Project team #22 - Comic Book Store

Team Members: Jacob Kedar Krevat, Noah Shaw, Kishan Singh, Omkar Nene

PROJECT PROPOSAL

Content, Scope and Objectives

The purpose of this database is to manage inventory and subscriptions for a comic book store. The potential application built atop this database would allow Subscribers to browse comic books by a number of filters, by publisher, by writer, by title, by artist, by storyline, by character, etc. Additionally, administrators working at the comic book store can manage Subscriber subscriptions and purchases. Subscriptions at comic book stores work in that a Subscriber can choose to subscribe to a certain title or a certain storyline, while also being allowed to reserve special books ahead of time.

There are various relationships between multiple different comic books in the inventory including what was stated above, being writer, artist, character, etc. Furthermore, comic books can have more complicated relationships with one another in terms of the storylines to which the books belong. Because of the nature of the comic book narratives, a single book could be a part of multiple storylines that interact with each other. The system will need to reflect how multiple books can belong to multiple storylines. Additionally the database can utilize real world data, better simulating an implementation experience.

A large part of a comic book store is not just in back issues, which are books which have already come out, but also in books that are yet to come out. The system will need to reflect this idea because Subscribers subscribe to books and preorder books which have not yet been released.

PROJECT ENVIRONMENT

Our database is hosted by Amazon Web Services as part of Amazon RDS. We are connecting to this database by using the MySQL Workbench. The username and password are shared among our group. The user interface is created in Java and utilizes the WindowBuilder 1.9.1 GUI builder to build the interface. The application uses the MySQL Connector J 8.0 to connect to the database. The project is available on GitHub so that the entire team has access to it.

HIGH LEVEL REQUIREMENTS

Initial user roles

User Role	**Description**
Manager	The manager manages the shop and all of its inventory. They have access to the comic book catalog, inventory, Employees, and Subscribers.
Subscriber	The Subscriber will browse the online catalog searching for comic books that they are interested in purchasing and subscribe to series. They have access to the comic book catalog.
Employee	The Employee interacts with the Subscribers/Free Users and facilitates transactions. The Employee has access to the comic book catalog and Subscribers.
Free User	The Free User will browse the online catalog searching for comic books they are interested in purchasing. They have access to the comic book catalog. (Free Users are not subscribers)

Initial user story descriptions

Story ID	**Story description**
US1	As a manager, I want to view the inventory so that I restock if inventory is low.

US2	As a manager, I want to modify the inventory so that I can reflect new purchases by the store.
US3	As a manager, I want to modify the catalog so that I can reflect new books that come out.
US4	As a manager, I want to view subscriptions so that I know how many books to order.
US5	As a manager, I want to view the users* so that I can manage and contact employees and Subscribers.
US6	As an Employee, I want to interface with the catalog and inventory to assist users*.
US7	As a Subscriber, I want to browse the catalog so that I can see what books are available.
US8	As a free user, I want to browse the catalog so that I can see what books are available.
US9	As an Employee, I want to record transactions so that inventory can be updated accordingly.
US10	As an Employee, I want to add multiple single issues to a transaction so that a Subscriber can buy multiple items at once.
US11	As a Free User, I want to become a Subscriber so that I can create a pull list.
US12	As a Subscriber, I want to add subscriptions to my pull list so that I can reserve books every month that I want.
US13	As a Subscriber, I want to see which issues a trade paperback has so that I can know whether or not I want it.
US14	As a Subscriber, I want to to see the content of the omnibus so that I know whether or not I want to it.
US15	As a Subscriber, I want to browse the catalog by series so that I can find the books that belong to a particular series.
US16	As a Subscriber, I want to browse the catalog by publisher so that I can find books published by publishers

	I like.
US17	As a Subscriber, I want to browse the catalog by storyline so that I can see which books I want to buy.
US18	As a Subscriber, I want to browse the catalog by character so that I can find books which have characters I like.
US19	As a Subscriber, I want to browse the catalog by writer so that I can find books written by writers I like.
US20	As a Subscriber, I want to browse the catalog by artist so that I can find books drawn by artists I like.
US21	As a Subscriber, I want to browse the catalog by inker so that I can find books inked by inkers I like.
US22	As a Subscriber, I want to browse the catalog by colorist so that I can find books colored by the colorist I like.

*This is representative of all tracked user roles (ie Subscribers, Employees, etc..)

HIGH-LEVEL CONCEPTUAL DESIGN

Entities:

Manager
Subscriber
Free User
Employee
Catalog
Inventory
Single Issue
Series
Storyline
Character
Trade
Omnibus
Writer
Artist
Inker
Colorist

Publisher
Transaction
Subscription
Pull List

Relationships:

Manager views users (ie Subscribers, Employees, etc..).
Manager views inventory.
Manager modifies Inventory.
Manager views Pull List.
Manager alters the Catalog.
Manager views Subscriptions.
Subscriber adds Subscriptions.
Subscriber views Omnibus.
Subscriber filters the catalog by character.
Subscriber filters the catalog by series.
Subscriber filters the catalog by storyline.
Subscriber filters the catalog by artist.
Subscriber filters the catalog by writer.
Subscriber filters the catalog by inker.
Subscriber filters the catalog by colorist.
Subscriber filters the catalog by publisher.
Employee queries the Inventory.
Employee can add multiple single issues to a transaction.
Employee records transactions.
Free User becomes a Subscriber
Employee interfaces with Catalog.
Free User can search books in the Catalog.

Sprint 1

REQUIREMENTS

Story ID	Story description
US1	As a manager, I want to view the inventory so that I can restock if inventory is low NOTE: Need inventory relation to represent owned copies of issues, manager should be able to query entire list.
US2	As a manager, I want to modify the inventory so that I can reflect new purchases by the store. NOTE: Manager needs the capability to modify the inventory to alter table entries.
US3	As a manager, I want to modify the catalog so that I can reflect new books that come out. NOTE: Manager should be able to update and insert into catalog
US4	As a manager, I want to view subscriptions so that I know how many books to order. NOTE: Manager should be able to query subscriptions to get an aggregation of subscriptions
US5	As a manager, I want to view the users* so that I can manage and contact employees and Subscribers. NOTE: Manager should be able to query subscriptions joined on users to see which users are subscribed to which books. Manager should be able to mail employees but not subscribers.
US6	As an Employee, I want to interface with the catalog and inventory to assist users*. NOTE: Employee can assist both free users and subscribers

US7	As a User* , I want to browse the catalog so that I can see what books are available. NOTE: User* refers to free user and subscriber who can both query the entire catalog to view issues and issue attributes
US8	As an Employee, I want to record multiple single issues in a transaction so that inventory can be updated accordingly. NOTE: Employee can record transactions with users and can record multiple purchases per transaction
US9	As a Free User, I want to become a Subscriber so that I can create a pull list. NOTE: Free user should be able to change their status from free user to subscriber within the database
US10	As a Subscriber, I want to add subscriptions to my pull list so that I can reserve books every month that I want. NOTE: Subscriber should be able to update and insert into their pull list
US11	As a Subscriber, I want to add an end date to a subscription in my pull list so that I do not have to cancel it myself NOTE: A start and end date will be kept track, if user elects to not select an end date that field will be left NULL
US12	As a Subscriber, I want to see which issues a trade paperback has so that I can know whether or not I want it.
US13	As a Subscriber, I want to to see the content of the omnibus so that I know whether or not I want to it.
US14	As a Subscriber, I want to browse the catalog by series so that I can find the books that belong to a particular series.
US15	As a Subscriber, I want to browse the catalog by publisher so that I can find books published by publishers

	I like.
US16	As a Subscriber, I want to browse the catalog by storyline so that I can see which books I want to buy.
US17	As a Subscriber, I want to browse the catalog by character so that I can find books which have characters I like.
US18	As a Subscriber, I want to browse the catalog by writer so that I can find books written by writers I like.
US19	As a Subscriber, I want to browse the catalog by pencilist so that I can find books drawn by pencilist I like.
US20	As a Subscriber, I want to browse the catalog by inker so that I can find books inked by inkers I like.
US21	As a Subscriber, I want to browse the catalog by colorist so that I can find books colored by the colorist I like.

