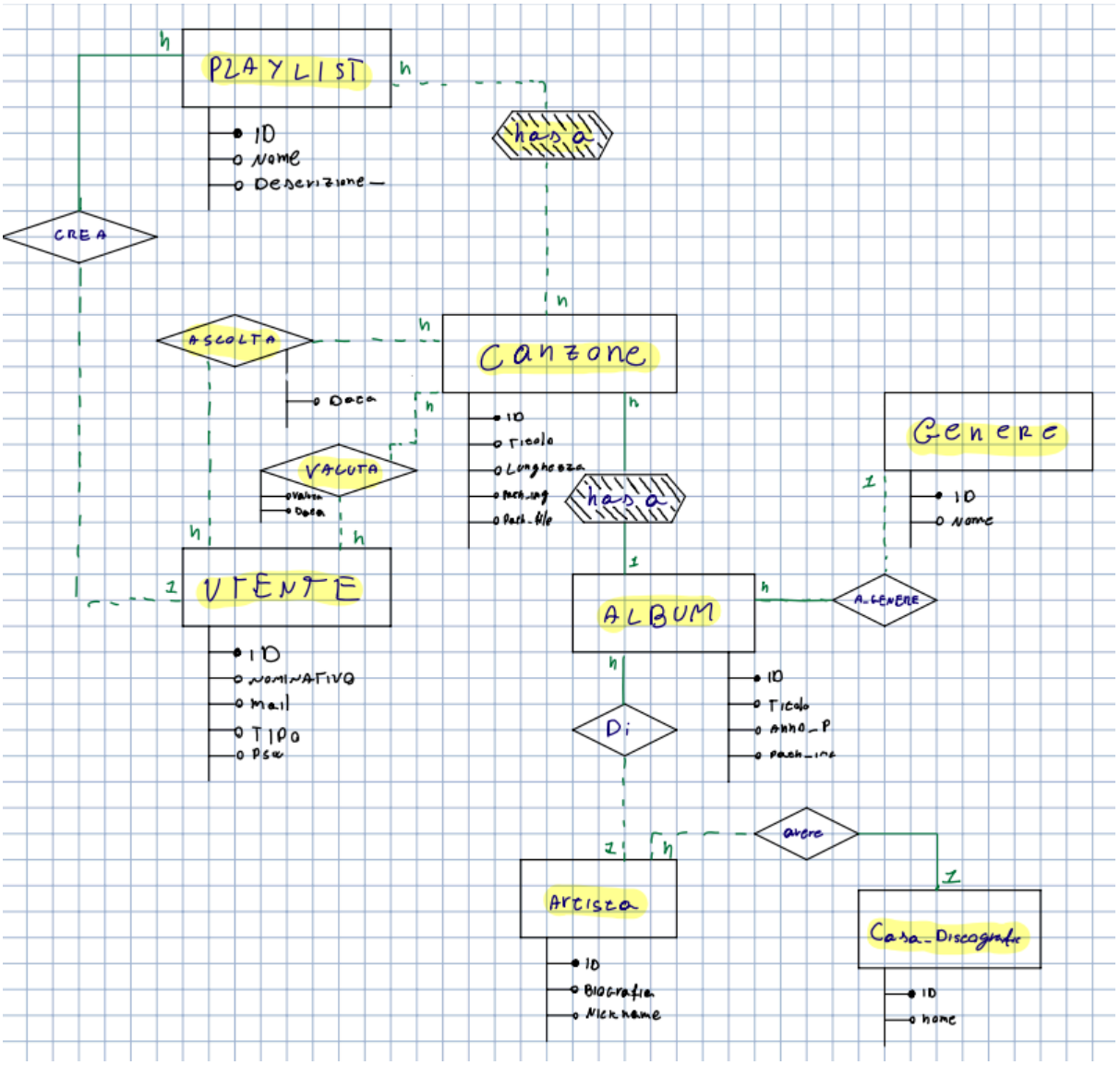
Music4all

Schema ER



Ipotesi Aggiuntive

- Possono esistere playlist vuote
- Non esistono canzoni senza genere
- Le canzoni possono non avere ascolti
- Gli utenti possono non ascoltare canzoni
- Gli album hanno un genere principale
- Ogni album deve essere caricato da un amministratore che ne specificherà l'artista

