*SQL and Relational Databases, Project Report, 2023*

Project Excelsior

**Florian Kenner - 22208397**

(Florian.Kenner@ucdconnect.ie)

Use a splash page image here [optional]

Use *LaTeX* if you wish but use the general spacing and font/style you find here (1.5 spacing, 12-point font for text, etc.).

Be sure to submit a PDF (not a .DOC file) as your report. Overall, it will be about **25-30 pages**, including diagrams and screenshots. A significant portion of the report should be textual. Do not rely on images to write your report for you.

**Remember**, your project this year concerns a database for a food laboratory. Your database is intended to support the workings of the innovation processes within the laboratory. As such, identify the place of the database in the overall laboratory, and tell us how you would support its operations at the SQL level.

**What to submit**: This report, as a PDF, *and* the necessary SQL files to allow us to examine your database constructs and test your queries.

# Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| Section 1: | Section Heading | ……………………………………………………………….. | 1 |
| Section 2: | Section Heading | ……………………………………………………………….. | 2 |
| Section 3: | Section Heading | ……………………………………………………………….. | 3 |
| Section 4: | Section Heading | ……………………………………………………………….. | 4 |
| Section 5: | Section Heading | ……………………………………………………………….. | 5 |
| Section 6: | Section Heading | ……………………………………………………………….. | 6 |
| Section 7: | Section Heading | ……………………………………………………………….. | 7 |

# List of Figures

|  |  |  |  |
| --- | --- | --- | --- |
| Figure 1: | Figure Caption | ……………………………………………………………….. | 1 |
| Figure 2: | Figure Caption | ……………………………………………………………….. | 2 |
| Figure 3: | Figure Caption | ……………………………………………………………….. | 3 |
| Figure 4: | Figure Caption | ……………………………………………………………….. | 4 |
| Figure 5: | Figure Caption | ……………………………………………………………….. | 5 |
| Figure 6: | Figure Caption | ……………………………………………………………….. | 6 |

# Introduction

The following project embodies my vision for the database of the comic retailer excelsior. The comic book industry relies on physical items, which have so far mostly been displayed on physical storefronts and sold in stores. The tradition and history of this domain must be adapted modern demands, in example online retailing. The heart of each online retailer is its underlying database system, which determines how the sold goods are being viewed, which aspects are emphasized and how all parts of the business come together. The main technical part of my vision is being as close to reality as possible when making design and implementation choices. The project should be lean, yet complex enough to consider the peculiarities and subtleties of comic collecting and selling so that every comic enthusiast would be satisfied with this design. Furthermore, the database should meet all technical criteria to be performant, non-redundant and scalable.

Comic books have become far more than simple entertainment products, which get read once and forgotten afterwards. Comic books have turned into collectables which carry both emotional and financial value. There are single very valuable pieces which got produced only a few times many years ago and even fewer which survived until today. At the same time there are many editions produces and sold today in large quantities. The difficult job of a comic retailer is to satisfy all these different target groups with their different needs and demands. Numerous traits are shared by all comics. These can be differentiated on the abstract level of a comic book batch and the level of each physical copy. On the abstract level, each comic book has a title, publisher, release date and so on. Each physical copy has a condition, which is rated by a uniform evaluation system, a buying and selling price, etc.. The customer must be provided with all relevant information so that he can make an educated buying decision and is satisfied in the long term. The intended application of the project is therefore providing a database which stores all information of the comic books the retailer excelsior sells on which a website can be built.

From a technical perspective, the different abstractions of the data have to be considered when designing the database. It can be expected that the stress on the database will be low at the beginning but may grow over time, but it’s not to be expected to enter the domain of big data. The database stores so far only textual information, meaning no pictures or videos. There won’t be an ingoing stream of telemetry data for example which fills up our database. Still, the database must be able to handle increasing traffic through website visitors and deliver results in a short time to provide a satisfying user experience.

The scope of this project is providing a useful solution for dealing with all the data concerning the stock of excelsior. This means keeping track of the current inventory of comics and providing the additional information an enthusiast needs. This additional information to what series a single comic issue belongs, who published this series, which character appears in a comic book and who worked on a comic book. Also all above mentioned relevant information of a single copy must be stored, such as the buying price, the selling price, the condition and the format, meaning the cover size and material. On top of that, the database should keep track of all customer information such as the name, date of birth and address. These two parts must be brought together to keep track of inventory changes to document previous purchases. The basis for two more functions on possible the way of a customer to buying a comic book should be provided with database foundations, a wish list and a shopping cart. Most customers are familiar with such functionalities from other online retailers and may therefore miss those in the online shop of excelsior if absent. Not part of the scope of this project is keeping track of logistic business information such as the physical storage of comics, the buying process of comics before they appear in our inventory and the shipping process. Also, no information regarding the payment except the prices should be stored.

