Assignment 3- DDL Script & Basic DML

Sep 2021

PART 1 - DDL

Goal: In this assignment, you are tasked with creating a fully functioning DDL script that the TAs or Clint could run to create a demo of your **Hotel Reservation System database** that you designed in the previous Database Design Assignment. Your script must not only build the database according to the requirements below but also seed it with some data for testing purposes. Lastly, your script must be able to run over and over, meaning it can drop database objects (e.g. tables, sequences, indexes), recreate those objects, and seed with test data in a single script.

Assignment Requirements:

Do your own work

- This is an individual assignment and you must do your own work and create your own DDL script based on a working/final version posted on Canvas HW2 instructions.

Coding Standards

- **Naming**: Use the final ERD as a reference on how to name tables and columns. For constraints, make the names clear and easy to understand. Avoid abbreviations.
- **Data Types / Lengths:**
  - *NUMBER* – Follow the standard of making IDs numeric and any column that involves arithmetic.
  - *VARCHAR* – All non-numeric fields should be VARCHAR except fields mentioned in CHAR section below.
  - *CHAR* – Phone numbers will all be defined as CHAR length of 12 to allow for 10 digits and 2 dashes (e.g. 512-999-1234). State is the abbreviation of a state so its CHAR length of 2. Zipcode only needs to store a length of 5. CardType on the payments table will always be 4 character abbreviations of cards. Confirmation number will always be 8 characters of random letters/numbers. Reservation status is used to track a simple U for *Upcoming*, I for *In Progress*, C for *Completed*, N for *No-show*, R for *Refunded* so this is always a length of 1. The Room Type will also store a single character for the following room types: D for double beds, Q for single queen, K for single king, S for suite that has two rooms and some form of kitchen, C for cabin.
  - *DATE* – Any dates fields should be DATE format
  - Note: Lengths can vary unless specified above.
- **Commenting**
  - Include comments at least with each section of code (i.e. DROP, CREATE, INSERT, INDEXES)
  - Comments should include a description of section and your name/uteid as the author.
  - If you want to add additional comments to single statements or lines feel free but it's not required. It's just a best practice to comment code well.

Constraints

- Assign primary and foreign keys per the design.
- Only the following can be NULL: *Address_line_2* since it's not required. Customers*' birthdates* since it's not always known. The reservation *Checkout Date, Discount Code, Customer Rating*, and *Notes* are also not required since they are allowed to be blank when the reservation is created. Note, if you see other fields that could be nullable, please clarify these assumptions on Slack.
- The following fields should be UNIQUE: *Customer email, Feature Name, Location Name,* and *Confirmation Number*. *Customer_ID* on payments should also be unique to force it to be a 1-to-1. Again, if you find other potential non-primary keys, bring this up on Slack. Also we need a composite unique constraint on Room table for the *location_id* and *room_number* to prevent duplicate rooms at a location. If you see a need for more non-primary key constraints, please raise that on Slack.
- DEFAULTS:
  - Stay credits earned and used should be set initially to 0 (zero).
  - The reservation *Date_Created* should default to the current date using the SYSDATE function. I know…we didn't cover this in class but you can figure this out if you dig a little in your book or online. You can do it!

- Make sure the following Check constraints are added:
    - Reservation status as of now can only be the following values detailed above: U, I, C, N, or R.
    - The Room Type as of now can only be the following values detailed above: D, Q, K, S, or C.
    - Stay Credits Used should never be greater than the Stay Credits Earned
    - Customer Emailed - emails should have a character length of at least 7 or more. Again, just because we didn't cover it in class doesn't mean you can't google it and figure it out. Name this constraint "email_length_check".

Other

- Create indexes on all foreign keys that are not also part of a primary key. Since primary keys are indexed we won't index a column that is both a PK and FK. Also, index at least 2 other fields in the schema to show you can properly discern which columns should have indexes per design rules discussed in class
- Create sequences that auto-increment the *payment_id, reservation_id, room_id, location_id, and feature_id* by 1 starting at 1. Create a sequence for *customer_id* that increments by 1 starting at 100001.

Format

- **Easy to Read Code:** All your code should be well spaced and indented. If it's not easy to read and messy, you could lose points.
- **Sections**: You must create your script with the three following sections:
    - Drop Sequence/Tables section - Area of the script that drops all tables and sequences in proper order
    - Create Sequence/Tables section - Area of the script that creates tables/sequences and adds constraints either via CREATE or ALTER TABLE statements
    - Insert Data section - Area of the script that inserts data into the tables using "INSERT INTO"
    - Create Index section – After you seed data, add in indexes for the database to optimize performance
- **Commenting**: You should add comments before each section that includes a *description* of what is happening in that section and *your name and UTEID*, which is a best practice to know who coded what.
- The script must build the database shown in the ERD exactly which means table names, column names, and constraints must match. That being said, we are not going to specify the exact names of constraints that you create unless stated above. Just be sure to use logical, clear names for constraints.

Data Requirements

Seed your tables based on the following requirement: NOTE: **Include commits** after each group of inserts for a particular table and don't forget to regularly commit to avoid taxing the server and causing NOWAIT error.

