**SVKM’s NMIMS**

**School of Technology Management & Engineering (Indore Campus)**

**Computer Engineering Department (B Tech/MBATech CE and B Tech AIDS Sem IV)**

**Database Management System**

**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | BTech CE | |
| Semester | 4th Sem | |
| Name of the Project: | Wonfour- the music and art workspace | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| B2 | D096 | Vihaan Singh |
| B2 | D032 | Harshit Shah |
|  |  |  |
| Date of Submission: | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| D032 | Harshit Shah |  |
| D096 | Vihaan Singh |  |

**Github link of your project:**

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

**Rubrics for the Project evaluation:**

|  |  |
| --- | --- |
| First phase of evaluation:  Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) | 10 marks |
| Final phase of evaluation  Implementation, presentation and viva, Self-Learning and Learning Beyond classroom | 10 marks |

**Project Report**

**Wonfour- the music art workspace**

**By**

**Harshit Shah, D032**

**Vihaan Singh, D096**

**Course: DBMS**

**AY: 2024-25**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Topic** | **Page no.** |
| **1** | Storyline |  |
| **2** | Components of Database Design |  |
| **3** | Entity Relationship Diagram |  |
| **4** | Relational Model |  |
| **5** | Normalization |  |
| **6** | SQL Queries |  |
| **7** | Learning from the Project |  |
| **8** | Project Demonstration |  |
| **9** | Self-learning beyond classroom |  |
| **10** | Learning from the project |  |
| **8** | Challenges faced |  |
| **9** | Conclusion |  |

**I. Storyline**

When a user visits the website, they can either log in or create an account. Upon registration, the backend securely stores their information in the user database with encrypted passwords using hashing (e.g., bcrypt). A unique user ID is assigned, and a session or token (e.g., JWT) is created for authentication.

Once logged in, users are redirected to their dashboard. If they choose to upload a music sample for sale, the backend processes the uploaded file, stores it on a secure cloud storage (e.g., AWS S3), and records metadata such as title, genre, price, and uploader ID in the database. The sample becomes visible to other users for browsing and purchasing.

When a user decides to purchase a sample, the backend verifies payment through a secure payment gateway (e.g., Stripe or PayPal). Upon successful payment, the system generates a downloadable link for the buyer and updates the seller’s earnings. The transaction details are logged in the order history.

Both buyers and sellers can view their activity, including uploads, purchases, earnings, and downloads. Engagement and transaction data contribute to artist rankings. The backend continuously ensures data consistency, user authorization, and secure access throughout the entire process.

**II. Components of Database Design**

**1. Users-**

user\_id, username, email, password\_hash

**2. Samples-**

sample\_id, title, genre, price

**3. Uploads-**

upload\_id , file\_url, upload\_date

**4. Downloads-**

download\_id , download\_date

**5. Transactions-**

transaction\_id , amount, transaction\_date

**6. Top\_Artists**

artist\_id, total\_uploads, total\_sales\_amount, total\_downloads, last\_updated

**Relationships, Cardinality, and Participation-**

**1. Users – Uploads**

Relationship: One user can upload many samples.

Cardinality: 1:N

Participation: Total on Uploads (each upload must be by a user), partial on Users.

**2. Samples – Uploads**

Relationship: One sample can be uploaded once (in this schema), and each upload is of one sample.

Cardinality: 1:1

Participation: Total on Uploads, total on Samples. (Assumes one upload per sample.)

**3. Users – Downloads**

Relationship: One user can download many samples.

Cardinality: 1:N

Participation: Total on Downloads, partial on Users.

**4. Samples – Downloads**

Relationship: One sample can be downloaded many times by many users.

Cardinality: M:N resolved via Downloads table

Participation: Total on Downloads, partial on Samples.

**5. Users – Transactions**

Relationship: One user (as buyer) can make many transactions.

Cardinality: 1:N

Participation: Total on Transactions, partial on Users.

**6. Samples – Transactions**

Relationship: One sample can be sold many times.

Cardinality: 1:N

Participation: Total on Transactions, partial on Samples.

