Hacettepe University
Computer Engineering Department

BBM 473 - Database Management Systems Laboratory
Experiments Phase 2

Subject: Stored Procedures, Views, Triggers and SQL Queries

Advisors: Res. Assist. Alaettin UÇAN,
Res. Assist. Cemil ZALLUHOĞLU

Project: E-book Subscription Service

Project Members: Ekrem CANDEMİR - 21327771
Berk Kemal ÖZATA - 21328298

1. Experiment

In this experiment, we created user objects such as tables, procedures, triggers, sequences and views using SQL. Then we performed insert, update and delete operations using procedures we created.

In this phase we have done insert, update and delete operations with a single SQL statement using stored procedures. Using triggers and sequences, we implemented auto increment keys. We have used views for hiding complexity and also providing security by creating different views for different users. [1]

NOTES: 'Insert.sql' has delete operations at the end. So there can be less than 10 data on the tables.

2. Procedures

2.1. Insert Procedures

Naming convention that we used in insert procedures as follows: `<table_name>_insert`

`-USER_INSERT`
`USERTBL` is parent of both `COMMERCIALUSER` and `SYSTEMUSER` tables. Before inserting these tables, it must be inserted into `USERTBL`.

`-SYSTEMUSER_INSERT`
This procedure adds the authorized users to the `SYSTEMUSER` table for system related operations. These users have a role and contact details. Since it is inherited from `USERTBL`, this procedure first inserts into `USERTBL`, then inserts into `SYSTEMUSER` and `CONTACT` tables. Also there must be a roleid from `ROLETBL`.

`-CONTACT_INSERT`
Inserts into `CONTACT` table which contains system user's contact information.

`-COMMERCIALUSER_INSERT`
This procedure adds the users who will use e-book services to the `COMMERCIALUSER` table. These users have payment details also. Since it is inherited from `USERTBL`, this procedure first inserts into `USERTBL`, then inserts into `COMMERCIALUSER` and `PAYMENTDETAIL` tables.

`-PAYMENTDETAIL_INSERT`
Inserts into `PAYMENTDETAIL` table which contains commercial user's payment details.

`-ROLE_INSERT`
This procedure creates a role with several permissions for system users.

-PERMISSION_INSERT
This procedure creates a permission that defines what the system user can do.

-ROLEPERMISSION_INSERT
It provides many-many relation between role and permission.

-AUTHOR_INSERT
Inserts into AUTHOR table.
Parameters: author name, author summary, author image, system user performing insert operation and date of the insert, system user performing the last update and date of the last update

-CATEGORY_INSERT
Inserts into CATEGORYTBL table.
Parameters: category name, system user performing insert operation and date of the insert, system user performing the last update and date of the last update

-CATEGORYINHERITANCE_INSERT
It provides many-many relation between parent category and subcategory.
Parameters: parent category, subcategory, system user performing insert operation and date of the insert, system user performing the last update and date of the last update

-BOOK_INSERT
This procedure inserts books into BOOK table.
Parameters: book name, book summary, cover image, system user performing insert operation and date of the insert, system user performing the last update and date of the last update

-BOOKAUTHOR_INSERT
It provides many-many relation between book and author.

-BOOKCATEGORY_INSERT
It provides many-many relation between book and category.

-PUBLISHER_INSERT
This procedure inserts a publisher into PUBLISHER table.
Parameters: publisher name, publisher summary, system user performing insert operation and date of the insert, system user performing the last update and date of the last update

-FILE_INSERT
Inserts published edition of a book in a e-book format or audio-book format. This procedure must be called before inserting e-book or audio-book since FILETBL is parent of them. It includes relations with book and publisher.

Parameters: file size, file language, page number, book id, publisher id, system user performing insert operation and date of the insert, system user performing the last update and date of the last update

-EBOOK_INSERT
This procedure inserts e-book edition of a book into EBOOK table. Since EBOOK table is inherited from FILETBL, this procedure first inserts into FILETBL, then inserts into EBOOK table.
Parameters: ebook file and FILE_INSERT's parameters

-AUDIOBOOK_INSERT
This procedure inserts audio-book edition of a book into AUDIOBOOK table. Since AUDIOBOOK table is inherited from FILETBL, this procedure first inserts into FILETBL, then inserts into AUDIOBOOK table.
Parameters: audio-book file, duration and FILE_INSERT's parameters

-LIBRARY_INSERT:
A commercial user can add e-book or audio-book to his/her library with this procedure.
Parameters: userid, fileid

-SHELF_INSERT
A commercial user can add a private or public shelf with this procedure.
Parameters: shelfname, userid

-SHELFFILE_INSERT:
Commercial user can add a file to his/her shelf. This procedure also inserts that file into user's library.
Parameters: userid, shelfid, fileid

2. 2. Update Procedures

Our approach when updating is to send all parameters belonging to that table. The columns that will not be updated should be sent with their old values. After adding the interface to the system, the fields will already be filled with old values. The user will change the fields he/she wants to change and after submit, all the fields will be overwritten.

