Music Store

Phase II

This report contains an overview of database design, entity relationship diagrams, functional dependencies, normal forms, design justifications, assumptions, and other design related information relating to the database / music store application for the Principles of Database Systems course at Metropolitan State University of Denver.

2012

Devin Brown & Brad Gill

Principles of Database Systems

11/18/2012

1. **Project Summary**

The Oracle music store serves a wide variety of musical interests. As a small store, it must maintain close control of its sale and inventory.  The store keeps an inventory of music in stock.  The store also keeps track of sales and customer special orders for items out of stock, which are placed through an employee of the store.

The primary goal when designing this database was to allow for ease of data accessibility, eliminate dependencies, minimize redundancies, while structuring the data in a form that was easily understood and still remained functionally orthogonal.

The overall goal of the Music Store Database / Application Interface is to enable maximum end-user, usability and flexibility, while allowing for queries and other DBA tasks to be assigned with relative ease. Another important requirement for this database was setting an infrastructure that allows for easy administration and user-access based restriction.

As is shown in the UML diagram, join tables are often the solution for allowing flexibility between option tables, which also contain the “one -to-many” type relationship. These join tables, for example, allow for **Customers** to have multiple **Transactions** of which contain multiple **Music**(s).

1. **Assumptions**

* Assume that the database is implemented for transactions on or after Jan. 1, 2000
* **Music**: Music titles can come in 3 different forms: cassette tape, compact disk, or sheet music.  At any one time the store may have all three forms in stock for a particular music title.  Therefore, the database must keep track of which forms of music are in stock for each music title
* **Vendors**: A vendor may publish several **music** titles; however, a particular music title can be published by one and only one vendor.
* Special **Orders** to **Vendors**: When placing a special order, all music by the same vendor may exist on the same order.  Music from different vendors must be placed on different orders: one vendor per order.  You can create a unique order ID by using the date as part of the order number.  Date should be in the format YYMMDD. **Orders** are placed by a store employee.
* For special orders, recording **Customer** information is mandatory
* For in-stock purchases, **Customer** information is optional.
* **Employees** of the store will be the users of this database system, not the customers themselves.  Information on every employee is in the database for management purposes.
* Each **Sale** is made by one **Employee** for one **Customer**.  The sale can include multiple music titles, and should also include quantity & price of each item and a grand total.  Upon making the sale, the system should *decrement* the music in stock appropriately.
* **Employee** and **Vendor** have required **Contact**(s), **Customer** is optional
* Each **Employee** and **Customer** may have multiple **Transactions**
* Each **Transaction** may have multiple **Music**(s)
* Each **Music** has one or more **Artists**
* Each **Order** may contain multiple **Music**(s)
* **Customers** may be linked to an **Order** (special order)
* An **Employee** may only have ONE **Job\_Title** (wage, wage\_type)
* Each **Music** may have zero or more **Promotions**
* **Vendors** have a **company\_contact** AND **representative\_contact**, for shipping and general information, respectively
* **Customers** may optionally subscribe for **Promotions** notifications
* **Employees**  may have multiple **Order**(s) linked to **Vendors**
* Each **Vendor** may fulfill multiple **Order**(s)

1. **Sample Statistics**

Sample statistics relevant to the applications include the following.  They would impact the real operational environment of the database system.  Here, it is offered as general information only.

