4. Relational Schemas

Songs(Name: string, Tempo: int)

Chords(Name: string)

Progressions(ID: int, Name: string, Name: string)

Notes(Octave: int, Tone: char, Accidental: string)

Motifs(ID: int, Name: string, ID: int, Rest?: Boolean, Name: string,

Octave: int, Tone: char, Accidental: string)

Rhythms(ID: int, Measures: int, Time_Signatures: int)

RhythmDuration(ID: int, Rest?: Boolean, Name: string)

Scales(Name: string, Mode: string)

Durations(Rest?: Boolean, Name: string)

Instruments(Name: string, Range: string, Role: string)

InstrumentsInSongs(Name: string, Name: string)

Many of these schemas don’t fit well into the ER model. For example, a progression can be an arbitrary sequence of chords, which can be made up of an arbitrary group of notes. Here, we are generating data on the fly instead of trying to keep referential integrity. However, once a chord progression has been identified, we would like to save it in the database with a unique ID. For example, the progression I-V-I in the key of C major would have the chords C G C, C consisting of the notes C, E, G, and G consisting of the notes G, B, D. Since a progression can be anywhere from 2 to an arbitrarily long number of chords and each chord can be 2 to 13 notes, it doesn’t make senses to try use the ER model for this. Many of our relations have this property. Another example is Motifs. Each song has a main motif, which consists of a few notes in a specified sequence, each with its own rhythmic value. We’re unsure how to represent this in the ER style. Perhaps most of our data will be in memory as we operate on it to produce our MIDI files. It’s difficult to say at this point.