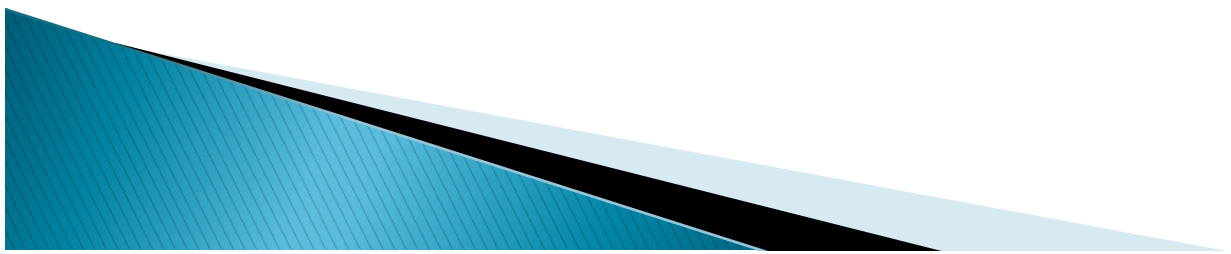
CSCI-UA.60-1 Database Design and Implementation

Relational Database Design

CSCI-UA.60-1
Prof Deena Engel
Department of Computer Science
deena.engel@nyu.edu

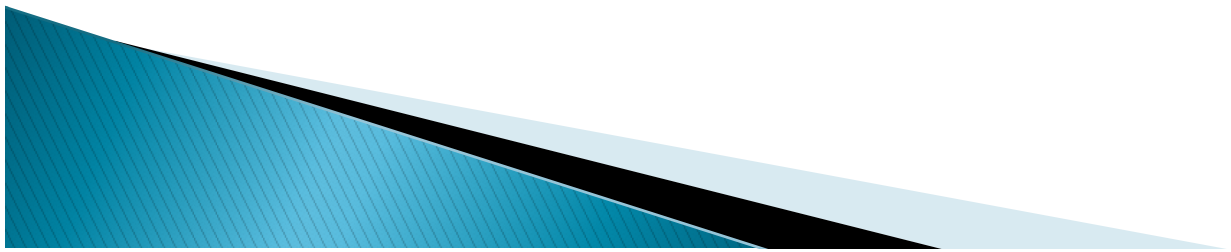
What can go wrong with data? Why the study of database design?

- Mishandled keywords and categories
- Repeated information across records
- Inflexible design; data sets that are created for a single report without a view towards future use
- For class discussion: what else can you think of based on our examples in class?



Review: Guidelines to creating a valid table in SQL

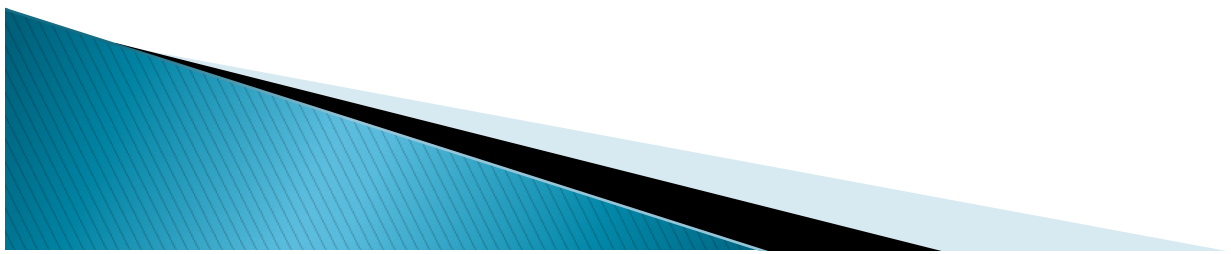
- Every row or record contains data about one specific entity.
- Every field contains data with attributes of (or information about) that entity.
- Any given field within a specific record may only contain one value.
- All entries of a specific column or field are of the same kind; i.e. they are of the same data type.
- Every column or field has a unique name within the table.
- The order of the columns or fields is not important.
- The order of the rows or records is not important.
- With respect to the data: No two records may be identical.



Tables and Data Types

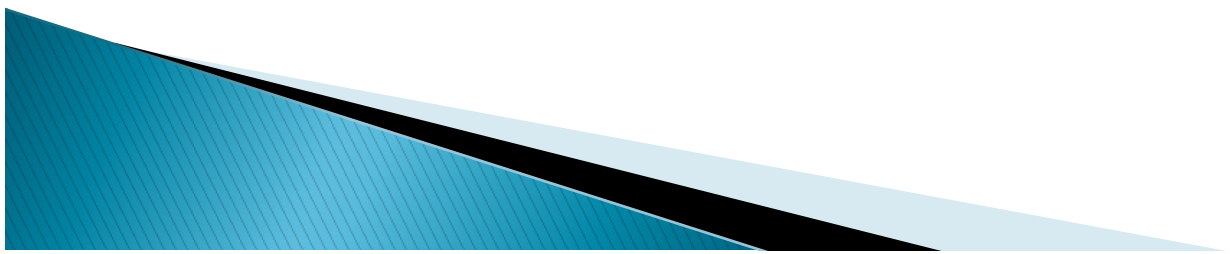
- Every field is associated with a specific data type. For example:
  - Character types for text
  - Integer types
  - Numbers with decimals
  - Dates / time

- Note that different data types sort differently. For example, numbers would be sorted as (1,2,12,20) but numbers in a text format would be sorted as (1,12,2,20).