Naming convention that we used in update procedures as follows: <table_name>_UPDATE

-USER_UPDATE
This procedure is used to update existing records in USERTBL. The record is updated using the userid of the record to be updated.
Parameters: userid, username, userpassword, firstname, surname, region

-SYSTEMUSER_UPDATE
This procedure is used to update existing records in SYSTEMUSER. It also changes the USERTBL, because the information of each system user is also kept in the USERTBL. As each system user has contact information, it also updates the information in the contact tab. The record is updated using the userid of the record to be updated.
Parameters: userid, username, userpassword, firstname, surname, region, roleid, isactive, country, city, phone

-CONTACT_UPDATE
This procedure is used to update existing records in CONTACT. When SYSTEMUSER_UPDATE is updated, CONTACT_UPDATE is also updated. This record is updated using the userid of the record to be updated.
Parameters: userid, country, city, phone

-COMMERCIALUSER_UPDATE
This procedure is used to update existing records in COMMERCIALUSER. It also changes the USERTBL, because the information of each commercial user is also kept in the USERTBL. The record is updated using the userid of the record to be updated.
Parameters: userid, username, userpassword, firstname, surname, region, ispremium

-PERMISSION_UPDATE
This procedure is used to update existing records in PERMISSION. The record is updated using the permissionid of the record to be updated.
Parameters: permissionid, permissionname, description

-ROLE_UPDATE
This procedure is used to update existing records in PERMISSION. The record is updated using the roleid of the record to be updated.
Parameters: roleid, rolename, description

-PAYMENTDETAIL_UPDATE
This procedure is used to update existing records in PAYMENTDETAIL. The record is updated using the userid of the record to be updated.
Parameters: userid, username, userpassword, firstname, surname, region, ispremium

-AUTHOR_UPDATE
This procedure is used to update existing records in AUTHOR. The record is updated using the authorid of the record to be updated.
Parameters: authorid, author name, author summary, author image, insert userid, last update userid, insert date, last update date

-BOOK_UPDATE
This procedure is used to update existing records in BOOK. The record is updated using the bookid of the record to be updated.

Paramaters: bookid, book name, book summary, book cover image, insert userid, last update userid, insert date, last update date

-CATEGORY_UPDATE
This procedure is used to update existing records in CATEGORY. The record is updated using the categoryid of the record to be updated.
Parameters: categoryid, categoryname, insert userid, last update userid, insert date, last update date

-FILE_UPDATE
This procedure is used to update existing records in FILETBL. The record is updated using the fileid of the record to be updated.
Parameters: fileid, file size, file language, page number, bookid, publisherid, insert userid, last update userid, insert date, last update date, publish date

-EBOOK_UPDATE
This procedure is used to update existing records in EBOOK. The record is updated using the fileid of the record to be updated. It also changes the FILETBL, because the information of each ebook is also kept in the FILETBL.
Parameters: fileid, file size, file language, page number, bookid, publisherid, insert userid, last update userid, insert date, last update date, publish date, ebook file

-AUDIOBOOK_UPDATE
This procedure is used to update existing records in AUDIOBOOK. The record is updated using the fileid of the record to be updated. It also changes the FILETBL, because the information of each ebook is also kept in the FILETBL.
Parameters: fileid, file size, file language, page number, bookid, publisherid, insert userid, last update userid, insert date, last update date, publish date, total duration, audiobook file
LIBRARY_UPDATE
This procedure is used to update existing records in LIBRARY. The record is updated using the fileid and userid of the record to be updated.
Parameters: userid, fileid, current page

-SHELF_UPDATE
This procedure is used to update existing records in SHELF. The record is updated using the shelfid of the record to be updated.
Parameters: shelfid, shelfname, userid, ispublic

-PUBLISHER_UPDATE
This procedure is used to update existing records in PUBLISHER. The record is updated using the publisherid of the record to be updated.
Parameters: publisherid, publisher name, publisher summary, insert userid, last update userid, insert date, last update date

2. 3. Delete Procedures

-PERMISSION_DELETE
This procedure is used to delete records in PERMISSIONTBL. The record is deleted using the permissionid of the record to be deleted. The deletion of a record from PERMISSIONTBL also deletes this record from the ROLEPERMISSION. Because ROLEPERMISSION has foreign key with cascade delete that parent is PERMISSIONTBL.

-ROLE_DELETE
This procedure is used to delete records in ROLETBL. The record is deleted using the roleid of the record to be deleted. The deletion of a record from ROLETBL also deletes this record from the ROLEPERMISSION. Because ROLEPERMISSION has foreign key with cascade delete that parent is ROLETBL.

