Wolf Inns
Hotel Chain Management System

CSC 540
Database Management Concepts and Systems

Project Report # 3

Team C

Abhay Soni	(asoni3)
Aurora Tiffany-Davis	(attiffan)
Manjusha Trilochan Awasthi	(mawasth)
Samantha Scoggins	(smscoggi)

Table of Contents

Project Assumptions

1. Customers

    1.1. A customer has exactly one phone number (that the system tracks at any one time).
    1.2. A customer has exactly one email address (that the system tracks at any one time).
    1.3. All Customers will have a United States Social Security Number.

2. Staff

    2.1. Each staff member has exactly one job title and department.
    2.2. A staff member's job title indicates what they are expected to do within the system. However, a manager may need to fill in for a colleague, and so for example a manager may provide any type of service regardless of job title. For example, if all staff members except for the manager are dedicated to presidential suites, then the hotel manager would have to run the rest of the hotel.
    2.3. A staff member's department is merely an attribute and is not expected to influence how the staff member interacts with the system. This attribute has no functional impact to the system, and is maintained purely due to its presence in the project narrative.
    2.4. Each staff member is assigned to at most one hotel.
    2.5. A hotel has exactly one manager, and a manager manages exactly one hotel. Therefore, if you wish to fire a hotel manager, you must update the hotel information with a new manager within the same transaction. Also, if you wish to add a new hotel, you may not assign a staff member to manage this new hotel, who already manages another hotel.
    2.6. Because "Front desk representatives… bill customers", "Billing Staff" is just another job title meaning "Front Desk Representative".
    2.7. Staff members know their own staff ID as well as the hotel ID of the hotel they are currently serving.
    2.8. A presidential suite is assigned exactly one room service staff member and exactly one catering staff member at check-in.
    2.9. Staff ID is unique across all hotels.
    2.10. If room category is presidential suite, then the room will be considered unavailable unless there is both a not-yet-dedicated room service staff member, and a not-yet-dedicated catering staff member assigned to the hotel.
    2.11. Staff dedicated to presidential suites are not necessarily reported as "staff serving during customer stay". If those staff members actually provide a service, then they will be represented in this query. If they do not, then they have not actually served the customer during the stay - rather, they were simply on standby to do so if needed.

3. People

    3.1. Several staff members and/or customers may have the same name, or the same phone number, or the same address, or the same email.

4. System Interaction

    4.1. The system has no GUI.

5. Rooms

5.1. Room requests may involve more than room category (“Basic”, “Deluxe”, etc.). That is, assigning rooms to customers according to their request criteria may include some or all of: hotel location, room category, room (nightly) rate, maximum occupancy.
5.2. Room prices vary by location and class of service, but are not strictly defined by location and class of service. That is, each room may be given a unique price when the room is added to the system.
5.3. Each hotel may have any number of rooms of any given category. For example, “Presidential Suite” is not necessarily unique within a hotel.
5.4. Reports of occupancy are expected to report occupancy by room, and not by guest (e.g. “4 rooms occupied” rather than “6 guests staying”).
5.5. “Total occupancy” refers to the total number of rooms occupied.
5.6. Maximum occupancy is not determined by room category (“Basic”, “Deluxe”, etc).

6. Services

6.1. Each instance of a service rendered for a customer is provided by one staff member. That is, staff members do not team up to bring room service, etc.
6.2. The only standard services offered by the hotel are phone bills, dry cleaning, gyms, room service, and catering. All other services are considered “special requests”.
    6.2.1. Every Wolf Inns hotel charges the same rate/minute for phone use.
    6.2.2. Every Wolf Inns hotel charges the same rate/item for dry cleaning.
    6.2.3. Every Wolf Inns hotel charges the same rate/use for gyms.
    6.2.4. Every Wolf Inns hotel charges the same rate/meal for catering.
    6.2.5. Every Wolf Inns hotel charges the same rate for special requests.
    6.2.6. No Wolf Inns hotel charges customers for room service (ordering a meal to be delivered to your room is handled as a catering charge).
6.3. All services must be provided by a staff member with a job title matching the service provided. There are only two exceptions to this rule. First, a manager may provide any type of service. Second, phone service or a special request may be provided by any staff member.
6.4. All services must be provided by a staff member assigned to the hotel where the customer’s stay is taking place.
6.5. A staff member dedicated to the service of one room may not provide a service to another room.
6.6. Wolf Inns does not provide the service of “splitting your bill”.
    6.6.1. Each instance of a service rendered is associated with a single hotel room stay. For example, multiple occupied rooms cannot jointly request catering. That is, there is one bill per room.
    6.6.2. Each hotel bill has one customer responsible for payment.