CONCEPTUAL DESIGN

Include your detailed conceptual design here. Use the format shown below.

Entity: **Employee**
Attributes:
- name [composite]
  - first_name
  - last_name
- email
- phone_number [multi-value]
- address [composite]
  - street_address
  - city
  - state
  - zip
  - country

Entity: **Manager**
Attributes:
- name [composite]
  - first_name
  - last_name
- email
- phone_number [multi-value]
- address [composite]
  - street_address
  - city
  - state
  - zip
  - country

Entity: **Subscriber**
Attributes:
- name [composite]
  - first_name
  - last_name
- email
- phone_number [multi-value]

Entity: **Single Issue**
Attributes:
- issue_name [composite]
  - series_name
  - issue_number
- price
- availability_date
- writer [multi-valued]
- pencilist [multi-valued]
- colorist [multi-valued]
- inker [multi-valued]
- genre [multi-valued]
- quantity
- availability [derived]

Entity: **Series**

Attributes:

series_name
publisher
total_issues [derived]

Entity: **Transaction**
Attributes:

transaction_items [multi-valued]
total_price [derived]
num_items [derived]
transaction_date
subscriber
employee

Entity: **Transaction Item**
Attributes:

transaction
issue_name
quantity
price [derived]

Entity: **Subscription**
Attributes:

series
pull_list
start_date
end_date

Entity: **Pull List**
Attributes:

subscriber
subscription_item [multi-value]

Relationship: A **Single Issue** is part of a **Series**
Cardinality: Many to One
Participation:
    Single Issue has partial participation
    Series has total participation

Relationship: A **Subscriber** owns a **Pull List**
Cardinality: One to One
Participation:
    Subscriber has total participation
    Pull List has total participation

Relationship: A **Pull List** is made up of **Subscriptions**
Cardinality: One to Many
Participation:
    Pull List has partial participation
    Subscription has total participation

Relationship: A **Subscription** is made up of a **Series**
Cardinality: Many to One
Participation:
    Subscription has total participation
    Series has partial participation

Relationship: An **Employee** facilitates a **Transaction**
Cardinality: One to Many
Participation:
    Employee has partial participation
    Transaction has total participation

Relationship: A **Subscriber** makes a **Transaction**
Cardinality: One to Many
Participation:
    Subscriber has partial participation
    Transaction has partial participation

Relationship: A **Transaction** is made up of **Transaction Items**
Cardinality: One to Many
Participation:

Transaction has total participation
Transaction Item has total participation

Relationship: A **Transaction Item** has an **Issue**
Cardinality: Many to One
Participation:

Transaction Item has total participation
Issue has partial participation

LOGICAL DESIGN

Table: **Employee***
Columns:
- employee_id
- first_name
- last_name
- email
- mobile_phone_number

Justification (if needed): Managers have the same attributes as Employees because Manager is a type of Employee, so they could be combined and to specify that an Employee is a Manager, there is a boolean attribute: is_manager. For the multivalued phone number, it was split into two different phone numbers: mobile and home.

Table: **Subscriber***
Columns:
- subscriber_id
- first_name
- last_name
- email
- mobile_phone_number
- home_phone_number

Justification (if needed): Because Subscriber to Pull List was a one-to-one relationship and Pull List did not have any fields besides for the relationship to subscriptions, Pull List does not need to be represented in the relation. For the multivalued phone number, it was split into two different phone numbers: mobile and home.

Table: **Issue***
Columns:
- issue_id
- issue_number
- price
- quantity

availability_date
series_id [foreign key; references **series_id** of **Series**]

Table: **Genre***
Columns:
genre_id
genre_name

Table: **Writer***
Columns:
writer_id
writer_last_name
writer_first_name

Table: **Pencilist***
Columns:
pencilist_id
pencilist_last_name
pencilist_first_name

Table: **Inker***
Columns:
inker_id
inker_last_name
inker_first_name

Table: **Colorist***
Columns:
colorist_id
colorist_last_name
colorist_first_name

Table: **IssueWriter***
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
writer_id [foreign key; references **writer_id** of **Writer**]

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Writer.

Table: **IssuePencilist***
Columns:
issue_id[foreign key; references **issue_id** of **issue**]
pencilist_id[foreign key; references **pencilist_id** of **Pencilist**]

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Pencilist.

Table: **IssueInker***
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
inker_id [foreign key; references **inker_id** of **Inker**]

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Inker.

Table: **IssueColorist***
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
colorist_id [foreign key; references **colorist_id** of **Colorist**]

Table: **IssueGenre***
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
genre_id [foreign key; references **genre_id** of **Genre**]

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Genre.

Table: **Series***
Columns:
- series_id
- publisher_id [foreign key; references **publisher_id** of **Publisher**]
- series_name

Table: **Publisher***
Columns:
- publisher_id
- publisher_name

Justification: To keep the relation in at least 3NF, Publisher Name needed to be extracted to its own relation

Table: **Transaction***
Columns:
- transaction_id
- employee_id [foreign key; references **employee_id** of **Employee**]
- subscriber_id [foreign key; references **subscriber_id** of **Subscriber**]
- transaction_date

Table: **TransactionItem***
Columns:
- transaction_item_id
- quantity
- issue_id [foreign key; references **issue_id** of **Issue**]
- transaction_id [foreign key; references **transaction_id** of **Transaction**]

Table: **Subscription***
Columns:
- subscription_id
- start_date
- end_date

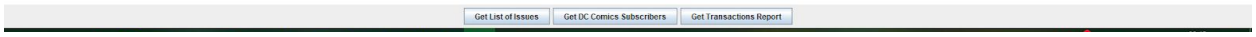
subscriber_id [foreign key; references **subscriber_id** of **Subscriber**]
series_id [foreign key; references **series_id** of **Series**]

SQL QUERIES

Query Issue Information:

```
SELECT
        CONCAT(series_name, ' #', issue_number) issue_name
    ,publisher_name
    ,GROUP_CONCAT(DISTINCT genre_name SEPARATOR ', ') genres
            ,GROUP_CONCAT(DISTINCT  CONCAT_WS('  ',  writer_first_name,
writer_last_name) SEPARATOR ', ') writers
          ,GROUP_CONCAT(DISTINCT  CONCAT_WS('  ',  pencilist_first_name,
pencilist_last_name) SEPARATOR ', ') pencilists
            ,GROUP_CONCAT(DISTINCT  CONCAT_WS('  ',  inker_first_name,
inker_last_name) SEPARATOR ', ') inkers
          ,GROUP_CONCAT(DISTINCT  CONCAT_WS('  ',  colorist_first_name,
colorist_last_name) SEPARATOR ', ') colorists
FROM
        Issue i
    LEFT OUTER JOIN Series s ON i.series_id=s.series_id
    LEFT OUTER JOIN Publisher pu ON s.publisher_id=pu.publisher_id
    LEFT OUTER JOIN IssueColorist ic ON i.issue_id=ic.issue_id
    LEFT OUTER JOIN Colorist c ON ic.colorist_id=c.colorist_id
    LEFT OUTER JOIN IssueGenre ig ON i.issue_id=ig.issue_id
    LEFT OUTER JOIN Genre g ON ig.genre_id=g.genre_id
    LEFT OUTER JOIN IssueInker ii ON i.issue_id=ii.issue_id
    LEFT OUTER JOIN Inker ink ON ii.inker_id=ink.inker_id
    LEFT OUTER JOIN IssuePencilist ip ON i.issue_id=ip.issue_id
    LEFT OUTER JOIN Pencilist p ON ip.pencilist_id=p.pencilist_id
    LEFT OUTER JOIN IssueWriter iw ON i.issue_id=iw.issue_id
    LEFT OUTER JOIN Writer w ON iw.writer_id=w.writer_id
GROUP BY i.issue_id
```

