Advanced Database Concepts
Project 2:
Administrator's Guide

By: Jordan Bechek

Index:

Overview

The database systems I created are designed to hold the data and metadata associated with a company that provides a DJ service. Day to day information regarding the DJ's, clients, and services provided, as well as information about the music the company offers to its clients is held in the transactional database. In addition, recent changes to band membership, and recent changes to DJ information, such as salary, are also tracked here. Historical information, up until present day, is held in the informational database (data warehouse). Changes to data are immediately reflected in the transactional database, as its focus is on the intake of relevant data. Those changes are then transposed to the informational database in a more report-friendly way. The scripts for both systems are fully commented in sections, allowing each database object to easily be referenced by this document.

Transactional Database: Summary

The transactional database is a fully normalized database focusing on three main aspects of the company: band information, music information, and business information. The band section holds information about each band the company offers music from, and information about their members, such as what positions he or she holds in which bands. The music section holds information about the albums releases by each band, as well as the record companies who produced them. In addition, the censored and uncensored versions of tracks of those albums that the company can offer are also held here. Lastly, the business section contains information about their DJ's, clients, and transactions between them. Additionally,

the specialty of each DJ (DJ Type), such as weddings or children's parties, is held here along with the genres of music respective to each type. Changes to DJ information and band membership are also logged here.

Transactional Database: Tables

Abstract

All of the tables in the database contain a field that captures the timestamp of the last insert. This field is used in the daily transfer of data to the data warehouse. Every table can be created with the execution of a single stored procedure (see sp_create_database in Transactional Database: Stored Procedures). Each table is a parent, child, or auditing table, and each table's relational notation is given in its respective section following this paragraph, along with the section of the script it can be found in. All of the relationships between these tables can be found in the ERD diagram found later in this document.

Parent tables

All parent tables contain basic information about various aspects of the system, and are used in relationships with other parent tables to give a comprehensive outlook on the entire system. The parent tables include:

- tblGenre (GenreID, Genre, LastUpdate) – Section 2A
- tblPosition (PositionID, Position, LastUpdate) – Section 2B
- tblBand (BandID, Band, FoundationYear, LastUpdate) – Section 2C
- tblArtist (ArtistID, FName, LName, LastUpdate) – Section 2D
- tblRecordCompany (RecordCompanyID, RecordCompany, FoundationYear, LastUpdate) – Section 2E
- tblDJType (DJTypeID, DJType, LastUpdate) – Section 2L
- tblDJ (DJID, FName, LName, Salary, LastUpdate) – Section 2M
- tblClient (ClientID, FName, LName, LastUpdate) – Section 2N

Child tables

All of the child tables serve to capture relationships between parent tables. Some of them strictly contain foreign keys in order to accomplish this, while others contain other information as well. The child tables include:

- tblArtistPosition (*ArtistID*, *PositionID*, LastUpdate) – Section 2F
- tblBandArtist (*BandID*, *ArtistID*, *PositionID*, LastUpdate) – Section 2G
- tblAlbum (AlbumID, Album, ReleaseDate, Price, *GenreID*, LastUpdate) – Section 2H
- tblRecordCompanyAlbum (*RecordCompanyID*, *AlbumID*, LastUpdate) – Section 2I
- tblAlbumBand (*BandID*, *AlbumID*, LastUpdate) – Section 2J
- tblTrack (Track, IsCensored, *AlbumID*, IsExplicit, Duration, LastUpdate) – Section 2K
- tblDJTypeGenre (*DJTypeID*, *GenreID*, LastUpdate) – Section 2O
- tblDJTypeAssignment (*DJID*, *DJType*, LastUpdate) – Section 2P
- tblTransaction (TransactionID, *ClientID*, *DJID*, *DJTypeID*, TransactionDate, Price, LastUpdate) – Section 2Q

Audit tables

The database contains two audit tables that track changes to tblBandArtist and tblDJ respectively. The audit tables include:

- tblBandAudit (ChangeID, BandID, ArtistID, PositionID, Event, TimeStamp) – Section 2R
- tblDJAudit (ChangeID, DJID, FName, LName, Salary, Event, LastUpdate) – Section 2S

Transactional Database: Stored Procedures

Abstract

There are two stored procedures in the transactional database that serve to create the table structure, and subsequently fill those tables with randomly

generated test data. They can be found in sections 2 and 5 of the script. The following is a description of each, along with recommendations on their use.

sp_create_database()

This stored procedure is used to fill the empty database with all the tables listed above. It can be found in section 2 of the script, and it must be run immediately after the database is created in order for the system to function properly. It has no parameters, and uses no other procedures or functions.

sp_generate_test_data(…)

This stored procedure is used to fill the tables with randomly generated test data. It can be found in section 5 of the script, and it cannot be run before the functions and triggers, found in sections 3 and 4 of the script respectively, have been created. The procedure has 8 integer type parameters to ensure that the test data is at the desired magnitude. The following is a list of those parameters along with the section of the script they can be hardcoded at, if necessary.