# Database Plan: A Schematic View

For the database design, an intense and deep dive into the topic is very important to be able to understand the context of the data and how everything comes together. Therefore, the first step was collecting as much information as possible about comic book collecting in general. This concerned the releases of new issues and completely new comic book series, how these get numbered. Another important domain is understanding how comic books and graphic novels are related, which properties they share and what distinguishes them. Furthermore, the grading of collectables plays an important role in pricing comic books. The most important key learnings from the research which influenced the data base design were:

Each comic book has a title, a year of publication and an issue number. In a series of comics, e.g. „Iron Man (2002)“ all issues share the same title and only the issue number changes. Additionally, there might be multiple runs of a series with different content, there might be an “Iron Man (1990)” and an “Iron Man (2002)” which have different contents. The numbering of comics may not be linear, it can be influenced if a character appears in another comic series and the original series may skip several issue numbers, how it comes to the numbering in detail is not important for this project. Graphic novels and comic books are closely related. Graphic novels are usually more extensive than comic books, they contain closed plot. A graphic novel may contain a collection of comic issues, such as “Iron Man (1990) #1-#5”, but graphic novels may also contain unique storylines which never got released as comics. Also, the edition of a graphic novel is very important to consider. There might be several reprints of the same graphic novel and only the first reprint could be of interest for comic enthusiasts. When it comes to the physical specimen of comic books and graphic novels, they share most attributes. There is a grading scale which applies to both which includes a numerical scale from 0.0 to 10.0 and a textual condition. There are numerical bins which result all in the same textual condition, for example comic books with a rating from 9.6 to 9.9 are graded with the condition “Near Mint/Mint” (NM/MT). To get a certain grade, a comic book has to fulfil certain criteria. The grading is done by independent companies and not by our comic book reseller excelsior, therefore we solely have to store this information about our stock.

Diagram

Description automatically generatedThe next step was to collect all requirements for this project. All mentioned features from the project white paper got collected and extended by research on mile high comics. The goal was carrying together, what information an online comic book retailer needed and derive the scope for this project from there. The resulting scope was presented in the first section of this report. The following entity relationship diagram is the core of this project and was designed by following the definitions given in "Database Systems: Design, Implementation, and Management" by Carlos Coronel.

Figure 1: Entity relation diagram

Let’s start by looking at the collectable entity. This entity abstracts the theoretical comic book or graphic novel. This level of abstraction describes all unique comic book issues or graphic novels, not referring to the physical specimen. Each collectable has the attributes title, storyline, year of publication and publisher. The storyline connects the two comic book series “Iron Man (1990)” and “Iron Man (2002)” since both tell the story of Iron man, but besides this share not much more. Each collectable has more traits, which are outsourced and displayed in the entities characters and creators.

The creators entity contains all people who worked on collectables, they possess a first name, a last name and a job type. The job type specifies what exactly their job was in example drawing or writing a story. Each collectable must have at least one creator who worked on it and each creator has worked on n collectables. As a consequence, there can be creators in our database, which have not worked on any collectables we store. This is a conscious decision, so we are able to store information about creators before we have any of their work in our data.

The other property a comic has which is stored in a different table are its character appearances. Each character has a name and a profession. In contrast to the creators, the character name consists only of one token which is implemented since characters might be stored not by referencing their real name, but their nickname such as “superman”. Each collectable has at least one character appearing, and each character can appear in n collectables, behind this lies the same design thought as behind the creators. These two entities are separated from the collectable entity since they can exist without it and are not a fixed part of them, in contrast the title is only part of a collectable and has no self-reliance apart from that. The same thought could be applied to the publisher, but since our scope is only to store his name and he has no further attributes in our database, he does not exist as an independent entity.

Now we move from the theoretical level of the collectable to the concrete physical of the stock. The stock entity represents all physical copies of comic books and graphic novels we sell. As mentioned before, one of their most important attributes is the condition, which consists of a numeric rating and a textual rating. Additionally, each stock item has a format, a buying and a selling price and an edition. The format refers to the size and material of a copy, which might be hard- or softcover, and the cover might be jumbo sized or medium sized and so on. The edition describes in which reprint the stock item was produced. Each stock item must be a collectable, so that all the information which does not belong to the single copy can be presented to the customer. Vice versa, each collectable can be n times in our stock, we might sell multiple copies of a collectable or none at a given time. Besides these traits, a stock item can have a comment in which more specific information is recorded. This information has no further restrictions, it could be about the condition or also tell if it's signed by a celebrity. From the entities collectables and stock items can be concluded, that the database does not store any information about the language of an item we sell. This choice was made based on the assumption, that the title of a collectible already specifies the language. The English comic book version of the amazing spiderman must have a different title than the German one.

All information needed about the goods we sell is can be stored in these entities, the following customer entity which hints at the online domain in which our store exists shows the general interactions our customers have with our web store. Each customer has a first and last name, a date of birth and an email address. Moreover, he has an address which consists of a street and house number, a city, a state, and a zip code. All this information should be self-explanatory, the customer entity is only a compressed abstraction of the real person interacting with our business and exists more for demonstration purposes. This customer may put any number of stock items on his wish list and every stock item can be on any number of wish lists. In the further process, there will be a more distinct separation of a wish list and a shopping cart, but for this model we can treat them both as the same. A customer either wants to buy an item which can be in his virtual shopping cart, his wish list and so on or he has bought an item. This is the second interaction displayed in the diagram. A customer may buy any number of stock items and logically, each stock item can only be bought once by a single customer.

The next step to designing a database concept is transforming these derived entities in an entity relationship diagram with tables, which shows how exactly they are stored in a relational database. From now on I will use the nomenclature to describe entities provided in “Database Design – 2nd Edition” by Adrienne Watt. This includes:

**Kernels**, which are independent entities, that means they don’t rely on another entity to exist.

**Derived entities**, which depend on other tables for their meaning and are used to connect two kernels together

**Characteristic entities**, which provide more information about another table.

Diagram

Description automatically generatedThe following diagram displays this information and was made following the guidelines from the same book as the first diagram:

Figure 2: Entity relation diagram (tables)

Let’s go through this diagram in the same order as the other ER-diagram to see how the previous entities changed and what stayed the same.

