Report

Physical Database Design and Database Tuning

Asian Songs Database

Presented

By

Purit Phan-udom Section 1 ID: 5988023

Pongpeera Sukasem Section 1 ID: 5988040

Boonyada Lojanarungsiri Section 1 ID: 5988153

Manisa Satravisut Section 1 ID: 5988209

ITCS413
Database Design

Faculty of Information and Communication Technology

Mahidol University

Data Requirements

Member

Members refer to the users who are interested in using the database's functions as a member (i.e. not as a guest). To become a Member, a person must first create an account with the system and login using such account. They are the ones who will be using the database the most in order to search for information about the songs and insert songs and their information into the database. The data describing a member include a unique member ID, full name, address, gender, email(s), and nationality. Furthermore, a Member record contains flags to indicate whether the Member is also a Song Owner and/or a Researcher.

Song

Songs are the main component of this database, as they are used to store various information related to the songs such as the ceremony they were played in and the artists who performed that song. The data stored on a Song include a unique ID number, title (name of the song), note key, lyrics, danceability, teachability, URL to the song's video, year released (in A.D.), type of copyright, sentiment(s), songwriter(s), and the date this song was posted. Members can search for many Songs, and a Song can also belong in multiple Countries. Only Song Owners (the member who originally insert that song into the database) can add and edit a song's information and can search for more information regarding all the songs.

Album

Albums refer to the collections of recordings issued as a single item on CD, record, or another medium. The data stored on an Album include a unique ID number, album name, description, genre (rock, pop, etc.), release year, and number of songs in the album. Note that a Song may or may not be in an album. An album can contain as many songs as it should be, as in reality. However, an normal music album in real life would contain from about 15 - 20 songs. Only Song Owners can add or edit information about an album through the system.

Artist

Artists refer to the people (one or many) who performed the Songs in the database. Information about an Artist may only be inserted into the database by the Song Owners. An Artist record contains a unique ID number, the artist's name, gender, nationality, gender, publisher, and date of birth. A Song may contain one or more artists, and an artist may perform many Songs in the database.

Musical Instrument

A Musical Instrument is an instrument created or adapted to make musical sounds in a Song. Only Song Owners can insert and update the information about a Musical Instrument. A Musical Instrument object contains a unique ID, name, link to the instrument's picture(s), and the family it belongs to (Brass, Strings, Woodwind, Percussion or Keyboard). A Musical Instrument can belong in multiple Countries.

Country

A Country refers to the labelling of specific land areas or territory. A country is used to distinguish which nations do Songs, Musical Instruments, Ceremony and Era come from. Country's primary key (Cou_ID), hence, is used as foreign for those relations. A tuple of Country instance contains Country ID, name, and the region(s) within a Country. Song Owners may add or edit information about a Country where their elements belong to.

Ceremony

A Ceremony refers to an event of (mostly religious) significance, performed on a special occasion. Songs can be played at Ceremonies, which can help researchers identify a Song more easily. A Ceremony record contains a unique ID, ceremony name, indicator whether the event was held indoor or outdoor, and the date the ceremony was held.

Review

An assessment or examination of something with the possibility or intention of instituting change of a Song if necessary. All types of users can leave reviews on all the Songs in the database, and a Song can have many Reviews in them. A Review object contains a unique ID, the comment string, rating from 1-5, and the date the review was posted.

