CST 363
Project 1

Scuzzy Records Database

Nathan Warren-Acord

Cody Young

Abby Packham

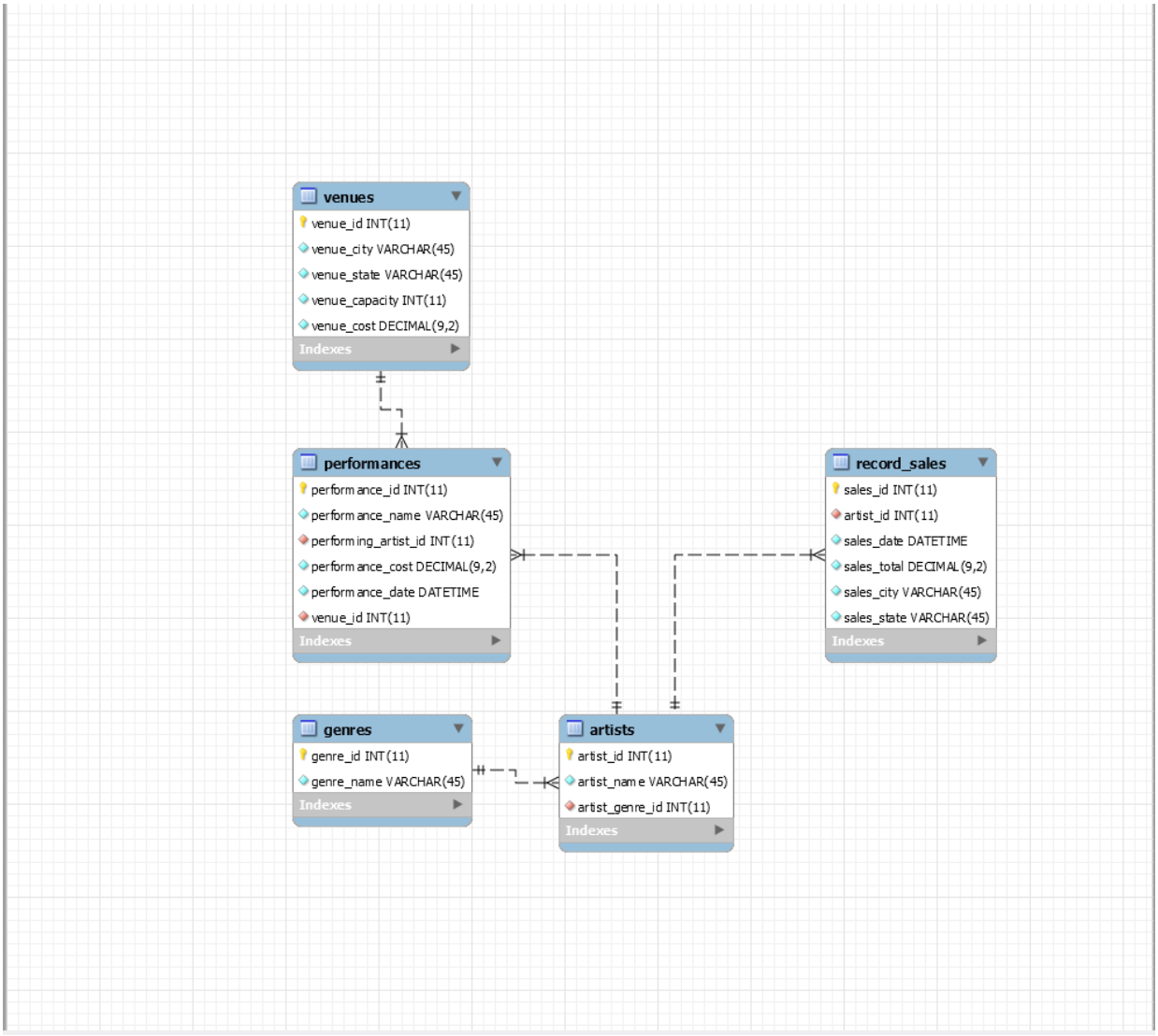
Application Overview

Our application is a database that manages data of artists signed to a fictional record label. With our application, the user, presumably a record label employee, can query the database and return information such as a list of artists signed to our label, their assigned genres, venues performed at, and sales metrics according to performance or venue. In a future implantation, it will also return statistics related to how well an artist is doing, such as sales and performance numbers.

From the main page, a user can select from the drop down menu whether they want to review data from certain categories, or perform write operations such as inserting or deleting artist or venue data from the existing database. When writing to the database, the user is prompted to enter in various fields new data values as strings for the corresponding rows.

Keeping track of performance metrics is essential in most modern businesses, and our database allows stakeholders to make important business decisions based off of metrics, or for other employees to simply update and keep track of large amounts of information accurately.

The E-R diagram is pictured as follows.


venues
venue_id INT(11)
venue_city VARCHAR(45)
venue_state VARCHAR(45)
venue_capacity INT(11)
venue_cost DECIMAL(9,2)
Indexes
performances
performance_id INT(11)
performance_name VARCHAR(45)
performing_artist_id INT(11)
performance_cost DECIMAL(9,2)
performance_date DATETIME
venue_id INT(11)
Indexes
record_sales
sales_id INT(11)
artist_id INT(11)
sales_date DATETIME
sales_total DECIMAL(9,2)
sales_city VARCHAR(45)
sales_state VARCHAR(45)
Indexes
genres
genre_id INT(11)
genre_name VARCHAR(45)
Indexes
artists
artist_id INT(11)
artist_name VARCHAR(45)
artist_genre_id INT(11)
Indexes

Normalization

Our database is normalized, despite the presence of a few columns in tables that are not normalized, such as multiple artists under the same genre in the "artists" table. This makes sense from a logical and real world perspective – one genre, such as jazz, can have multiple artists classified under it, and inventing a new genre for each artist would be a waste of time and effort. Most importantly, our database is normalized because it allows for efficient read/write operations. Record labels often have many artists under a large number of genres, and having an efficient database saves a company time and money on administrative tasks. Additionally, it allows for ease of future maintenance in case the company decides to switch its focus and branding (e.g. switching from a rock/pop label to a strictly jazz/electronic label), which can require large movements of data, or with managing invoices and returning sales data.