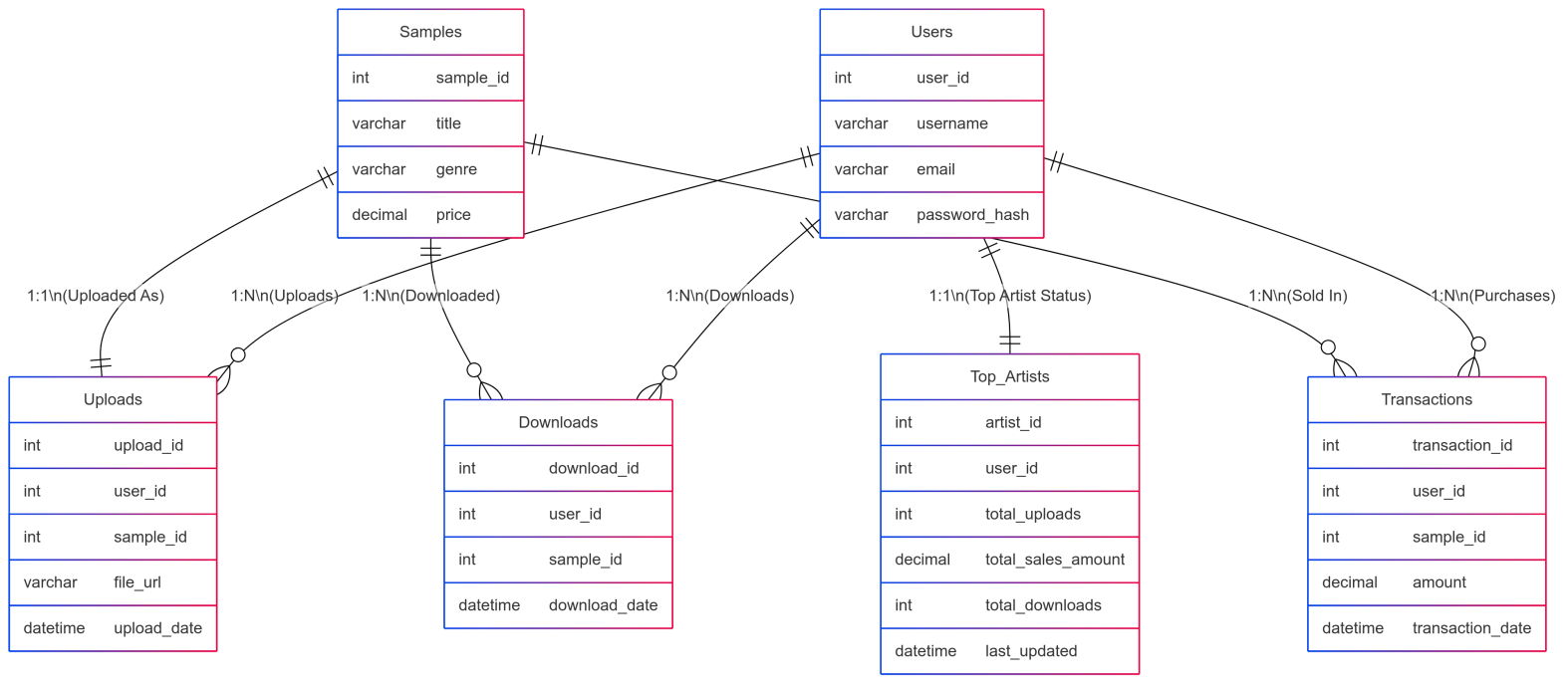
**7. Users – Top\_Artists**

Relationship: One user may be listed as a top artist.

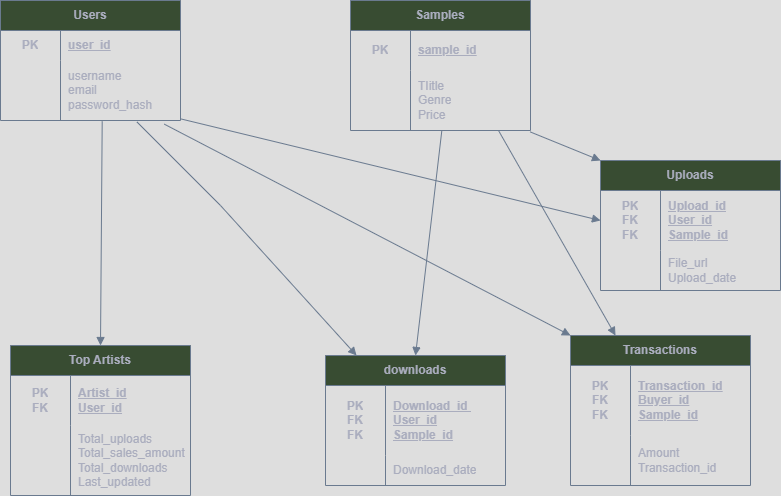
Cardinality: 1:1 or 0:1

Participation: Partial on Users, total on Top\_Artists.

**III. Entity Relationship Diagram**



**IV. Relational Model**

****

**Tables**

### ****1. Users****

| **user\_id** | **username** | **email** | **password\_hash (hashed example)** |
| --- | --- | --- | --- |
| 1 | beatmaker101 | [bm101@example.com](https://mailto:bm101@example.com/) | 2a2*a*10$N9qo8uLOickgx2ZMRZo... |
| 2 | synthqueen | [sq@example.com](https://mailto:sq@example.com/) | 2a2*a*10$ZR3ho8kLxOjGJZs4XQ... |
| 3 | drumlord | [dl@example.com](https://mailto:dl@example.com/) | 2a2*a*10$9KJp5fGv2yNjU7Wm1Y... |
|  |  |  |  |

