Digital Music Store

COMP 421 – Database Systems

Nicholas Colotouros, Rita Lu, Alexandre Sincennes

Requirements Analysis

Introduction

Purpose

This application will store the information and constrains required to sell and distribute music digitally through an online interface. Digital distribution is a great way to sell products which do not require a physical medium eliminates cost of production and ensures nothing will ever be out of stock.

With so much music from around the world it's impossible for a store to physically carry them all. Even within a single store it's often difficult to find something a client will truly enjoy. A database can be used to adapt recommendations based on the client's previous purchases and song reviews.

It's also quite difficult for salesmen to effectively know all of this music. With a database storing user reviews, customer preferences and previous purchases, a recommender can be constructed to display songs that appeal to the user and entice them to buy more.

Scope and Special Requirements

The music being sold in the online database will cover music from all around the world, in various languages. The system will support trade regions to allow for differences in what music can be sold to customers in different countries.

We will focus on effective organization and categorization of the music, allowing customers to find music based on everything from artist name, language and genre to BPM, country of origin and song file format. A user will also be able to view the most popular songs being sold and filter them based on any criteria they wish. We will also have a recommender system which will find music similar to a customer's purchases and reviews of songs.

Terminology

Album artist: The primary artist that produced a given album.

Artist: Any person or group of people can consider themselves artists provided they have a song listed in the database.

BPM: Beats per minute, the speed of the song.

Rating: A customer's review of a song from 1 to 5, 1 being the worst and 5 being the best.

Format: The data format that the song is stored. For example: mp3, flac and wav are possible formats.

Inspirations

Our database is based on the iTunes and Amazon digital music stores.

Database Description

Entities and their Attributes

The following entities will be stored in the tables of a relational database:

Song: a song is a digital-format piece of music. Its attributes consists of a title, the song length, its date of creation or publishing, its speed (in beats per minute, bpm), as well the its lyrics. It is identified by a unique primary key called sID.

Artist: an artist is a person or group who publish songs. They have a name, a description and a date of formation. It is identified by a unique primary key called artID.

Album: an album is a collection of one or more songs published together. It has a title, a date of publishing, and its artwork as attributes, and can be identified by the unique primary key albID.

Genre: a genre is a category of music. It has a name as an attribute, can be identified by the unique primary key genID.

Customer: a customer is a person who purchases songs and albums in the store, and who can rate them. A customer is defined by the attributes that are his first name, last name, email address, his credit card number, birth date and password. The customer can be identified by the unique primary key cID.

Order: a record of transactions between the customer and the system. It has the date of purchase as well as price as attributes and can be identified by the unique primary key oID.

Product: a record of which songs and/or album is a part of an order. It has the price of the song and/or album as an attribute and can be identified by the unique primary key pID.

Country: a country is a territory which has sovereignty over digital music publishing and purchasing. It has a name as an attribute, and can be identified by the unique primary key coID.

Language: a language is defined by its name attribute and its unique primary key langID.

Format: the file format of the song. its attribute is the name of the file extension, and can be identified by the unique primary key fID.

Relationships

wrote: An *artist* wrote a *song.* This is a many-to-many relationship because an artist can write many songs and a song can be written by many artists.

album artist in: An *artist* is an album artist in an *album*. This is a one-to-many relationship because an artist can be an album artist in many albums and each album can have only one album artist. This is distinct from an artist writing a song because an album can contain songs written by artists that are not the album artist.

track in: A *song* is a track in an *album*. This is a one-to-many relationship because each song can only belong to one album while an album can consist of multiple songs.

associated with: A *song* can be associated with a *genre*. This is a many-to-many relationship as a single song can have multiple genres and a genre can be associated with multiple songs.

places: A *customer* places an *order*. This is a one-to-many relationship since a single customer can place many orders while each order can only be placed by one customer.

contains product: An *order* contains a *product*. This is a many-to-many relationship because each order can contain multiple products and each product can be a part of multiple orders.

song is a: A *song* is a *product*. This is a one-to-one relationship because each product correlates to one song and each song correlates to one product.

album is a: An *album* is a *product*. This is a one-to-one relationship because each product correlates to one album and each album correlates to one song

rates: A *song* is rated by a *customer.* This is a many-to-many relationship because each song can be rated by multiple customers while each customer can rate multiple songs. This relationship also has an attribute called rating to store the numerical value of the customer's rating of the song.