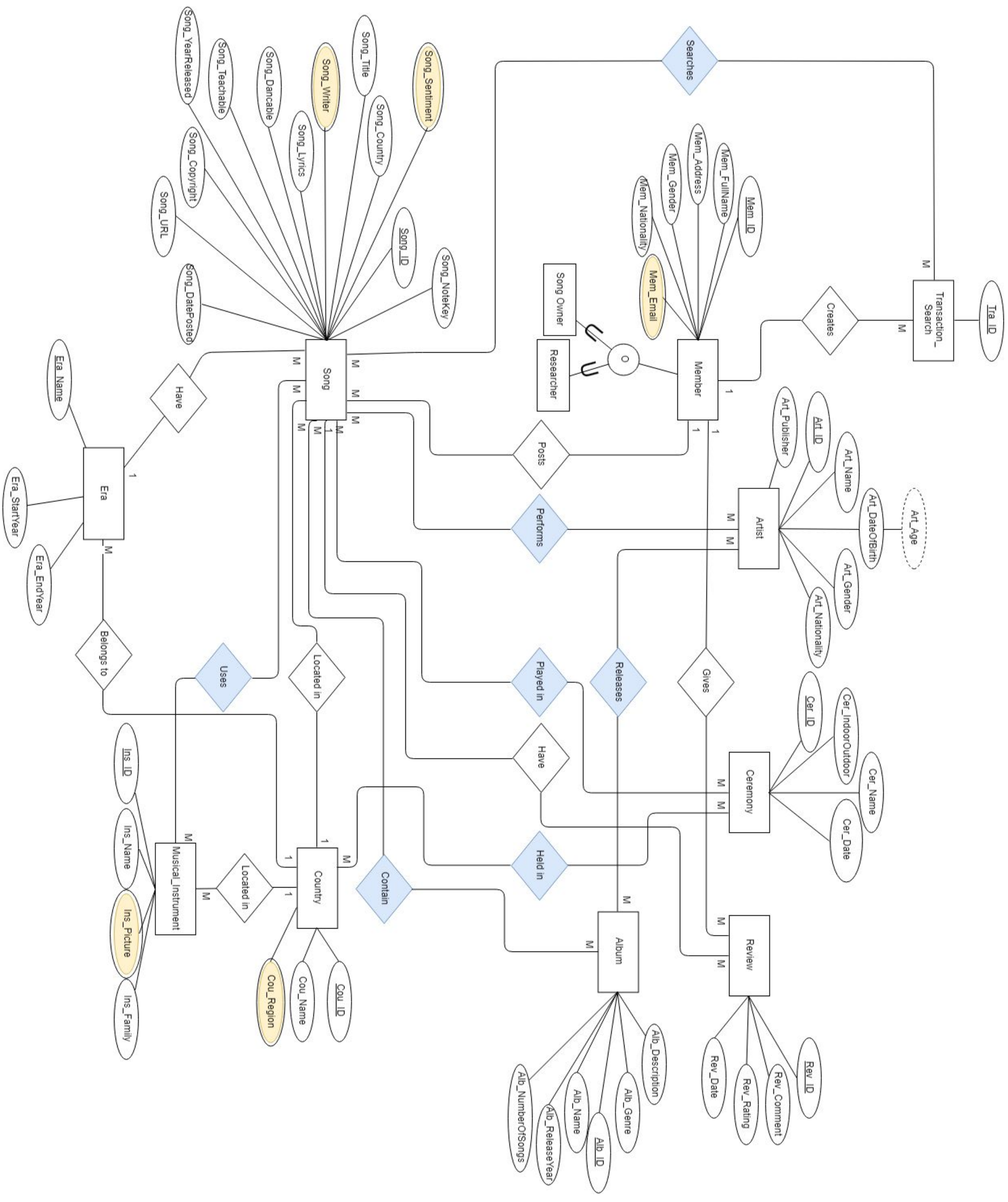
Era

An Era refers to a calling name used to represent a certain period of time (over years) in the history of a nation(s) or country(s). Era helps to provide more information about song in terms of historical timestamp which can be further related to culture, demography and other information of that era. A tuple of Era instance contains the Era's name, starting year, ending year, Song's ID and Country's ID. Song Owners may add or edit information about an era to help provide more details about their Songs. There can be many Songs in an Era, and an Era may be for many Countries.

Search Transaction

A Search Transaction refers to the records of the Members' past song searching history. These transactions can be useful to identify the searching trend and make adjustments accordingly. A Search Transaction object contains a unique ID, and the ID of the Member who performed the search.

B. Final ER Diagram



C. Final Relational Database Schema

Strong Relations	
Member (Mem_ID, Mem_FullName, Mem_Address, Mem_Gender, Mem_Nationality) **Primary Key** Mem_ID **Alternate Key** Mem_FullName	**Song** (Song_ID, Song_NoteKey, Song_Title, Song_Lyrics, Song_Dancable, Song_Teachable, Song_URL, Song_YearReleased, Song_Copyright, Song_DatePosted) **Primary Key** Song_ID **Alternate Key** {Song_Title, Song_Country} **Foreign Key** Mem_ID **references** Member(Mem_ID) **Foreign Key** Cou_ID **references** Country(Cou_ID)
Album (Alb_ID, Alb_Name, Alb_Description, Alb_Genre, Alb_ReleaseYear, Alb_NumberOfSongs) **Primary Key** Alb_ID **Alternate Key** Alb_Name	**Artist** (Art_ID, Art_Name, Art_Gender, Art_Nationality, Art_Publisher, Art_DateOfBirth) **Primary Key** Art_ID **Alternate Key** Art_Name
Musical_Intrument (Ins_ID, Ins_Name, Ins_Family) **Primary Key** Ins_ID **Alternate Key** Ins_Name **Foreign Key** Reg_ID **references** Country(Cou_ID)	**Country** (Cou_ID, Cou_Name) **Primary Key** Cou_ID **Alternate Key** Cou_Name
Ceremony (Cer_ID, Cer_Name, Cer_IndoorOutdoor, Cer_Date) **Primary Key** Cer_ID **Alternate Key** Cer_Name	**Review** (Rev_ID, Rev_Comment, Rev_Rating, Rev_Date) **Primary Key** Rev_ID **Alternate Key** {Rev_ID, Rev_Date} **Foreign Key** Mem_ID **references** Member(Mem_ID) **Foreign Key** Song_ID **references** Song(Song_ID)
Era (Era_Name, Era_StartYear, Era_EndYear) **Primary Key** Era_Name **Foreign Key** Song_ID **references** Song(Song_ID) **Foreign Key** Cou_ID **references** Country(Cou_ID)	**Search_Transaction** (Tra_ID) **Primary Key** Tra_ID **Foreign Key** Mem_ID **references** Member(Mem_ID)
Subclass Relations	
Member_Details (Mem_ID, songOwner_Flag, researcher_Flag) **Primary Key** Mem_ID **Foreign Key** Mem_ID **references** Member(Mem_ID)	