-USER_DELETE
This procedure is used to delete records in USERTBL. The record is deleted using the userid of the record to be deleted. The deletion of a record from USERTBL also may delete this record from many tables such as COMMERCIALUSER, PAYMENTDETAIL, SYSTEMUSER, CONTACT, SHELF, LIBRARY. Because all of them has foreign key with cascade from the USERTBL.

-SYSTEMUSER_DELETE
System user has dependencies with other tables. Because of that we cannot delete the system user. Instead we inactivate the system user.

-COMMERCIALUSER_DELETE
This procedure is used to delete records in COMMERCIALUSER. The record is deleted using the userid of the record to be deleted. The deletion of a record from COMMERCIALUSER also deletes this record from PAYMENTDETAIL, SHELF, SHELFFILE and LIBRARY.

-FILE_DELETE
This procedure is used to delete records in FILETBL. The record is deleted using the fileid of the record to be deleted. The deletion of a record from FILETBL also deletes this record from EBOOK, AUDIOBOOK and SHELFFILE.

-EBOOK_DELETE
This procedure is used to delete records in EBOOK. The record is deleted using the fileid of the record to be deleted. The deletion of a record from EBOOK also deletes this record from FILETBL.

-AUDIOBOOK_DELETE
This procedure is used to delete records in AUDIOBOOK. The record is deleted using the fileid of the record to be deleted. The deletion of a record from AUDIOBOOK also deletes this record from FILETBL.

-SHELF_DELETE
This procedure is used to delete records in AUDIOBOOK. The record is deleted using the shelfid and userid of the record to be deleted. This takes two parameters because a user can have many shelfs.

-LIBRARY_SINGLE_DELETE
This procedure is used to delete records in LIBRARY. The record is deleted using the userid and fileid of the record to be deleted. The procedure deletes only one record from LIBRARY.

-LIBRARY_ALL_DELETE
This procedure is used to delete records in LIBRARY. The record is deleted using the userid of the record to be deleted. The procedure deletes all records from LIBRARY.

-AUTHOR_DELETE
This procedure is used to delete records in AUTHOR. The record is deleted using the authorid of the record to be deleted.

-BOOK_DELETE
This procedure is used to delete records in BOOK. The record is deleted using the bookid of the record to be deleted.
The deletion of a record from BOOK also deletes this record from BOOKCATEGORY and BOOKAUTHOR.

-CATEGORY_DELETE
This procedure is used to delete records in CATEGORYTBL. The record is deleted using the categoryid of the record to be deleted. The deletion of a record from CATEGORY also may delete this record from CATEGORYINHERITANCE.

-PUBLISHER_DELETE
This procedure is used to delete records in PUBLISHER. The record is deleted using the publisherid of the record to be deleted. The deletion of a record from PUBLISHER also may delete this record from BOOKAUTHOR.

3. Views

3. 1. Aggregate Views

-FILE_ADDINGNUMBER_VIEW
This view shows how many users have added each edition to their shelf or library. Contains both e-book and audio-book.

-AUDIOBOOK_ADDINGNUMBER_VIEW
This view shows how many users have added each audio-book.

-EBOOK_ADDINGNUMBER_VIEW
This view shows how many users have added each e-book.

-AUTHOR_BOOKNUMBER_VIEW
This view shows the number of books of each author registered in the system.
-BOOK_ADDINGNUMBER_VIEW
This view shows how many users have added each book to their shelf or library.

-CATEGORY_BOOKNUMBER_VIEW
This view shows how many books are stored in each category.

-LANGUAGE_BOOKNUMBER_VIEW
This view shows how many editions are registered for each language.

-USER_BOOKNUMBER_VIEW
This view shows how many books are added to each user's shelf or library.

3. 2. System User Views

System user views show the system users who performs transactions on the tables. These views can only be seen by authorized system users. The id and name of the system user who performed the insert, the id and name of the system user who performed the last update are shown in these views.

- *These are the system user views:*
    AUTHOR_ADMINVIEW, BOOK_ADMINVIEW, CATEGORY_ADMINVIEW
    FILE_ADMINVIEW, PUBLISHER_ADMINVIEW, USER_VIEW
    COMMERCIAL_USER_VIEW, SYSTEM_USER_VIEW

3. 3. Commercial User Views

These views provide direct access to the user to view relevant information instead of id's. Also it hides the system user information from the unauthorized users.

-EBOOK_VIEW
This view shows book name, publisher, publish date, page number and ebook file.

-AUDIOBOOK_VIEW
This view shows book name, publisher, publish date, page number, duration and audio file.

-AUTHOR_USERVIEW
This view shows author name, author summary and author image.

-BOOK_USERVIEW
This view shows book name, book summary and cover image.

-PUBLISHER_USERVIEW
This view shows publisher name and publisher summary.