- ArtistAmount – amount of randomly generated artists desired
  - Recommended value – 50 – Section 5E
- BandAmount – amount of randomly generate bands desired
  - Recommended value – 10 – Section 5G
- RCAmount – amount of randomly generated record companies desired
  - Recommended value – 20 – Section 5I
- AlbumAmount - amount of randomly generate albums desired
  - Recommended value – 25 – Section 5J
- TrackAmount – amount of randomly generated tracks desired
  - Recommended value – 300 – Section 5L
- DJAmount – amount of randomly generated DJ's desired
  - Recommended value – 15 – Section 5M
- ClientAmount – amount of randomly generated clients desired
  - Recommended value – 35 – Section 5N
- TransactionAmount – amount of randomly generated transactions desired
  - Recommended value – 100 – Section 5P

At section 5L, it is important to note that each track has an equal chance of being each genre, but specific probabilities of being explicit depending on its genre. For example, there is a 0% chance of a track being explicit if it is of the 'Blues' genre, but there is an 80% chance if the track is of the 'Rap' genre. These probabilities can be changed by altering the integer to the right of the inequality in the if statements that compare against the variable 'Probability'.

Transactional Database: Functions

Abstract

All of the functions in this database serve to randomly generate test data, and are therefore all called in sp_generate_test_data(…). The functions can be created before or after the triggers, but must be created before sp_generate_test_data(…) is run. They can be found in section 4 of the script.

fnRandomInt(…)

This function accepts two integer type parameters, 'min' and 'max'. The result is a randomly generated integer that is between the 'min' and 'max' values. It can be found in section 4A.

fnRandomName()

This function has no parameters, and returns a randomly generated first and last name concatenated as one field. The names are randomly selected from temporary tables, which can hold as many names as desired. It can be found in section 4B.

fnRandomDate(…)

This function accepts two date type parameters, 'StartDate' and 'EndDate'. The result is a randomly generated date that is between the two specified dates. It can be found in section 4C.

fnRandomBand()

This function has no parameters, and returns a randomly generated adjective and noun concatenated as one field. The adjectives and nouns are randomly selected from temporary tables, which can hold as many items as desired. It can be found in section 4D.

fnRandomDecimal(…)

This function accepts two decimal type parameters, 'min' and 'max'. The result is a randomly generated decimal that is between the 'min' and 'max' values. It can be found in section 4E.

Transactional Database: Triggers

Abstract

All of the triggers in the database serve to either enforce business logic, such as making sure future dates are not inputted inappropriately, or to log changes to specific tables of interest. They can be found in section 3 of the script.

Business logic enforcers

The following is a list of triggers that enforce business logic, along with what tables they apply to, what business rule they enforce, and what section of the script they can be found in.

- trCheckBandDate – Checks FoundationYear before insert on tblBand to ensure it is not a future date – Section 3A
- trCheckReleaseDate – Checks ReleaseDate before insert on tblAlbum to ensure it is not a future date – Section 3B
- trCheckRCDate – Checks FoundationYear before insert on tblRecordCompany to ensure it is not a future date – Section 3C
- trExplicitCheck – Checks IsExplicit and IsCensored fields before insert on tblTrack to ensure that they correlate – Section 3D
- trCheckTransactionDate – Check TransactionDate before insert on tblTransaction to ensure it is not a future date – Section 3E

Logging/Auditing

The following is a list of triggers that log changes to tables, along with what tables they apply to, and what section of the script they can be found in.

- trBandArtistInsert – Logs inserts into tblBandArtist in tblBandAudit – Section 3F
- trBandArtistUpdate – Logs updates to tblBandArtist in tblBandAudit – Seciton 3G
- trBandArtistDelete – Logs deletes from tblBandArtist in tblBandAudit – Section 3H
- trDJInsert – Logs inserts into tblDJ in tblDJAudit – Section 3I
- trDJUpdate – logs updates to tblDJ in tblDJAudit – Section 3J
- trDJDelete – Logs deletes from tblDJ in tblDJAudit – Section 3K

Transactional Database: Indices

Abstract

The database contains indices on certain fields (other than primary and foreign keys) to expedite querying the data. These can be found in section 6 of the script.