Many-to-Many Relations	
Searches (Tra_ID, Mem_ID) **Primary Key** {Tra_ID, Mem_ID} **Foreign Key** Mem_ID **references** Member(Mem_ID) **Foreign Key** Tra_ID **references** Search_Transaction(Tra_ID)	**Performs** (Song_ID, Art_ID) **Primary Key** {Song_ID, Art_ID} **Foreign Key** Song_ID **references** Song(Song_ID) **Foreign Key** Art_ID **references** Artist(Art_ID)
Played_In (Cer_ID, Song_ID) **Primary Key** {Cer_ID, Song_ID} **Foreign Key** Song_ID **references** Song(Song_ID) **Foreign Key** Cer_ID **references** Ceremony(Cer_ID)	**Uses** (Ins_ID, Song_ID) **Primary Key** {Ins_ID, Song_ID} **Foreign Key** Song_ID **references** Song(Song_ID) **Foreign Key** Ins_ID **references** Musical Instrument(Ins_ID)
Held_In (Cer_ID, Cou_ID) **Primary Key** {Cer_ID, Cou_ID} **Foreign Key** Cer_ID **references** Ceremony(Cer_ID) **Foreign Key** Cou_ID **references** Country(Cou_ID)	**Contain** (Alb_ID, Song_ID) **Primary Key** {Alb_ID, Song_ID} **Foreign Key** Alb_ID **references** Album(Alb_ID) **Foreign Key** Song_ID **references** Song(Song_ID)
Release (Alb_ID, Art_ID) **Primary Key** {Alb_ID, Art_ID} **Foreign Key** Alb_ID **references** Album(Alb_ID) **Foreign Key** Art_ID **references** Artist(Art_ID)	
Multivalued Relations	
Mem_Emails (Mem_ID, Mem_Email) **Primary Key** Mem_Email **Foreign Key** Mem_ID **references** Member(Mem_ID)	**Ins_Pictures** (Ins_ID, Ins_Picture) **Primary Key** Ins_Picture **Foreign Key** Ins_ID **references** Instrument(Song_ID)
Country_Regions (Cou_ID, Cou_Region) **Primary Key** Cou_Region **Foreign Key** Cou_ID **references** Country(Cou_ID)	**Song_Sentiments** (Song_ID, Song_Sentiment) **Primary Key** Song_Sentiment **Foreign Key** Song_ID **references** Song(Song_ID)
Song_Writers (Song_ID, Song_Writer) **Primary Key** Song_Writer **Foreign Key** Song_ID **references** Song(Song_ID)	

D. Transaction Analysis

The chosen transactions from Project 2 to be analyzed are as follows:
(b) List the details of all instruments that are used in each song
(m) Identify the total number of artist in each song

Cross-referencing Transaction and Relation Matrix

Transaction/ Relation	**(b)**				**(m)**			
	I	**R**	**U**	**D**	**I**	**R**	**U**	**D**
Member								
Song		X				X		
Album								
Artist						X		
Musical_Instrument		X						
Country		X						
Ceremony								
Review								
Era								
Perform						X		
Search_Transaction	X							

I = Insert, R = Read, U = Update, D = Delete

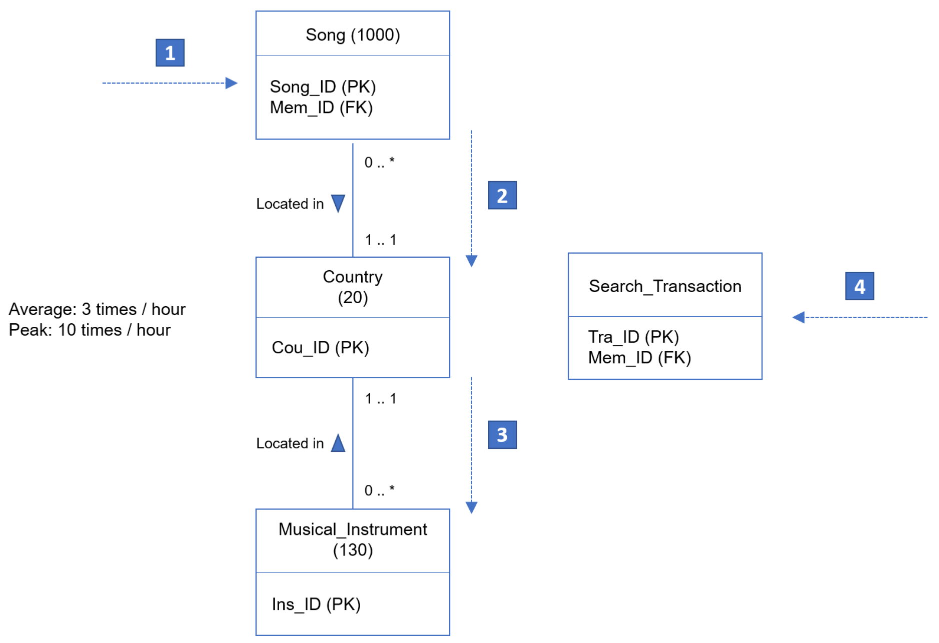
Transaction Analysis Forms

Transaction: (b) List the details of all instruments that are used in each song	
Transaction Volume Average: 3 times per hour Peak: 10 times per hour (12.00 - 18.00 Saturday - Sunday)	
CREATE PROCEDURE proc_SongInstrument @SongName varchar(255) AS BEGIN SET NOCOUNT ON SELECT Ins_Name,Ins_Family FROM Musical_Instrument m JOIN Uses ON m.Ins_ID = Uses.Ins_ID WHERE Uses.Song_ID IN (SELECT (Song_ID) FROM Song WHERE Song_Title =	**Predicate**: none **Join attributes:** - Country.Cou_ID = Song.Cou_ID - Country.Cou_ID = Musical_Instrument.Cou_ID **Ordering attribute:** none **Grouping attribute:** none **Built-in functions:** none **Attributes updated:** none

```
@SongName);
END


EXEC proc_SongInstrument
@SongName = 'Kiss Kiss'
```

