Lab #8.1: Create a database and set up tables

Database: Library, database implementation

Overview

Objective

Use basic SQL DDL statements to set up a database and tables in a database. Based on the logical database design for a business:

- Interpret a logical ERD.
- Finalize the physical design.
- Create the database for the library.
- Create the tables.
- Populate the tables.
- Verify the database setup.

Outcomes

- Setup a database with all tables.
- Populate all tables with records.

Knowledge expected

- Create a database using SQL syntax.
- Create tables using SQL syntax.
- Work with the following `psql` commands: `\q`, `\?`, `\c`, `\l`, `\dt`, `\d [`*`table`*`]`
- To execute SQL statements from a file with the `psql \i` sub-command.

Submission instructions

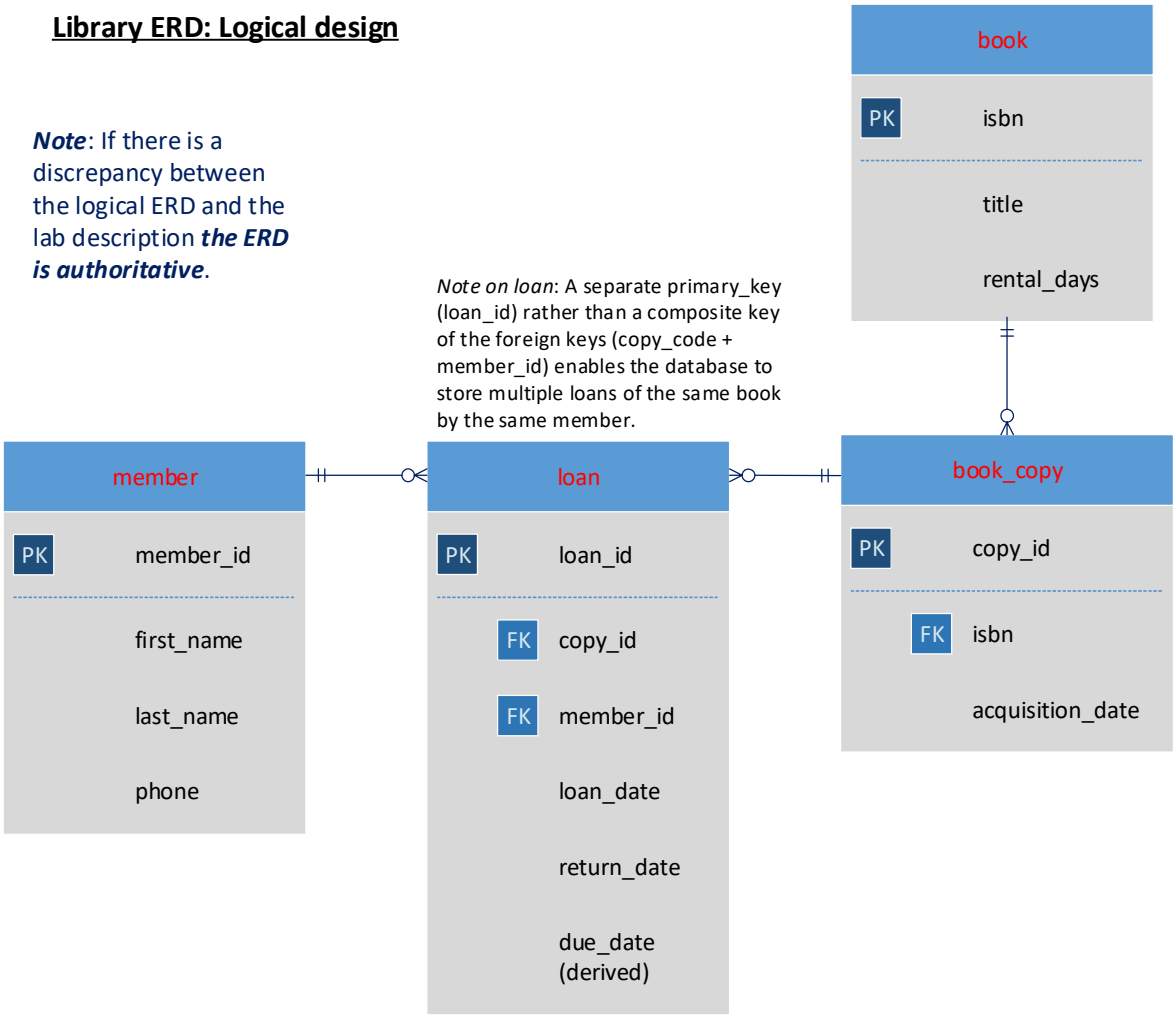
- **READ ALL THE WORDS**
- **You must follow ALL submission instructions**: Submission instructions are explained in the "Lab 8 submission details" document, posted on BrightSpace. ANY submission instructions below NOT followed result in a grade of zero.
  Note: This includes extra functionality that has not been requested.
- You are expected to complete all exercises, even those not are not required to be submitted.

