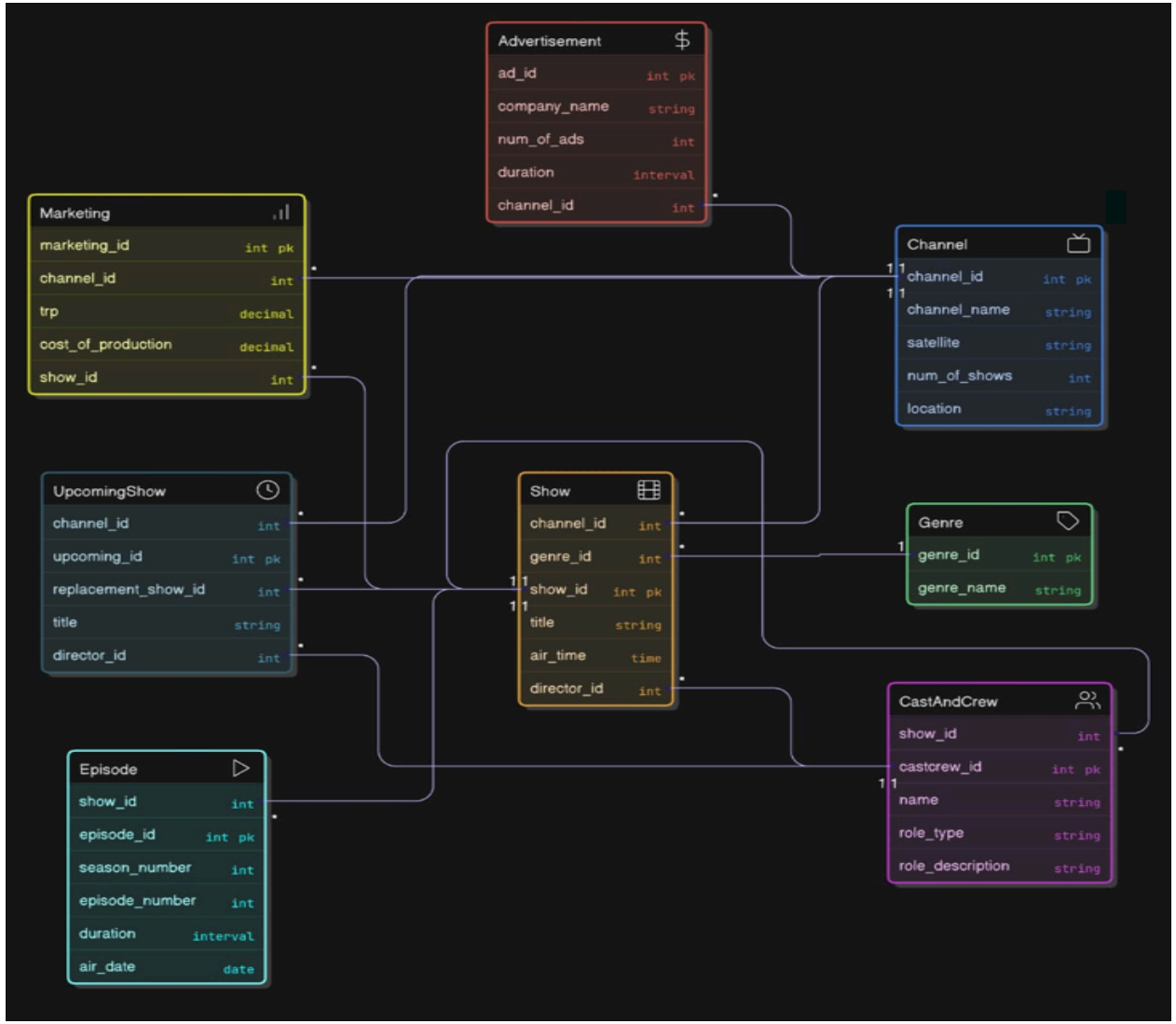
2NF – Second Normal Form (No Partial Dependency)

1. Table must already satisfy **1NF**.
2. Every **non-key attribute** must depend on the **entire primary key**, not just part of it.
3. Removes **partial dependency**, which happens when a non-key attribute depends on **part of a composite key**.
4. Applicable **only if the primary key is composite**.
5. Decomposition step: split the table so that attributes fully depend on their key.
6. Result → eliminates redundant data and improves data consistency.

3NF – Third Normal Form (No Transitive Dependency)

1. Table must already be in **2NF**.
2. **No transitive dependency** should exist between non-key attributes.
    - i.e., a non-key attribute should not depend on another non-key attribute.
3. Each **non-prime attribute** must depend **only on the primary key**.
4. Removes dependency chains like **A** → **B** → **C**, where **C** depends indirectly on **A**.
5. Decomposition ensures **each table stores one kind of fact**.
6. Result → data redundancy minimized and update anomalies avoided.

ER DIAGRAM


Advertisement
ad_id int pk
company_name string
num_of_ads int
duration interval
channel_id int
Marketing
marketing_id int pk
channel_id int
trp decimal
cost_of_production decimal
show_id int
Channel
channel_id int pk
channel_name string
satellite string
num_of_shows int
location string
UpcomingShow
channel_id int
upcoming_id int pk
replacement_show_id int
title string
director_id int
Show
channel_id int
genre_id int
show_id int pk
title string
air_time time
director_id int
Genre
genre_id int pk
genre_name string
CastAndCrew
show_id int
castcrew_id int pk
name string
role_type string
role_description string
Episode
show_id int
episode_id int pk
season_number int
episode_number int
duration interval
air_date date