### ****2. Samples****

| **sample\_id** | **title** | **genre** | **price** |
| --- | --- | --- | --- |
| 101 | "Epic Trap Brass" | Trap | 9.99 |
| 102 | "LoFi Chill Keys" | LoFi | 7.50 |
| 103 | "DnB Neuro Bass" | Drum & Bass | 12.00 |

### ****3. Uploads****

| **upload\_id** | **user\_id** | **sample\_id** | **file\_url** | **upload\_date** |
| --- | --- | --- | --- | --- |
| 1001 | 1 | 101 | <https://storage.com/brass.wav> | 2024-01-15 10:30:00 |
| 1002 | 2 | 102 | <https://storage.com/lofikeys.wav> | 2024-01-20 14:15:00 |
| 1003 | 3 | 103 | <https://storage.com/neurobass.wav> | 2024-02-05 09:45:00 |

### ****4. Downloads****

| **download\_id** | **user\_id** | **sample\_id** | **download\_date** |
| --- | --- | --- | --- |
| 5001 | 2 | 101 | 2024-01-16 11:20:00 |
| 5002 | 3 | 101 | 2024-01-18 16:45:00 |
| 5003 | 1 | 102 | 2024-01-25 13:10:00 |

### ****5. Transactions****

| **transaction\_id** | **user\_id** | **sample\_id** | **amount** | **transaction\_date** |
| --- | --- | --- | --- | --- |
| 2001 | 2 | 101 | 9.99 | 2024-01-16 11:20:00 |
| 2002 | 3 | 101 | 9.99 | 2024-01-18 16:45:00 |
| 2003 | 1 | 102 | 7.50 | 2024-01-25 13:10:00 |

### ****6. Top\_Artists****

| **artist\_id** | **user\_id** | **total\_uploads** | **total\_sales\_amount** | **total\_downloads** | **last\_updated** |
| --- | --- | --- | --- | --- | --- |
| 10 | 1 | 5 | 150.00 | 120 | 2024-03-01 08:00:00 |
| 11 | 2 | 3 | 85.50 | 90 | 2024-03-01 08:00:00 |

**V. Normalization**

**Checking for 1NF-**

|  |  |  |
| --- | --- | --- |
| **Table** | **1NF Status** | **Explanation** |
| Users | Yes | All values are atomic (username, email, password\_hash |
| Samples | Yes | Each field holds a single value (no multiple genres, etc.). |
| Transactions | Yes | No repeating groups or multi-values. |
| Downloads | Yes | One user downloads one sample at a time. |
| Engagements | Yes | One engagement per record. |
| Top Artists | Yes | One rank per artist. |

**Checking for 2NF-**

|  |  |  |
| --- | --- | --- |
| **Table** | **2NF Status** | **Explanation** |
| Users | Yes | PK is user\_id; all other fields depend fully on it. |
| Samples | Yes | PK is sample\_id; all fields depend on it. |
| Transactions | Yes | PK is transaction\_id; other fields depend on it. |
| Downloads | Yes | PK is download\_id; other fields depend on it. |
| Engagements | Yes | PK is engagement\_id; no partial dependency. |
| Top Artists | Yes | PK is artist\_id; score depends fully on it. |

**Checking for 3NF-**

|  |  |  |
| --- | --- | --- |
| **Table** | **3NF Status** | **Explanation** |
| Users | Yes | No transitive dependencies. |
| Samples | Yes | user\_id is a FK, rest are direct sample attributes. |
| Transactions | Yes | All attributes relate directly to the transaction. |
| Downloads | Yes | Same—no derived or transitive values. |
| Engagements | Yes | type is atomic; no transitives. |
| Top Artists | Yes | Score directly relates to artist ID. |

**3NF Final normalized tables-**

|  |  |
| --- | --- |
| **Table** | **Attributes** |
| Users | user\_id (PK), username, email, password\_hash |
| Samples | sample\_id (PK), user\_id (FK), title, genre, price, upload\_date |
| Transactions | transaction\_id (PK), user\_id (FK), sample\_id (FK), amount\_paid, transaction\_date |
| Downloads | download\_id (PK), user\_id (FK), sample\_id (FK), download\_date |
| Engagements | engagement\_id (PK), user\_id (FK), sample\_id (FK), type, timestamp |
| Top Artists | ranking\_id (PK), user\_id (FK), score |

**Normalized Tables (3NF)**

**1. Users**

| Field | Type | Description | Constraints |
| --- | --- | --- | --- |
| user\_id | INT | Unique identifier for the user. | PK, NOT NULL |
| username | VARCHAR(50) | User's display name. | NOT NULL, UNIQUE |
| email | VARCHAR(100) | User's email address. | NOT NULL, UNIQUE |
| password\_hash | VARCHAR(255) | Hashed password for security. | NOT NULL |

**Normalization**:

* **1NF**: Atomic values (no repeating groups).
* **2NF**: All fields depend fully on user\_id (no partial dependencies).
* **3NF**: No transitive dependencies (e.g., email → username).