Transaction Usage Map



Access	Entity	Type of Access	Number of References		
			Per Transaction	Avg / Hour	Peak / Hour
1	**Song**	R	1000	3000	10000
2	**Country**	R	20	60	200
3	**Musical Instrument**	R	130	390	1300
4	**Search_Transaction**	I	1	3	10
Total References			**1151**	**3453**	**11510**

Transaction: (m) Identify the total number of artist in each song

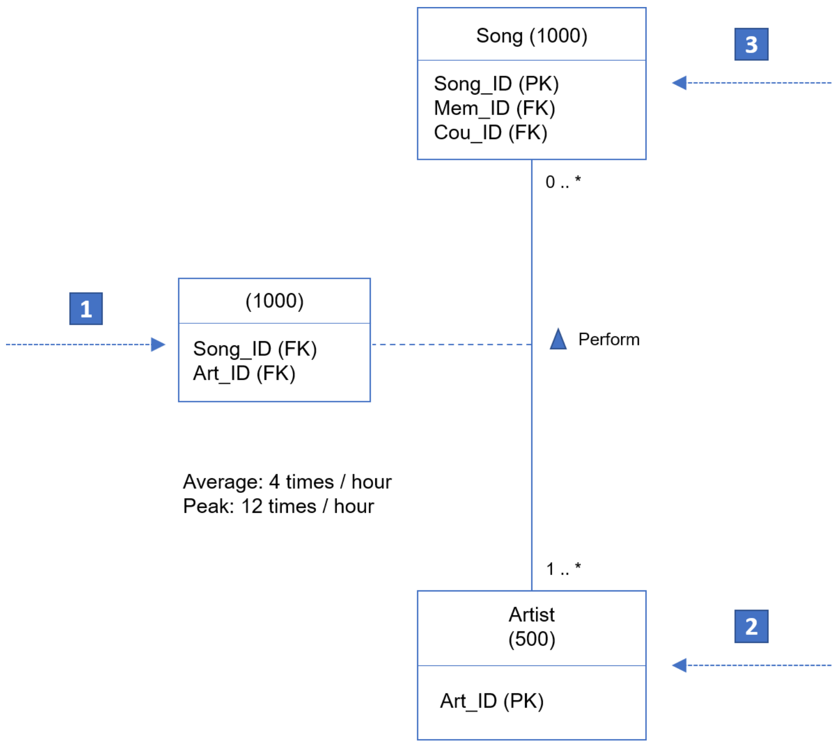
Transaction Volume
Average: 1.25 times per hour
Peak: 6.25 times per hour (18.00 - 21.00 Saturday - Sunday)

```
SELECT Song_Title, COUNT(Artist.Art_ID)
AS [Total Artist] FROM Performs JOIN
Artist ON Performs.Art_ID = Artist.Art_ID
JOIN Song ON Song.Song_ID =
Performs.Song_ID GROUP BY Song_Title;
```

Predicate: none
Join attributes:
- Performs.Art_ID = Artist.Art_ID
- Performs.Song_ID = Song.Song_ID

Ordering attribute: none
Grouping attribute: Song_Title
Built-in functions: none
Attributes updated: none

Transaction Usage Map



Access	Entity	Type of Access	No. of References		
			Per Transaction	Avg Per Hour	Peak Per Hour
1	**Song**	R	1000	4000	12000
2	**Artist**	R	500	2000	6000
3	**Perform**	R	1000	3000	12000
Total References			**2500**	**9000**	**30000**

E. Index Analysis

Table	Transaction	Field	Frequency (per day)
Song	(b),(g),(j),(l),(r)	PREDICATE: Song_ID	30 - 260
	(b),(c),(d),(e),(f),),(h),(i),(j),(k),(l),(m),(n),(o),(q),(r),(s),(t),(u),(v)	PREDICATE: Song_Title	30 - 1055
	(g),(j)	PREDICATE: Song_NoteKey	30 - 70
	(g),(v)	PREDICATE: Song_Lyrics	30 - 70
	(g),(o)	PREDICATE: Song_Dancable	30 - 70
	(g),(n),(o)	PREDICATE: Song_Teachable	30 - 70
	(g)	PREDICATE: Song_URL	30
	(g),(t)	PREDICATE: Song_YearReleased	30 - 70
	(g),(q)	PREDICATE: Song_CopyRight	30 - 70
	(u)	PREDICATE: Song_DatePosted	70
	(d),(s)	JOIN: Song_Writer ON Song_ID	30 - 70
	(e),(g),(m)	JOIN: Performs ON Song_ID	30 - 130
	(f)	JOIN: Cou_Regions ON Cou_ID	30
	(h)	JOIN: Song_Sentiment ON Song_ID	70
	(i)	JOIN: Era ON Song_ID	70
	(k)	JOIN: Played_in ON Song_ID	45
	(l)	JOIN: Contain on Song_ID	40
	(r)	JOIN: Album on Alb_ID	70
	(e),(i)	ORDER BY: Song_Title	30 - 60
	(q)	ORDER BY: Song_CopyRight	30
	(t)	ORDER BY: Song_YearReleased	30
	(m)	GROUP BY: Song_Title	30 - 60
Album	(l)	PREDICATE: Alb_Name	40
	(r)	PREDICATE: Alb_Genre	70