issue_name	publisher_name	genres	writers	pencilists	inkers	colorists
Batman #0001	DC Comics	Superhero	Tom King	David Finch	Matt Banning	
Batman #0002	DC Comics	Superhero	Tom King	David Finch	Danny Miki, Matt Banning	Jordie Bellaire
Batman #0003	DC Comics	Superhero	Tom King	David Finch	Danny Miki, Matt Banning	Jordie Bellaire
Batman #0004	DC Comics	Superhero	Tom King	David Finch	Matt Banning, Sandra Hope	Jordie Bellaire
Batman #0005	DC Comics	Superhero	Tom King	David Finch	Matt Banning, Sandra Hope	Jordie Bellaire
Superman #0001	DC Comics	Superhero	Peter Tomasi	Patrick Gleason	Mick Gray	
Superman #0002	DC Comics	Superhero	Patrick Gleason, Peter Tomasi	Patrick Gleason	Mick Gray	John Kalisz
Superman #0003	DC Comics	Superhero	Patrick Gleason, Peter Tomasi	Jorge Jimenez	Jorge Jimenez	Alejandro Sanchez
Superman #0004	DC Comics	Superhero	Patrick Gleason, Peter Tomasi	Patrick Gleason	Mick Gray	John Kalisz
Superman #0005	DC Comics	Superhero	Patrick Gleason, Peter Tomasi	Doug Mahnke	Jaime Mendoza	Will Quintana
Amazing Spider-Man #0001	Marvel Comics	Superhero	Nick Spencer	Ryan Ottley	Ryan Ottley	Ryan Ottley
Amazing Spider-Man #0002	Marvel Comics	Superhero	Nick Spencer	Ryan Ottley	Ryan Ottley	Ryan Ottley
Amazing Spider-Man #0003	Marvel Comics	Superhero	Nick Spencer	Ryan Ottley	Ryan Ottley	Ryan Ottley
Amazing Spider-Man #0004	Marvel Comics	Superhero	Nick Spencer	Ryan Ottley	Ryan Ottley	Ryan Ottley
Amazing Spider-Man #0005	Marvel Comics	Superhero	Nick Spencer	Ryan Ottley	Ryan Ottley	Ryan Ottley

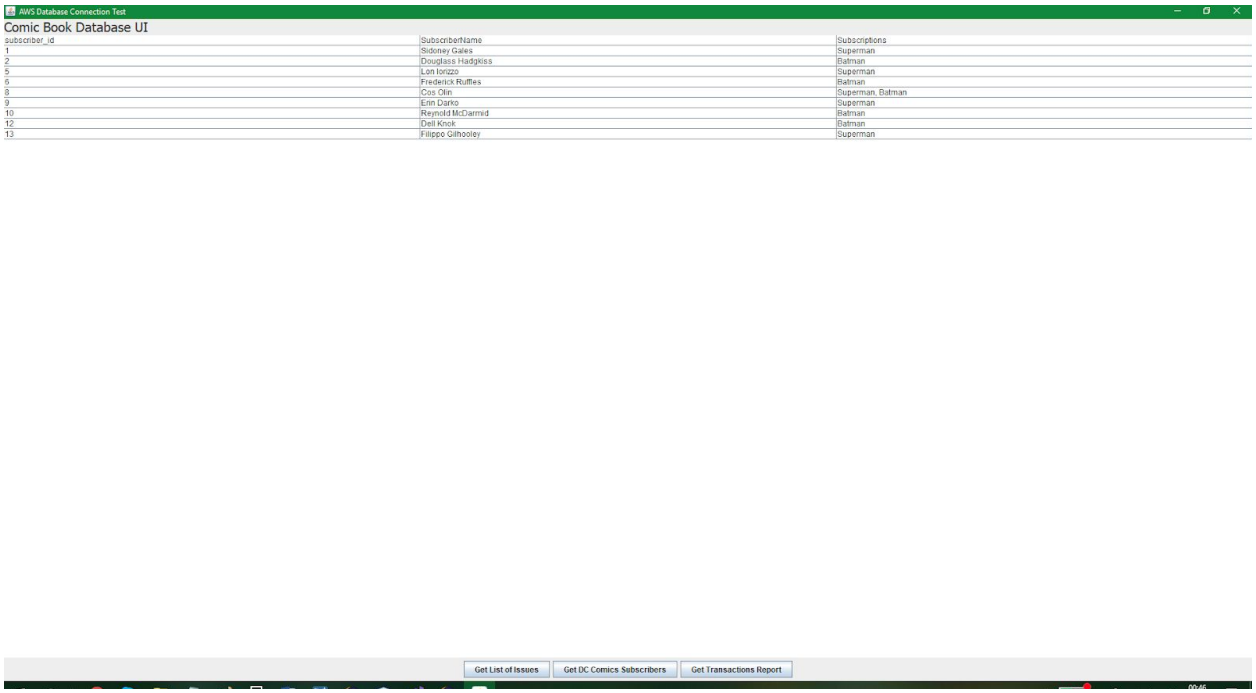


List of DC Comics subscriptions per subscriber subscribed to DC Comics

```
SELECT
        Subscriber.subscriber_id
    ,CONCAT_WS(' ', first_name, last_name) SubscriberName
    ,GROUP_CONCAT(series_name SEPARATOR ', ') Subscriptions
FROM
        Subscriber
        INNER JOIN Subscription USING (subscriber_id)
    INNER JOIN Series USING (series_id)
        INNER JOIN Publisher USING (publisher_id)
WHERE publisher_name = 'DC Comics'
GROUP BY subscriber_id
```

;

	subscriber_id	SubscriberName	Subscriptions
▶	1	Sidoney Gales	Superman
	2	Douglass Hadgkiss	Batman
	5	Lon Iorizzo	Superman
	6	Frederick Ruffles	Batman
	8	Cos Olin	Superman, Batman
	9	Erin Darko	Superman
	10	Reynold McDarmid	Batman
	12	Dell Knok	Batman
	13	Filippo Gilhooley	Superman