**2. Samples**

| Field | Type | Description | Constraints |
| --- | --- | --- | --- |
| sample\_id | INT | Unique identifier for the sample. | PK, NOT NULL |
| user\_id | INT | ID of the user who uploaded it. | FK → Users, NOT NULL |
| title | VARCHAR(100) | Name of the sample. | NOT NULL |
| genre | VARCHAR(50) | Genre (Trap, LoFi, etc.). |  |
| price | DECIMAL(10,2) | Cost to purchase. | NOT NULL, ≥ 0 |
| upload\_date | DATETIME | When the sample was uploaded. | NOT NULL |

**Normalization**:

* **1NF**: Single-valued fields (e.g., no multiple genres).
* **2NF**: All fields depend on sample\_id (even user\_id is part of the record).
* **3NF**: No transitive dependencies (e.g., price doesn’t depend on genre).

**3. Transactions**

| Field | Type | Description | Constraints |
| --- | --- | --- | --- |
| transaction\_id | INT | Unique transaction ID. | PK, NOT NULL |
| user\_id | INT | ID of the buyer. | FK → Users, NOT NULL |
| sample\_id | INT | ID of the purchased sample. | FK → Samples, NOT NULL |
| amount\_paid | DECIMAL(10,2) | Price paid at time of purchase. | NOT NULL, ≥ 0 |
| transaction\_date | DATETIME | Date/time of purchase. | NOT NULL |

**Normalization**:

* **1NF/2NF/3NF**: All fields depend solely on transaction\_id.

**4.** Downloads

| Field | Type | Description | Constraints |
| --- | --- | --- | --- |
| download\_id | INT | Unique download ID. | PK, NOT NULL |
| user\_id | INT | ID of the user who downloaded. | FK → Users, NOT NULL |
| sample\_id | INT | ID of the downloaded sample. | FK → Samples, NOT NULL |
| download\_date | DATETIME | Date/time of download. | NOT NULL |

Normalization:

1NF/2NF/3NF: Atomic values, no partial/transitive dependencies.

5. Engagements

| Field | Type | Description | Constraints |
| --- | --- | --- | --- |
| engagement\_id | INT | Unique engagement ID. | PK, NOT NULL |
| user\_id | INT | ID of the engaging user. | FK → Users, NOT NULL |
| sample\_id | INT | ID of the sample. | FK → Samples, NOT NULL |
| type | VARCHAR(50) | Action (like, comment, share). | NOT NULL |
| timestamp | DATETIME | When the action occurred. | NOT NULL |

Normalization:

1NF/2NF/3NF: All fields depend on engagement\_id.

6. Top\_Artists

| Field | Type | Description | Constraints |
| --- | --- | --- | --- |
| ranking\_id | INT | Unique ranking ID. | PK, NOT NULL |
| user\_id | INT | ID of the top artist. | FK → Users, NOT NULL, UNIQUE |
| score | DECIMAL(10,2) | Performance metric (sales, etc.). | NOT NULL, ≥ 0 |

Normalization:

1NF/2NF/3NF: score depends only on ranking\_id.

VI. SQL Queries

-- Create the database

CREATE DATABASE sample\_marketplace;

-- Use the created database

USE sample\_marketplace;

-- Create Users table

CREATE TABLE Users (

user\_id INT AUTO\_INCREMENT,

username VARCHAR(100) NOT NULL,

email VARCHAR(255) NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

PRIMARY KEY (user\_id),

UNIQUE KEY (email),

UNIQUE KEY (username)

);

-- Create Samples table

CREATE TABLE Samples (

sample\_id INT AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

Genre VARCHAR(100),

Price DECIMAL(10, 2) NOT NULL,

PRIMARY KEY (sample\_id)

);

-- Create Uploads table

CREATE TABLE Uploads (

Upload\_id INT AUTO\_INCREMENT,

User\_id INT NOT NULL,

Sample\_id INT NOT NULL,

File\_url VARCHAR(255) NOT NULL,

Upload\_date DATETIME NOT NULL,

PRIMARY KEY (Upload\_id),

FOREIGN KEY (User\_id) REFERENCES Users(user\_id) ON DELETE CASCADE,

FOREIGN KEY (Sample\_id) REFERENCES Samples(sample\_id) ON DELETE CASCADE,

UNIQUE KEY (File\_url)

);

-- Create downloads table

CREATE TABLE downloads (

Download\_id INT AUTO\_INCREMENT,

User\_id INT NOT NULL,

Sample\_id INT NOT NULL,

Download\_date DATETIME NOT NULL,

PRIMARY KEY (Download\_id),

FOREIGN KEY (User\_id) REFERENCES Users(user\_id) ON DELETE CASCADE,

FOREIGN KEY (Sample\_id) REFERENCES Samples(sample\_id) ON DELETE CASCADE

);