	(l)	JOIN: Contain ON Alb_ID	40
	(r)	JOIN: Song ON Alb_ID	70
Artist	(g),(m)	PREDICATE: Art_ID	70 - 140
	(e),(g)	PREDICATE: Art_Name	30 - 100
	(e),(g),(m)	JOIN: Perform ON Art_ID	30 - 130
Song_Writer	(d),(s)	PREDICATE: Song_Writer	70 - 140
	(d),(s)	JOIN: Song ON Song_ID	70 - 140
	(d),(s)	ORDER BY: Song_Writer	70 - 140
Song_Sentiment	(h)	PREDICATE: Song_Sentiment	70
	(h)	JOIN: Song ON Song_ID	70
Musical_Instrument	(b)	JOIN: Uses on Ins_ID	50
Cou_Regions	(f)	PREDICATE: Cou_Region	30
	(f)	JOIN: Song ON Cou_ID	30
Ceremony	(k)	PREDICATE: Cer_Name	30
	(k)	JOIN: Played_in ON Cer_ID	30
Performs	(e),(g),(m)	JOIN: Song ON Song_ID	30 - 130
	(e),(g)	JOIN: Artist ON Art_ID	30 - 100
Played_In	(k)	JOIN: Song ON Song_ID	30
	(k)	JOIN: Ceremony ON Cer_ID	30
Uses	(b)	JOIN: Musical_Instrument ON Ins_ID	50
	(b)	PREDICATE: Song_ID	50
Contain	(l)	JOIN: Album ON Alb_ID	40
	(l)	JOIN: Song ON Song_ID	40
Era	(i)	PREDICATE: Era_Name	70
	(i)	PREDICATE: Era_StartYear	70
	(i)	PREDICATE: Era_EndYear	70
	(i)	JOIN: Song ON Song_ID	70

Additional indexes to be created in MS-SQL on the query transactions for the AllUsersSearch user views of MusicDB

Table	**Index**
Song	Song_ID Song_Title
Musical_Instument	Ins_ID Ins_Name
Artist	Art_ID Art_Name
Uses	Song_ID Ins_ID
Perform	Song_ID Art_ID
Album	Alb_ID Alb_Name
Ceremony	Cer_ID Cer_Name

F. User View Analysis

The first phase of the database design methodology involved the production of a conceptual data model for either the single user view or a number of combined user views identified during the requirements collection and analysis stage. In Project 1 we identified three types of user views, namely Member, Song Owner, and Researcher. Following an analysis of the data requirements for these user views, we used the centralized approach to merge the requirements for the user views as follows:

- **AllUsersSearch**, consisting of the Member, Researcher and Song Owner user views.
- **StaffOwnerInsert**, consisting only of the Song Owner user view.

To explain, the AllUsersSearch user view contains the view for searching the records of the database such as songs, albums and instruments. This view is accessible to all three types of users, as the search function of the database is meant to be accessible to all types of users. As for the StaffOwnerInsert user view, it is only meant to be accessible only to Song Owners, as this type of member is the only type that is allowed to insert and update some tables such as Song, Artist and Ceremony.

The reason for using multiple types of user views is to make sure that each type of user has correct permissions to the database's records. Furthermore, this will ensure the security of the database, as all user types cannot access the data that are not allowed to them.

2. The MS-SQL script for creating indexes.

```
Use MusicDB
CREATE INDEX index_Artist on Artist(Art_ID,Art_Name);
CREATE INDEX index_Song on Song(Song_ID,Song_Title);
CREATE INDEX index_Instrument on Musical_Instrument(Ins_ID, Ins_Name)
CREATE INDEX index_Uses on Uses(Song_ID,Ins_ID)
CREATE INDEX index_Perform on Performs(Song_ID, Art_ID)
CREATE INDEX index_Album on Album(Alb_ID,Alb_Name)
CREATE INDEX index_Ceremony on Ceremony(Cer_ID, Cer_Name)
```