Generate Transactions Report

```
SELECT
        t.transaction_id
    ,CONCAT(e.first_name, ' ', e.last_name) EmployeeName
       ,IFNULL(CONCAT(s.first_name, ' ', s.last_name), 'Not a Subscriber')
SubscriberName
    ,SUM(tir.quantity) 'Number of Items'
            ,GROUP_CONCAT(tir.transaction_item_record SEPARATOR ', ')
'Transaction Description'
    ,SUM(tir.Price_Per_Item) AS 'Total Price'
    ,t.transaction_date
FROM
        Transaction t
    INNER JOIN Employee e USING (employee_id)
    INNER JOIN Subscriber s USING (subscriber_id)
    INNER JOIN (
            SELECT
                    CONCAT(ti.quantity, ' copies of ', series_name, ' #',
issue_number) transaction_item_record
          ,transaction_id
          ,ti.quantity
          ,ROUND(ti.quantity * price, 2) AS Price_Per_Item
            FROM
                    Transaction st
                    INNER JOIN TransactionItem ti USING (transaction_id)
                    INNER JOIN Issue USING (issue_id)
                    INNER JOIN Series USING (series_id)
            ) tir ON t.transaction_id = tir.transaction_id
GROUP BY t.transaction_id
ORDER BY t.transaction_date DESC
;
```

transaction_id	EmployeeName	SubscriberName	Number of Items	Transaction Description	Total Price	transaction_date
11	Andrew Dalton	Erin Darko	1	1 copies of Batman #0001	2.99	2018-10-20 11:01:06
5	Arabella Valdez	Reynold McDarmid	9	6 copies of Superman #0005, 3 copies of Superman #0002	26.91	2018-10-15 14:05:36
9	Kathryn Watts	Frederick Ruffles	4	4 copies of Batman #0003	11.96	2018-10-08 19:02:13
8	Azaria Hanna	Jeannie Tompkins	8	8 copies of Batman #0004	23.92	2018-09-10 14:31:45
4	Azaria Hanna	Frederick Ruffles	13	7 copies of Amazing Spider-Man #0001, 6 copies of Superman #0004	59.87	2018-07-01 16:45:15
10	Arabella Valdez	Cos Olin	6	6 copies of Batman #0002	17.94	2018-06-20 17:40:09
3	Andrew Dalton	Burch McCandless	1	1 copies of Amazing Spider-Man #0002	3.99	2018-06-15 10:16:45

AWS Database Connection Test

Comic Book Database UI

transaction_id	EmployeeName	SubscriberName	Number of Items	Transaction Description	Total Price	transaction_date
11	Andrew Dalton	Erin Darko	1	1 copies of Batman #0001	2.99	2018-10-20 11:01:06
5	Arabella Valdez	Reynold McDermid	9	6 copies of Superman #0005, 3 copies of Super...	26.91	2018-10-15 14:05:36
9	Kathryn Watts	Frederick Ruffles	4	4 copies of Batman #0003	11.96	2018-10-08 19:02:13
8	Azaria Hanna	Jeannie Tompkins	8	8 copies of Batman #0004	23.92	2018-09-10 14:31:45
4	Azaria Hanna	Frederick Ruffles	13	7 copies of Amazing Spider-Man #0001, 6 copie...	59.87	2018-07-01 16:45:15
10	Arabella Valdez	Cos Olin	6	6 copies of Batman #0002	17.94	2018-06-20 17:40:09
3	Andrew Dalton	Burch McCandless	1	1 copies of Amazing Spider-Man #0002	3.99	2018-06-15 10:16:45
2	Kathryn Watts	Douglass Hadgkiss	10	2 copies of Amazing Spider-Man #0004, 8 copie...	55.9	2018-04-28 15:05:36
1	Owen Costa	Sidoney Gales	2	2 copies of Amazing Spider-Man #0005	7.98	2018-04-25 18:00:56
7	Lilah Salas	Filippo Gilhooley	2	2 copies of Superman #0001	5.98	2018-02-18 19:08:04
6	Owen Costa	Dell Knok	19	3 copies of Superman #0001, 8 copies of Super...	64.81	2018-01-11 09:32:23

Get List of Issues | Get DC Comics Subscribers | Get Transactions Report

Sprint 2 - Comic Book Store (Team #22)

REQUIREMENTS

List your updated user stories in decreasing order of priority. Highlight the stories for which database design was completed in Sprint 1 in one color. Highlight the updated/new stories chosen for Sprint 2 in a different color. *There is no need to explicitly show your story refinement process.* Use the format shown below.

Story ID	Story description
US1	As a manager, I want to view the inventory so that I can restock if inventory is low NOTE: Need inventory relation to represent owned copies of issues, manager should be able to query entire list.
US2	As a manager, I want to modify the inventory so that I can reflect new purchases by the store. NOTE: Manager needs the capability to modify the inventory to alter table entries.
US3	As a manager, I want to modify the catalog so that I can reflect new books that come out. NOTE: Manager should be able to update and insert into catalog
US4	As a manager, I want to view subscriptions so that I know how many books to order. NOTE: Manager should be able to query subscriptions to get an aggregation of subscriptions
US5	As a manager, I want to view the users* so that I can manage and contact employees and Subscribers. NOTE: Manager should be able to query subscriptions joined on users to see which users are subscribed to which books. Manager should be able to mail employees but not subscribers.

US6	As an Employee, I want to interface with the catalog and inventory to assist users*. NOTE: Employee can assist both free users and subscribers
US7	As a User* , I want to browse the catalog so that I can see what books are available. NOTE: User* refers to free user and subscriber who can both query the entire catalog to view issues and issue attributes
US8	As an Employee, I want to record multiple single issues in a transaction so that inventory can be updated accordingly. NOTE: Employee can record transactions with users and can record multiple purchases per transaction
US9	As a Free User, I want to become a Subscriber so that I can create a pull list. NOTE: Free user should be able to change their status from free user to subscriber within the database
US10	As a Subscriber, I want to add subscriptions to my pull list so that I can reserve books every month that I want. NOTE: Subscriber should be able to update and insert into their pull list
US11	As a Subscriber, I want to add an end date to a subscription in my pull list so that I do not have to cancel it myself NOTE: A start and end date will be kept track, if user elects to not select an end date that field will be left NULL
US12	As a Subscriber, I want to see which issues a trade paperback has so that I can know whether or not I want it. NOTE: Trade paperback is a collection of single issues, so trade should be related to issues

US13	As a Subscriber, I want to to see the content of the omnibus so that I know whether or not I want to it. NOTE: Omnibus is a large collection of single issues.
US14	As a Subscriber, I want to browse the catalog by series so that I can find the books that belong to a particular series. NOTE: This is a filtered view of the Database which filters by series
US15	As a Subscriber, I want to browse the catalog by publisher so that I can find books published by publishers I like. NOTE: This is a filtered view of the Database which filters by publisher
US16	As a Subscriber, I want to browse the catalog by storyline so that I can see which books I want to buy. NOTE: This is a filtered view of the Database which filters by storyline
US17	As a Subscriber, I want to browse the catalog by character so that I can find books which have characters I like.
US18	As a Subscriber, I want to browse the catalog by writer so that I can find books written by writers I like.
US19	As a Subscriber, I want to browse the catalog by pencilist so that I can find books drawn by pencilist I like.
US20	As a Subscriber, I want to browse the catalog by inker so that I can find books inked by inkers I like.
US21	As a Subscriber, I want to browse the catalog by colorist so that I can find books colored by the colorist I like.

CONCEPTUAL DESIGN

Include your complete updated conceptual design here. Use the format shown below.