-- Create Transactions table

CREATE TABLE Transactions (

Transaction\_id INT AUTO\_INCREMENT,

Buyer\_id INT NOT NULL,

Sample\_id INT NOT NULL,

Amount DECIMAL(10, 2) NOT NULL,

Transaction\_ref VARCHAR(50),

PRIMARY KEY (Transaction\_id),

FOREIGN KEY (Buyer\_id) REFERENCES Users(user\_id) ON DELETE RESTRICT,

FOREIGN KEY (Sample\_id) REFERENCES Samples(sample\_id) ON DELETE RESTRICT

);

-- Create Top\_Artists table

CREATE TABLE Top\_Artists (

Artist\_id INT AUTO\_INCREMENT,

User\_id INT NOT NULL,

Total\_uploads INT DEFAULT 0,

Total\_sales\_amount DECIMAL(15, 2) DEFAULT 0.00,

Total\_downloads INT DEFAULT 0,

Last\_updated DATETIME NOT NULL,

PRIMARY KEY (Artist\_id),

FOREIGN KEY (User\_id) REFERENCES Users(user\_id) ON DELETE CASCADE,

UNIQUE KEY (User\_id)

);

-- Insert data into Users table

INSERT INTO Users (username, email, password\_hash) VALUES

('beatmaster', 'beatmaster@example.com', 'hash1'),

('melodymaker', 'melody@example.com', 'hash2'),

('bassking', 'bass@example.com', 'hash3'),

('drumqueen', 'drums@example.com', 'hash4'),

('synthwave', 'synth@example.com', 'hash5'),

('vocalpro', 'vocals@example.com', 'hash6'),

('guitargenius', 'guitar@example.com', 'hash7'),

('producerpro', 'producer@example.com', 'hash8'),

('djmixer', 'dj@example.com', 'hash9'),

('soundfx', 'fx@example.com', 'hash10'),

('pianoplayer', 'piano@example.com', 'hash11'),

('orchestral', 'orchestra@example.com', 'hash12');

-- Insert data into Samples table

INSERT INTO Samples (Title, Genre, Price) VALUES

('Deep House Beat', 'House', 19.99),

('Epic Orchestral Theme', 'Orchestral', 29.99),

('Hip Hop Drums', 'Hip Hop', 14.99),

('EDM Drop', 'EDM', 24.99),

('Acoustic Guitar Loop', 'Acoustic', 9.99),

('Trap 808 Bass', 'Trap', 12.99),

('Vocal Chops Pack', 'Vocals', 19.99),

('Synthwave Melody', 'Synthwave', 15.99),

('Lo-Fi Piano', 'Lo-Fi', 11.99),

('Jazz Saxophone Solo', 'Jazz', 17.99),

('Rock Drum Fills', 'Rock', 13.99),

('Ambient Textures', 'Ambient', 22.99),

('Reggae Bass Line', 'Reggae', 10.99),

('Dubstep Wobble Bass', 'Dubstep', 16.99);

-- Insert data into Uploads table (ensure Users and Samples exist first)

INSERT INTO Uploads (User\_id, Sample\_id, File\_url, Upload\_date) VALUES

(1, 1, 'https://samples.com/deephouse.wav', '2024-03-15 10:30:00'),

(2, 2, 'https://samples.com/orchestral.wav', '2024-03-16 11:45:00'),

(3, 3, 'https://samples.com/hiphopdrums.wav', '2024-03-17 09:20:00'),

(4, 4, 'https://samples.com/edmdrop.wav', '2024-03-18 14:10:00'),

(5, 5, 'https://samples.com/acoustic.wav', '2024-03-19 16:25:00'),

(6, 6, 'https://samples.com/trap808.wav', '2024-03-20 12:40:00'),

(7, 7, 'https://samples.com/vocalchops.wav', '2024-03-21 08:50:00'),

(8, 8, 'https://samples.com/synthwave.wav', '2024-03-22 17:15:00'),

(9, 9, 'https://samples.com/lofi.wav', '2024-03-23 13:30:00'),

(10, 10, 'https://samples.com/jazz.wav', '2024-03-24 15:45:00'),

(1, 11, 'https://samples.com/rockdrums.wav', '2024-03-25 10:20:00'),

(2, 12, 'https://samples.com/ambient.wav', '2024-03-26 11:35:00'),

(3, 13, 'https://samples.com/reggae.wav', '2024-03-27 09:50:00'),

(4, 14, 'https://samples.com/dubstep.wav', '2024-03-28 14:05:00');