Query

A)

```
Select path_img, c.titolo, ar.nick, ar.bio
from artisti ar, valuta v, canzoni c, album al
where ar.id = al.id_artista
and c.id_album = al.id
and v.id_canzone = c.id
and al.annoP BETWEEN ? and ? + 10
group by v.id_canzone
having avg(v.valutazione) > 3
```

B)

```
 SELECT c.titolo
FROM canzoni c
WHERE c.id NOT IN (
   SELECT v.id_canzone
   FROM valuta v
   WHERE (v.valutazione = 1 OR v.valutazione = 2)
     AND DATE(v.data) = CURRENT_DATE()
)
```

C)

```
SELECT c.titolo, c.lung
FROM Canzoni c, Ascolta a
WHERE a.id_utente = 1 AND c.id = a.id_canzone
GROUP BY c.id, c.titolo, c.lung
ORDER BY COUNT(*) DESC
LIMIT 1;
```

- **Indice**

---

1 Analisi dei Requisiti

1.1 Requisiti funzionali

- Gestione Account Utenti:
  - L'utente può decidere se abbonarsi con un pagamento ricorrente mensilmente o usare l'app gratuitamente con le pubblicità.
  - Sistema di login per amministratori e utenti. Gli amministratori avranno credenziali già pre-inserite nel database.
  - Sistema di registrazione per gli utenti.
  - Sistema di logout per gli utenti.
  - Autenticazione basata su sessioni e gestione delle autorizzazioni.

- Gestione Album e canzoni:
  - L'Utente può visualizzare le informazioni generali sugli album, informazioni generali sull'artista/i (Nickname, biografia), informazioni sulla casa discografica di appartenenza nonché il suo genere.
  - L'utente può visualizzare le caratteristiche delle canzoni come: titolo, valutazione, durata, tutto durante il suo ascolto.
  - Gli utenti volendo possono valutare le canzoni con un voto da 1 a 5.
  - Gli amministratori possono creare nuovi album inserendo i vari campi tra cui le canzoni (Tramite l'inserimento della path di dove è localizzato l'audio nel server)
  - Un utente può ascoltare le canzoni presenti all'interno dell'album.
  - Un utente può ricercare un album tramite la barra di ricerca situata nella home.

- Gestione delle playlist:
  - Un utente del sito può CREARE un playlist cliccando nell'header la funzione "crea playlist". Questo verrà poi reindirizzato in una pagina che gli permetterà di inserire canzoni già presenti nell'app, le quali andranno ad appartenere nella nuova playlist.

    - Un utente del sito può ascoltare una playlist generata automaticamente dall'applicazione seguendo le valutazioni precedentemente inserite dall'utente.
    - Un utente può ascoltare le playlist precedentemente create da lui.
    - Un utente può modificare la playlist precedentemente creata scegliendo se togliere o aggiungere nuove canzoni.

1.2 Requisiti non funzionali

- **Validazione lato client**: i dati inseriti dall'utente devono essere verificati direttamente sul lato client tramite controlli con script HTML (campi obbligatori).
- **Prestazioni:** il caricamento e la riproduzione delle tracce musicali devono avvenire con latenza minima, garantendo uno streaming fluido anche su connessioni medie.
- **Affidabilità e disponibilità:** il sistema deve garantire un tempo di attività elevato, riducendo al minimo i periodi di inattività.
- **Manutenibilità:** il codice e l'architettura devono essere strutturati in modo da permettere facile aggiornamento, debugging e miglioramenti futuri.
- **Usabilità:** l'interfaccia utente deve essere **intuitiva e user-friendly**, sia per utenti gratuiti che per quelli a pagamento.
- **Sicurezza**
    - I dati sensibili devono essere inviati al server in modo sicuro, usando il metodo **POST**.
    - L'accesso alle funzionalità riservate (come gestione account e pagamenti) deve avvenire tramite **autenticazione sicura** Infatti le comunicazioni di dati avvengono tramite una api di paypal o applicazioni di scambio di denaro digitale le quali sono criptate e sicure.

---

2 Progettazione.

Livello di Presentazione (Client)

Rappresenta l'interfaccia visibile all'utente finale, ovvero il **browser web**. È qui che avviene l'interazione con l'utente, sia gratuito, sia premium, sia d'amministrazione.