- Create the 3 locations mentioned in HW1 and make-up details on address, phone, and URL.
- Create up to 3 features that can be shared or unique to the locations but make sure at least one location has multiple features assigned to it
- Create 2 rooms for each location (even though in reality there should be more)
- Create 2 customers that have payments attached. The first customer should have **your** first and last name. Their email should be your uteid + "utexas.edu". (e.g. abd123@utexas.edu). The rest of the data about you can be fake. Make up data for the 2nd customer.
- For your customer account, create a single room reservation. For the 2nd, create two separate reservations that are on different dates.
- NOTE: If you need to clarify any data requirements, ask on Slack.

Testing before you turn in your work

- Make sure your script runs without errors before you submit it. Test it by dropping all the tables and running the script. Then run the script to ensure no unexpected errors occur.

What to turn in

- Turn in your DDL script in a (.sql) file format. If you have issues doing this, at least save it as a .txt file. Do not submit in any other format other than .sql and .txt

PART 2 - DML

Instructions:

- **Do your own work:** This is an individual assignment and you must do your own work and create your own SQL statements. If you are caught cheating on this or using someone else's work, you will receive a zero on this assignment and be reported to the Dean of Students. Also, this homework prepares you for the exam coming up so doing the work now will help you learn and do well on when it counts more.
- **What to turn in**
  - Clearly separate your code for each question. Save your code into one SQL file with the naming format: LastName_FirstName_UTEid. Please make sure the lastname and firstname you use matches what is in Canvas.
  - **Save your file either as a .sql file or as a .txt file. If you need help doing this, refer to the page linked in the Canvas assignment. Files saved in a different format will be 50% and files in a different format that cannot be read into SQL (example: PDF) will result in a 0%.**
  - Submit your .sql file on Canvas before the deadline. Late submissions receive 50% off. No submissions will be accepted 24 hours after the deadline.
  - Do not include the DDL in your submission. If you do, you will lose 5 points. Only provide SQL with comments and nothing else. Do this going forward on all other assignments unless noted.
- The SQL problems below will be based on the DDL script that is posted on the Canvas instructions. Download that script and run it before you start.

Problems:

1. Write a SELECT statement that returns following columns from the *Title* table: genre, title, publisher, number_of_pages. Then, run this statement to make sure it works correctly.
   Add an ORDER BY clause to this statement that sorts the result set by number_of_pages in ascending order. Then, run this statement again to make sure it works correctly. This is a good way to build and test a statement, one clause at a time.

2. Write a SELECT statement that returns one column from the *Author* table named author_full_name that combines the first_name and last_name columns. Leave out middle name for now.
   Format this column with the first name, a space, and last name like this:
   ```
   Michael Jordan
   ```
   Sort the result set by last name in descending sequence.
   Use the "IN" operator to return only the authors whose first name begins with letters of A, B, or C.

3. Write a SELECT statement that returns these columns from the *Checkouts* patron_id, title_copy_id, date_out, due_back_date, and date_in. Return only the rows for checkouts that went out in Feb 2021. That means to filter where only date_out between the beginning Feb 1st and Feb 28th. Use the BETWEEN operator. Sort the result by date_in and the date_out so we can see which are in first and then which are not in.

4. Create a duplicate of the previous query but this time update the WHERE clause to use only the following operators (<, >, <=, or >=). Keep the rest of the query the same.

5. Write a SELECT statement that returns these column names and data from the ***Checkouts*** table:

checkout_id	The checkout_id column
title_copy_id	The title_copy_id column
renewals_left	This is calculated as 2 minus the times_renewed. Assign an alias of **renewals_left**

   Use the ROWNUM pseudo column so the result set contains only the first 5 rows from the table.
   Sort the result set by the column alias **renewals_left** in ascending order.

6. Write a SELECT statement that returns the title and genre from Title table and also a third column called "Book_Level". The book level is calculated by dividing the number_of_pages by 100 and the rounding it to have 1 decimal. So a book that is around 100 pages would be a level 1, ea book that is 500 pages would be level 5, etc…
   Once you have this filter to only show books that are greater than a level 9

7. Write a SELECT statement that returns these columns from the *Author*: first_name, middle_name, last_name. Return only rows for users that have a value in middle_name using a NULL operator. Sort by column positions 2 and then 3 in ascending order

8. Using the DUAL table write a SELECT statement that uses SYSDATE function to create a row with these columns:

today_unformatted	The SYSDATE function unformatted
today_formatted	The SYSDATE function in this format: MM/DD/YYYY

This displays a number for the month, a number for the day, and a four-digit year. Use a FROM clause that specifies the Dual table. *Hint: You will need to implement the TO_CHAR function to format the sysdate in the format designated above.*

After you write this add the following columns to the row:

Days_Late	5
Late_fee	.25
Total_late_fees	5 * .25
Late_fees_until_lock	5-(5*.25). This assumes that after you accrue $5 in late fees, your account locks

Your result table contains only one row.

9. Write a query that returns a distinct list of genres form the title table sorted by genre ascending

10. Write a query that returns all the rows from *Titles* for books that contains the word 'Bird'. The problem is that we don't know if the word will be capitalized in the title or not so in your where clause use the LOWER() function to compare the lowercase version of title to filter only rows that contain the word 'bird'