-- Insert data into downloads table

INSERT INTO downloads (User\_id, Sample\_id, Download\_date) VALUES

(5, 1, '2024-04-01 09:15:00'),

(6, 1, '2024-04-02 10:30:00'),

(7, 1, '2024-04-03 11:45:00'),

(8, 2, '2024-04-04 13:00:00'),

(9, 2, '2024-04-05 14:15:00'),

(10, 3, '2024-04-06 15:30:00'),

(11, 4, '2024-04-07 16:45:00'),

(12, 5, '2024-04-08 18:00:00'),

(1, 6, '2024-04-09 09:30:00'),

(2, 7, '2024-04-10 10:45:00'),

(3, 8, '2024-04-11 12:00:00'),

(4, 9, '2024-04-12 13:15:00'),

(5, 10, '2024-04-13 14:30:00'),

(6, 11, '2024-04-14 15:45:00'),

(7, 12, '2024-04-15 17:00:00'),

(8, 13, '2024-04-16 18:15:00');

-- Insert data into Transactions table

INSERT INTO Transactions (Buyer\_id, Sample\_id, Amount, Transaction\_ref) VALUES

(5, 1, 19.99, 'TX001'),

(6, 1, 19.99, 'TX002'),

(7, 1, 19.99, 'TX003'),

(8, 2, 29.99, 'TX004'),

(9, 2, 29.99, 'TX005'),

(10, 3, 14.99, 'TX006'),

(11, 4, 24.99, 'TX007'),

(12, 5, 9.99, 'TX008'),

(1, 6, 12.99, 'TX009'),

(2, 7, 19.99, 'TX010'),

(3, 8, 15.99, 'TX011'),

(4, 9, 11.99, 'TX012');

-- Insert data into Top\_Artists table (normally this would be computed, but for demo purposes we'll insert directly)

INSERT INTO Top\_Artists (User\_id, Total\_uploads, Total\_sales\_amount, Total\_downloads, Last\_updated) VALUES

(1, 2, 59.97, 3, '2024-04-20 00:00:00'),

(2, 2, 59.98, 2, '2024-04-20 00:00:00'),

(3, 2, 30.98, 2, '2024-04-20 00:00:00'),

(4, 2, 49.98, 2, '2024-04-20 00:00:00'),

(5, 1, 9.99, 1, '2024-04-20 00:00:00'),

(6, 1, 19.99, 0, '2024-04-20 00:00:00'),

(7, 1, 19.99, 0, '2024-04-20 00:00:00'),

(8, 1, 15.99, 0, '2024-04-20 00:00:00'),

(9, 1, 11.99, 1, '2024-04-20 00:00:00'),

(10, 1, 17.99, 1, '2024-04-20 00:00:00');

SELECT user\_id, username, email FROM Users;

SELECT \* FROM Samples WHERE Genre = 'House';

SELECT Title, Genre, Price FROM Samples ORDER BY Price DESC;

SELECT Genre, COUNT(\*) AS SampleCount FROM Samples GROUP BY Genre ORDER BY SampleCount DESC;

SELECT s.Title, u.username, up.Upload\_date

FROM Uploads up

JOIN Samples s ON up.Sample\_id = s.sample\_id

JOIN Users u ON up.User\_id = u.user\_id

WHERE up.Upload\_date >= DATE\_SUB(NOW(), INTERVAL 7 DAY)

ORDER BY up.Upload\_date DESC;

SELECT DISTINCT u.username, u.email

FROM Users u

JOIN Uploads up ON u.user\_id = up.User\_id

JOIN downloads d ON u.user\_id = d.User\_id;

SELECT s.Title, s.Genre, COUNT(d.Download\_id) AS DownloadCount

FROM Samples s

JOIN downloads d ON s.sample\_id = d.Sample\_id

GROUP BY s.sample\_id, s.Title, s.Genre

ORDER BY DownloadCount DESC

LIMIT 1;

SELECT s.Genre, SUM(t.Amount) AS TotalRevenue

FROM Samples s

JOIN Transactions t ON s.sample\_id = t.Sample\_id

GROUP BY s.Genre

ORDER BY TotalRevenue DESC;

SELECT u.username, u.email

FROM Users u

JOIN Uploads up ON u.user\_id = up.User\_id

WHERE u.user\_id NOT IN (SELECT Buyer\_id FROM Transactions);

SELECT s.Title, s.Genre

FROM Samples s

LEFT JOIN downloads d ON s.sample\_id = d.Sample\_id

WHERE d.Download\_id IS NULL;

SELECT s.Title,

COUNT(DISTINCT d.Download\_id) AS TotalDownloads,

COUNT(DISTINCT t.Transaction\_id) AS TotalPurchases,

CASE

WHEN COUNT(DISTINCT d.Download\_id) > 0

THEN (COUNT(DISTINCT t.Transaction\_id) / COUNT(DISTINCT d.Download\_id)) \* 100

ELSE 0

END AS ConversionRate

FROM Samples s

LEFT JOIN downloads d ON s.sample\_id = d.Sample\_id

LEFT JOIN Transactions t ON s.sample\_id = t.Sample\_id

GROUP BY s.sample\_id, s.Title

HAVING TotalDownloads > 0

ORDER BY ConversionRate DESC;