The collectable entity is now a kernel, in which all graphic novels and comic books are. Since comic books have an issue number and graphic novels don’t, we need a characteristic entity to define which collectables have issue numbers.

This is the characteristic entity comics. Each entry in the comic table refers to one single collectable and a collectable can be either a comic or not. The appearance of a collectable in this entity is enough to identify a collectable either as comic or as graphic novel. The other attributes didn’t change, now we introduced attribute types which should be so far self-explanatory, if not, there will be further explanation provided in the text. Furthermore, the specific choice of primary keys will be discussed in the next chapter when talking about data normalization.

The storyline got outsourced as well, it can be treated a kernel hence it can exist on its own. The kernel contains only a storyline title and the connection between a collectable and its storyline is drawn in the derived entity “storyline\_mappings”. Each storyline can be in n mappings, while each mapping connects one storyline with one collectable. Every collectable has to be at least part of one storyline.

The creators remain the same with a first and last name, and the derived entity feature\_work connects the kernel of creators and the collectable kernel. The job type is specified in feature\_work since a creator can do different jobs working on the same or different projects. Storing the job type in creators would lead to duplicate entries in the creators kernel except the job type, this solution is much more elegant. Each feature\_work entry connects a single creator with a single collectable, each creator can work on n collectables and each collectable had at least one creator working on it.

The characters table shares the structure of creators. The kernel character defines all characters by their name and their profession, which could be superhero for example. Analogue to the creators, the derived entity character\_appearances connects per entry one character with one graphic novel. Each character can appear in any number of collectables and each collectable has at least one character appearance.

Now we move onto the characteristic entity stock. It falls under this category since a stock item which means a physical comic or graphic novel can’t exist neither in the real world, nor in this relational data base model without getting created as abstract, not yet produced and printed collectable. Therefore, the stock entity just adds more information to a collectable in form one physical copy. Each stock item must be a collectable and each collectable can be any number of times in our stock. Consequentially we don’t need a derived entity to connect these two tables and can simply refer by using a foreign key from each stock item to its abstract collectable above. We introduced the Boolean in\_stock to keep track of the availability of stock items, which may at first sound counter intuitive to the thought, that each item in the stock entity is in stock but makes sense in the long rung when it comes to keeping track of the previously sold items. To avoid redundancy and dependencies in the stock table, each stock item has a foreign key condition\_id which is the earlier in this text explained numerical rating. This foreign key enables us to connect the stock items with their numerical and textual condition, such as the criteria it has to fulfil to be graded with this condition.

This information is stored in the kernel numerical\_conditions, which connects each possible numerical condition with its textual shortform. The numerical condition consists of a decimal from one to ten with one digit after the decimal point, what explains the used data type. Since the longest textual equivalent consists of 5 characters like “NM/MT”, the textual condition is a varchar of 5 characters. Each stock item has a numerical condition and n stock items can fall under any numerical condition.

The characteristic entity condition\_descriptions provides the matching descriptions for each numerical condition. Since we don’t know the exact maximum length of a description, its data type is text. Each numerical condition has a condition description, and each condition description is valid for 1 to 19 numerical descriptions. This is true because of the fact, that the largest bin for the same condition is “Fine / Very Fine” which applies for all stock items rated in between 6.0 and 8.0.

The comment property a stock item could have got outsourced in the characteristic entity comments which to avoid NULL values. This is due to the fact that stock item can have a comment but isn’t forced to, vice versa each comment belongs to one stock item.

The previous customer entity got split up for later on explained normalization reasons into the two kernels Customers and Addresses. The customer kernel stores the customers first and last name, their date of birth and email address.

The addresses kernel stores a street name, a house number, a city, a state and a zip code. This entity can be classified as a kernel since an address can exist on its own without belonging to a person. This is also represented by the cardinalities of these kernels: An address can belong to n customers while each customer must have exactly one address.

Let’s now inspect the interactions of the customers kernel with our stock. As mentioned in scope at the beginning, the database should provide simple standard features a customer expects when online shopping. First of all, there is the derived entity whish list, which lets customer put stock items on their wish list. Each stock item can be on n wish lists while each wish list contains one customer and one stock item. Each customer can have n wish lists.

The next logical step for a customer on the way to making their purchase is putting a stock item in their shopping cart. This relation is mapped by the kernel “shopping\_carts”. Each shopping cart entry maps one stock item to one customer, while each stock item can be in one or zero shopping carts and each customer can have n shopping carts. This differs from the way a customer would see it from the web interface since he has only one shopping cart there with multiple items at once, but this doesn’t come true on the database level and can be easily presented the common way to the customer.

The last step is the finished purchase of a stock item, which is represented in the derived entity “sold\_items”. This entity provides the additional information of a purchase date, but besides that acts in the completely same way as the shopping carts. Each store item can be sold or not be sold and therefore be one or zero times in the sold\_items entity. Each entry consists of exactly one stock item and one customer, meanwhile each customer can have bough n items.

