Author:	Joel Kemp
File:	README.doc
Project:	Database Systems 1 Final Project, Fall 2011, CCNY
Professor:	Jie Wei
Purpose:	Details the inner workings of the system.

Introduction

The system was built purely using Matlab. The particular version of Matlab used was R2011b, although, previous versions should still run the system fine.

The program can be launched via the rdbms.m file. The function rdbms() is the entry point to the entire system.

Architecture

The system maintains a save-file that maintains a recent copy of the database after any major operations. The save file is located in the root directory of the source code and named *database.mat*. This auto-saving feature allows the user to retain his/her work even after the system terminates!

The architecture of the system is broken down into the following main sections:

1. Define Table: allows the user to define new tables that will be managed by a collection of tables.

2. Extend Table: allows the user to define constraints (boolean conditions, functional dependencies, multivalue dependencies, and keys) and tuples for any table in the database.

3. Modify Table: allows a user to perform a plethora of set (table) operations (union, difference, intersection, cross-join, etc).

4. Delete Table: allows a user to delete any table in the database.

Data Structures

The system contains elementary structures used throughout the implementation:

Table Structure:

name:	The name of the table.
schema:	TreeMap of attribute to datatype associations.
constraints:	A list of constraints (fd, mvd, conditions, keys) See Constraint.m for the constraint structure.
tuples:	A column array of dynamic structures accessed by the set of attributes for the table.

Constraint Structure:

conditions:	List of boolean conditions
fds:	List of functional dependencies
mvds:	List of multi-value dependencies
keys:	List of attribute keys

Dependency Structure:
Represents a functional dependency or multi-value dependency

lhs:	Attribute
rhs:	Attribute

Condition Structure:
Represents a single condition. Can be "chained" using boolean "AND" or "OR".

lhs:	Attribute (Left Hand Side)
operator:	Boolean Operator (<, >, ==, <>, <=, >=)
rhs:	Int or String Value (Right Hand Side)

Tuples:
Tuples did not have a pre-defined structure, however a table's list of tuples was represented as an array of dynamic structures. Each tuple's structure was defined at runtime and used runtime-variable indexing to store the tuple values. The key to this scheme is the table's schema (attribute -> datatype pairing).

Complete Feature Listing

Note: The **bolded** *features were not on the spec sheet, but were included for a thorough implementation.*

- Define new tables (attrib names and types)
- Input constraints (conditions, fds (**and transitive dependencies**), mvds, and keys)
    - Ignore conflicting conditions
    - Remove trivial FDs
    - Table must be in 4th Normal Form
    - Remove Trivial MVDs
    - Remove MVDs trivialized by an existing FD
    - **Keys must be valid keys (closure equals the set of all attributes)**
- Input Tuples
    - Remove tuples that violate constraints
        - Conditions
        - FDs
        - MVDs
            - Auto-add missing tuples if violated
        - **Keys**
- Delete tuples based on condition
    - **Checks that the modified table does not violate the MVDs**
- Select tuples by condition
- Group tuples by attribute
- Set operations
    - Cross Join
    - Natural Join
    - Union
    - Intersection
    - Difference
- Delete tables

Coding Remarks

Matlab lacks strings!

Unfortunately, Matlab does not support strings natively. This constraint imposes (sometimes) impractical design decisions – almost always necessitating the use of (fragile) cell matrices to hold collections of varying-length strings. Cell matrices were used heavily in the system, although, there are places (namely, the storage of FDs and MVDs) that should have used a structure array for a more reliable implementation.

It's hard to show certain features!

The strict validation for constraints (namely 4th normal form enforcement for MVDs) makes it very difficult to input tuples that would require additional (missing) tuples to be added if the tuples list violates the MVDs.

This MVD tuple validation feature was tested in isolation (bypassing constraints) and correctly generates tuples that make the MVD violation non-existent. These tuples are added to the list of tuples.

Helpers were made global when necessary!

The directory structure for the project's source contains subdirectories. These subdirectories primarily group the functions associated with a particular section of the system. In Matlab, in order for functions to be accessible to other functions (in different files), the functions need to be defined in separate M-files.

The M-files in the subdirectories are made available on the Matlab path at runtime.

Only those functions that needed to be global were made global. Hence, there are certain files (constraints.m) that are rather large and could be cleaned up by migrating code to separate files.