CMPUT 291
Project 1 Design Report

Falon Scheers, Alanna McLafferty, Isabella Lin

March 25, 2015

Contents

1 Overview

Our application system has been designed such that each component is contained in its own module. Components are accessed via a gateway function in each module, and are otherwise independent of each other. The one exception is the module dealing with creating new people records, which is inherited and accessed by several modules. Error checking and verification is first performed against the user input; only when we are sure that the input is logically valid do we incorporate it into a database query or statement.

The main modules in our application are:

- `main`
- `new_vehicle`
- `new_person`
- `new_auto_transaction`
- `new_driver`
- `violation_record`
- `record_search`

2 main

This module handles login and connection with the Oracle database, as well as the initial user interface for the program.

2.1 Interfaces

This module should be called to initiate the program. `menu` may be called from another module if the cxOracle connection and cursor arguments are provided.

2.2 Functions

2.2.1 Menu()

Description:
Displays the menu of user options, and upon user input calls the appropriate entry function from other modules. Passes the `connection` and `curs` parameters to these other modules to enable interfacing with the database.

Parameters: None

Return value: None

2.2.2 Exit()

Description:
Gracefully exits the program.

Parameters: None

Return value: None

2.3 Inherits

From `cx_Oracle`: `DatabaseError, connect`
From `new_vehicle`: `NewVehicle`
From `new_auto_transaction`: `AutoTransaction`
From `new_driver`: `NewDriver`
From `violation_record`: `ViolationRecord`
From `record_search`: `RecordSearch`

3 new_vehicle

This module handles the registration of a new vehicle.

3.1 Interfaces

The module should be entered via calling the function `NewVehicle`. It is called by the main module. References `new_driverperson` via `NewPerson`.

3.2 Functions

3.2.1 NewVehicle

Description:

This function obtains user information and checks the validity of the information sometimes by querying the database. If the user input is valid, adds the new vehicle information into the vehicle table. Then it adds the new primary owner information into owner table. If a secondary owner is given it then adds the new secondary owner into the owner table.

Information needed from the user:

- serial number of vehicle (int CHAR(15))
- maker (VARCHAR(20))
- model (VARCHAR(20))
- year (Number(4,0))
- color (VARCHAR(10))
- type_id (integer)
- primary owner sin (CHAR(15))
- secondary owner sin (CHAR(15))

Assumptions:

- It is only possible to have one secondary owner
- All vehicles must have one primary owner
- User enters SINs that are 9 digits long
- There are some valid type_ids already entered in the vehicle_type table

Parameters:

`connection, curs` - from an active connection to the database

Return value: None

3.2.2 `OwnerErrCheck`

Description:
Prompts the user for a SIN and carries out checks on it to ensure validity.

Parameters:
`connection, curs` - from an active connection to the database
`owner_type` - either 'primary' or 'secondary'

Returns:
String containing the owner's SIN if a valid SIN was entered. If not, the string "EXIT" is returned.

3.2.3 `CheckIfIdExists`

Description:
Checks if the SIN already exists in the people table. If not, prompts for insertion of a new person through the module `new_person`.

Parameters:
`connection, curs` - from an active connection to the database
`some_id` - string containing a SIN to query

Returns:
`False` if the ID exists or if a person was successfully added, `True` if the ID does not exist and the user chose not to add a new person. The string "EXIT" is returned if a new person was not successfully added.

3.3 Undocumented helper functions

`TypeErrCheck, CheckIfTypeExists, YearErrCheck, StrErrCheck, SerialErrCheck, CheckLen, CheckIfVehicleExists, CheckIfInt`

3.4 Inherits

From `new_driver`: `NewDriver`

4 new_person

This module deals with the creation of a new person record.

4.1 Interfaces

This module is accessed via the `NewPerson` function. It is used by the `new_vehicle` and `new_auto_transaction` modules.