Now it comes together why each sold stock item is kept in the stock entity. This way it can be easily kept track of previously sold items and all their additional information without enlarging the whole data base model by a significant count of tables.

# Database Structure: A Normalized View

Describe the main tables in your database and the role played by each. Show that your database meets the definitions of 1NF, 2NF and 3NF normal forms. Is your database also in BCNF normal form? If so, explain how and why.

The following chapter discusses the normalization of the database. At first, there will be a brief overlook why normalization is useful, then the first three normal forms and boyce codd normal form will be defined and in the last part, it will be proven that our database project fulfils all these requirements.

Data base normalization is a process which works through the normal forms. Each subsequently higher normal form requires that all the conditions by the lower normal forms are fulfilled. As Carlos Coronel and Steven Morris state in "Database Systems: Design, Implementation, and Management", data base normalization helps to eliminate data redundancy by organizing data into smaller tables which also reduces storage space. Furthermore, normalization improves data consistency across the tables improves maintenance since as much data as possible gets outsourced into its own tables and may therefore only be updated in these tables.

For this project, the first three normal forms are required to be fulfilled and the boyce codd normal form. The following definitions where created by combining the given definitions from the books "Database Systems: Design, Implementation, and Management" by Coronel and Morris, such as “Database Design and Relational Theory” by C.J. Date.

**First normal form** (1NF) requires a table that all its attributes are atomic, meaning they can’t be split into smaller attributes. Additionally, there are no duplicate rows (entries) in a table and that all attributes are dependent on a key, which can be a single column or a set of columns. This concludes that the primary key is unique.

**Second normal form** (2NF) demands that a table is in 1NF and that there are no partial dependencies on the primary key, every non key attribute must be dependent on the whole key.

**Third normal form** (3NF) demands a table to be in 2NF and all other dependencies between non key columns must be eliminated, this form of dependencies is called transitive dependency.

**Boyce Codd normal form** (BCNF) all primary key attributes must not be dependent on a non-primary key attribute

As seen in figure two, many surrogate keys meaning synthetically produced primary keys with no occurrence in the real world were introduced to most of the tables. This had most of the time individual reason per entity, but for some entity types, there were general logics followed in this design:

“Database Design – 2nd Edition” von Adrienne watt suggests chapter in 8 to introduce a surrogate key for all derived entity, which "Fundamentals of Database Systems" by Ramez Elmasri and Shamkant B. Navathe advocates aswell on page 298.

For Kernels, the case is not that clear, but it can be said in general that a surrogate key was introduces if a primary key would have to be a composite key of multiple columns which would complicate the mapping in the derived entities. This idea is also supported by “Database Design and Relational Theory” by C.J. Date on page 168.

Surrogate keys were introduced to characteristic entities if the only option was to use all columns as composite primary key.

In general surrogate keys help to fulfil the requirements of many normal forms. When talking about the requirements of 1NF, a surrogate key makes sure that there is a primary key and that every row is unique. 2NF is automatically ensured aswell when introducing a surrogate key, hence it prevents the case, that columns might depend only on a part of the primary key. The requirements of 3NF have to be ensured whether there is a surrogate key or not, but if a table is in 3NF and has a surrogate key, it’s in BCNF since the primary key is not derived and can therefore not be partially dependent on non-key attributes. This argumentation won’t be elaborated each time a table has a surrogate key. If this is the case, it will simply be discussed if all attributes are atomic and if there are no transitive dependencies.

Let’s now take a closer look at the tables and demonstrate why they meet the requirements of BCNF and let’s also keep the usual order. The collectables kernel has a surrogate key, since a composite key and consequently a primary key would not be possible otherwise since there can be multiple collectables sharing the same title and publication year. Each attribute can’t be divided into sub attributes and is therefore atomic. Furthermore, there are no transitive dependencies, this relies on the assumption that the publisher of a comic can change over time and can therefore not be derived by the title and release year.

If a collectable is a comic is stored via its appearance in the Comics table, this table can use the collectable id as primary key, since each collectable appears once at maximum. Both of the ids can not be split into smaller attributes and therefore the table is in BCNF.

The storyline has to be in its own table because it would harm 3NF since its dependent on the title. The storyline table is self-explanatory in BCNF hence it contains only a single attribute.

The table “storyline\_mappings" is a derived entity using a surrogate key. The foreign keys in it are atomic and there are no transitive dependencies. There can be multiple mappings sharing a storyline with different collectables and vice versa. Therefore, it’s in BCNF.

The character kernel uses a surrogate key as well, the character name can’t be divided into multiple parts, in our view of comics and graphic novels, it functions only as a whole. Furthermore, the profession is not dependent on the character name since there could be a character having multiple professions, as example Superman could be a superhero as well as an alien, also this table is in BCNF.

The same goes for the “creators” kernel, but here the name can be divided into a first and a last name which was done. Also, there can be multiple creators sharing the same first and or last name and so, there is no functional dependency between these attributes. That’s also the reason for the introduction of a surrogate key which ensures that the entity meets the requirements of BCNF.

The “character\_appearances“ entity undergoes as the same logic as the “storyline\_mappings” which can be applied to all the derived entities which do not provide more information than the two foreign keys they contain.

The derived entity “feature\_work” is the other rubric of a derived entity in this case, which provides additional information per entry, in this case the job type. This additional information does not contain on the collectable, nor on the creator it connects since a creator can work on the same collectable for example as designer and as story writer. As conclusion, this table is in BCNF as well.