- **JavaScript**: Per gestire l'interattività (aggiunta/rimozione brani da playlist, player musicale, aggiornamenti dinamici tramite AJAX).
- **HTML**: Per la creazione e la gestione della struttura della pagina.

Livello Logico/Applicazione (Web Server)

Questo livello si occupa della logica applicativa e dell'elaborazione delle richieste provenienti dal client. È qui che viene gestito il flusso tra utente e dati.

Tecnologie utilizzate:

- **PHP**: linguaggio server-side che gestisce operazioni fondamentali come autenticazione utenti, accesso riservato, gestione playlist, salvataggio delle valutazioni;

- **Apache**: web server che interpreta il codice PHP, riceve le richieste HTTP e invia risposte dinamiche al client.

Livello Dati (Database Server)

Questo livello è responsabile della **gestione dei dati persistenti** dell'applicazione: utenti, tracce, album, ascolti, valutazioni, playlist, artisti, generi, ecc.

Tecnologia utilizzata:

- **MySQL**: sistema di gestione di basi di dati relazionali , scelto per la sua affidabilità ed efficienza. Utilizza il linguaggio **SQL** per interrogazioni, aggiornamenti e inserimenti.

Tecnologie utilizzate

Web Server - Apache

Il progetto utilizza **Apache HTTP Server**, un server web modulare e multipiattaforma, adatto alla gestione di applicazioni dinamiche. Apache riceve le richieste HTTP dai client (browser), interpreta il codice PHP lato server e restituisce all'utente contenuti HTML aggiornati in tempo reale.

La sua architettura modulare consente di gestire in modo efficiente operazioni come la gestione di sessioni utente, i pagamenti sicuri e lo streaming audio.

Database - MySQL

Per la gestione delle informazioni relative a utenti, playlist, artisti, canzoni, valutazioni e ascolti, è stato scelto **MySQL,** che consente di:

- Archiviare in modo strutturato i dati del sistema;

- Eseguire interrogazioni complesse in linguaggio **SQL**;

- Garantire **integrità referenziale** tra le tabelle;

- Ottimizzare le performance di accesso ai dati.

Linguaggio lato server - PHP

Il linguaggio utilizzato per la logica lato server è **PHP**, pensato per la creazione di applicazioni web dinamiche.

Il codice PHP si occupa di:

- Ricevere ed elaborare le richieste degli utenti;
- Accedere e modificare i dati presenti nel database MySQL;
- Gestire l'autenticazione, le sessioni e le funzioni riservate agli utenti registrati o paganti.
- Generare pagine HTML personalizzate in base al profilo utente o alle sue interazioni (es. traccia più ascoltata, playlist consigliate).

Linguaggi lato client

HTML – Struttura dei contenuti

HTML (HyperText Markup Language) è il linguaggio utilizzato per strutturare il contenuto delle pagine web. Ogni pagina di Music4All è costruita con elementi HTML come titoli, immagini (es. copertine album), pulsanti di interazione, form di ricerca, elenchi di brani ecc.

CSS – Stile e impaginazione

CSS viene utilizzato per definire l'aspetto grafico delle pagine. Permette di:

- rendere l'interfaccia moderna e responsiva.
- personalizzare colori, layout, font e immagini (es. copertine e player),
- distinguere l'esperienza utente tra versione gratuita e premium.

JavaScript – Interattività

JavaScript è il linguaggio di programmazione lato client che rende l'interfaccia interattiva e dinamica. Su Music4All viene utilizzato per:

- Aggiornare i contenuti in tempo reale (es. aggiunta/rimozione di brani da playlist senza ricaricare la pagina),
- Validare i form (es. login, registrazione),
- Gestire player audio e controlli multimediali in modo asincrono.

Altri strumenti utilizzati

- **Editor di codice**: Visual Studio Code, pannello di controllo e XAMPP.
- **Prova client:** Browser (Chrome, Microsoft Edge, Opera)

2.2 Struttura Database

L'analisi dei requisiti ha spostato il focus della progettazione in una parte ben delineata ed inserita in un progetto più ampio dello sviluppo di un'applicazione di streaming di audio.

