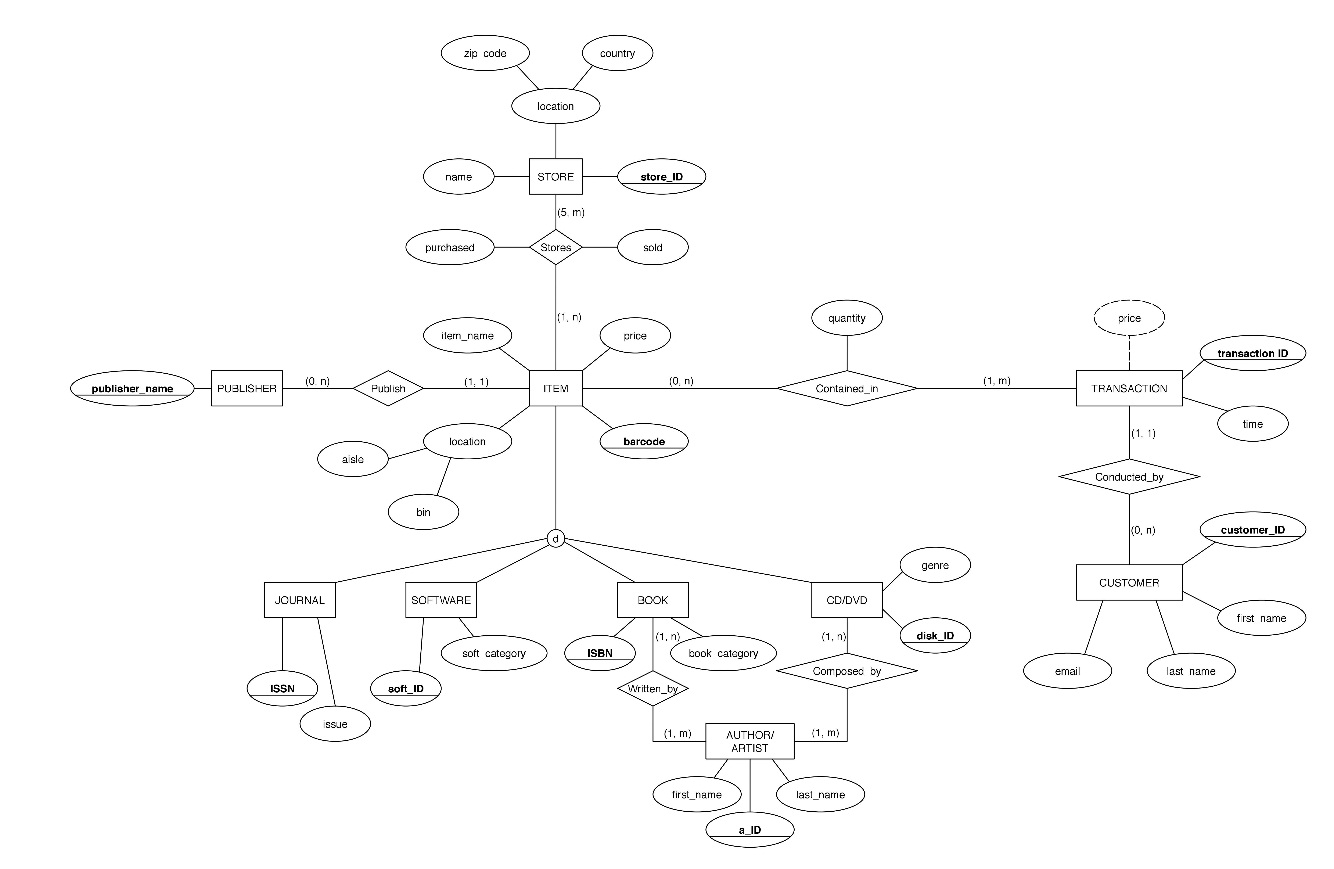
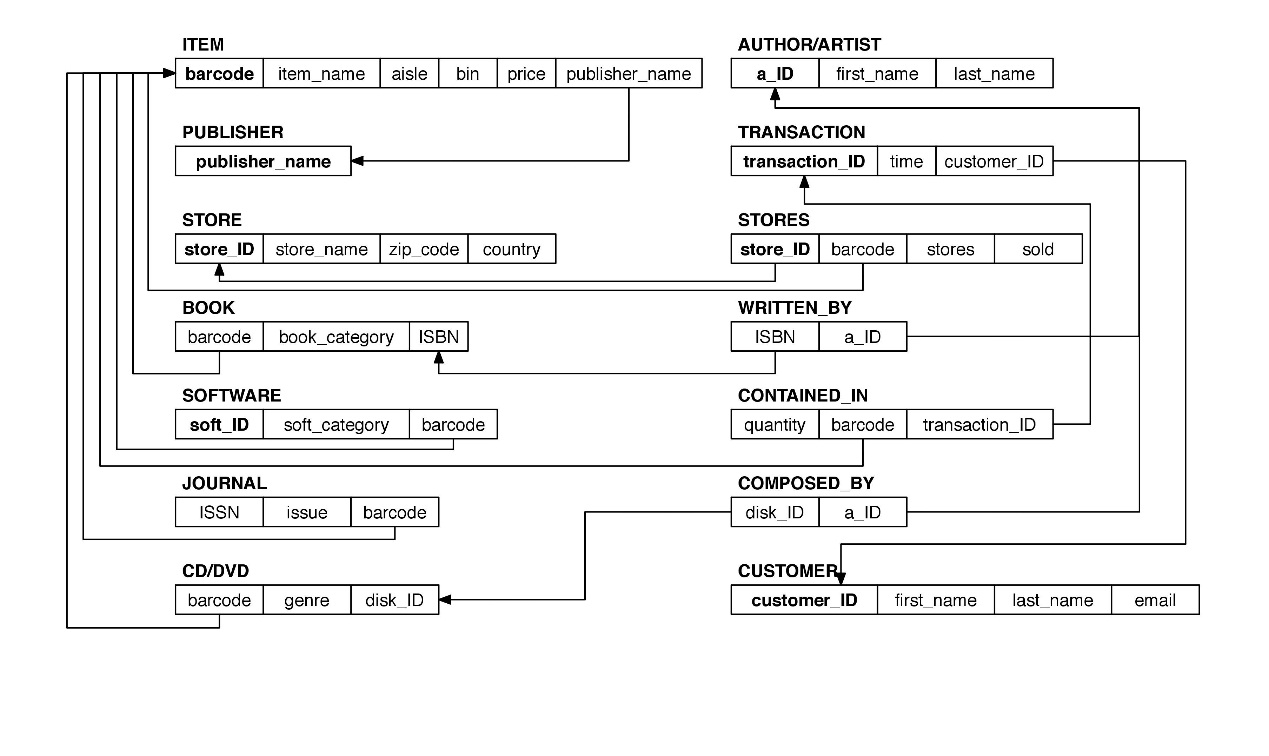
**Section 1 - Database Descriptiona. A professionally presented, well-formatted ER-model that reflects the updates you have made during the semester.  Do not submit a hand-drawn diagram.**