Background

Logical design for relational library database

Library ERD: Logical design

Note: If there is a discrepancy between the logical ERD and the lab description ***the ERD is authoritative***.



Section A - Physical design

- The most functional data types for the attributes of the library relations are listed below:
  - `int` or `smallint`: integer (4-byte) or 2-byte integer
    Example: rental_days: smallint
    Note: The PostgreSQL data type `serial` is an integer that is auto-incremented by the database: it is used to generate surrogate primary keys.
  - `char(`*`length`*`)` : where length is a fixed number of characters
    Example: isbn: char(13)
  - `varchar(`*`maximum_length`*`)` : variable number of characters up to maximum_length
    Example: email: varchar(200)
  - `date`: date
    Example: borrow_date: date

- Below find the completed table that implements the physical design by specifying the datatype for each attribute:

Entity	**Relation name**	**Attribute names** - **identify PK** - **identify FK**	**Data type**

Book	`book`	`PK: isbn`
		`- title`
		`- rental_days`
			`char(13)`
			`varchar(255)`
			`smallint`

Book copy	`book_copy`	`PK: copy_id`
		`FK: isbn`
		`- acquisition_date`
			`int -> serial`
			`char(13)`
			`date`

Member	`member`	`PK: member_id`
		`- first_name`
		`- last_name`
		`- phone`
			`int -> serial`
			`varchar(50)`
			`varchar(200)`
			`varchar(20)`

Book loan	`loan`	`PK: loan_id`
		`FK: copy_id`
		`FK: member_id`
		`- loan_date`
		`- return_date`
		`- due_date (derived)`
			`int -> serial`
			`int`
			`int`
			`date`
			`date`
			n/a: NOT stored in database

SQL overview

SQL syntax

- The semicolon at the end of the statement is part of the SQL syntax.
- SQL commands are case-insensitive.

Common syntax failures

The `psql` prompt gives an indication if there is a SQL syntax failure.

- of a regular prompt for a superuser: `dabase_name=#`
- *Example* of a regular prompt for a non-superuser: `dabase_name=>`
- *Example* of prompt indicating a missing closing bracket: `dabase_name(#`
- *Example* of prompt indicating a missing closing semi-colon: `dabase_name-#`
- To cancel a bad command use: `\r`.

Section B – Create a database

***Syntax practice #1*: Create a new database using SQL syntax.**

To create a database as **dbadmin** with the SQL `CREATE DATABASE` statement:

- Connect to the **postgres** database as the linux user **dbadmin**.
- At the `psql` prompt type: `CREATE DATABASE lib_your_network_ID;`
    - ***Example****:* `CREATE DATABASE lib_smit0001;`
    - *Note*: Don't forget the semicolon at the end of the statement – it is part of the syntax.
    - *Note*: To remove a database, use: `DROP DATABASE database`
- List all databases to verify that it has been created using the `psql \l` sub-command.

Switch to the newly created database using the subcommand `\c database`: the `psql` prompt displays the name of the database you are currently connected to.

Section C – Create tables

Overview of database creation options

- We will setup tables for the new database using SQL statements.
  Note: To drop a table: `DROP TABLE table_name`;
- The database accepts SQL statements from the `psql` prompt or by importing a text file that contains SQL statements.
- Either method is acceptable: the instructions given here are for Linux text file creation and importing. To enter SQL statements on the `psql` prompt see the section labeled "Alternative method" at the end of the lab.