The characteristic entity stock is the biggest and the most complex to normalize. It has a surrogate key because there could be copies of the same collectable in our database which share all attributes and would be therefore not uniquely identifiable. All attributes are atomic and there are no transitive dependencies. The thought may arise that the buying price determines the selling price but since our comic retailer doesn’t have a constant markup which is applied to all items we sell, this logic doesn’t apply as well, and the table is in BCNF.

The numerical conditions kernel is the first kernel which doesn’t need a surrogate key, the attribute “condition\_id” is a super key since each numeric condition does only have one textual equivalent. This textual equivalent is unique, it’s a short form for a longer term, the condition “NM/MT” can not be distributed into “NM” and “MT” since the meaning comes from combining these two and the table meets the criteria to be in BCNF.

The characteristic entity “condition\_descriptions” disposes of a super key as well, the previously mentioned textual shortform of a condition. Hence the condition description is atomic, this entity is in BCNF.

In contrary, the “comments” does provide the foreign key “stock\_id” which can also be used as primary key because each stock item can only have one comment. Both the id and the comment are atomic, and the table is in BCNF.

The customers kernel has a surrogate key as well, since the attributes first and last name, dob and e mail address would be needed to uniquely identify a customer. Which would result in a very large primary key. This bases on the assumption that there can be multiple customer accounts in our database sharing one email address, this could be through a function which lets children join their parents account without providing a new email address, but they would still enter their name and so on. The address got outsourced because there can be multiple customers living in the same building which would result in redundancy. We assume that a zip code does not identify a specific building since in countries apart from Ireland, it only refers to a city. Furthermore bases this design on the assumption that there could be the same zip code for different cities in different countries. Therefore, we need a surrogate key and there are no partial dependencies. Both customer and address are in BCNF.

The derived entities whish lists and shopping carts follow the same principle as the storyline mappings and the character appearances. They are in BCNF for the same reasons as the other tables, since all four of them share the same structure and the same cardinalities towards all other tables they connect.

The derived entity sold items can be compared in its being to the comments entity, hence both of them refer to a stock item which can only be contained only a single time. Therefore, the stock id can here be used as super key as well, this means simply that each stock item can be sold only once and so the customer and the date are dependent on the item. Furthermore, all attributes are atomic and the customer alone doesn’t set the date and vice versa, therefore this table is in BCNF.

1. **Database Views**

As the title indicates, the views will be discussed in the in the following section. Each view will get mentioned, justified, and explained.

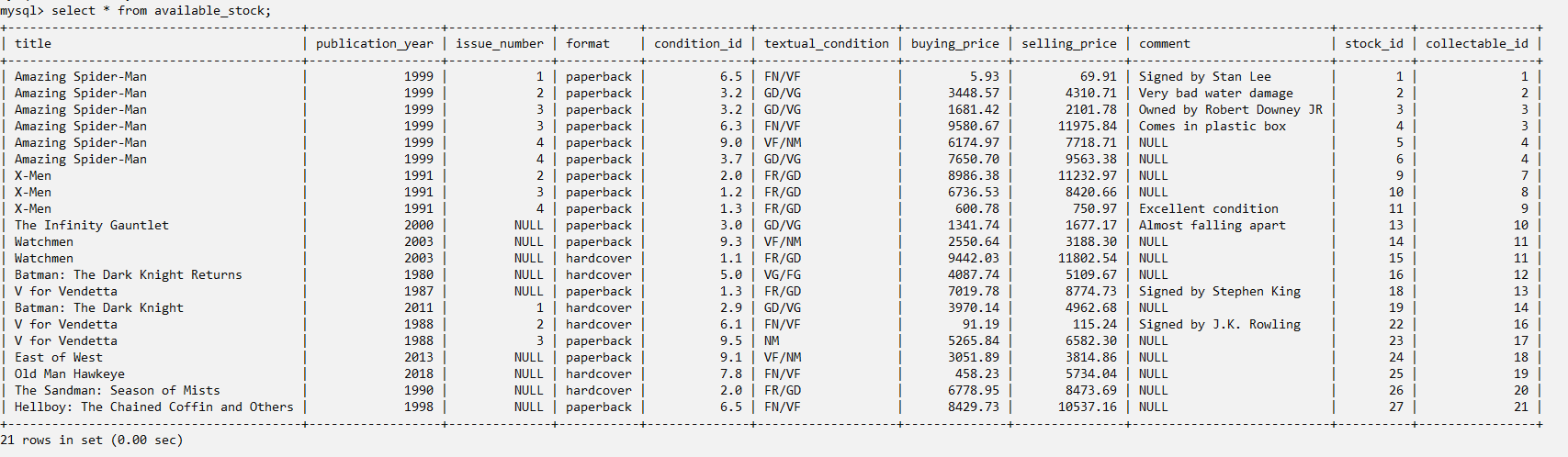
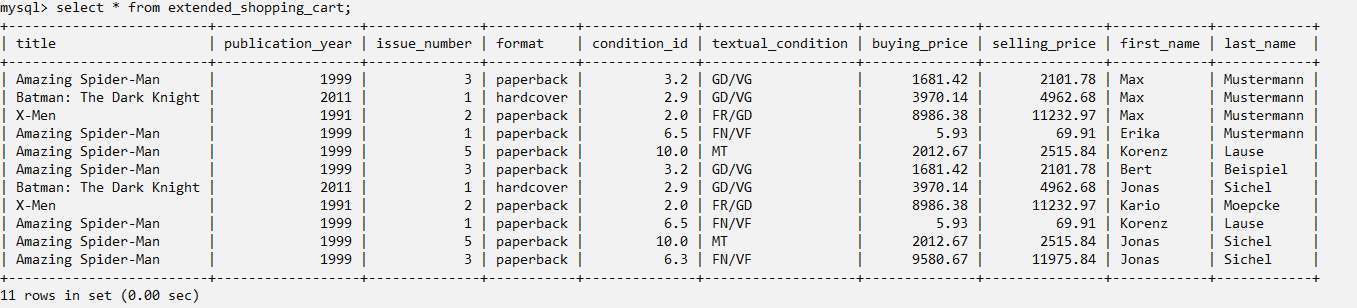
The first logical step is to puzzle all the information together we distributed over several tables in the normalization process, which is what the “available\_stock” view does. It displays all information which belongs to a physical copy we sell:

Figure 3: View “available\_stock”

We can see all items in stock, which is useful on its on to get an overview for us the operators of excelsior as well as our customers. Additionally this view is the technical foundation for all further querying of our stock, if a customer might want to see all spiderman comics which are for sale, this is the starting point. This view was created by joining the stock, the collectables, and the conditions table. This provides the title, publication year, format, condition, buying and the selling price of a stock item. Since we also want to know the issues of the comics in stock, we left join the comics table so that the items we sell without issue numbers (graphic novels) are also displayed. The same goes for the comments table. Lastly, we filter to only display the items which are actually in stock.

The next view shows us how many customers are very close to buying a specific copy, this can be a valuable information when keeping track of highly demanded comics. In these days, many online retailers face huge demands for certain collectables. For example, when new really rare Nike shoes get released on the Nike website, there are many people trying to get those. The following view provides us with this information which we can use to design our prices of very special items.

 Figure 4: view “extended\_shopping\_cart”

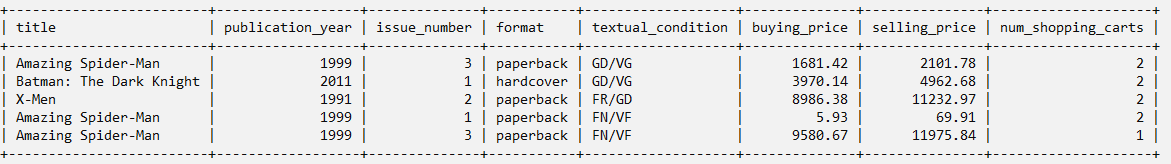
As we can see, in this view is every store item which is in a shopping cart, shown with its necessary information to identify, as well as the customers trying to buy it. This was simply done buy using the previously defined view “available\_stock” joined with the tables shopping cart and customers to get the information, which customer has which stock item in his shopping cart and how the first and last name of these customers are. A possible next step might be grouping the view by each stock item for obtaining the number of shopping carts it is in. This might look like the following:

Figure 5: grouped view “extended\_shopping\_cart”

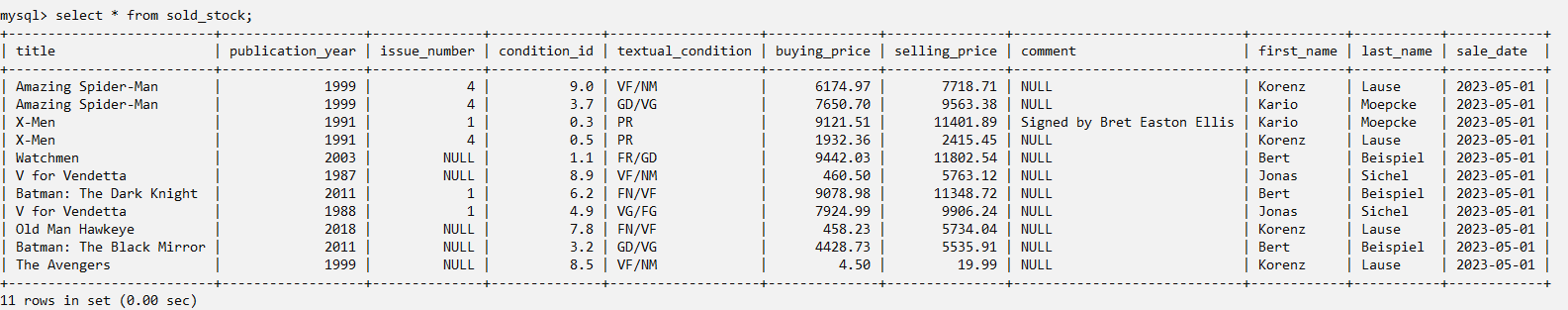
The next view is the counterpart to the " available\_stock “ view, namely the sold stock. This provides us with all the information we might need about previously made purchases.

Figure 6: view “sold\_stock”

It also functions as a basis on top of which can be built. It can be queried for previous customers, collectables, dates and far more. It was created by joining our stock with collectables, comics and comments the same way as used for the available stock view. Furthermore the sold items table was joined to gain the information mapping the customers to the sold items and to filter out the not sold items by using a normal join instead of a left join. In the last step, the customers were joined on the customer id given in the shopping cards to retrieve their first and last name.