**b. A professionally presented, well-formatted relational schema for the database.  This schema must be annotated with the primary key for each table, all foreign keys on all tables, and all functional dependencies on all tables.  Make sure that connections between FKs and PKs are clear.  **

**c. For each table, give a brief description of the level of normalization achieved for that table.  If the table is not in BCNF, explain why.**

ITEM: **BCNF**; Every other attributes dependent on the single primary key “barcode”.

PUBLISHER: **BCNF**; This table only has one attribute and it’s the primary key.

STORE: **BCNF**; Every other attributes dependent on the single primary key “store\_ID”.

BOOK: **BCNF**; “book\_category” is dependent on superkey.

SOFTWARE: **BCNF**; Every other attributes dependent on the single primary key “barcode”.

JOURNAL: **BCNF**; Every other attributes dependent on the single primary key “barcode”.

CD/DVD: **BCNF**; Every other attributes dependent on the single primary key “barcode”.

AUTHOR/ARTIST: **BCNF**; Every other attributes dependent on the single primary key “a\_ID”.

TRANSACTION: **BCNF**; Every other attributes dependent on the single primary key “transaction\_ID”.

STORES: **BCNF**; {store\_ID, barcode} determine the number of stores and sold.

WRITTEN \_BY: **BCNF**;

CONTAINED\_IN: **BCNF**;

COMPOSED\_BY: **BCNF**;

CUSTOMER: **BCNF**; Every other attributes dependent on the single primary key “customer\_ID”.

**d. A description of each of the indexes that you have chosen to implement on your database, along with rationale for each.**

1. Use clustering for ITEM, and make the item-name sorted for easier searching.

2. Use Tree-based for Inventory in table Stores, easy to check some range for inventory.

3. Use Hash-based index for Time in table Transaction, easy to check the number of transactions made in one specific day.

4. Use clustering for last\_name in table Author/Artist, and for publisher\_name in table Publisher, and for issue in table Journal, and for a\_ID in table Composed\_by, and for last\_name in table Customer.

5. Use Hash-based index for genre in table CD/DVD. It’s easy to check what we have in each genre, and for book\_category in table Book, and for software\_category in table Software.

6. Use Tree-based index for quantity in table Contained\_in. It’s fast to search for some range for quantity of each transaction made.

**e. For each view that you have implemented, provide the following:**

**i. A brief description in English of what this view produces, and why it would be useful.**

View 1, Popular Genres, which displays the most popular music genres according to the sale of CDs belong to different genres.

View 2, Popular Author, which displays the most popular author according to the sale of CD belong to different authors.