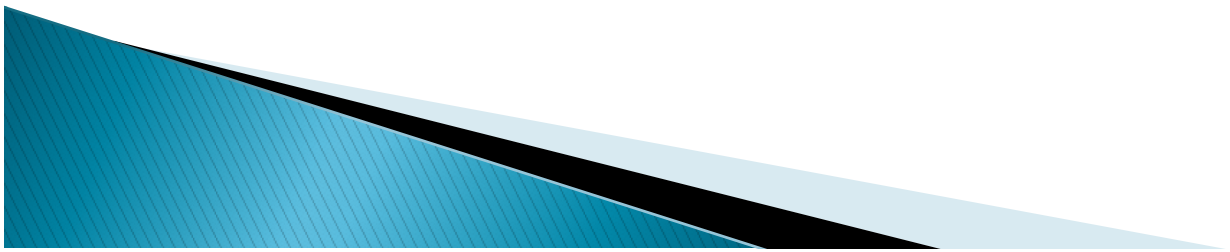
What is Database Normalization ?

- Database normalization is the process of designing a relational database schema that reduces data duplication and complies with data integrity.
- This is done by creating a schema that satisfies the six normal forms (1NF – 6NF).
- From a practical perspective, satisfying the first three normal forms (1NF – 3NF) is generally sufficient.



Database Normalization: 1NF

- First Normal Form (1NF)
  - Each column must contain only one value.
    - For example, an address field should be broken up into one or more fields for the street address; and one field each for the city, state, zip code and country.
  - *For class discussion*: We will consider a valid table to comply with 1NF.

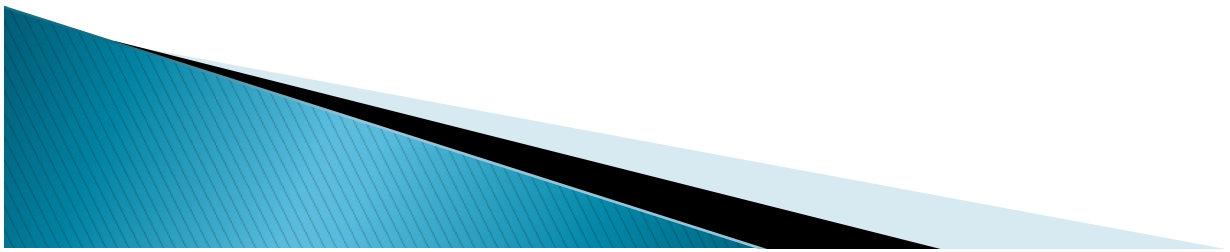


Database Normalization: 2NF

- Second Normal Form (2NF)
  - The table must already satisfy 1NF.
  - Here we examine prime and non-prime attributes.
  - This is where we identify foreign keys and create new tables in order to avoid duplication of data.
    - For example, information about an artist should not be stored in a table about paintings on exhibit: *only* the artist's name or ID as a foreign key should be stored in the table containing information about the paintings while the information about the artists (year born, year died, etc.) should be moved to a separate table in which the artist's name or ID is the primary to avoid data duplication.

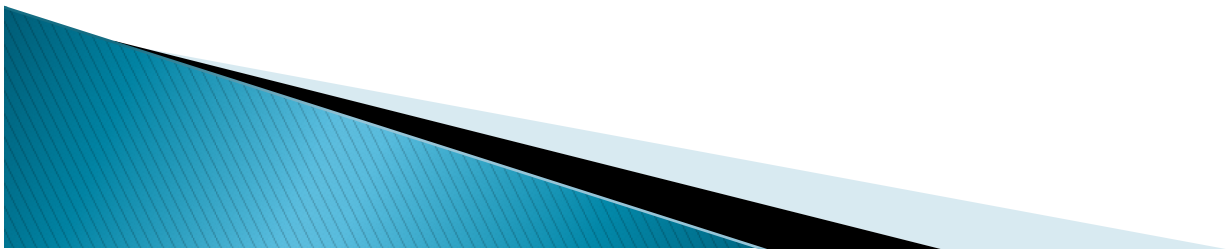
Database Normalization: 3NF

- Third Normal Form (3NF)
  - The table must already satisfy 1NF.
  - The table must already satisfy 2NF.
  - When a table is 3NF, every column is independent of every other non-key column.
  - We say that we eliminate transitive dependency in this step; for example, we might not need to store the city for an address if we have the zip code.
  - From a practical perspective, If the database already satisfies 2NF, this step might not require further changes but is still a good review of all fields.