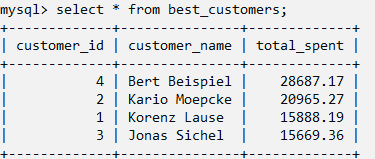
The last view is also of high importance in our business context. It lists all customers with their total amount of money spent in our online shop:

Figure 7: view “best\_customers”

We can see which customer spent the most money in our shop and is therefore the best or most valuable customer. This information is a key insight, in the customers reality they all get the same service and treated equally, but we as a retailer know that this does not correspond to the truth. A customer who spent only a few euros on comics in our store during the last years is of way lower priority to us. A customer in contrast who spends thousands of dollars each month might get premium service. This could mean special offers through reduces pricing or more goodwill when it comes to returning an article he bought in our store. This view was created by joining tables customers, shopping carts and stock and then grouping by customer and summing up the selling prices of the items. The customer names was concatenated into one column since we don’t need it to be atomic and in the last step the values got sorted by the total amount spent in ascending order to show our most valuable customer at the top.

# Procedural Elements

Hence the heart of this project is keeping track of our inventory and all its information, the first function simplifies the process of creating new inventory items. The thought behind it was to give all parameters shown in the extended stock view as well as the storyline of the stock item into a single function which takes care of inserting this data into all the right tables and creating the needed derived entities. Therefore the “create\_stock” function was written. It acts kind of a top-level function which manages the call of the sub level functions which create the storyline, the collectable and a comment. All these functions got out factored into their own functions and procedures since they might be executed on their own for example by just creating a storyline and a collectable without creating a stock item. The procedure which creates the storyline takes the storyline name as single input parameter and inserts it into the storyline table if the storyline name is not null and if it doesn’t already exist in the table. It’s a procedure and not a function because we don’t need it to return anything. The function to create a collectable does the same, it first checks if a collectable already exists and in the case of a given issue number it checks if this collectable exists with the specific issue number, if not it creates the collectable and the entry in the comics table for the issue number. It returns the collectable id either of the already existing collectable or of the newly create entry for further use. The procedure to create a comment works the same as the procedure to create a storyline, it checks whether a comment text was given, then it checks if the stock item already has a comment and if not, it creates the comment. All of these necessary functions to create side information about stock items get managed and called in the create stock function, which calls the create storyline, create collectable and create comment functions and procedures. Furthermore, it creates the stock item itself and an entry in the storyline mappings if it doesn’t exist yet.

Next on we have to procedures which function as antagonists to create and revert discounts. Both of them take a discount percentage and apply / revert it either for all stock items or just a single item if a stock id is provided. If this an idea is given, a check is performed whether the id actually exists, and an error gets signaled which this is not the case. This automates an otherwise often repeated step since discounts are one of the most often used tool to control prices and therefore demand and our stock.

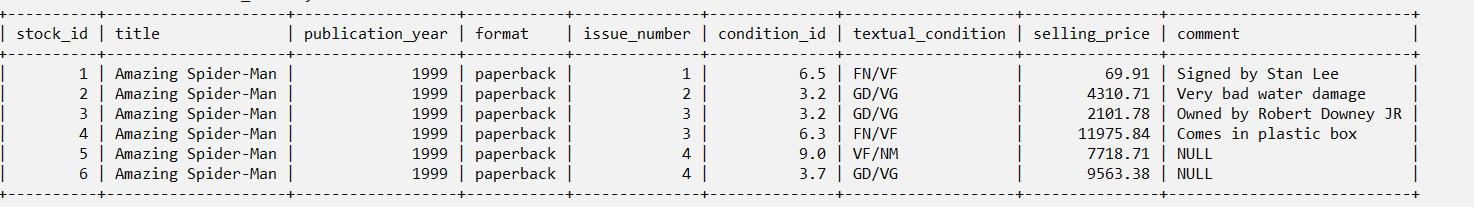
As the before mentioned functions, the following function should provide a simpler way to handle the everyday tasks when managing our database. The create purchase does exactly what its name suggests and checks the same way as the other functions if a stock id and a customer id actually exist in our database. In the last step, the availability of the sold item gets set to false.

The last procedure is both for practical real-world use, as well as demonstration purposes. It functions completely automatically and shows how automation like these can help maintaining the database and making life easier for the employees of excelsior. Every time a new item is sold, it appears in the sold stock view and triggers this procedure. The procedure sets the availability of the item to false and deletes it from every other shopping cart to make sure that it can’t be bought multiple times.

# Example Queries: Your Database In Action

Out first query is a simple yet meaningful way to present how customers can browse our database using filters such as price, condition, storyline and creators of a collectable to find their desired product. This application of filters and ordering is a part of most online shopping experiences, which definitely has to be provided in our online shop as well:

Figure 8: query 1 – filtered stock

In example, this query provides us with all cold collectables on which Stan Lee worked. This was obtained by joining our available stock with the feature work table and setting the filter that the person id had to match Stan Lee’s. Further filtering and sorting of our stock will not be displayed in this project paper since it provides nothing new. As seen, the foundation is there and displaying only the desired information for a customer can easily be done.

