CS 6400 Database Project: **LEOFURN Sales Reporting System (LSRS)**

Spring 2021

Project Overview

The purpose of this project is to analyze, specify, design, and implement a reporting system for the US branch of the Danish furniture company, LEOFURN. The project will proceed in three phases as outlined in the methodology for database development: Analysis & Specification; Design; and Implementation & Testing. The system will be implemented using a Database Management System (DBMS) that supports standard SQL queries.

LEOFURN Sales Reporting System

LEOFURN is a furniture manufacturer from Denmark, specializing in Scandinavian-style furniture, whose subsidiary has stores throughout the United States. Your team has been asked to design and build a prototype sales reporting system for LEOFURN USA. This section describes in detail the requirements for the LEOFURN Sales Reporting System (LSRS, pronounced like "lasers").

What makes LSRS different is that it is meant to import and aggregate sales data. Unlike *transactional* databases which are generally designed to record repetitive day-to-day business transactions (e.g., point of sale, buy and sell stock orders, online shopping carts, etc.), LSRS is meant for reporting and analysis over millions of records to support enterprise-wide decision making. As an example, a large online merchant like amazon.com or bestbuy.com relies on a transactional (also called *operational*) database system for recording customer orders and payments in real time. A data analyst tasked with generating a report that compares sales of a certain product among the different regions of the United States will typically query a reporting database for the report instead of accessing the transactional databases directly. There are several reasons for this: the system can store data from multiple transactional databases in a consolidated form, its schema is designed to support complex queries aggregating millions of rows, and queries do not impact the performance of the transactional database which must support high transaction throughput and availability.

For this project, you will design the database schema for LSRS and attach it to a rudimentary user interface. ***For this project you should create a normalized schema with as little redundancy as possible***. You also need not be concerned with the details of the transactional databases that we assume exist to support the point-of-sale system at each LEOFURN store. Instead, you will design the schema to support a consolidated view of the products offered and sold in all LEOFURN stores across the country. What follows is a description of the requirements for the system in terms of what information must be stored to support a set of reports defined by the LEOFURN executive team.

Data Requirements

entity: store

LSRS maintains information about each *store*, including a unique *store number*, the store's *phone number*, and the store's *street address*. (You do not need to store individual components of the address, such as street number, prefix, etc., as the addresses will be unparsed.) Stores may also provide *childcare* while customers are shopping, have a *restaurant* on site, a *snack bar*, or any combination of these. You will not need to load any data regarding restaurant or snack bar sales. Stores offering childcare have an individually determined *limit* as to the maximum number of minutes of childcare offered per customer.

entity: childcare (weak—>store), verb: provide

prop of store: restaurant existence (value: T or F), for report 8

prop of store: snack bar existence (value: T or F)

Reminder: each entity should be uniquely identified in some way ?
Not sure whether it "must" in EER, thus use the word "should"

The limit is chosen by each store from predetermined values, and all limits may need to be updated if it changes (such as from 45 minutes to 60) or a new limit requires manual updating outside of a data load.

city name in one state not unique?

LSRS should also maintain information about each store's *city*, including the *city name*, the *state* in which the city is located, and the *population* of the city. A city may have multiple stores.

r:store-city
verb: locate

entity: city, id: city name + state + population

derived prop: pop size ctgy

LSRS contains information about every *product* for sale at LEOFURN stores. Products have a numeric unique identifier (*PID*), similar to a UPC barcode, as well as the *name* of the product. Assume that all products are available and sold at all stores—that is, there is no need to specify that a certain product is only available at a certain store.

entity: product ~~r:store-product~~

A product must be assigned to one or more *categories*. Each category has a *name*, which we assume to be unique. A category may or may not have products associated with it.

r:category-product
verb: include

entity: category

Every product has a retail *price*. The retail price is in effect unless there is a *discount price*. LSRS maintains the *discount date* and *discount price* of any product that goes on discount. If a product is discounted for multiple days in a row, then a record is stored for each day. It is possible that the same product is discounted multiple times (i.e., different days) with different prices. **If a product is discounted, it is for the same price in all stores—i.e., stores are not allowed to discount items independently or have store-specific discount prices.**

price: disjoint

verb: offer

verb: has

entity: discount (weak -> day & product), price (weak -> product & discount) ~~r:discount-discount price~~

~~verb: has~~

The LEOFURN executive team would like the ability to compare sales data on *holidays* versus non-holidays, so LSRS should maintain information about which dates contain holidays. The *name(s)* of the holiday should be stored. If a day has multiple holidays, their names can be combined, such as "Halloween, Harry Potter Day" as we only need to know that the date had holidays associated to it.

entity: day (identified by date), holiday r:day-holiday verb: contain

LSRS stores information about which products are *sold*, including the *store* where it is sold, the *date* of the sale, and the *quantity* of the product purchased. The total amount of the sale is not stored explicitly but can be derived based on the date purchased and the quantity, as individual item prices can be determined from either the retail price or the discount price if one is in effect. For reporting purposes sales tax values are ignored. Also, the system is not required to store which products were purchased together during a single sales transaction.

entity: sale (weak -> store & day & product, partial key: quantity, derived prop:tot amt)

Advertising campaigns occur and are active for a specific set of dates. LEOFURN's executives want to have a rudimentary understanding on the impact of these advertising campaigns, so your system should store that information as well – this will include the *description* of the campaign (such as "Daily Prophet July Print Ad", "Triwizard Tournament Promotion Postcard", "Summer Radio Ads", and is unique) and each specific *date* it was active. Campaign dates may overlap.

entity: advertising campaign r: advertising campaign-day

verb: occur

The LEOFURN executive team understands that some details provided here may not be utilized in the reports they initially request but would like to have that data stored in the event they decided to enhance the system with more reports, or for any ad-hoc querying after you have built the system.

An important item to consider is that despite a "date" being a relatively simple piece of data, it may be optimal to treat dates as a set of values distinct from discount information, sale information, and campaign information, as discounts could occur on dates where there are no sales, sales could occur on dates where there are no campaigns, etc. Many of the reports requested can be simplified by referencing an established common set of dates instead of individual date information stored on each

Textbook P69:
Value Sets (Domains) of Attributes. Each simple attribute of an entity type is associated with a value set (or domain of values)
-> entity is day, attribute is date, which has a set of values dates