La progettazione porterà allo sviluppo delle seguenti tabelle

- **Utenti(id, nome, cognome, email, tipo, psw, id_artista*-)**
- **Playlist(id, nome, descr-, id_utente*)**
- **Album(id, titolo, annoP, path_img, id_genere*, id_artista*)**
- **Canzoni(id, titolo, lung, id_album*)**
- **Generi(id, nome)**
- **Artisti(id, nick, bio, id_casa_d*)**
- **Case_discografiche(id, nome)**
- **Valuta(id_utente*, id_canzone*, valutazione, data)**
- **Ascolta(id_ascolto, id_utente*, id_canzone*, data)**
- **Contiene(id_playlist*, id_canzone*)**
-

Le tabelle verranno create con i seguenti script SQL

- **Utenti**

```
CREATE TABLE Utenti (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(25) NOT NULL CHECK (nome REGEXP '^[A-Z][a-z]*$'),
  cognome VARCHAR(25) NOT NULL CHECK (cognome REGEXP '^[A-Z][a-z]*$'),
  email VARCHAR(50) UNIQUE NOT NULL,
  tipo INT NOT NULL,
        //0 -> amministratore
        //1 -> non abbonato
        //2 -> abbonato
  psw VARCHAR(15) NOT NULL
);
```

- **Generi**

```
CREATE TABLE Generi (
  id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    nome VARCHAR(30) NOT NULL
);
```

- **Case Discografiche**

```
CREATE TABLE Case_discografiche (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL
);
```

- **Artisti**

```
CREATE TABLE Artisti (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nick VARCHAR(25) NOT NULL,
    bio TEXT NOT NULL,
    id_casa_d INT,
    FOREIGN KEY (id_casa_d) REFERENCES Case_discografiche(id)
        ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- **Album**

```
CREATE TABLE Album (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titolo VARCHAR(30) NOT NULL,
    annoP YEAR NOT NULL,
    path_img VARCHAR(100) NOT NULL,
    id_genere INT,
    id_artista INT,
    FOREIGN KEY (id_genere) REFERENCES Generi(id)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (id_artista) REFERENCES Artisti(id)
        ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- **Canzoni**

```
CREATE TABLE Canzoni (
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
    titolo VARCHAR(30) NOT NULL,
    lung TIME NOT NULL,
    id_album INT,
    FOREIGN KEY (id_album) REFERENCES Album(id)
        ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- **Playlist**

```
CREATE TABLE Playlist (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(30) NOT NULL,
    `desc` TEXT,
    id_utente INT,
    FOREIGN KEY (id_utente) REFERENCES Utenti(id)
        ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- **Valuta**

```
CREATE TABLE Valuta (
    id_utente INT,
    id_canzone INT,
    valutazione TINYINT NOT NULL CHECK (valutazione BETWEEN 1 AND 5),
    data DATE NOT NULL,
    PRIMARY KEY (id_utente, id_canzone),
    FOREIGN KEY (id_utente) REFERENCES Utenti(id)
        ON UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (id_canzone) REFERENCES Canzoni(id)
        ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- **Ascolta**

```
CREATE TABLE Ascolta (
    id_ascolto INT AUTO_INCREMENT PRIMARY KEY,
```

```
    id_utente INT,
    id_canzone INT,
    data DATETIME NOT NULL,
    FOREIGN KEY (id_utente) REFERENCES Utenti(id)
      ON UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (id_canzone) REFERENCES Canzoni(id)
      ON UPDATE CASCADE ON DELETE RESTRICT
);
```

- **Contiene**