SELECT u.username, SUM(t.Amount) AS TotalSales

FROM Users u

JOIN Uploads up ON u.user\_id = up.User\_id

JOIN Samples s ON up.Sample\_id = s.sample\_id

JOIN Transactions t ON s.sample\_id = t.Sample\_id

GROUP BY u.user\_id, u.username

ORDER BY TotalSales DESC

LIMIT 1;

SELECT Genre, AVG(Price) AS AveragePrice

FROM Samples

GROUP BY Genre

ORDER BY AveragePrice DESC;

SELECT t.Transaction\_id, t.Transaction\_ref, u.username AS Buyer,

s.Title AS SampleName, t.Amount

FROM Transactions t

JOIN Users u ON t.Buyer\_id = u.user\_id

JOIN Samples s ON t.Sample\_id = s.sample\_id

ORDER BY t.Transaction\_id;

SELECT DISTINCT u.username, u.email

FROM Users u

JOIN downloads d ON u.user\_id = d.User\_id

WHERE u.user\_id NOT IN (SELECT Buyer\_id FROM Transactions);

SELECT

DATE\_FORMAT(t.Transaction\_id, '%Y-%m') AS Month,

COUNT(t.Transaction\_id) AS TransactionCount,

SUM(t.Amount) AS TotalSales

FROM Transactions t

GROUP BY Month

ORDER BY Month;

SELECT u.username, COUNT(d.Download\_id) AS DownloadCount

FROM Users u

JOIN downloads d ON u.user\_id = d.User\_id

GROUP BY u.user\_id, u.username

ORDER BY DownloadCount DESC

LIMIT 1;

SELECT u.username, ta.Total\_sales\_amount

FROM Top\_Artists ta

JOIN Users u ON ta.User\_id = u.user\_id

ORDER BY ta.Total\_sales\_amount DESC

LIMIT 5;

UPDATE Top\_Artists ta

SET

Total\_uploads = (SELECT COUNT(\*) FROM Uploads WHERE User\_id = ta.User\_id),

Total\_sales\_amount = (

SELECT COALESCE(SUM(t.Amount), 0)

FROM Uploads up

JOIN Transactions t ON up.Sample\_id = t.Sample\_id

WHERE up.User\_id = ta.User\_id

),

Total\_downloads = (

SELECT COUNT(\*)

FROM Uploads up

JOIN downloads d ON up.Sample\_id = d.Sample\_id

WHERE up.User\_id = ta.User\_id

),

Last\_updated = NOW()

WHERE EXISTS (SELECT 1 FROM Users u WHERE u.user\_id = ta.User\_id);

SELECT s.Title, u.username AS Uploader, up.Upload\_date

FROM Samples s

JOIN Uploads up ON s.sample\_id = up.Sample\_id

JOIN Users u ON up.User\_id = u.user\_id

LEFT JOIN downloads d ON s.sample\_id = d.Sample\_id

WHERE d.Download\_id IS NULL;

SELECT AVG(TIMESTAMPDIFF(HOUR, up.Upload\_date, MIN(d.Download\_date))) AS AvgHoursToFirstDownload

FROM Uploads up

JOIN downloads d ON up.Sample\_id = d.Sample\_id

GROUP BY up.Upload\_id;

SELECT

u.username,

COUNT(DISTINCT up.Upload\_id) AS UploadsCount,

COUNT(DISTINCT d.Download\_id) AS DownloadsCount,

COUNT(DISTINCT t.Transaction\_id) AS PurchasesCount,

COALESCE(SUM(t.Amount), 0) AS TotalSpent

FROM Users u

LEFT JOIN Uploads up ON u.user\_id = up.User\_id

LEFT JOIN downloads d ON u.user\_id = d.User\_id

LEFT JOIN Transactions t ON u.user\_id = t.Buyer\_id

GROUP BY u.user\_id, u.username

ORDER BY UploadsCount DESC, DownloadsCount DESC;

A white rectangular object with black lines

AI-generated content may be incorrect.