Entity: **Employee**
- Attributes:
  - name [composite]
    - first_name
    - last_name
  - email
  - phone_number [multi-value]
  - address [composite]
    - street_address
    - city
    - state
    - zip
    - country

Entity: **Manager**
- Attributes:
  - name [composite]
    - first_name
    - last_name
  - email
  - phone_number [multi-value]
  - address [composite]
    - street_address
    - city
    - state
    - zip
    - country

Entity: **Subscriber**
- Attributes:
  - name [composite]

first_name
last_name
email
phone_number [multi-value]

Entity: **Single Issue**
Attributes:
issue_number
price
availability_date
writer [multi-valued]
pencilist [multi-valued]
colorist [multi-valued]
inker [multi-valued]
genre [multi-valued]
quantity
availability [derived]

Entity: **Series**
Attributes:
series_name
publisher
total_issues [derived]

Entity: **Storyline**
Attributes:
storyline_name

Entity: **Transaction**
Attributes:
transaction_items [multi-valued]
total_price [derived]
num_items [derived]
transaction_date
subscriber
employee

Entity: **Transaction Item**

Attributes:
- transaction
- issue_name
- quantity
- price [derived]

Entity: **Subscription**
Attributes:
- series
- pull_list
- start_date
- end_date

Entity: **Pull List**
Attributes:
- subscriber
- subscription_item [multi-value]

Entity: **Trade**
Attributes:
- trade_name
- price
- writer [multi-valued]
- pencilist [multi-valued]
- colorist [multi-valued]
- inker [multi-valued]
- genre [multi-valued]
- avalability_date
- quantity
- availability [derived]

Entity: **Omnibus**
Attributes:
- omnibus_name
- price
- writer [multi-valued]
- pencilist [multi-valued]
- colorist [multi-valued]
- inker [multi-valued]

genre [multi-valued]
avalability_date
quantity
availability [derived]

Relationship: A **Single Issue** is part of a **Series**
Cardinality: Many to One
Participation:
Single Issue has partial participation
Series has total participation

Relationship: A **Subscriber** owns a **Pull List**
Cardinality: One to One
Participation:
Subscriber has total participation
Pull List has total participation

Relationship: A **Pull List** is made up of **Subscriptions**
Cardinality: One to Many
Participation:
Pull List has partial participation
Subscription has total participation

Relationship: A **Subscription** is made up of a **Series**
Cardinality: Many to One
Participation:
Subscription has total participation
Series has partial participation

Relationship: An **Employee** facilitates a **Transaction**
Cardinality: One to Many
Participation:
Employee has partial participation
Transaction has total participation

Relationship: A **Subscriber** makes a **Transaction**
Cardinality: One to Many
Participation:

Subscriber has partial participation
Transaction has partial participation

Relationship: A **Transaction** is made up of **Transaction Items**
Cardinality: One to Many
Participation:
Transaction has total participation
Transaction Item has total participation

Relationship: A **Transaction Item** has an **Issue**
Cardinality: Many to One
Participation:
Transaction Item has partial participation
Issue has partial participation

Relationship: A **Transaction Item** has a **Trade**
Cardinality: Many to One
Participation:
Transaction Item has partial participation
Trade has partial participation

Relationship: A **Transaction Item** has an **Omnibus**
Cardinality: Many to One
Participation:
Transaction Item has partial participation
Omnibus has partial participation

Relationship: A **Storyline** has an **Issue**
Cardinality: Many to Many
Participation:
Storyline has total participation
Issue has partial participation

LOGICAL DESIGN WITH NORMAL FORM IDENTIFICATION

Include your complete updated logical design here. Use the format shown below.

LOGICAL DESIGN

Table: **Employee**
Columns:

- employee_id
- first_name
- last_name
- email
- mobile_phone_number
- home_phone_number
- street_address
- city
- country
- is_manager
- zip_code

Highest normalization level: 4NF

Justification (if needed): Managers have the same attributes as Employees because Manager is a type of Employee, so they could be combined and to specify that an Employee is a Manager, there is a boolean attribute: is_manager. For the multivalued phone number, it was split into two different phone numbers: mobile and home.

Table: **Subscriber**
Columns:

- subscriber_id
- first_name
- last_name
- email
- mobile_phone_number
- home_phone_number

Highest normalization level: 4NF

Justification (if needed): Because Subscriber to Pull List was a one-to-one relationship and Pull List did not have any fields besides for the relationship to subscriptions, Pull List does not need to be represented in the relation. For the multivalued phone number, it was split into two different phone numbers: mobile and home.

Table: **Product**
Columns:
- product_id
- product_name
- price
- quantity
- availability_date

Highest normalization level: 4NF

Table: **Issue**
Columns:
- issue_id
- issue_number
- product_id [foreign key; references **product_id** of **Product**]
- series_id [foreign key; references **series_id** of **Series**]

Highest normalization level: 4NF

Table: **Collection**
Columns:
- collection_id
- is_omnibus
- product_id [foreign key; references **product_id** of **Product**]

Highest normalization level: 4NF

Table: **CollectionIssue**
Columns:

collection_id [foreign key; references **collection_id** of **Collection**]
issue_id [foreign key; references **issue_id** of **Issue**]

Highest normalization level: 4NF

Table: **Genre**
Columns:
genre_id
genre_name

Highest normalization level: 4NF

Table: **Writer**
Columns:
writer_id
writer_last_name
writer_first_name

Highest normalization level: 4NF

Table: **Pencilist**
Columns:
pencilist_id
pencilist_last_name
pencilist_first_name

Highest normalization level: 4NF

Table: **Inker**
Columns:
inker_id
inker_last_name
inker_first_name

Highest normalization level: 4NF

Table: **Colorist**
Columns:

colorist_id
colorist_last_name
colorist_first_name

Highest normalization level: 4NF

Table: **IssueWriter**
Columns:

issue_id [foreign key; references **issue_id** of **Issue**]
writer_id [foreign key; references **writer_id** of **Writer**]

Highest normalization level: 4NF

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Writer.

Table: **IssuePencilist**
Columns:

issue_id[foreign key; references **issue_id** of **issue**]
pencilist_id[foreign key; references **pencilist_id** of **Pencilist**]

Highest normalization level: 4NF

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Pencilist.

Table: **IssueInker**
Columns:

issue_id [foreign key; references **issue_id** of **Issue**]
inker_id [foreign key; references **inker_id** of **Inker**]

Highest normalization level: 4NF

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Inker.

Table: **IssueColorist**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
colorist_id [foreign key; references **colorist_id** of **Colorist**]

Highest normalization level: 4NF

Table: **IssueGenre**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
genre_id [foreign key; references **genre_id** of **Genre**]

Highest normalization level: 4NF

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Genre.

Table: **Series**
Columns:
series_id
publisher_id [foreign key; references **publisher_id** of **Publisher**]
series_name

Highest normalization level: 4NF

Table: **Publisher**
Columns:
publisher_id
publisher_name
Justification: To keep the relation in at least 3NF, Publisher Name needed to be extracted to its own relation

Highest normalization level: 4NF

Table: **Storyline**
Columns:
storyline_id
storyline_name

Highest normalization level: 4NF

Table: **StorylineIssue**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
storyline_id [foreign key; references **storyline_id** of **Storyline**]

Highest normalization level: 4NF

Table: **Transaction**
Columns:
transaction_id
employee_id [foreign key; references **employee_id** of **Employee**]
subscriber_id [foreign key; references **subscriber_id** of **Subscriber**]
transaction_date

Highest normalization level: 4NF

Table: **TransactionItem**
Columns:
transaction_item_id
quantity
product_id [foreign key; references **product_id** of **Product**]
transaction_id [foreign key; references **transaction_id** of **Transaction**]

Highest normalization level: 4NF

Table: **Subscription**
Columns:

subscription_id
start_date
end_date
subscriber_id [foreign key; references **subscriber_id** of **Subscriber**]
series_id [foreign key; references **series_id** of **Series**]

Highest normalization level: 4NF

SQL QUERIES

List at least **three** SQL queries that perform data retrievals relevant to the features chosen in the current sprint. For each query, paste a **screenshot** of the output, as shown through your user interface.