```
CREATE TABLE Contiene (
    id_playlist INT,
    id_canzone INT,
    PRIMARY KEY (id_playlist, id_canzone),
    FOREIGN KEY (id_playlist) REFERENCES Playlist(id)
      ON UPDATE CASCADE ON DELETE RESTRICT,
    FOREIGN KEY (id_canzone) REFERENCES Canzoni(id)
      ON UPDATE CASCADE ON DELETE RESTRICT
);
```

2.3 Struttura Filesystem su Server

- Music4all
  - Conf
    - Db_config.php
  - Templates
    - Header.php
    - ascolta_temp.php
    - valuta.php
    - tue_playlist_temp.php
    - modifica_playlist_temp.php
    - home_temp.php
    - home_adm_temp.php
    - crea_playlist_temp
    - aggiungi_a_album_temp.php
    - crea_album_temp.php
    - dettagli_album.php

- Css
  - Style.css
- Php
  - pagamento.php
  - logincheck.php
  - aggiungi_canzone.php
  - rimuovi_canzone.php
  - regitra_ascolti.php
  - logout.php
  - cerca.php
  - elimina.php
- img
  - pop vibes.jpg
  - classical.jpg
  - aday.jpg
- Index.php
- pagamento_base.php
- ascolta.php
- aggiungi_a_album.php
- tue_playlist.php
- modifica_playlist.php
- home_adm.php
- home.php
- dettagli_album.php
- crea_playlist.php
- crea_album.php
- registra.php

Connessione al Database

Il file db_config.php ha lo scopo di **stabilire una connessione tra l'applicazione web e il database MySQL**.

Queste quattro variabili contengono le **informazioni necessarie per connettersi al database**:

- `$servername = "localhost";` → indica che il database si trova sulla **stessa macchina (localhost)** del server web.

- `$username = "root";` → è l'**utente del database**, in questo caso l'amministratore (`root`).

- `$password = "";` → è la **password dell'utente**;

- `$dbname = "music4all";` → nome del **database da utilizzare** per l'applicazione.

SECONDA PARTE

I.

Per garantire la sicurezza da eventuali attacchi al DB, è necessario innanzitutto mettere delle credenziali di accesso robuste, in modo che nel caso di un eventuale attacco di brute force, sia comunque molto difficile arrivare a bucare il DB.

Gli attacchi SQL injection sono attacchi estremamente semplici e veloci, con cui inserendo una stringa nell'input della password o dello username del tipo '1 OR '1' = "1" ' il db ci autentica ad accedere perché il controllo nella ricerca viene soddisfatto, allora si previene utilizzando i punti interrogativi nella query (esempio: … "where id = ?") e la funzione di PHP "bind_param" che associa ai punti interrogativi, i corrispettivi valori che si vogliono controllare.

Importantissimo è anche utilizzare i metodi HTTP corretti quando si passano dei valori tra un file PHP e un altro (esempio: utilizzare la GET per mandare le password è sbagliato, perché si vede in chiaro sulla URL, si deve in quel caso usare la POST), ma la sicurezza non è ancora garantita, perché i messaggi HTTP non sono criptati.

HTTPS serve proprio a codificare i messaggi HTTP in modo che non siano leggibili da terze parti, utilizza SSL come protocollo sicuro per la codifica e decodifica dei messaggi.

II.

- L'obbiettivo è permettere a chi necessita di questi dati di accederci tramite le api e ottenere un json
    - L'api che si andrebbe ad usare è **GET /api/statistiche_ascolto.php?artista_id=ID**

➢ Un esempio di risposta potrebbe essere il seguente:

{

```
"artista_id": 12,
"nome_artista": "Nome artista",
"tracce": [
 {
  "id_traccia": 1,
  "titolo": "Titolo Traccia",
  "utenti_distinti": 134
 },
 ...
]
}
```

Il codice che genererebbe questo codice

```
<?php
// Imposta il tipo di contenuto della risposta come JSON
header("Content-Type: application/json");

require("../conf/db_config.php");

// Recupera l'id dell'artista passato come parametro GET
$id_artista = isset($_GET['id_artista']) ? intval($_GET['id_artista']) : 0;

if ($id_artista <= 0) {
    echo json_encode(["errore" => "ID artista non valido"]);
    exit;
}

// Query per ottenere, per ogni canzone dell'artista, il numero di utenti distinti che l'hanno ascoltata
$sql = "
    SELECT C.id AS id_canzone, C.titolo, COUNT(DISTINCT A.id_utente) AS num_utenti
    FROM Canzoni C
```

```
    JOIN Album Al ON C.id_album = Al.id
    LEFT JOIN Ascolta A ON C.id = A.id_canzone
    WHERE Al.id_artista = ?
    GROUP BY C.id, C.titolo
    ORDER BY C.titolo
";

$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $id_artista);
$stmt->execute();
$result = $stmt->get_result();

// Prepara l'array di risposta
$risposta = [];
while ($row = $result->fetch_assoc()) {
    $risposta[] = $row;
}

// Restituisce il risultato in formato JSON
echo json_encode($risposta, JSON_PRETTY_PRINT);