A white and blue line

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

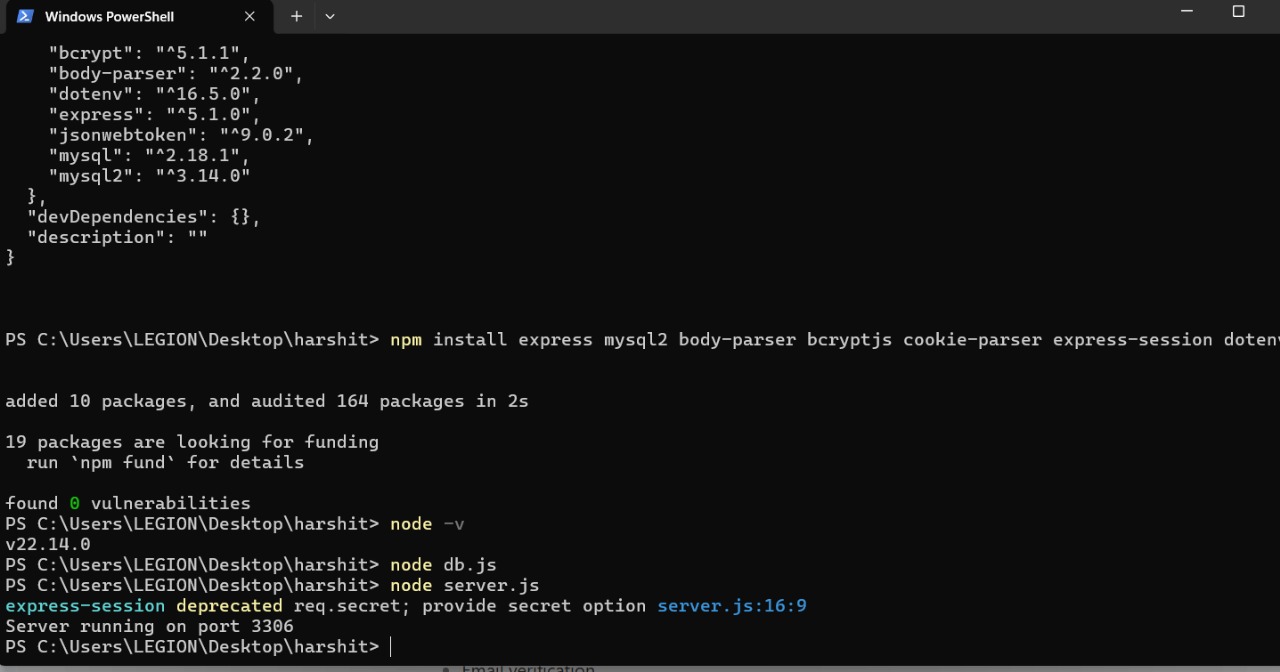
**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**VI. Project demonstration**

****

● Tools/Software/Libraries Used

• MySQL Workbench 8.0 – For schema creation, query execution, and transaction handling.

• XAMPP (Apache + MySQL) – Local server to host the database (optional).

• DBeaver / phpMyAdmin – GUI alternative for database visualization (optional).

• Microsoft Word – Documentation.

• Visual Studio Code – For organizing .sql files and writing queries.

• Tables were created using DDL commands with proper normalization up to 3NF.

• Data was inserted into all tables (15+ rows).

• More than 50 queries were executed demonstrating:

o Filtering, Joins, Aggregation

o Subqueries, Views, Transactions

o DCL and user management

­­• Screenshots include results of:

o Customer Orders & Payments Join

o Aggregated product sales

o Transaction ROLLBACK example

o View creation and usage

**VII. Self -Learning beyond classroom**

During this project, I independently explored and learned several new concepts that strengthened my understanding of databases. I learned how to normalize complex database schemas to Third Normal Form (3NF) and effectively eliminate transitive dependencies, which helped in designing clean and efficient data models. I also gained hands-on experience with real-world transaction control using START TRANSACTION, SAVEPOINT, and ROLLBACK TO, which are essential for maintaining data consistency in multi-step operations. Since MySQL does not directly support INTERSECT and EXCEPT operations, I learned how to simulate these functionalities using alternative query techniques. Additionally, I developed a practical understanding of foreign key constraints, including how to resolve common errors like Error 3734.

**VIII. Learning from the Project**

**How this project helped me:**

* Built a strong foundational knowledge of database design and normalization.
* Understood the importance of relational integrity, especially when working with **FOREIGN KEYS** and cascading updates/deletes.
* Learned how to optimize queries using indexing techniques and proper table relationships.
* Gained confidence in handling large SQL scripts and debugging errors such as constraint violations and syntax issues.
* Developed a project that simulates a real-world merchandise management system, helping bridge the gap between academic concepts and industry-relevant practices.

**IX. Challenges Faced**

Encountered foreign key constraint errors (Error Code: 3734) when dropping/recreating tables in incorrect order. Initial confusion in designing many-to-many relationships like Orders and Products using a junction table. Needed to refactor the schema multiple times to achieve full 3NF compliance, especially around subtotal, total\_amount, and discount\_price. Implementing transactions and ensuring no data inconsistency was tricky at first, especially understanding ROLLBACK TO SAVEPOINT. Creating realistic dummy data across all 9 tables while maintaining referential integrity was time-consuming.

**X. Conclusion**

What are the key takeaways from the project?

Designing a normalized database ensures data integrity, scalability, and ease of querying.

Practical implementation of SQL gives hands-on exposure to what is taught theoretically.

Transaction handling and DCL concepts are critical in real-world applications like payment systems and user management.

The project has enhanced my ability to analyze requirements, convert them into an ER diagram, normalize the schema, and write robust SQL queries.

I now understand the full cycle of database application development — from design to querying to troubleshooting.