Copy down other updated queries

All omnibuses with the involved series

```
  SELECT DISTINCT
        p.product_name 'Omnibus'
   ,GROUP_CONCAT(DISTINCT s.series_name SEPARATOR ', ') 'Series Involved'
   ,p.price 'Price'
   ,p.quantity 'Quantity'
   ,p.availability_date 'Availability Date'
FROM Collection c
        INNER JOIN Product p ON p.product_id = c.product_id
        INNER JOIN CollectionIssue ci ON ci.collection_id = c.collection_id
        INNER JOIN Issue i ON ci.issue_id = i.issue_id
        INNER JOIN Series s ON s.series_id = i.series_id
WHERE c.is_omnibus = 1
GROUP BY c.collection_id
;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Omnibus	Series Involved	Price	Quantity	Availability Date
Infinite Crisis Omnibus	Countdown to Infinite Crisis, Day of Vengeance, Infinite Crisis, Rann-Thanagar, The OMAC Project, Villains United	79.99	1	2008-06-14

Comic Book Database UI

product_name	Series Involved	price	quantity	availability_date
Infinite Crisis Omnibus	Countdown to Infinite Crisis, Day of Vengeance, Infin...	79.99	1	2008-06-14

Storyline bundle with total price at 10% off

```
SELECT
        s.storyline_name 'Storyline'
   ,GROUP_CONCAT(i.issue_number SEPARATOR ', ') 'Issue(s)'
        ,FORMAT(SUM(p.price),2) 'Regular Price'
   ,FORMAT(SUM((p.price) - (p.price * .10)),2) '10% Off Storyline-Bundle'
FROM Storyline s
        INNER JOIN StorylineIssue si USING (storyline_id)
        INNER JOIN Issue i ON si.issue_id = i.issue_id
        INNER JOIN Product p ON i.product_id = p.product_id
```

```
GROUP BY s.storyline_id
;
```

Storyline	Issue(s)	Regular Price	10% Off Storyline-Bundle
Day of Vengeance	1, 2, 3, 4, 5, 6	11.94	10.75
Rann/Thanagar War	1, 2, 3, 4, 5, 6	11.94	10.75
The OMAC Project	1, 2, 3, 4, 5, 6	11.94	10.75
Villains United	1, 2, 3, 4, 5, 6	11.94	10.75
Infinite Crisis	1, 2, 3, 4, 5, 6, 7	13.93	12.54

Comic Book Database UI

storyline_name	Issue(s)	Regular Price	10% Off Storyline-Bundle
Day of Vengeance	1, 2, 3, 4, 5, 6	11.94	10.75
Rann/Thanagar War	1, 2, 3, 4, 5, 6	11.94	10.75
The OMAC Project	1, 2, 3, 4, 5, 6	11.94	10.75
Villains United	1, 2, 3, 4, 5, 6	11.94	10.75
Infinite Crisis	1, 2, 3, 4, 5, 6, 7	13.93	12.54

Number of issues each storyline has:

```
SELECT
        stl.storyline_name
   ,COUNT(i.issue_id) AS number_of_issues
   ,GROUP_CONCAT(DISTINCT CONCAT_WS(' ', w.writer_first_name, w.writer_last_name)
SEPARATOR ', ') Writers
   ,GROUP_CONCAT(DISTINCT CONCAT_WS(' ', p.pencilist_first_name,
p.pencilist_last_name) SEPARATOR ', ') Pencilists
FROM Storyline stl
        INNER JOIN StorylineIssue stli ON stl.storyline_id = stli.storyline_id
        INNER JOIN Issue i ON stli.issue_id = i.issue_id
   INNER JOIN IssueWriter iw ON i.issue_id = iw.issue_id
   INNER JOIN Writer w ON iw.writer_id = w.writer_id
   INNER JOIN IssuePencilist ip ON i.issue_id = ip.issue_id
   INNER JOIN Pencilist p ON ip.pencilist_id = p.pencilist_id
GROUP BY i.series_id
;
```

storyline_name	number_of_issues	Writers	Pencilists
Day of Vengeance	6	Bill Willingham	Justiniano, Ron Wagner
Rann/Thanagar War	6	Dave Gibbons	Ivan Reis
Villains United	6	Gail Simone	Dale Eaglesham, Val Semeiks
The OMAC Project	6	Peter Tomasi	Jorge Jimenez
Infinite Crisis	20	Geoff Johns	Andy Lanning, George Perez, Ivan Reis, Jerry Ordway, Joe Bennett, Phil Jimenez

Comic Book Database UI

storyline_name	number_of_issues	Writers	Pencilists
Day of Vengeance	6	Bill Willingham	Justiniano, Ron Wagner
Rann/Thanagar War	6	Dave Gibbons	Ivan Reis
Villains United	6	Gail Simone	Dale Eaglesham, Val Semeiks
The OMAC Project	6	Peter Tomasi	Jorge Jimenez
Infinite Crisis	20	Geoff Johns	Andy Lanning, George Perez, Ivan Reis, Jerry Ordway, Joe Bennett...

Updated Transaction Report Query from Sprint 1:

```
SELECT
        t.transaction_id
   ,CONCAT(e.first_name, ' ', e.last_name) EmployeeName
   ,IFNULL(CONCAT(s.first_name, ' ', s.last_name), 'Not a Subscriber') SubscriberName
   ,SUM(tir.quantity) 'Number of Items'
   ,GROUP_CONCAT(tir.transaction_item_record SEPARATOR ', ') 'Transaction Description'
   ,SUM(tir.Price_Per_Item) AS 'Total Price'
   ,t.transaction_date
FROM
        Transaction t
   INNER JOIN Employee e USING (employee_id)
   INNER JOIN Subscriber s USING (subscriber_id)
   INNER JOIN (
                SELECT
                        CONCAT(ti.quantity, ' copies of ', p.product_name)
transaction_item_record
        ,transaction_id
        ,ti.quantity
        ,ROUND(ti.quantity * price, 2) AS Price_Per_Item
                FROM
                        Transaction st
                        INNER JOIN TransactionItem ti USING (transaction_id)
        INNER JOIN Product p USING (product_id)
                ) tir ON t.transaction_id = tir.transaction_id
GROUP BY t.transaction_id
ORDER BY t.transaction_date DESC
;
```

Sprint 3

REQUIREMENTS

List your updated user stories in decreasing order of priority. Highlight the stories that were completed in Sprint 1 in one color. Highlight the stories that were completed in Sprint 2 in a different color. Highlight the updated/new stories chosen for Sprint 3, if any, in a third color. *There is no need to explicitly show your story refinement process.* Use the format shown below.

Story ID	**Story description**
US1	As a manager, I want to view the inventory so that I can restock if inventory is low NOTE: Need inventory relation to represent owned copies of issues, manager should be able to query entire list.
US2	As a manager, I want to modify the inventory so that I can reflect new purchases by the store. NOTE: Manager needs the capability to modify the inventory to alter table entries.
US3	As a manager, I want to modify the catalog so that I can reflect new books that come out. NOTE: Manager should be able to update and insert into catalog
US4	As a manager, I want to view subscriptions so that I know how many books to order. NOTE: Manager should be able to query subscriptions to get an aggregation of subscriptions
US5	As a manager, I want to view the users* so that I can manage and contact employees and Subscribers. NOTE: Manager should be able to query subscriptions joined on users to see which users are subscribed to which books. Manager should be able to mail employees

	but not subscribers.
US6	As an Employee, I want to interface with the catalog and inventory to assist users*. NOTE: Employee can assist both free users and subscribers
US7	As a User* , I want to browse the catalog so that I can see what books are available. NOTE: User* refers to free user and subscriber who can both query the entire catalog to view issues and issue attributes
US8	As an Employee, I want to record multiple single issues in a transaction so that inventory can be updated accordingly. NOTE: Employee can record transactions with users and can record multiple purchases per transaction
US9	As a Free User, I want to become a Subscriber so that I can create a pull list. NOTE: Free user should be able to change their status from free user to subscriber within the database
US10	As a Subscriber, I want to add subscriptions to my pull list so that I can reserve books every month that I want. NOTE: Subscriber should be able to update and insert into their pull list
US11	As a Subscriber, I want to add an end date to a subscription in my pull list so that I do not have to cancel it myself NOTE: A start and end date will be kept track, if user elects to not select an end date that field will be left NULL
US12	As a Subscriber, I want to see which issues a trade paperback has so that I can know whether or not I want it. NOTE: Trade paperback is a collection of single issues, so

	trade should be related to issues
US13	As a Subscriber, I want to to see the content of the omnibus so that I know whether or not I want to it. NOTE: Omnibus is a large collection of single issues.
US14	As a Subscriber, I want to browse the catalog by series so that I can find the books that belong to a particular series. NOTE: This is a filtered view of the Database which filters by series
US15	As a Subscriber, I want to browse the catalog by publisher so that I can find books published by publishers I like. NOTE: This is a filtered view of the Database which filters by publisher
US16	As a Subscriber, I want to browse the catalog by storyline so that I can see which books I want to buy. NOTE: This is a filtered view of the Database which filters by storyline
US17	As a Subscriber, I want to browse the catalog by character so that I can find books which have characters I like.
US18	As a Subscriber, I want to browse the catalog by writer so that I can find books written by writers I like.
US19	As a Subscriber, I want to browse the catalog by pencilist so that I can find books drawn by pencilist I like.
US20	As a Subscriber, I want to browse the catalog by inker so that I can find books inked by inkers I like.
US21	As a Subscriber, I want to browse the catalog by colorist so that I can find books colored by the colorist I like.

CONCEPTUAL DESIGN

Include your complete updated conceptual design here. Use the format shown below.

Entity: **Employee**
- Attributes:
  - name [composite]
    - first_name
    - last_name
  - email
  - phone_number [multi-value]
  - address [composite]
    - street_address
    - city
    - state
    - zip
    - country

Entity: **Manager**
Attributes:
- name [composite]
  - first_name
  - last_name
- email
- phone_number [multi-value]
- address [composite]
  - street_address
  - city
  - state
  - zip
  - country

Entity: **Subscriber**
Attributes:
- name [composite]
  - first_name
  - last_name

email
phone_number [multi-value]

Entity: **Single Issue**
Attributes:
issue_number
price
availability_date
writer [multi-valued]
pencilist [multi-valued]
colorist [multi-valued]
inker [multi-valued]
genre [multi-valued]
quantity
availability [derived]

Entity: **Series**
Attributes:
series_name
publisher
total_issues [derived]

Entity: **Storyline**
Attributes:
storyline_name

Entity: **Trade**
Attributes:
trade_name
price
writer [multi-valued]
pencilist [multi-valued]
colorist [multi-valued]
inker [multi-valued]
genre [multi-valued]
avalability_date
quantity
availability [derived]

Entity: **Omnibus**
Attributes:
- omnibus_name
- price
- writer [multi-valued]
- pencilist [multi-valued]
- colorist [multi-valued]
- inker [multi-valued]
- genre [multi-valued]
- avalability_date
- quantity
- availability [derived]

Entity: **Character**
Attributes:
- character_name

Relationship: A **Single Issue** is part of a **Series**
Cardinality: Many to One
Participation:
- Single Issue has partial participation
- Series has total participation

Relationship: A **Single Issue** features a **Character**
Cardinality: Many to Many
Participation:
- Single issue has partial participation
- Character has partial participation

Relationship: A **Subscriber** owns a **Pull List**
Cardinality: One to One
Participation:
- Subscriber has total participation
- Pull List has total participation

Relationship: A **Subscription** is made up of a **Series**
Cardinality: Many to One
Participation:

Subscription has total participation
Series has partial participation

Relationship: An **Employee** facilitates a **Transaction**
Cardinality: One to Many
Participation:
Employee has partial participation
Transaction has total participation

Relationship: A **Subscriber** makes a **Transaction**
Cardinality: One to Many
Participation:
Subscriber has partial participation
Transaction has partial participation

Relationship: A **Storyline** has an **Issue**
Cardinality: Many to Many
Participation:
Storyline has total participation
Issue has partial participation

LOGICAL DESIGN WITH HIGHEST NORMAL FORMS AND INDEXES

Table: **Employee**
Columns:
employee_id
first_name
last_name
email
mobile_phone_number
home_phone_number
street_address
city
country
is_manager
zip_code

Highest normalization level: 4NF

Indexes:

Index #: 1 Clustered

Columns: employee_id

Justification: employee_id is the primary key hence will be indexed by default.

Index #: 2 non-clustered

Columns: last_name, first_name

Justification: {last_name, first_name} The 'Full Name' of an Employee is commonly used for numerous queries and views where employee information is relevant.

Justification (if needed): Managers have the same attributes as Employees because Manager is a type of Employee, so they could be combined and to specify that an Employee is a Manager, there is a boolean attribute: is_manager. For the multivalued phone number, it was split into two different phone numbers: mobile and home.

Table: **Subscriber**

Columns:

<u>subscriber_id</u>

first_name

last_name

email

mobile_phone_number

home_phone_number

Highest normalization level: 4NF

Indexes:

Index #: 1 Clustered

Columns: subscriber_id

Justification: subscriber_id is the primary key hence will be indexed by default.

Index #: 2 non-clustered

Columns: last_name, first_name

Justification: {last_name, first_name} The 'Full Name' of an Subscriber is commonly used for numerous queries and views where subscriber information is relevant.

Justification (if needed): Because Subscriber to Pull List was a one-to-one relationship and Pull List did not have any fields besides for the relationship to subscriptions, Pull List does not need to be represented in the relation. For the multivalued phone number, it was split into two different phone numbers: mobile and home.

Table: **Product**
Columns:
product_id
product_name
price
quantity
availability_date

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: product_id
Justification: product_id is the primary key hence will be indexed by default.

Index #2: non-clustered
Columns: product_name
Justification: There are often searches for a particular product based on the name

Index #3: nonclustered
Columns: price
Justification: The products are often ordered by issue

Table: **Issue**

Columns:

issue_id
issue_number
product_id [foreign key; references **product_id** of **Product**]
series_id [foreign key; references **series_id** of **Series**]

Highest normalization level: 4NF

Indexes:

Index 1: Clustered
Columns: issue_id
Justification: issue_id is the primary key and hence will be indexed by default.

Table: **Collection**
Columns:

collection_id
is_omnibus
product_id [foreign key; references **product_id** of **Product**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered
Columns: collection_id