**ii. Relational algebra expression to produce this view.**

**View 1, PopularGenres**

R1=CD\_DVD\*CONTAIN\_IN

R2= πgenre,quantity(R1);

R3=genre**ℱ**sum quantity(R1)

R4= **ρ**geren,sold (R3)

**View 2, PopluarAuthor**

R1=WRITTEN\_BY\*BOOK

R2=R1\*CONTAIN\_IN

R3=a\_id**ℱ**sum quantity (R2)

R4=**ρ** a\_id,sold (R3)

R5=AUTHOR\_ARTIST\*R4

R6=πfirst name, last name, sold(R5)

**iii. SQL statements to produce the view.**

CREATE VIEW PopularGenres

AS

SELECT CD.genre AS genre, SUM(quantity) AS sold

FROM CD\_DVD AS CD, CONTAINED\_IN AS CI

WHERE CD.barcode = CI.barcode

GROUP BY sold

ORDER BY genre DESC

CREATE VIEW PopularAuthor

AS

SELECT AA.a\_ID AS a\_ID, first\_name AS fname, last\_name AS lname, SUM(quantity) AS sold

FROM AUTHOR\_ARTIST AS AA, CONTAINED\_IN AS CI, WRITTEN\_BY AS WB, BOOK AS B

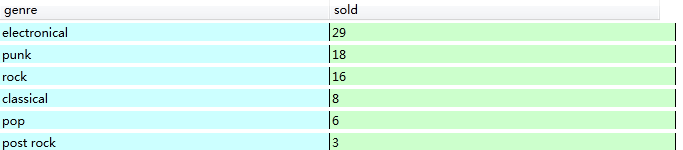
WHERE AA.a\_ID = WB.a\_ID

AND WB.ISBN = B.ISBN

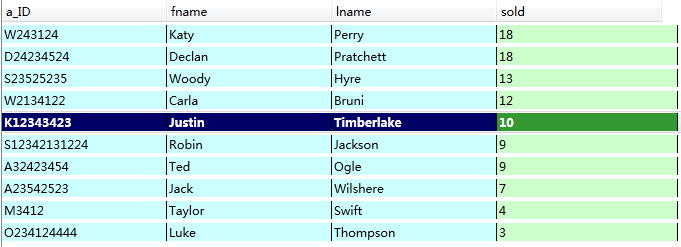
AND B.barcode = CI.barcode

**iv. Sample output from the view, with 5-10 lines of data records shown.**

Results: PopularGenre



Result: PopularAuthor



**f. A professionally presented description of three sample transactions useful for your database. This should include the sample SQL code for each transaction as well as an English language description of what “unit of work” the transaction represents. Remember – a transaction is a sequence of SQL statements taken as a unit – this can be reads and writes together or just a sequence of writes.**

**Transaction 1**

**BEGIN TRANSACTION** CREATE\_TABLE

CREATE TABLE CUSTOMER (

customer\_ID char(10) not null,

first\_name varchar(15) not null,

last\_name varchar(15) not null,

email varchar(30) not null,

PRIMARY KEY (customer\_ID))

IF error THEN GO TO UNDO; END IF;

CREATE TABLE TRANSACTIONS (

transaction\_ID char(20) not null,

time date not null,

customer\_ID char(10) not null,

PRIMARY KEY(transaction\_ID)

FOREIGN KEY(customer\_ID) REFERENCES CUSTOMER(customer\_ID))

IF error THEN GO TO UNDO; END IF;

**COMMIT**;

GO TO FINISH;

UNDO:

ROLLBACK;

FINISH:

**END TRANSACTION;**

When creating table, we should consider the creating order since the new table my reference other tables’ primary keys as its foreign key. Thus we should firstly create talbe CUSTOMER then create table TRANSACTIONS.

If CREATE TABLE CUSTOMER fails, do not do the CREATE TABLE TRANSACTIONS

If CREATE TABLE TRANSACTIONS fails, undo the CREATE TABLE CUSTOMER

**Transaction 2**

**BEGIN TRANSACTION** CHANGE\_NAME

UPDATE AUTHOR\_ARTIST

SET a\_id = ‘A23542529’

WHERE last\_name=' Wilshere';

IF error THEN GO TO UNDO; END IF;

UPDATE WRITTEN\_BY

SET a\_id = ‘A23542529’

WHERE a\_id = ‘A23542523’;

IF error THEN GO TO UNDO; END IF;

**COMMIT**;

GO TO FINISH;

UNDO:

ROLLBACK;

FINISH:

**END TRANSACTION;**

When changing the value of a\_id in AUTHOR\_ARTIST, we should also change the a\_id in WRITTEN\_BY.

If UPDATE AUTHOR\_ARTIST fails, do not do the UPDATE WRITTEN\_BY

If UPDATE WRITTEN\_BY fails, undo the UPDATE AUTHOR\_ARTIST.

**Transaction 3**

**BEGIN TRANSACTION** DELETE\_DATA

DELETE FROM AUTHOR\_ARTIST

WHERE a\_id=’ D24234524’;

IF error THEN GO TO UNDO; END IF;

DELETE FROM COMPOSED\_BY

WHERE a\_id=’ D24234524’;

IF error THEN GO TO UNDO; END IF;

**COMMIT**;

GO TO FINISH;

UNDO:

ROLLBACK;

FINISH:

**END TRANSACTION;**

When deleting an artist in AUTHOR\_ARTIST, we should also delete his/her works in WRITTEN\_BY.

If DELETE FROM AUTHOR\_ARTIST fails, do not do the DELETE FROM COMPOSED\_BY

If DELETE FROM COMPOSED\_BY fails, undo the DELETE FROM AUTHOR\_ARTIST.