Steps to setup a table

- To create a table follow the physical design:
    - Use the proper naming convention for table and attribute names.
    - Use appropriate data types.
    - Set the primary key.
    - Set up foreign key references where required.
    - Decide when to use the NOT NULL (and optionally other) constraint.
- Create tables in proper order: a table has to be created before it can be referenced.
- Verify the successful creation of each table in the database.

Subcommands of `psql` to verify table creation

The interactive `psql` utility has a sub-command used to list all tables.

- To list all tables: **`\dt`**
  Note: `\d` lists tables, views (virtual tables – *not covered*), and sequences (created for each serial data type).
- To list the fields of a table: **`\d table_name`**.

- Lists attributes, data types and modifiers
- Lists indexes
- Lists foreign key constraints
- ***Example***: `\d member`

Preparation of data

- Create a library directory named `library_db` in the home directory of the user account responsible for managing the library database, dbadmin.
- In your library directory, create a subdirectory named `sql_create_table.d`.
- Within that directory set up one text file for each table and name it `create_<table_name>.sql`.
- *Example*: `~/library_db/sql_create_table.d/create_book.sql`

Verify that you are connected to the newly created database ***before*** creating tables.

***Syntax practice #2*: Create tables using SQL syntax.**

Each file in the `sql_create_table.d` directory contains the SQL statement to create one table.

Table: member
Set the following constraints: PK, NOT NULL

```
 CREATE TABLE member (
   member_id       serial PRIMARY KEY,
   first_name      varchar(100) NOT NULL,
   last_name       varchar(200) NOT NULL,
   phone           varchar(20)
);
```

Table: book
Set the following constraints: PK, NOT NULL

```
  CREATE TABLE book (
    isbn         char(13) PRIMARY KEY,
    title        varchar(255) NOT NULL,
    rental_days  int NOT NULL
  );
```

Table: book_copy
Set the following constraints: PK, FK referencing the book table.

```
  CREATE TABLE book_copy (
    copy_id          serial PRIMARY KEY,
```

```
  isbn             char(13) REFERENCES book(isbn),
  acquisition_date date
);
```

***Syntax practice #3*: Execute SQL statements from a file and verify the result.**

Import all tables into the library database and verify the result.

- The interactive `psql` utility has a sub-command used to execute SQL statements from a file.
  - ***Syntax***: `\i` *`file`*
  - ***Example***: `\i create_member.sql`
    Note: This assumes that you started `psql` from the directory that contain the files; alternative use the Linux path statement to locate the file in the Linux directory structure.
- Verify the successful creation of all tables in the library database, including all relevant columns, data types and constraints.

***Exercise #4*: Create tables to complete the database design requirements.**

Follow the steps below to create the table for `loan` based on the physical design.

- Use SQL syntax to create the `loan` table in the file in the `create_loan.sql` with all required attributes.
- In addition, set the following constraints:
  - Set the primary key.
  - Set a foreign key that references the `book_copy` table.
    Note: When referencing an attribute of data type *serial*, associate the data type *integer* with the foreign key attribute.
  - Set a foreign key that references the `member` table.
- Import the file into the database: this executes the SQL statement.
- Verify the successful creation of all tables in the library database, including all relevant columns, data types and constraints.

***Alternative method*: SQL statement on the `psql` prompt**

- To enter a multi-line SQL statement:
  - Statements in `psql` can span multiple lines: the '**;**' is the "end-of-statement" marker.
  - To enter an SQL statement over multiple lines you can press [Enter] to split up the statement into multiple lines.
    Note: You will notice that *the prompt changes* when entering a multi-line SQL statement.

- Once a table has been created successfully `psql` displays `CREATE TABLE`. Else, errors are identified explicitly.
- *FYI*: After a table definition is entered, `psql` provides "notices" to inform you of additional actions it performs on the table.
  Example for book_copy table:
  - Creates a 'sequence' for the primary key because we are using the `serial` (auto-incrementing integer) data type.
  - Creates an 'index' on the primary key.