* 1. **CUSTOMERS:** 3,000 customers in the database
  2. **QUERIES**: sales reports are done once weekly by music title
  3. **MUSIC:** 5,000 different music titles, between 1 and 20 copies of any music title is in stock at any given time
  4. **VENDORS:** 100 different vendors supply music
  5. **CUSTOMER ORDERS/SALES:**
     + 1. 50 customer special orders per day are submitted (each order has on average 2 titles)
       2. 30 orders are queried per day to check on the status of the order
       3. 50 sale transactions per day in the store with an average of 3 titles per sale

1. **Data Requirements** 
   1. The Oracle music store sells music in 3 different forms: cassette, compact disk, and sheet music.  A given music title has several attributes associated with it: examples are title, artist(s), producer, musical subject, physical type of music (cassette, compact disc, or sheet music), year of recording, etc.   An employee can query the music in stock by searching on any one or more attributes of the music.
   2. Music vendors can distribute several music titles; however, each music title can only be distributed by one vendor.  Vendor information (name, address, phone, contact person, etc.) is also maintained in the database.
   3. **Employees** are in charge of two tasks:
      1. selling music to customers
      2. placing special order from vendors for customers.
   4. **Employee** information maintained: ID, name, address, pay level, job title, etc.
   5. The store can place **Orders** to **Vendors** when stock runs low or when a customer requests a special order.
   6. A particular **Order** can only go to one vendor: this means that all items being ordered from a particular vendor can exist on the same order.
   7. If items are distributed by differing **Vendors**, an order form for each vendor must be filled out.  Example attributes of orders are the item(s) to be ordered, date of order, vendor to whom the order is going, and customer information if applicable.
   8. **Customer** information is maintained when special orders are placed.
   9. **Customer** information can also be maintained optionally when selling music: for example, customers can fill out a card to be put onto a mailing list.
   10. Sales information must be maintained whenever music items are sold: amount of purchase, each item purchased, customer information (if applicable), the employee making the sale, etc.
   11. **Music** may have one or more **Artists**
   12. **Music** may have one or more **Promotions**
2. **Functional Requirements**

There are four types of processes that are relevant:

[query] This process allows store employees to query the database with regard to music in stock and music on order.

[order] This process generates special orders by customers for music.  A store employee inputs orders.

[sell] This process modifies the database appropriately, regarding the item(s) being sold and the employee making the sale.  It is typically the operation done at the cash register. An invoice is generated for every order placed by a customer.  A receipt is printed for every in-store sale transaction.

[admin] This process modifies the database information about employees, customers, vendors, etc.  It may have other management report features which are left out for this project.

1. **Entity Relational Diagrams**

*(attached, last two pages) – updated based on Phase I feedback.*

1. **Dependencies, Multi-Valued Dependencies, and Normal Forms**

**Employees dependency**:

Key: active , ID

ID - > Employee\_contact, Job\_title

Job\_title -> wage\_type, wage

Nothing determines active.

*BCNF:*

{Job\_title,wage\_type, wage}

{ ID, Active, Employee\_contact, Job\_title}

**Contact dependency**:

Key: ID

ID -> Name, Address, City, State

Address, City, State -> zip

Name, Address -> email, phone

*BCNF:*

{Address, City, State, zip}

{name,Address,email,phone}

{ID,Name,city,State}

*This would be worse than keeping them all together since we would have to update multiple tables when adding a contact.*

**Vendors Dependency:**

Key: ID

ID -> Represntative\_contact

Representative\_contact -> company\_contact

*Normalized*

**Customer dependency**:

Key: Customer\_contact, promotions, ID

*Normalized*

**Employee\_transactions dependency**:

Key: Transaction\_ID

Transaction\_ID -> employee\_ID, Customer\_ID

*Normalized*

**Transaction dependency**:

Key: ID

ID -> sale\_amount, date\_sold, employee\_ID, music\_ISBN

*Normalized*

**Music Transactions dependency**:

Key: Transaction\_ID,Quantity

Transaction\_ID -> Music\_ISBN, Quantity

*Normalized*

**Music dependency:**

Key: ISBN, Type

ISBN -> producer, genre, year, vendor\_ID

Type -> price, quantity

*Normalized*

**Artists dependency:**

Key: Music\_ISBN

Music\_ISBN -> artist name

*Normalized*

**Order dependency:**

Key: ID,Status

ID -> Music\_ISBN, Date\_Ordered

Music\_ISBN -> Vendor\_ID

*Normalized*

**Music\_Orders:**

Key: Order\_ ID, quantity

Order\_ID -> Music\_ISBN

*Normalized*

**Promotions dependency:**

Key: ID

ID -> start\_date, End\_date, Music\_ISBN, Discount

*Normalized*

1. **Database Implementation**

**Users / Privileges**

Each user roll inherits all powers from the previous roll (e.g. a manager can make sales).