Indices

The following is a list of the indices in the transactional database, including the field and table to apply to, and the section of the script they can be found in.

- idxBand – index on Band field of tblBand – Section 6A
- idxRecordCompany – index on RecordCompany field of tblRecordCompany – Section 6B

Transactional Database: Views

Abstract

The views in the transactional database serve to portray important information that is likely to be repeatedly queried. These can be found in section 7 of the script.

Views

The following is a list of the views in the transactional database, including their function and the fields they display, and the section of the script they can be found in.

- vwArtistBandUnion – Provides a view of all bands and artists labeled as either 'Artist', 'Band', or 'Artist/Band' – Section 7A
- vwRecordCompanyAlbums – Provides a view of record companies with the albums they have produced – Section 7B
- vwArtists – Provides a view of all artists by band, including those without bands – Section 7C
- vwBandMembership – Provides a view of all the changes logged in tblBandAudit (see Transactional Database: Tables) – Section 7D
- vwDJ – Provides a view of all the changes logged in tblDJAudit (see Transactional Database: Tables) – Section 7E
- vwTransByGenre – Provides a view of all the transactions made by genre – Section 7F

Informational Database: Summary

The informational database is a de-normalized reconstruction of the transactional database built to serve reporting based needs. The structure contains two fact tables, focusing on the two main aspects of reporting needed by the

company. These aspects are music and business, at the grain levels of tracks and transactions respectively. All information regarding individual tracks, such as the album they are from, are stored in one fact table, and all information regarding individual transactions, such as the clients and DJ's involved, is stored in the other fact table. Aggregate tables have been allocated as well to store frequently queried aggregated data.

Informational Database: Tables

Abstract

The table structure follows an extended snowflake schema, meaning it uses multiple fact tables, as mentioned above. There are multiple dimension tables and aggregate tables, and all of the relationships between these dimension and fact tables can be found in the ERD diagram found later in this document.

Dimension tables

All of the dimension tables are directly related to one of the fact tables. In addition, one of the dimensions is used in both areas of reporting, and a time dimension is used to track the dates of transactions. The dimension tables also are used in the aggregation of data. The following is a list of all the dimension tables, their relational notation, and what section of the script they can be found in.

- tblPosition (PositionID, Position) – Section 2A
- tblArtist (ArtistID, FName, LName) – Section 2B
- tblRecordCompany (RecordCompanyID, RecordCompany, FoundationYear) – Section 2C
- tblBand (BandID, Band, FoundationYear) – Section 2D
- tblGenre (GenreID, Genre) – Section 2E
- tblAlbum (AlbumID, Album, ReleaseDate, Price, *GenreID*) – Section 2F

- tblClient (ClientID, FName, LName) – Section 2G
- tblDJType (DJTypeID, DJType) – Section 2H
- tblDJ (DJID, FName, LName, Salary) – Section 2I
- tblTime (TimeID, Time) – Section 2J

Fact tables

The following is a list of the two fact tables, along with their relational notation, and the section of the script they can be found in.

- tblTrack (Track, IsCensored, *AlbumID*, IsExplicit, Duration, *RecordCompanyID*, *ArtistID*, *PositionID*, *BandID*) – Section 2K
- tblTransaction (*ClientID*, *DJID*, *DJTypeID*, Price, *GenreID*, *TimeID*) – Section 2L

Aggregate tables

Like the fact tables, there is one aggregate table for each aspect of reporting, each focusing on a specific part of that aspect that frequently requires aggregation. The following is a list of all the aggregate tables, their relational notation, which fields are aggregated and how, and what section of the script they can be found in.

- tblAlbumAggregate (*BandID*, *AlbumID*, TrackCount, Runtime) – Section 2M
  - TrackCount is a count of the tracks in each album
  - Runtime is a summation of the duration of every track in each album
- tblDJAggregate (*DJID*, *DJTypeID*, TransactionCount, TotalPrice) – Section 2N
  - TransactionCount is a count of the transactions by each DJ
  - TotalPrice is a summation of the price of every transaction by each DJ

Informational Database: Stored Procedures

Abstract

The stored procedures involved with the informational database serve to create and maintain the database. Additionally, there are stored procedures to migrate data from the transactional database daily, and to empty the warehouse, if necessary. The following is a description of each, along with recommendations on their use.

sp_create_warehouse()

This stored procedure is used to fill the empty database with all the tables listed above. It can be found in section 2 of the script, and it must be run immediately after the database is created in order for the system to function properly. It has no parameters, and uses no other procedures or functions.

sp_etl_all()