Justification: collection_id is the primary key and hence will be indexed by default.

Table: **CollectionIssue**
Columns:

issue_id [foreign key; references **issue_id** of **Issue**]
collection_id [foreign key; references **collection_id** of **Collection**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered
Columns: {issue_id,collection_id}
Justification: {issue_id,collection_id} is the primary key and hence will be indexed by default.

Table: **Genre**
Columns:
genre_id
genre_name

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: genre_id
Justification: genre_id is the primary key and hence will be indexed by default.

Table: **Writer**
Columns:
writer_id
writer_last_name
writer_first_name

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: writer_id
Justification: writer_id is the primary key and hence will be indexed by default.

Index #2: non-clustered
Columns: writer_last_name, writer_first_name
Justification: {writer_last_name, writer_first_name} The 'Full Name' of a Writer is commonly used for numerous queries and views where Writer information is relevant.

Table: **Pencilist**
Columns:
pencilist_id
pencilist_last_name
pencilist_first_name

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: pencilist_id
Justification: pencilist_id is the primary key and hence will be indexed by default.

Index #2: non-clustered
Columns: pencilist_last_name, pencilist_first_name
Justification: {pencilist_last_name, pencilist_first_name} The 'Full Name' of a Pencilist is commonly used for numerous queries and views where Pencilist information is relevant.

Table: **Inker**
Columns:
inker_id
inker_last_name
inker_first_name

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: inker_id
Justification: inker_id is the primary key and hence will be indexed by default.

Index #2: non-clustered
Columns: inker_last_name, inker_first_name
Justification: {inker_last_name, inker_first_name} The 'Full Name' of a inker is commonly used for numerous queries and views where Inker information is relevant.

Table: **Colorist**
Columns:
colorist_id
colorist_last_name
colorist_first_name
Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: colorist_id
Justification: colorist_id is the primary key and hence will be indexed by default.

Index #2: non-clustered
Columns: colorist_last_name, colorist_first_name
Justification: {colorist_last_name, colorist_first_name} The 'Full Name' of a Colorist is commonly used for numerous queries and views where Colorist information is relevant.

Table: **Character**
Columns:
character_id
character_name

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: character_id
Justification: character_id is the primary key and hence will be indexed by default

Table: **IssueWriter**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
writer_id [foreign key; references **writer_id** of **Writer**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered

Columns: {issue_id, writer_id}

Justification: {issue_id, writer_id} is the primary key and hence will be indexed by default.

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Writer.

Table: **IssuePencilist**

Columns:

issue_id[foreign key; references **issue_id** of **issue**]

pencilist_id[foreign key; references **pencilist_id** of **Pencilist**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered

Columns: {issue_id,pencilist_id}

Justification: {issue_id, pencilist_id} is the primary key and hence will be indexed by default.

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Pencilist.

Table: **IssueInker**

Columns:

issue_id [foreign key; references **issue_id** of **Issue**]

inker_id [foreign key; references **inker_id** of **Inker**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered

Columns: {issue_id, inker_id}

Justification: {issue_id, inker_id} is the primary key and hence will be indexed by default.

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Inker.

Table: **IssueColorist**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
colorist_id [foreign key; references **colorist_id** of **Colorist**]

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: {issue_id, colorist_id}
Justification: {issue_id, colorist_id} is the primary key and hence will be indexed by default.

Table: **IssueGenre**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
genre_id [foreign key; references **genre_id** of **Genre**]

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: {issue_id,genre_id}
Justification: {issue_id, genre_id} is the primary key and hence will be indexed by default.

Justification (if needed): To keep in 4NF the table is created to maintain the many-to-many relationship that exist between Issue and Genre.

Table: **IssueCharacter**

Columns:

issue_id [foreign key; references **issue_id** of **Issue**]
character_id [foreign key; references **character_id** of **Character**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered
Columns: {issue_id, character_id}
Justification: {issue_id, character_id} is the primary key and hence will be indexed by default.

Table: **Series**
Columns:

series_id
publisher_id [foreign key; references **publisher_id** of **Publisher**]
series_name

Highest normalization level: 4NF

Indexes:

Index #1: Clustered
Columns: series_id
Justification: series_id is the primary key and hence will be indexed by default.

Index #2: non-clustered
Columns: series_name
Justification: The series are often searched by name

Table: **Publisher**
Columns:

publisher_id
publisher_name

Justification: To keep the relation in at least 3NF, Publisher Name needed to be extracted to its own relation

Highest normalization level: 4NF
Indexes:
Index #1: Clustered
Columns: publisher_id
Justification: publisher_id is the primary key and hence will be indexed by default.

Table: **Storyline**
Columns:
storyline_id
storyline_name

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: storyline_id
Justification: storyline_id is the primary key and hence will be indexed by default.

Index #2: non-clustered
Columns: storyline_name
Justification: The storyline is often searched and ordered by name

Table: **StorylineIssue**
Columns:
issue_id [foreign key; references **issue_id** of **Issue**]
storyline_id [foreign key; references **storyline_id** of **Storyline**]

Highest normalization level: 4NF

Indexes:
Index #1: Clustered
Columns: {issue_id, storyline_id}
Justification: {issue_id, storyline_id} is the primary key and hence will be indexed by default.

Table: **Transaction**
Columns:

transaction_id
employee_id [foreign key; references **employee_id** of **Employee**]
subscriber_id [foreign key; references **subscriber_id** of **Subscriber**]
transaction_date

Highest normalization level: 4NF
Indexes:

Index #1: Clustered
Columns: transaction_id
Justification: transaction_id the primary key and hence will be indexed by default.

Table: **TransactionItem**
Columns:

transaction_item_id
quantity
product_id [foreign key; references **product_id** of **Product**]
transaction_id [foreign key; references **transaction_id** of **Transaction**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered
Columns: transaction_item_id
Justification: transaction_item_id the primary key and hence will be indexed by default.

Table: **Subscription**
Columns:

subscription_id
start_date
end_date
subscriber_id [foreign key; references **subscriber_id** of **Subscriber**]

series_id [foreign key; references **series_id** of **Series**]

Highest normalization level: 4NF

Indexes:

Index #1: Clustered

Columns: subscription_id

Justification: subscription_id the primary key and hence will be indexed by default

VIEWS AND STORED PROGRAMS

View: vw_issue_information
Goal: Provides a list of all issues names along with the names of the publishers and all creative talent who worked on the book. This could be used in a list view of the entire catalog.

View: vw_dccomics_subscribers
Goal: Provides a list of subscriber ids, names, and DC Comics Series from Subscribers who are subscribed to DC Comics. This could be used to generate a list of subscribers to know how many books to order in an upcoming month to meet subscriber needs.

View: vw_transactions_report
Goal: Provides a list of transaction information ordered by date including transaction id, employee name, subscriber name, the total number of items in the transaction, details about the quantity of different products involved, the total price, and the date. This could be used for the owner to view financial information.

View: vw_storyline_bundle
Goal: Provides a list of all storylines with a discounted bundle price for each. This could be used to allow customers to see the benefit of buying an entire storyline by displaying the associated prices.

View: vw_storyline_num_issues
Goal: Provides the number of issues for each storyline, as well as, the associated writers and pencilist. This could be used to allow customers to

view the total number of issues, to get a better understanding of the length of each storyline.

View: vw_omnibus_series
Goal: Provides a list of all series involved in the omnibuses which are stored inside the catalog. This could be used to generate a list of omnibuses that contain a particular series that a customer is interested in.

Stored function: CalculateTotalProductPrice
Parameters: prod_id (Product to purchase), quant (number of items to purchase)
{product_id and quantity from Product table}
Goal: Calculate the total price of all products with the particular id. This is accomplished by multiplying the price by the quantity. This information can be used when creating a transaction with the particular product.

Stored procedure: CreateTransaction
Parameters: emp_id (IN), sub_id (IN)
{employee and subscriber id}
Goal: The purpose of this stored procedure is to create an entry in the transaction table. This will be used to facilitate transactions when they occur.

Stored procedure: AddTransactionItem
Parameters: trans_id (IN), prod_id (IN), quant (IN)
{Transaction and product id as well as quantity}
Goal: The purpose of this stored procedure is to create a transaction item for the desired purchase. This will be inserted into the transactionItem table for a particular transaction. Additionally this will update the product table's quantity to reflect the transaction (ie the quantity will be decremented to reflect the total number of products remaining after the purchase is complete).

Trigger: after_issue_insert
Goal: The purpose of this trigger is to add an entry in product once a new issue has been created. For example: If a new series is created each time a new issue is added to the particular series this will be reflected in the product table.