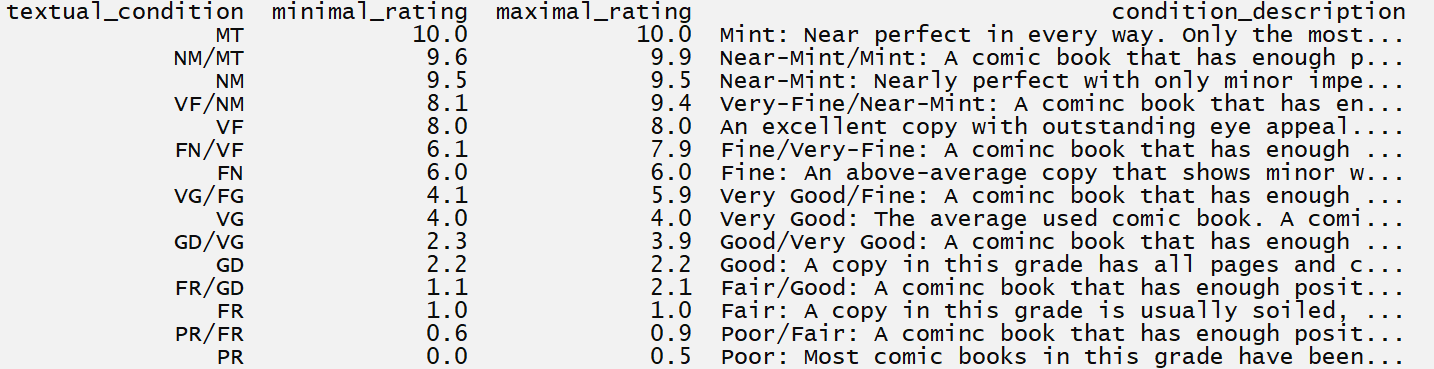
The following query is a brief overview of the interaction of textual condition and numeric condition. It shows each textual condition and in which range of numerical conditions a copy must be to get that rating. Furthermore, provides a detailed condition of the damage an item can have at maximum to be in that range.

Figure 9: query 2 – condition ranges and criteria

This was achieved by joining the conditions with their descriptions, grouping by textual condition and displaying the maximum and minimum numerical rating. The displayed information can be for the fans as well as for the staff when introduced to the topic of collecting comic books and graphic novels. It makes no sense to create this as a view since there won’t be any further querying and specifying of the displayed information.

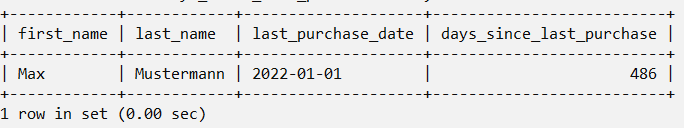
The next query is a demonstration of how useful information to our business could be extracted from the sold items view. It shows all customers whose last purchase was at least a given count of days ago.

Figure 10: query 3 - >n days since last purchase

In combination with the information displayed in the most valuable customers view could this query be used derive information which customer could be contacted. For example, if there is a customer who spent tens of thousands of dollars on collectables via our online shop but didn’t buy anything in the last year, we could write them an email with special offers etc. to win them back as a customer of ours again. The view was created by joining the customers and sold items tables, grouping by customer, and calculating the difference in days since each customer’s last purchase. In the last step, the results were filtered to show only the customers with a time difference greater than x, in our case 250 days.

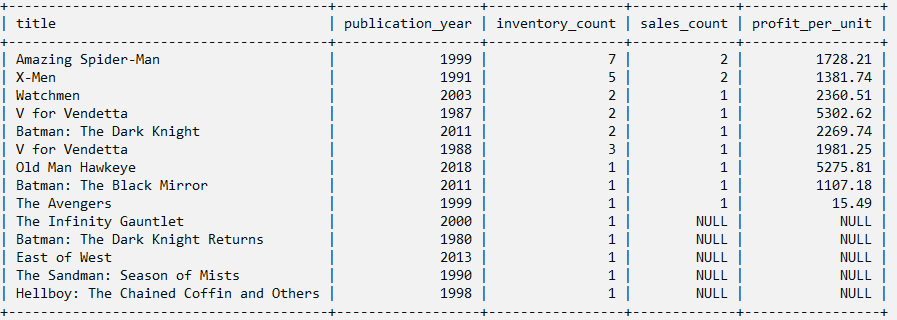
Out last query shows as well how valuable business information could be obtained by querying a previously created view, in this case the sold stock view. Each collectable series is shown, how many different copies of all issues we have in stock, how many we already sold and what the average profit is per sold item:

Figure 11: query 4 – sold and available issues of all series

This can show us which series generate us the most profit and if we should think about buying more stock of a specific series. It could as well be presented to the customer without the inventory and sales count and without the profit per unit to show the most bought series of comic books and graphic novels. The view was created by joining our stock with the sold stock, grouping by title and year and calculating the profit per unit by dividing the difference of the sum of sale prices and the sum of buying prices by the count of sold copies.

# Conclusions

Provide any concluding thoughts here. How might you build on this work for the future? How might your database support future developments?

# Acknowledgements

Name check any person who helped you with this work. Acknowledge that the work is entirely your own, and that every sentence in this report was written by you and you alone. If you wish to quote another person or piece of work, place the quoted work in quotation marks and cite the author inline. Plagiarism is a very serious infraction that must be dealt with severely. Avoid any ambiguity on this point by citing things carefully!

# References

List any bibliographical citations here [for people and work that you quote/cite in the main text of your report; please do include meaningful citations]