4. Revisions

4. 1. Transaction logs

In our previous design, we were trying to keep the system user actions on a single table named ACTIVITY. We tried to do this by holding the type of table in which the operation was performed and the id of the row in which the operation was performed. Since there are tables with more than one primary key, it is a problem to keep a single id for each row.

ACTIVITY and ACTIVITYOBJECTTYPE tables are no longer available in the current design. Instead, we keep the user who performed the insert operation and the user who performed the update on all the tables that have relation with the system user.

4. 2. Category

In the previous phase we created our design in ER model. After we heard that the diagram should be in relational model, so we translated our diagram into relational diagram. We have forgotten to include the category related tables when doing so.

A category can have multiple subcategories, and vice versa. So that CATEGORY table should have many-many relationship with itself. This relation is kept in the CATEGORYINHERITANCE table.

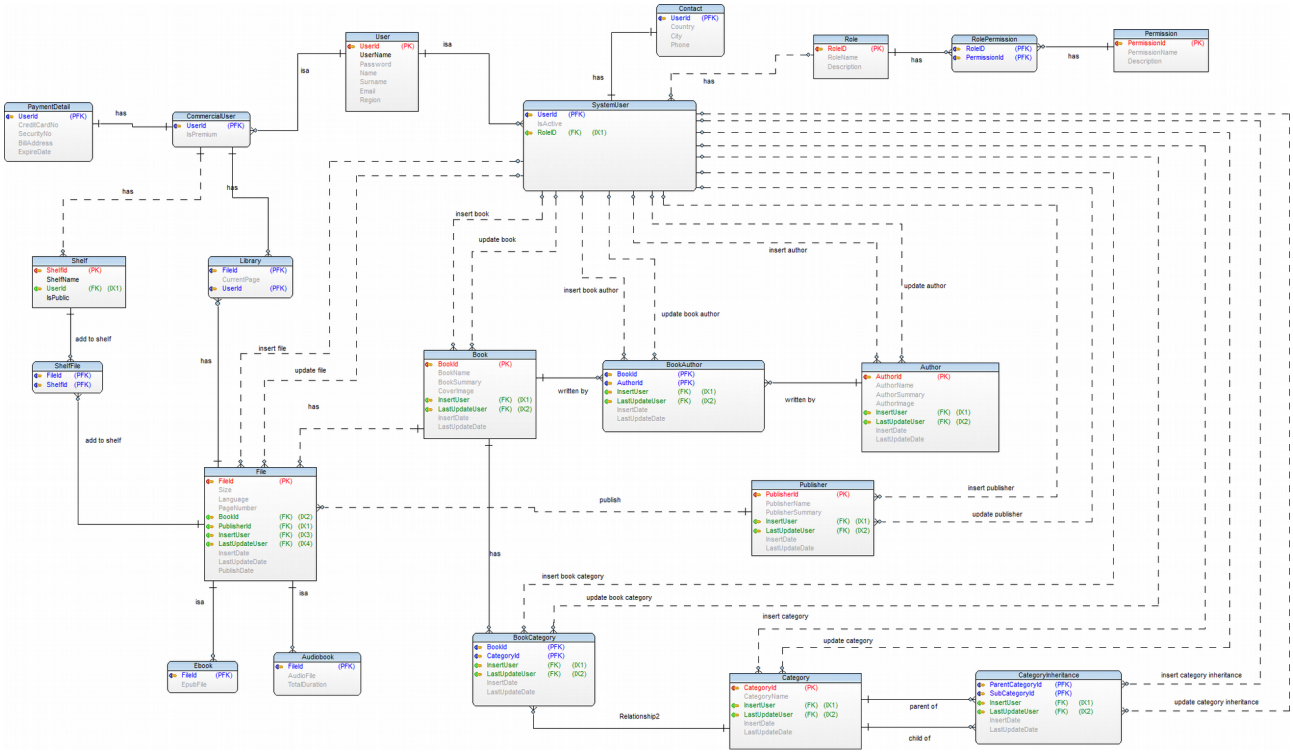
A book can have multiple categories and a category can belong to multiple books. There is many-many relationship between BOOK and CATEGORY tables. The name of the relation is BOOKCATEGORY in the diagram.

4. 3. Primary Key Changes

In the previous design, we thought username should be the primary key since we think each username is unique. We decided to add userid as primary key because of performance issues and possible further changes. For example numeric indexes are much faster than varchar indexes. Also we may allow the username to be changed. [2]

We changed the primary key of SHELF table. We wanted to allow users to create a shelf with the same name so that we added shelf id as primary key.

5. Diagram



6. References

[1] Why do you create a View in a database?, Dave Carlile, Aug 14 2009, stackoverflow, [url]

[2] Why use an auto-incrementing primary key when other unique fields exist?, meagar, May 25 2011, stackoverflow, [url]