is located in: A *customer* located in a *country*. This is a one-to-many relationship since each customer is located in a single country while each country can contain multiple customers.

is from: An *artist* is from a *country*. THis is a one-to-many relationship since each artist is located in a single country while each country can contain multiple artists.

speaks: A *country* speaks a *language*. This is a many-to-many relationship since each country can speak multiple languages and each language can be spoken in multiple countries.

is written in: A *song* is written in a *language.* This is a many-to-many relationship as each song can be written in multiple languages and a language can be used in multiple songs.

communicates in: A *customer* communicates in a *language*. This is a one-to-many relationship since a customer only communicates in one language while each language can be used by multiple customers for communication.

is in format: A *song* is in a *format*. This is a one-to-many relationship as each song only comes in one format while each format can belong to multiple songs.

Functional Requirements

topSongs(Genre g, int numberOfSongs, date startDate, date endDate)
This function returns a list of songs in a genre *g*, sorted according to the amount of orders of that song placed between *startDate* and *endDate*, and in the number determined by the argument *numberOfSongs*.

topArtists(Genre g, int numberOfArtists, date startDate, date endDate)
This function returns a list of artists of a genre *g*, sorted according to the amount of orders of the artist's songs in that genre placed between *startDate* and *endDate*, and in the number determined by the argument *numberOfSongs*.

checkout(List<Product> p, Customer c)
This function creates an *Order consisting of* products specified as arguments, which is associated to the Customer c. This function calculates the total *price* of the order from the individual *prices* of each order and returns the new order created.

getPurchased(Customer c)
This function returns a list of *Orders* associated with the customer *c*.

getRecommended(Customer c, int numberOfSongs)
This function returns a list of songs based on the customer *c*'s purchases and ratings in the number determined by the argument *numberOfSongs*. The recommendations are of top songs in genres where the user has made the most purchases and highest ratings.

Unique and Difficult Aspects

A unique aspect of making this database is figuring out how to store songs when they are written by multiple artists. Ultimately a single artist will get credit for the album but it's possible many different artists appeared on a song. Having an album artist allows for collaborations on songs on a given artists' album and allows for compilation and cover albums.

Another tricky aspect was figuring out how to make sure an order had at least one song or album in an order. We ensured this by having making an entity called product which has an ID and a price. Each song and album have an "is a" relationship to product and an order requires at least one product.

Entity Relationship Diagram

The only constraint that cannot be shown through the ER diagram is that a product should not exist on its own. When a product is entered into the database it must be in the form of a song or an album.

email
birthdate
name
cID
password
credit_card_number
Customer
price
date
oID
Order
places
is located in
communicates in
rates
rating
genID
Genre
name
associated with
has purchased
title
song_length
date_recorded
BPM
Song
is a
Product
pID
price
Extentsion
Format
fID
is in format
is written in
lyrics
sID
Wrote
track_number
is a track in
albID
title
Album
is a
date_released
artwork
Language
langID
name
Speaks
description
date_formed
name
artID
Artist
is album artist in
is from
Country
tax_rate
name
coID

Relations Based on E/R Diagram

Relations

Entities

Song(sID, title, length, date_recorded, bpm, lyrics)
Artist(artID, name, description, date_formed)
Album(albID, title, date_released, artwork)
Product(pID, price)
Genre(genID, name)
Customer(cID, name, email, credit_card_number, birthdate, password)
Order(oID, date, price)
Country(coID, name)
Language(langID, name)
Format(fID, extension)

Relationships

wrote(artID, sID)
album_artist_in(artID, albID)
track_in(sID, albID, track_number)
associated_with(sID, genID)
places(cID, oID)
contains_product(oID, pID)
song_is_a(pID, sID)
album_is_a(pID, albID)
rates(sID, cID, rating)
is_located_in(cID, coID)
is_from(artID, coID)
speaks(coID, langID)
communicates_in(cID, langID)
is_written_in(sID, langID)
is_in_format(sID, fID)

Opportunities to Combine Redundancies

As far as we can tell we aren't storing any redundant any information. Arguments can be made for removing certain ID attributes from some entities but music and everything associated to it are not constant. What may be called heavy metal today may not be the same heavy metal 20 years from now. Our justification for IDs on every entity allows the database to keep with the ever changing music without having to constantly change the database entries.