4.2 Functions

4.2.1 NewPerson

Description:

This function obtains user information and checks the validity of the information. If the user input is valid, it adds the new person into the people table in the database

Information needed from the user:

- person's name (CHAR(40))
- person's height (number(5,2))
- person's weight (number(5,2))
- person's eyecolor (VARCHAR(10))
- person's haircolor (VARCHAR(10))
- person's addresss (VARCHAR(50))
- person's gender (CHAR(1))
- person's birthday (DATE)

Assumptions:

- The user will enter valid alpha strings (spaces permitted)
- The SIN is valid (9 digits long) and does not already exist in the database

Parameters:

`connection, curs` - from an active connection to the database
`SIN` - a string containing the person's SIN

Returns:

`True` if the operation is a success.

4.3 Undocumented helper functions

`DateErrCheck, GenderErrCheck, FloatErrCheck, StrErrCheck`

4.4 Inherits

From `datetime`: `datetime.datetime.strptime` for date error checking.

5 new_auto_transaction

This module is used by a registering officer to create an auto sale transaction record.

5.1 Interfaces

This module should be accessed via calling `AutoTransaction`. It is called by the `main` module.

5.2 Functions

5.2.1 AutoTransaction

Description:
This function obtains user information and checks the validity of the information sometimes by querying the database. If the user input is valid, it removes all previous vehicle owner information from owner table. Then it adds the new owner information into owner table. Finally, it adds a new auto sale transaction into the auto sale table.
Information needed from the user:

- buyer's SIN (int CHAR(15))
- seller's SIN (int CHAR(15))
- vehicle serial number (int CHAR(15))
- date of sale (DATE)
- vehicle price (numeric(9,2))

Information generated by database:

- transaction id (int)

Assumption:

- The user is entering a valid price.

Parameters:
`connection, curs` - from an active connection to the database

Returns: None.

5.3 Undocumented helper functions

`OwnerCheck, PriceErrCheck, DateErrCheck, VehErrCheck, CheckIfVehExists, IdErrCheck, CheckLen, CheckIfIdExists, GenerateTransaction, CheckIfInt`

5.4 Inherits

From `decimal`: `Decimal` for price format conversion.
From `datetime`: `datetime.datetime.strptime` for date error checking.
From `new_person`: `NewPerson` for new person record creation.

6 new_driver

This module creates a new driver's license record.

6.1 Interfaces

This module should be accessed through the `NewDriver` function. It is called by the `main` module.

6.2 Functions

6.2.1 NewDriver

Description:
This function serves as a gateway to creating a new driver's license record. It does so by requesting the SIN to be input and then deals with the following cases:

- SIN already associated to a license - return to menu.
- SIN already associated with a person; no existing licence - proceed to licence creation.
- No records exist - prompt for person record creation.

Parameters:
`connection, curs` - from an active connection to the database

Returns: None.

6.2.2 NewPeopleUI

Description:
This function serves as the gateway through which a new person record is created. We assume that the SIN entered at this point is logically correct, and prompt for the remaining required information:

- name VARCHAR(40)
- height number(5,2)
- weight number(5,2)
- eyecolor VARCHAR(10)
- haircolor VARCHAR(10)
- addr VARCHAR2(50)
- gender CHAR ('m' or 'f')
- birthday DATE

Parameters:
`connection, curs` - from an active connection to the database
`sin` - string containing a valid SIN

Returns:
`1` if the operation was successful; `0` if not.

6.2.3 `newDriverUI`

Description:
This function is the interface for creating a new driving licence record, once we have verified that the SIN has a person associated in the database. Correctness of data length is handled by truncating the data if necessary. We assume that the driver licence number is determined externally; that is, we do not obtain it from the database.
The user will be prompted for the following info:

- licence_no CHAR(15)
- class VARCHAR(10)
- filename of photo BLOB
- issuing_date DATE
- expiring_date DATE

Parameters:
`connection, curs` - from an active connection to the database
`sin` - string containing a valid SIN

Returns:
`1` if the operation was successful; `0` if not.