3. The MS-SQL script for creating user views.

```
Use MusicDB
GO

CREATE VIEW ListSong AS
SELECT Song_ID, Song_NoteKey,Song_Title,Song_Lyrics,
Song_Dancable,Song_Teachable,Song_URL,Song_YearReleased,
Song_Copyright,Song_DatePosted
FROM Song
GO

CREATE VIEW InstrumentOfSong AS
SELECT Song_Title,Ins_Name,Ins_Family
FROM Musical_Instrument m
JOIN Uses ON m.Ins_ID = Uses.Ins_ID
JOIN Song ON Uses.Song_ID = Song.Song_ID
GO

CREATE VIEW WriterOfSong AS
SELECT Song_Writer,Song_Title FROM Song_Writers
JOIN Song ON Song_Writers.Song_ID = Song.Song_ID
GO

CREATE VIEW ArtistOfSong AS
SELECT  Song.Song_Title,Art_Name FROM Performs
JOIN Song ON Performs.Song_ID = Song.Song_ID
JOIN Artist ON Performs.Art_ID = Artist.Art_ID
GO

CREATE VIEW RegionOfSong AS
SELECT Song_Title, Cou_Region FROM Song
JOIN Cou_Regions ON Song.Cou_ID = Cou_Regions.Cou_ID
GO

CREATE VIEW ArtistPerforms AS
SELECT Art_Name, Song_NoteKey,Song_Title,
Song_Lyrics,Song_Dancable,Song_Teachable,Song_URL,
Song_YearReleased,Song_CopyRight FROM Song
JOIN Performs ON Song.Song_ID = Performs.Song_ID
join Artist on Performs.Art_ID = Artist.Art_ID;
GO

CREATE VIEW SentimentOfSong AS
SELECT Song_Title,Song_Sentiment from Song_Sentiments
JOIN Song ON Song.Song_ID = Song_Sentiments.Song_ID;
GO

CREATE VIEW EraOfSong AS
SELECT Song.Song_ID,Song_Title,Era_Name,Era_StartYear,Era_EndYear
FROM Era JOIN Song ON Song.Song_ID = Era.Song_ID;
GO

CREATE VIEW PatternNoteOfSong AS
SELECT Song_ID,Song_Title,Song_NoteKey FROM Song;
GO

CREATE VIEW CeremonyOfSong AS
SELECT Song_Title, Cer_Name FROM Song JOIN Played_in ON Played_in.Song_ID =
Song.Song_ID JOIN Ceremony ON Ceremony.Cer_ID = Played_in.Cer_ID
GO

CREATE VIEW AlbumsContainSong AS
SELECT Song.Song_ID,Song_Title,Alb_Name FROM Album
JOIN Contain ON Contain.Alb_ID = Album.Alb_ID
JOIN Song ON Contain.Song_ID = Song.Song_ID
GO

CREATE VIEW NumberOfSingerInEachSong AS
SELECT Song_Title,COUNT(Artist.Art_ID)AS TotalArtist
FROM Performs JOIN Artist ON Performs.Art_ID = Artist.Art_ID
JOIN Song ON Song.Song_ID = Performs.Song_ID GROUP BY Song_Title
GO

CREATE VIEW SongTeachable AS
SELECT Song_Title,Song_Teachable FROM Song;
GO

CREATE VIEW SongDancable AS
SELECT Song_Title, Song_Dancable FROM Song;
GO

CREATE VIEW CopyRightOfSong AS
SELECT Song_Title,Song_CopyRight FROM Song;
GO
```

```
CREATE VIEW GenreOfSong AS
SELECT Song_ID,Alb_Genre,Song_Title FROM Song JOIN Album ON Album.Alb_ID =
Song.Alb_ID;
GO

CREATE VIEW PublisherOfSong AS
SELECT Song_Writer,Song_Title FROM Song_Writers JOIN Song ON
Song_Writers.Song_ID = Song.Song_ID
GO

CREATE VIEW ReleasedYearOfSong AS
SELECT Song_Title, Song_YearReleased FROM Song
GO

CREATE VIEW PostedDateOfSong AS
SELECT Song_Title, Song_DatePosted FROM Song
GO

CREATE VIEW LyricsOfSong AS
SELECT Song_Title,Song_Lyrics FROM Song
GO
```

4. The MS-SQL script for updating views and handling transactions that manipulate databases. In this script, the creation of triggers and assertion may be considered.

Transaction	**Procedure**

(b) List the details of all instruments that are used in a song	CREATE PROCEDURE proc_SongInstrument
	@SongName varchar(255)
	AS
	BEGIN
	    SET NOCOUNT ON
	    SELECT Ins_Name,Ins_Family
	FROM Musical_Instrument m JOIN Uses ON m.Ins_ID = Uses.Ins_ID
	WHERE Uses.Song_ID IN (SELECT (Song_ID)
	FROM Song
	WHERE Song_Title = @SongName
	)
	END

(c) List songs that contain the query word (e.g. return songs that contain the word “love” in title)	CREATE PROCEDURE proc_SongQueryWord
	@QueryWord varchar(255)
	AS
	BEGIN
	    SET NOCOUNT ON
	    SELECT * FROM Song WHERE Song_Title LIKE
	'%'+@QueryWord+'%';

	END

(g) Identify songs that are performed by an artist	CREATE PROCEDURE proc_ArtistSong
	@ArtistName varchar(255)
	AS
	BEGIN
	        SET NOCOUNT ON
	        SELECT Song.Song_ID, Song_NoteKey,Song_Title, Song_Lyrics,Song_Dancable,Song_Teachable,Song_URL,Song_YearReleased,Song_CopyRight
	FROM Song JOIN Performs ON Song.Song_ID = Performs.Song_ID join Artist on Performs.Art_ID = Artist.Art_ID WHERE Artist.Art_ID IN (SELECT (Art_ID)
	FROM Artist WHERE Art_Name = @ArtistName
	)
	END