// Chiude la connessione
$stmt->close();
$conn->close();
?>
```

- Quindi una qualsiasi casa discografica può effettuare una richiesta di questo genere:

  http://tuosito/canzoni_artista.php?artista=3 e otterrà risultati di questo tipo

  [

  {

  "id_canzone": 7,

  "titolo": "Onda Sonora",

```
    "num_utenti": 12
   },
   {
    "id_canzone": 8,
    "titolo": "Vento",
    "num_utenti": 0
   }
  ]
```


III.

- HTML (HyperText Markup Language)
  - È il linguaggio di struttura di ogni pagina web.
  - Permette di definire gli elementi dell'interfaccia utente, come pulsanti, form, testi, immagini, tabelle.
  - Non ha logica dinamica: definisce cosa appare, ma non come cambia
- CSS (a supporto, non richiesto nella traccia)
  - Anche se non richiesto, è bene menzionarlo brevemente.
  - Permette di personalizzare lo stile: colori, layout, spaziature, animazioni.
- JavaScript
  - Linguaggio di programmazione lato client.
  - Permette di interagire dinamicamente con l'HTML e di modificare l'interfaccia utente in tempo reale, senza ricaricare la pagina.
  - Esempi: cambiare un tema, nascondere/mostrare elementi, validare input, aggiornare contenuti dinamicamente.
- PHP (Hypertext Preprocessor)
  - Linguaggio di programmazione lato server.
  - Utilizzato per gestire: autenticazione utenti, generazione dinamica dell'interfaccia HTML, salvataggio e recupero dati da database (es. MySQL), perrsonalizzazione in base alle preferenze salvate di ciascun utente.

➢ **Esempio pratico con GET + HTML + PHP**
  - Obiettivo: Visualizzare un messaggio personalizzato in base al nome utente passato nell'URL.

HTML + PHP (file: saluto.php)

```
<!-- saluto.php -->

<?php

 // Controllo se il parametro GET 'nome' è presente
```

```
 $nome = isset($_GET['nome']) ? htmlspecialchars($_GET['nome']) : 'ospite';
?>

<!DOCTYPE html>
<html>
<head>
 <title>Saluto personalizzato</title>
</head>
<body>
 <h1>Ciao, <?= $nome ?>!</h1>
 <p>Benvenuto nel nostro sito personalizzato per te.</p>
</body>
</html>
```

- La personalizzazione dell'interfaccia utente è fondamentale per offrire un'esperienza coinvolgente e su misura all'utente. HTML definisce la struttura, JavaScript gestisce le interazioni dinamiche lato client, mentre PHP elabora i dati e genera interfacce personalizzate lato server. L'uso combinato di queste tecnologie permette di costruire siti web moderni, interattivi e adattabili alle preferenze di ciascun utente. Tutte queste

IV.

DF

1. codCorso -> desCorso, codInsegnante, nomeInsegnante (per come è scritto nel testo)

2. codSala -> desSala, codCorso, data, oraInizio, oraFine

3. codAlunno -> nomeAlunno

PK (codCorso, codSala, codAlunno, data)

1FN

● Non ci sono attributi multivalore o composti

no perche abbiamo nomeInsegnante e nomeAlunno con nome e cognome

● I valori di ciascun campo appartengono allo stesso dominio

● Tutte le tuple presentano lo stesso numero di colonne

● L’ordine delle colonne è irrilevante

● Si elegge come chiavi primarie: (codCorso, codSala, codAlunno, data)

tab(CodCorso(PK), desCorso, codInsegnante, nomeInsegnante, cognomeInsegnante, data, oraInizio, oraFine, codSala(PK), desSala, codAlunno(PK), nomeAlunno, cognomeAlunno)

2FN

non sono in 2FN la 1 e 3 riga

palestra(CodCorso(PK)*, data, oraInizio, oraFine, codSala(PK), desSala, codAlunno(PK)*)

Insegnante(codCorso(PK), codInsegnante, nomeInsegnante, cognomeInsegnante, desCorso)

alunno(CodAlunno(PK), nomeAlunno, cognomeAlunno)

3FN

La tabella è già in terza forma normale(perchè la DF 2 non rispetta le richieste della 3FN)

E’ in BCFN in quanto nel database non esistono determinanti che non siano chiave

(perche nelle dipendeze funzionali la parte di destra(cioe la determinante) e una PK o chiave candidata)

La tabella è in boyce-code perchè la 2° riga delle DF ha codCorso e data.