6.2.4 `CreateRestrictions`

Description:
Creates the entries in the database for restrictions on driving licences. Called after the driving licence record has been created. We assume no restrictions currently exist for the driver's licence, and keep track of which ones have already been entered. We verify that a new restriction must exist in the driving_condition table, and that it has not already been entered. We assume that any driver can have any number of restrictions (that is, some restrictions do not preclude or include others).

Parameters:
`connection, curs` - from an active connection to the database
`licence_no` - string containing a valid licence number

Returns:
`1` if the operation was successful; `0` if not.

6.3 Undocumented helper functions

`verifyRestrictions`, `newRecUI`, `verifyYN`, `requestPhoto`, `requestSIN`, `requestLicence`, `requestDate`, `requestGender`, `requestAddr`, `requestBodyStats`, `requestCharT`, `requestName`, `recordExists`, `licenceExists`

6.4 Inherits

From `cx_Oracle`: `BLOB` specification for input of photo file.
From `datetime`: `datetime.datetime.strptime` for date error checking.

7 violation_record

This component is used by a police officer to issue a traffic ticket and record the violation.

7.1 Interfaces

This module should be called through the `ViolationRecord` function. It is called by the `main` module.

7.2 Functions

7.2.1 `ViolationRecord`

Description:

Obtain the informaton needed from user and database to create a ticket record in the database table 'ticket'. Need to check user input for validity. The information entered needs to match the formats listed above and the SIN, VIN, officer no., and violation type has to exist in the system already to be be valid.

Information needed from the user (police officer):

- violator's SIN (9 digit integer CHAR)
- violator's VIN (integer CHAR)
- officer's ID (integer CHAR)
- violation type (char)
- date (YYYY-MM-DD date)
- location of violation (varchar)
- any notes about the incident (varchar)

Information that can be obtained from in the database already:

- Ticket number

Parameters:

`connection, curs` - from an active connection to the database

Returns: None

7.3 Undocumented helper functions

`GetValidSin, GetValidVin, GetValidDate`

7.4 Inherits

From `datetime`: `datetime.datetime.strptime` for date error checking.

8 record_search

This module contains the search engine for the car registry database.

8.1 Interfaces

This module should be invoked via the `RecordSearch` function. It is called by the `main` module.

8.2 Functions

8.2.1 RecordSearch

Description:
Allows a user search for one of the following and displays the output:

- The name, licence_no, addr, birthday, driving class, driving_condition, and the expiring_data of a driver by entering either a licence_no or a given name
- All violation records received by a person if the drive licence_no or sin of a person is entered.
- The vehicle_history, including the number of times that a vehicle has been changed hand, the average price, and the number of violations it has been involved by entering the vehicle's serial number

Parameters:
`connection, curs` - from an active connection to the database

Returns: None

8.2.2 DriverRecord

Description:
Driver Record gives the name, licence_no, addr, birthday, class, driving_condition, and expiring_date for a driver that is searched via a query into the database using what the user entered as the name or licence number.

Input requested from user:

- name (case sensitive), or
- licence number

Output:
A list of all of the results or a message saying the record doesn't exist.

Parameters:
`connection, curs` - from an active connection to the database

Returns: None

8.2.3 `DriverAbstract`

Description:
Driver Abstract lists all of the violations for a driver that is searched via a query into the database using what the user entered as the driver's SIN or licence number.

Input requested from user:

- SIN, or
- licence number

Output:
A list of all of the violations or a message saying there is no violation history.

Parameters:
`connection, curs` - from an active connection to the database

Returns: None

8.2.4 `VehicleHistory`

Description:
Vehicle History lists the number of times that a vehicle has been changed hand, the average price, and the number of violations it has been involved in by entering the vehicle's serial number. History is searched for via a query into the database using what the user entered as the vehicle's serial number.

Input requested from user:

- Vehicle serial number

Output:
A list of all of the vehicle's history or a message saying there is no history available for that vehicle.

Parameters:
`connection, curs` - from an active connection to the database

Returns: None

8.3 Undocumented helper functions

`exit`

8.4 Inherits

From `datetime`: `datetime.datetime.strptime` for date error checking.
From `decimal`: `Decimal` for price format conversion.