7. Check-In / Check-Out

7.1. A check-in cannot proceed without necessary billing information available:
- 7.1.1. Customer SSN
- 7.1.2. Customer name
- 7.1.3. Payment method
- 7.1.4. Card Type if payment is card:
  - 7.1.4.1. Accepted values: ‘VISA’, ’MASTERCARD’, ’HOTEL’
- 7.1.5. Card # if payment method is card
- 7.1.6. Customer billing address if payment method is card

7.2. A check-in can proceed without unnecessary billing information available:
- 7.2.1. Card type if payment method is not card
- 7.2.2. Card # if payment method is not card
- 7.2.3. Customer billing address if payment method is not card

7.3. The customer who checks in is responsible for paying for their stay. Information about guests physically staying in the rooms is not tracked, with the exception of tracking the number of such guests.

7.4. The customer may not wish to use the same method of payment for multiple Wolf Inns hotel stays.

7.5. A bill, and therefore a customer stay, is considered to take place in a single room. That is, if a customer books many rooms (family reunion, etc), separate bills will be generated for each room.

7.6. Each customer stay will be identified by a generated ID which is unique across all stays regardless of which hotel the stay takes place in.

7.7. Check-Out Time and End-Date will be entered at the time when a Customer checks out. And for this reason, these attributes are allowed to be NULL prior to the guest checking out.

7.8. If End-Date = Start-Date then the customer is not charged for the room, as charges are per night. However, it is still possible to incur charges from services provided.

8. Reports

8.1. Updating basic information about rooms may include updating a room’s nightly rate. Therefore, at the end of each stay we must preserve the total amount owed for the stay. This is in order to accurately generate information about revenue earned from past stays.

8.2. In the operation “For each customer stay, return information on all the staff members serving the customer during the stay”, “for each customer stay” can best be translated to “for a given customer stay”, rather than “for every single customer stay”. This assumption is made due to the perceived usefulness of the former, and the likely overwhelming length of the latter, once the Wolf Inns hotel chain is blessed with many customer stays.

9. Removing Records

9.1. Aside from the assumptions listed below, removing a record from the Wolf Inns database system will be isolated and will not “cascade” removal to other records.
9.2. If a hotel is removed from the Wolf Inns system, then all rooms within that hotel are also removed.
9.3. If a room is removed from the Wolf Inns system, then all information associated with this room is removed from the system, entirely. For example, all stays associated with that room are also removed.
9.4. If a customer is removed from the Wolf Inns system, then all information about this customer is removed from the system, entirely. For example, all stays associated with that customer are also removed.
9.5. If a stay or a service type is removed from the Wolf Inns system, then any record of an instance of a service provided that is associated with that stay or service type shall also be removed.
9.6. If a staff member is removed from the Wolf Inns system, then any record of an instance of a service provided by that staff member shall have the staff ID associated with the record set to null.
9.7. Any hotel, room, customer, staff member, or service type associated with a current guest stay may not be deleted.
9.8. Special note: As noted in a previous project assumption, a hotel has exactly one manager, and a manager manages exactly one hotel. Therefore, if you wish to fire a hotel manager, you must update the hotel information with a new manager within the same transaction. If this is not done, then removing a manager may result in also removing a hotel. Please note that this would indicate a serious problem, and is not intended to occur.