(l) Identify total number of a word or a phrase that are appeared in title or lyrics	CREATE PROCEDURE proc_AlbumContainSong
	@SongName varchar(255)
	AS
	BEGIN
	        SET NOCOUNT ON
	        SELECT distinct(Alb_Name)
	FROM Album JOIN Contain ON Contain.Alb_ID = Album.Alb_ID JOIN Song ON Contain.Song_ID IN (SELECT Song_ID
	FROM Song
	WHERE Song_Title = @SongName
	)
	END

(n) Identify whether songs are teachable or not	CREATE PROCEDURE proc_SongTeachable
	@SongName varchar(255)
	AS
	BEGIN
	        SET NOCOUNT ON
	        DECLARE @teach char(1)
	        SET @teach = (SELECT top 1 Song_Teachable
	FROM Song WHERE Song_Title = @SongName)
	        IF @teach = '1'
	                SELECT top 1 Song_Title, 'Yes' AS Teachable FROM Song WHERE Song_Title = @SongName;
	        ELSE IF @teach = '0'
	                SELECT top 1 Song_Title, 'No' AS Teachable FROM Song WHERE Song_Title = @SongName;
	END

(o) Identify whether songs are danceable or not	CREATE PROCEDURE proc_SongDancable @SongName varchar(255) AS

	BEGIN
	        SET NOCOUNT ON
	        DECLARE @dance char(1)
	        SET @dance = (SELECT top 1 Song_Teachable FROM Song WHERE Song_Title = @SongName)
	        IF @dance = '1'
	                SELECT top 1 Song_Title, 'Yes' AS Dancable FROM Song WHERE Song_Title = @SongName;
	        ELSE IF @dance = '0'
	                SELECT top 1 Song_Title, 'No' AS Dancable FROM Song WHERE Song_Title = @SongName;
	END

Search song	CREATE PROCEDURE proc_Search
	@Song_Lyrics varchar(255), @Song_Title varchar(255),
	@Song_Sentiment varchar(255), @Song_Instrument varchar(255),
	@Song_Writer varchar(255), @Song_Country varchar(255)
	AS
	BEGIN
	        SET NOCOUNT ON
	        IF(@Song_Lyrics is null)
	                SET @Song_Lyrics = '%'
	        IF(@Song_Title is null)
	                SET @Song_Title = '%'
	        IF(@Song_Sentiment is null)
	                SET @Song_Sentiment = '%'
	        IF(@Song_Instrument is null)
	                SET @Song_Instrument = '%'
	        IF(@Song_Writer is null)
	                SET @Song_Writer = '%'
	        IF(@Song_Country is null)
	                SET @Song_Country = '%'
	
	        SELECT * FROM Song JOIN Song_Sentiments
	ON Song_Lyrics LIKE '%'+ @Song_Lyrics +'%'
	AND Song_Title LIKE '%'+ @Song_Title +'%'
	AND Song_Sentiment LIKE '%' + @Song_Sentiment +'%'
	AND Song.Song_ID IN (SELECT Song_ID
	FROM Uses JOIN Musical_Instrument
	ON Uses.Ins_ID IN (SELECT Ins_ID
	FROM Musical_Instrument
	WHERE Musical_Instrument.Ins_Name LIKE
	'%'+@Song_Instrument+'%'
	        )
	)
	AND Song.Song_ID IN (SELECT Song.Song_ID
	FROM Song JOIN Song_Writers
	ON Song.Song_ID IN (
	SELECT Song_ID FROM Song_Writers

	WHERE Song_Writer LIKE '%' + @Song_Writer +'%')) AND Song.Song_ID IN (SELECT Song.Song_ID FROM Song JOIN Era ON Era.Song_ID = Song.Song_ID JOIN Country ON Era.Cou_ID IN (SELECT Cou_ID FROM Country WHERE Cou_Name LIKE '%'+@Song_Country+'%')) END
Insert reviews	CREATE PROCEDURE proc_InsertReview @SongName varchar(255), @MemberName varchar(255), @Comment varchar(255), @rating int, @date date AS BEGIN SET NOCOUNT ON INSERT INTO Review VALUES((SELECT CONCAT('Re0000', CAST(SUBSTRING(MAX(Review.Rev_ID), CHARINDEX('Re', MAX(Review.Rev_ID), 0)+2, 7) AS INT) + 1) FROM Review) ,@Comment, @rating, @date, (SELECT top 1 Mem_ID FROM Member WHERE Mem_FullName LIKE '%'+@MemberName+'%'), (SELECT top 1 Song_ID FROM Song WHERE Song_Title LIKE '%'+@SongName+'%')); END
Insert new song (only for SongOwnerSelect user view)	CREATE PROCEDURE proc_InsertSong @noteKey char(3), @title varchar(255), @lyrics varchar(255), @dancable char(1), @teachable char(1), @url varchar(255), @yearreleased int, @copyright char(50), @membername char(30), @countryname char(20), @albumname char(20) AS BEGIN SET NOCOUNT ON DECLARE @songID char(8), @memID char(8), @couID char(8), @albID char(8);