This stored procedure is used to initialize the database or, fill the tables created by sp_create_warehouse() with the data from the transactional database, and create the necessary indices on that data. It can be found in section 3 of the script, and it should only be run immediately after the database is created, or emptied (see sp_clean_warehouse() later in this section). This will bring all of the data from the transactional database over, so it may take some time to complete, depending on the amount of data being migrated. It has no parameters, and uses no other procedures or functions.

sp_maintenance()

This stored procedure is used to reorganize tables and recreate indices to improve performance. It can be found in section 4 of the script, and it should be used after major changes to the tables occur, for example, after sp_etl_all() or sp_etl_daily() (see next paragraph about this procedure for more information) are run. Doing this after those stored procedures are run could greatly improve the performance of future statements run on the effected tables. It has no parameters, and uses no other procedures or functions. It is highly recommended that the call

statement for this procedure be scheduled to run daily, which can be done by running the statements found in section 4A of the script.

sp_etl_daily()

This stored procedure is used to migrate data acquired the previous day to the informational system and should be scheduled, like sp_maintenance(). It can be found in section 5 of the script, and can be scheduled to run by running the statements found in section 5A of the script. It has no parameters, and uses no other procedures or functions.

sp_clean_warehouse()

This stored procedure is used to truncate all data from the entire system. The metadata is maintained, leaving an empty database. This should be used if the database needs to be reset for some reason, and can be found in section 8 of the script. It has no parameters, and uses no other procedures or functions.

Informational Database: Reports

Abstract

Due to the de-normalized nature of the data warehouse, complex reporting can be accomplished much easier. The reports focus on important music and business aspects, and utilize views or stored procedures to give a comprehensive report of the relevant data.

Reports as views

The following is a list of the reports as views, including their function and the fields they display, and the section of the script they can be found in.

- vwAlbumAggregate – Provides a view of the contents of tblAlbumAggregate (see Aggregate tables in Informational Database: Tables) – Section 6A
    - BandID and AlbumID are replaced with their actual names
- vwDJAggregate – Provides a view of the contents of tblDJAggregate (see Aggregate tables in Informational Database: Tables) – Section 6B
    - DJID and DJTypeID are replaced with their actual names
- vwTransactions – Provides a view of all the transactions from the last 4 weeks (starting from the time it is executed) – Section 6C
    - Full client and DJ names are given

Reports as stored procedures

The following is a list of the reports as stored procedures, including their function and the fields they display, and the section of the script they can be found in.

- sp_track_report() – Provides a view of all the tracks in the database by album by band – Section 6D
    - Tracks are labeled as 'Censored' or 'Uncensored'
    - The total tracks per band are given as subtotals
- sp_transaction_report() – Provides a view of all the transactions by client – Section 6E
    - The total transactions per client are given as subtotals

Informational Database: Imports/Exports

Abstract

Aggregated data stored in the data warehouse could be useful for different analytical processes. The following is an example of an export and import of aggregated data to show how data could be manipulated outside of the system and loaded back in.

Export

The export statement takes the entire contents of tblAlbumAggregate (see Aggregate tables in Informational Database: Tables) and exports it to a csv file in the

public folder. From there, it can be analyzed by other programs, or altered in different environments. The statements needed to execute the exportation can be found in section 7A of the script.

Import

The import statement takes the entire contents of the csv file originally exported in section 7A, and loads it back into tblAlbumAggregate, replacing the data already inside. The statements needed to execute the importation can be found in section 7B of the script.

Security Recommendations

Data hiding

None of the views in either system divulge sensitive information, with the exception of vwDJ in the transactional system. To keep this information hidden, 'select' should only be granted to administration. Alternatively, DJ information can be portrayed excluding salary by using a query such as the one found in section 9A of the informational database script.

Triggers

Triggers are in place to enforce various business rules, and it is recommended that they are created immediately after the database is, to ensure data quality. They can be found in section 3 of the transactional database script.

Logging/Auditing

As discussed earlier in the document, the transactional database contains two audit tables (see Audit tables in Transactional Database: Tables). These log

important changes and can be viewed with views previously outlined, found in section 7 of the transactional database script.

Transactions

Transactions can be used to maintain consistency in data in the transactional database. An example of the type of transaction that could be used can be found in section 8A of the script. In the example, the transaction serves to ensure that, if an artist needs to be removed from the database, his or her respective ArtistID is removed from all the tables it is found in.

Maintenance Recommendations

Scheduled events

As mentioned previously, events should be scheduled to maintain and migrate data daily. In addition, events should be scheduled that append to the backup of the data warehouse daily.