10. Updating Records

10.1. Any room associated with a current guest stay may not be updated.
10.2. Any staff member currently dedicated to serving a presidential suite may not have their job title changed.
10.3. While the Wolf Inns system supports updating most aspects of a record, the primary key of any record can never be changed - this is not supported by the Wolf Inns system.
10.3.1. These are aspects of a record which by definition define the record, and so if they are incorrect, the record itself can be considered invalid.
10.3.2. Such examples are the name of a service type, and the combination of room number and hotel ID for a room.
10.4. Service records may be updated only for current stays. Once the customer has checked out, service records may no longer be updated for that stay.
10.4.1. The customer is billed when they check out. If a service record is changed afterwards, it may make the amount owed and/or the itemized receipt, already provided to the customer, inaccurate.
10.4.2. The amount owed is saved so that in future, revenue reports can be run. If a service record is changed after the customer checked out, the revenue report cannot be made accurate. Consider this scenario: A customer gets “Gym” service at $15. The customer is billed accordingly at check-out. Then, Wolf Inns decides to update the cost of the “Dry Cleaning” service from $16 to $18. Finally, the service record for the already-ended stay is updated to say that the customer got “Dry Cleaning” service. Now, the system cannot know how to amend the amount owed associated with that stay. It does not remember the cost of “Dry Cleaning” at the time of the stay, indeed, it does not even remember that this value has changed in the past.
10.4.3. In order to avoid these inaccuracies, the system disallows updating service records for already-ended stays.

11. Miscellaneous

11.1. Wolf Inns does not accept reservations.
11.2. Wolf Inns will never be exposed to a hacking attempt, such that security related concerns, including access control, is not a necessary concern.
11.3. Wolf Inns handles employee payroll, etc. through a separate system.
11.4. If information that Wolf Inns should maintain is computable from other already explicitly stored information, then it is considered “maintained” and there is no need to explicitly store it.
11.5. If a question about a customer, stay, hotel, etc. is not required by the project narrative, then there is no need to support it.
11.6. A hotel’s name needn’t be unique, for example we may have several hotels named / branded “Econo Wolf”, or “Wolf Deluxe Stay”.
11.7. A phone number cannot belong to more than one hotel.

Code & Design Documentation

```
*
* Compilation:
* -    add mysql
* -    javac WolfInns.java
*
* Running:
* -    java WolfInns
*
* Operation:
* -    When the application starts, it will populate the DB with the demo data.
* -    Thereafter, interaction is enabled by means of menus
*      -   FRONTDESK Menu
*          -   Perform hotel staff operations such as checking guests in / out, entering service records, etc.
*      -   REPORTS Menu
*          -   Perform reporting operations such as revenue, occupancy, and staff reports.
*          -   Perform extra reporting operations such as reporting on all hotels, all rooms, etc.
*      -   MANAGE Menu
*          -   Perform management operations such as adding / updating / deleting hotels, staff, etc.
* -    Each menu shows each available command, along with a brief description of what the command accomplishes.
* -    If you wish to close the application, each menu includes a QUIT option.
*
```

```
 * Organization:
 *
 * -   "main" function
 *     -   Welcomes the user
 *     -   States available commands
 *     -   Listens to and acts on user commands
 *     -   Closes resources upon "QUIT"
 *
 * -   "startup_..." functions
 *     -   Perform startup work such as connecting to the DBMS, creating tables
 *
 * -   "populate_..." functions
 *     -   Populate tables with demo data
 *
 * -   "user_..." functions
 *     -   Interact with the user to get details about their query / update needs
 *
 * -   "db_..." functions
 *     -   Interact with the database to run queries and updates
 *
 * -   "support_..." functions
 *     -   Perform calculations and checks that may be needed by any other functions
 *
 * -   "error_handler" function
 *     -   Generalized error handler intended to give human-understandable feedback if something goes wrong
 *
 * -   Large source code file
 *     -   The discerning reader will notice that our application is written in one single file which is > 7k lines long.
 *     -   Our priority during development was on functionality and reliability, such that designing a file hierarchy fell by the wayside.
 *     -   We took to heart the guidance from course instructors that assessment would be based on functionality,
 *         and that "extra" work would yield no extra points.
 *     -   Certainly if this code were to be supported long-term we would have made a different choice here.
 *
```