	SET @songID = (SELECT CONCAT('S000', CAST(SUBSTRING(MAX(Song.Song_ID), CHARINDEX('S', MAX(Song.Song_ID), 0)+1, 7) AS INT) + 1) FROM Song)
	   SET @memID = (SELECT TOP 1 Mem_ID FROM Member WHERE @membername = Mem_FullName)
	   SET @couID = (SELECT TOP 1 Cou_ID FROM Country WHERE @countryname = Cou_Name)
	   SET @albID = (SELECT TOP 1 Alb_ID FROM Album WHERE @albumname = Alb_Name)
	   INSERT INTO MusicDB.dbo.Song
	   VALUES (@songID, @noteKey, @title, @lyrics, @dancable, @teachable, @url, @yearreleased, @copyright, GETDATE(), @memID, @couID, @albID)
	END

Meaning	SQL script
List the details of songs	SELECT * FROM Song
List the details of all instruments that are used in a song (suppose would like to search instruments of song 'Kiss Kiss')	EXEC proc_SongInstrument @SongName = 'Kiss Kiss'
List songs that contain the query word (e.g. return songs that contain the word “love” in title)	EXEC proc_SongQueryWord @QueryWord = 'love'

Identify song writers of songs in order	SELECT Song_Writer,Song_Title
	FROM Song_Writers JOIN Song
	ON Song_Writers.Song_ID = Song.Song_ID
	ORDER BY Song_Writer

Identify who is an artist of a song	SELECT Song.Song_Title,Art_Name
	FROM Performs JOIN Song
	ON Performs.Song_ID = Song.Song_ID JOIN Artist
	ON Performs.Art_ID = Artist.Art_ID
	ORDER BY Song_Title;

Identify region of the songs (i.e. created in what region)	SELECT Song_Title, Cou_Region
	FROM Song JOIN Cou_Regions
	ON Song.Cou_ID = Cou_Regions.Cou_ID

Identify songs that are performed by an artist	EXEC proc_ArtistSong @ArtistName = 'Paulo Coelo'

Identify the sentiment of a song	SELECT Song_Title,Song_Sentiment
	FROM Song_Sentiments JOIN Song

	ON Song.Song_ID = Song_Sentiments.Song_ID

Identify era of songs order by song title	SELECT Song.Song_ID, Song_Title, Era_Name, Era_StartYear, Era_EndYear
	FROM Era JOIN Song
	ON Song.Song_ID = Era.Song_ID
	ORDER BY Song_ID;

Identify pattern of notes that songs use	SELECT Song_ID,Song_Title,Song_NoteKey
	FROM Song;

Identify ceremony of songs	SELECT Song_Title, Cer_Name
	FROM Song JOIN Played_in
	ON Played_in.Song_ID = Song.Song_ID JOIN Ceremony
	ON Ceremony.Cer_ID = Played_in.Cer_ID

Identify albums that contain a song	EXEC proc_AlbumContainSong @SongName = 'lemon'

Identify total number of singers	SELECT Song_Title, COUNT(Artist.Art_ID) AS TotalArtist
	FROM Performs JOIN Artist
	ON Performs.Art_ID = Artist.Art_ID JOIN Song
	ON Song.Song_ID = Performs.Song_ID
	GROUP BY Song_Title;

Identify whether song is teachable or not	EXEC proc_SongTeachable @SongName = 'Lemon'
Identify whether song is danceable or not	EXEC proc_SongDancable @SongName = 'Kiss Kiss'

Identify what kind of copyright of songs	SELECT Song_Title,Song_CopyRight
	FROM Song Order by Song_CopyRight

Identify genre of songs (eg. Jazz)	SELECT Song_ID,Alb_Genre,Song_Title
	FROM Song JOIN Album
	ON Album.Alb_ID = Song.Alb_ID

Identify publisher of songs	SELECT Song_Writer,Song_Title
	FROM Song_Writers JOIN Song
	ON Song_Writers.Song_ID = Song.Song_ID
	ORDER BY Song_Writer

Identify released date of songs	SELECT Song_Title, Song_YearReleased
	FROM Song Order by Song_YearReleased

Identify posted date of songs (post in system)	SELECT Song_Title, Song_DatePosted
	FROM Song

Identify lyrics of songs	SELECT Song_Title,Song_Lyrics
	FROM Song

Search Song with full inputs	EXEC proc_Search @Song_Lyrics = 'get' , @Song_Title = 'I' , @Song_Sentiment = 'cheerful' , @Song_Instrument = 'chakhe', @Song_Writer = 'Vera Ward', @Song_Country = 'Thailand'
Search Song with missing some inputs	EXEC proc_Search @Song_Lyrics = null , @Song_Title = null , @Song_Sentiment = 'cheerful' ,@Song_Instrument = null, @Song_Writer = null, @Song_Country = null
Insert Review	EXEC proc_InsertReview @SongName = 'Lemon', @MemberName = 'Freya' , @Comment = 'Nice song!', @rating = 5, @date = '1993-12-21'
Insert new song (only for SongOwnerInsert user view)	EXEC proc_InsertSong @noteKey = 6, @title = 'I want to break free', @lyrics = 'I get on with life as a doctor, I''m a kind and generous kinda person.', @dancable = 1, @teachable = 1, @url = '134.132.99.199', @yearreleased = 1986, @copyright = 'Purchasing', @membername = 'Freya Smith', @countryname = 'Thailand', @albumname = 'Tennessee Williams'