Part 2 Update

We refactored our database using the "data warehouse" approach. In contrast to its original design, our database is now simplified and relies on a single fact table, artist_stats, with two dimension tables, events and artists. By designing our database around these tables, a user can send and receive queries more efficiently from the database while also having the capability to request queries such as best selling artists, highest profits by city, and other complex queries that OLAP databases are well suited for.

Additionally, the new design improves data integrity. For example, a user who tries to delete a sales entry without specifying an artist or event cannot do so.

The updated E-R diagram is pictured on the following page.

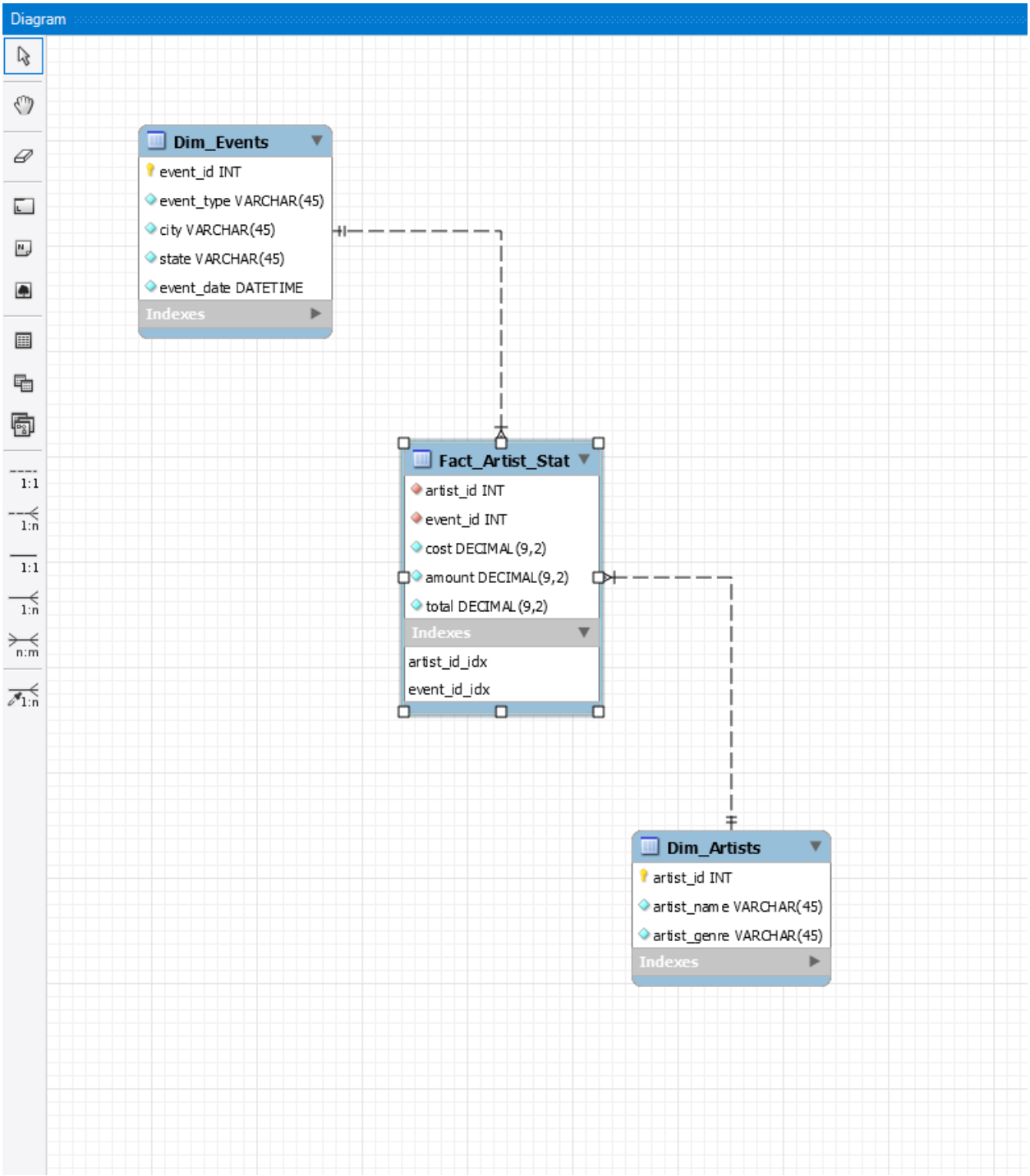


Diagram
Dim_Events
event_id INT
event_type VARCHAR(45)
city VARCHAR(45)
state VARCHAR(45)
event_date DATETIME
Indexes
Fact_Artist_Stat
artist_id INT
event_id INT
cost DECIMAL(9,2)
amount DECIMAL(9,2)
total DECIMAL(9,2)
Indexes
artist_id_idx
event_id_idx
Dim_Artists
artist_id INT
artist_name VARCHAR(45)
artist_genre VARCHAR(45)
Indexes
1:1
1:n
1:1
1:n
n:m
1:n