* Design:
*
* - Prepared Statements
* - We used prepared statements for most SQL interactions with the database.
* - Although efficiency is not a requirement of the project, we wanted to explore this topic as a best practice.
* - Our initial intention was to used prepared statements for all such interactions, but time did not allow for that.
*
* - Constraints
* - We implemented uniqueness, primary key, and foreign key constraints
* as noted in project report 2 and as implied by project assumptions.
* - We did not implement CHECK constraints as these are not fully supported by the version of MariaDB suggested for use in the project.
* - We did not implement ASSERTIONS as these are not fully supported by the version of MariaDB suggested for use in the project.
* - We implemented constraints needed for logical operation on the data,
* which otherwise might have been implemented via CHECK or ASSERTION, through two means:
* - In SQL where practical
* - In application code where SQL was not practical
* In some cases we are covered by checks at both the SQL and application levels
* This is a result of the development evolution and we are comfortable with extra protection
*
* - Transactions
* - For safety, our intention was to implement transactions wherever there are multiple DB modification steps within one function.
* - Functions are scoped in a logical way, such that they should represent actions on the database that should be handled atomically.
*
* - User Friendliness
* - We have made some effort to have a user-friendly interface.
* - There are cases where the application gives the user hints.
* - One example is showing a list of all service types before asking which service type the user wants to change the cost of.
* - Although the team members will be the users during the demo, these hints / helps are intended to help us:
* - Move through code development and debug smoothly
* - Move through the demo smoothly
* - We are certainly not claiming that this application is user-friendly enough for mass market adoption,
* it just has a few of the rough edges sanded down.
*

```
* -  Room Availability
*    -  As indicated by our design as documented in project report 2,
*       we calculate room availability by examining check-in / check-out records.
*    -  During phase 3 of the project we became aware via
*       https://classic.wolfware.ncsu.edu/wrap-bin/mesgboard/csc:540::001:1:2018?task=ST&Forum=13&Topic=7
*       that "A room can be unavailable for a variety of reasons - for instance, for being renovated".
*    -  We have not re-designed our system to accommodate a room which is unoccupied nonetheless being considered unavailable.
*
* -  Beyond-demo-data table population
*    -  Service Types
*       -  "Catering" is included as a service type although it is not found in the demo data.
*       -  Catering is a service mentioned in the project narrative.
*       -  Presidential suites must have dedicated catering staff again as noted in the project narrative.
*       -  We designed our system with catering in mind as an offered service.
*    -  Staff
*       -  Two staff members are included beyond what is found in the demo data in order to
*          have dedicated staff available to serve the presidential suite which is found in the demo data.
*          -  As noted in the project narrative, "At time of check-in, the presidential suite is assigned dedicated ... staff."
*          -  For this reason, a presidential suite is considered unavailable if there are no staff that we can dedicate to it.
*          -  Yet, the suite is noted as available in the demo data.
*          -  The added staff were necessary, given our project design & assumptions, to resolve this conflict.
*    -  Stays
*       -  One stay is included beyond what is found in the demo data in order to
*          have one room considered unavailable, as is noted in the demo data.
*    -  Because our changes to the demo data are purely additions, and there are no alterations or deletions,
*       we are comfortable that we will be best able to explain and defend the behavior of our application
*       with these changes in place.
*
* -  Other
*    -  Other design decisions are best understood by reviewing our team's project report 2, and project assumptions.
*
```

Functional Roles

Role		Project Phase 1	Project Phase 2	Project Phase 3
Software Engineer	Prime	n/a	n/a	Aurora
	Backup	n/a	n/a	Manjusha
DB Designer / Admin	Prime	Manjusha	Abhay	Abhay
	Backup	Samantha	Aurora	Samantha
Application Programmer	Prime	n/a	Aurora	Manjusha
	Backup	n/a	Abhay	Aurora
Test Plan Engineer	Prime	Aurora	Manjusha	Samantha
	Backup	Abhay	Samantha	Abhay

Transactions

As noted in the code / design documentation section of this report, we used transactions widely. All of the transactions may be found in the code by searching the code for “jdbc_connection.setAutoCommit(false)”. Here, we discuss just two of these transactions.