**Sales Employee**

Sales Employees are responsible for handling day-to-day affairs. This primarily includes making sales, searching inventory, viewing customer contact information, and adding that information if necessary.

* View customer contact information
* Search music stock for a customer
* Search orders placed by a customer
* Adding customer contact information
* Create a transaction when making a sale

**Manager**

A manager oversees all sales employees, and primarily has the power to add and modify contact information. Managers are also responsible ordering placing orders for customers, or for the sake of replenishing a diminished inventory. Managers may also view all tables, which may be helpful in the event of needing to view old data (e.g. find an old transaction for a return).

* View any table
* Create any type of contact
* Create an order for either a contact, or to replenish the inventory

**Owner**

Owners can view, modify, or add data to any table. The owner is also responsible for creating promotions.

* Modify any table
* Add data to any table
* Create promotions

**DBA**

The DBA has power to do anything, notably delete tables. The delete action is saved exclusively for the DBA due to the potentially catastrophic consequences of accidental deletion.

* Drop any table

**User Queries**

Each user roll inherits all powers from the previous roll (e.g. a manager can make sale).

1. **Sales Employee**

SELECT \* FROM customer\_contacts;

SELECT \* FROM music;

SELECT \* FROM order;

INSERT INTO customer\_contacts VALUES (…);

INSERT INTO transaction VALUES (..); -- this will be deeply abstracted into the GUI

1. **Manager**

SELECT \* FROM \*; -- View any table

INSERT INTO contact VALUES (…);

INSERT INTO order VALUES (…);

1. **Owner**

UPDATE \* SET \* WHERE (…); -- modify any table

INSERT INTO \* VALUES (…); -- add data to any table

INSERT INTO promotions (…); -- specific functionality requirement

1. **DBA**

DELETE FROM \*; -- delete anything from any table

**Indexes**

**Database Population**

Our goal was to great a customized, flexible system of generating *life like* data that could be easily scaled. For this portion of the assignment, we ended up creating a customized program that automates the randomization and script generation process. This program generates X number of entries for each applicable table, generating random, and unique where necessary, data that scales gracefully as we wish to increase the load testing on our database.

For initial testing the database is populated with:

* 100 Artists
* 300 Contacts
* 50 Employees (random number of sales employees, managers, owners, and DBAs)
* 25 Vendors
* 25 Representatives
* 200 Customers
* 4 Job Title / Rolls
* 5000 Music Products
* 100 Promotions

For the sake of saving a large quantity of paper, the scripts and the program that generate these scripts may be [found in this online repository.](https://github.com/dgbsco/music-store/) We will plan to increase the number of items in the database after initial testing is complete.

**Transactions**

The use of an isolated, atomic, transaction should occur in two places: (1) making sales between the customer and the Music Store, and (2) placing orders between vendors and the Music Store. Due to the critical nature of money and products being exchanged this information needs to be robust in the case of failure, and must occur atomically to avoid potentially selling or buying items that may have already been sold.

**Procedures**

* Add Customer
* Remove Customer
* Hire Employee
* Fire Employee

**Functions**

* Joining the contact table with customers (customer\_contacts)
* Joining the contact table with vendors (vendor\_contacts)
* Joining the contact table with employee (employee\_contacts)
* Retrieve all items that are on sale / promotion *today*

**Triggers**

* Update (remove) music table’s quantity when a sale / transaction is made
* Update (add) music table’s quantity when an order is made

1. **Applications For the Oracle Music Store**

*(Phase III)*