1. Populate the Customers table

```
public static void populate_Customers() {
    try {
        /* Start transaction
         * In this function, we add several customer tuples to the Customers table
         * If there is a problem with any tuple,
         * we feel it is safest to tell the user there is a problem, and leave the table empty
         */
        jdbc_connection.setAutoCommit(false);
        try {
            /* Populating data for Customers
             * Signature for method called here:
             * String ssn,
             * String name,
             * String dob,
             * String phoneNumber,
             * String email,
             * boolean reportSuccess
             */
              db_manageCustomerAdd("5939846", "David",    "1980-01-30", "123", "david@gmail.com",     false);
              db_manageCustomerAdd("7778352", "Sarah",    "1971-01-30", "456", "sarah@gmail.com",     false);
              db_manageCustomerAdd("8589430", "Joseph",   "1987-01-30", "789", "joseph@gmail.com",    false);
              db_manageCustomerAdd("4409328", "Lucy",     "1985-01-30", "213", "lucy@gmail.com",      false);
            // If success, commit
            jdbc_connection.commit();
            // Tell the user that the table is loaded
```

```
            System.out.println("Customers table loaded!");
        }
        catch (Throwable err) {
            // Handle error
            error_handler(err);
            // Roll back the entire transaction
            jdbc_connection.rollback();
        }
        finally {
            // Restore normal auto-commit mode
            jdbc_connection.setAutoCommit(true);
        }
    }
    catch (Throwable err) {
        error_handler(err);
    }
}
```

2. Release a room upon check-out

```
public static void db_frontDeskReleaseRoom (String hotelID, String roomNum) {
    try {
        /* Start transaction
         * In this function, we first update the check out time and end date in the Stays table
         * Then we release the dedicated staff assigned to the room in the Rooms table
         * Either operation by itself does not make sense
         * We want both operations to succeed together, or fail together
         */
        jdbc_connection.setAutoCommit(false);
        try {
            jdbcPrep_getStayIdForOccupiedRoom.setLong(1, Long.parseLong(hotelID));
            jdbcPrep_getStayIdForOccupiedRoom.setLong(2, Long.parseLong(roomNum));
            // Get the Stay ID for given room
            ResultSet rs = jdbcPrep_getStayIdForOccupiedRoom.executeQuery();
            int stayId = 0;
            while (rs.next()) {
                stayId = rs.getInt("ID");
            }
            // Now update the CheckOutTime and EndDate to current date
            jdbcPrep_updateCheckOutTimeAndEndDate.setInt(1, stayId);
            jdbcPrep_updateCheckOutTimeAndEndDate.executeUpdate();
            // Now release the dedicated staff assigned to the room
            jdbcPrep_releaseDedicatedStaff.setLong(1, Long.parseLong(hotelID));
            jdbcPrep_releaseDedicatedStaff.setLong(2, Long.parseLong(roomNum));
            jdbcPrep_releaseDedicatedStaff.executeUpdate();
            // Once both actions (Updating Checkout and EndDate for Stay & Releasing the dedicated staff)
            // are successful, commit the transaction
            jdbc_connection.commit();
            // Tell the user the room was released
            System.out.println("\nThe room has been successfully released!");
        }
        catch (Throwable err) {
```

```
                // Handle error
                error_handler(err);
                // Roll back the entire transaction
                jdbc_connection.rollback();
            }
            finally {
                // Restore normal auto-commit mode
                jdbc_connection.setAutoCommit(true);
            }
        }
        catch (Throwable err) {
            error_handler(err);
        }
    }
```