Database Normalization: BCNF

- BCNF: Boyce Codd Normal Form
  - BCNF is a slightly stronger version of 3NF.
  - BCNF is rarely used and relates to specific data models.
  - BCNF roughly means that every foreign key has only one possible match and could be a primary key elsewhere.

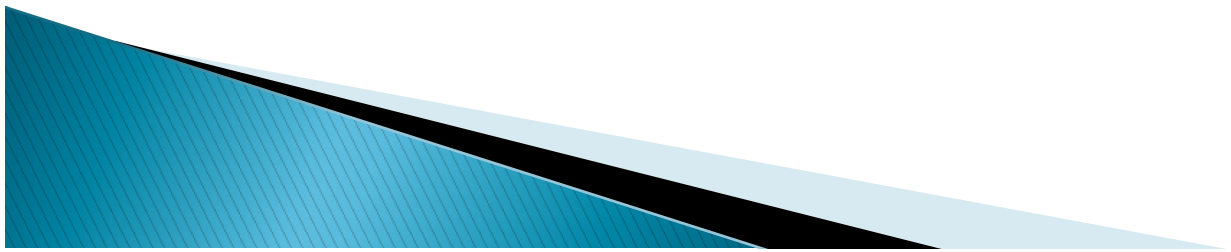


Database Normalization

- Here is a good test for 3NF and BCNF:

A table is based on
the key,
the whole key,
And nothing but the key

From Beginning Database Design by Clare Churcher, page 122

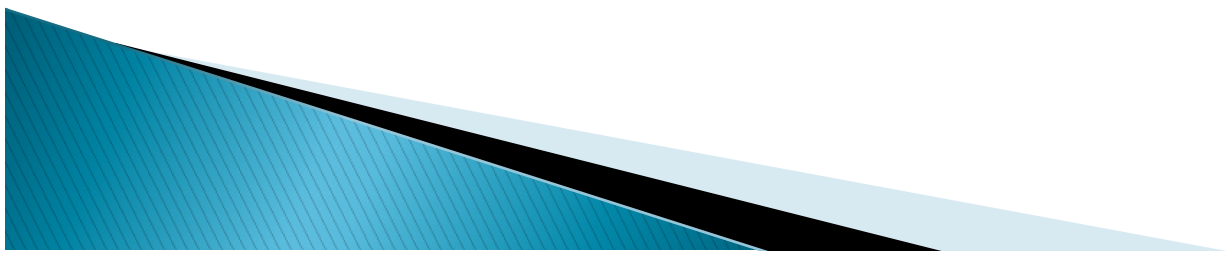


Database Normalization: 4NF, 5NF and 6NF

- The 4th, 5th and 6th Normal Forms are used much less often. In every case, the database must already satisfy 1NF, 2NF and 3NF.
- Fourth Normal Form (4NF)
  - Here we evaluate non-trivial multi-value dependencies; e.g. in this step, we might decide to move all addresses into a separate address table.
- 5th normal form (5NF)
  - It is rare for a table to comply with 4NF and not with 5NF and typically relates to specific model constraints based on keys used for joins.
- 6th normal form (6NF)
  - one primary key + one attribute;
  - Note that 6NF is not necessarily desirable!

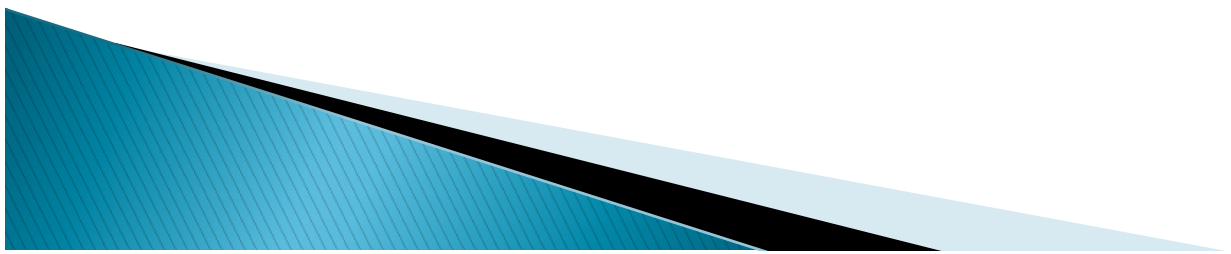
Database Normalization

- *For class discussion:*
- Remember that in the “real world”, practical concerns about convenience and performance will override some of the theoretical notions of normalization.
- A design that includes some controlled data duplication to speed up performance (such as storing a city which otherwise could be looked up by zip code) will be preferred over a design that strictly goes “by the rules”. This is part of the **art** of database design: where theory and practice meet.



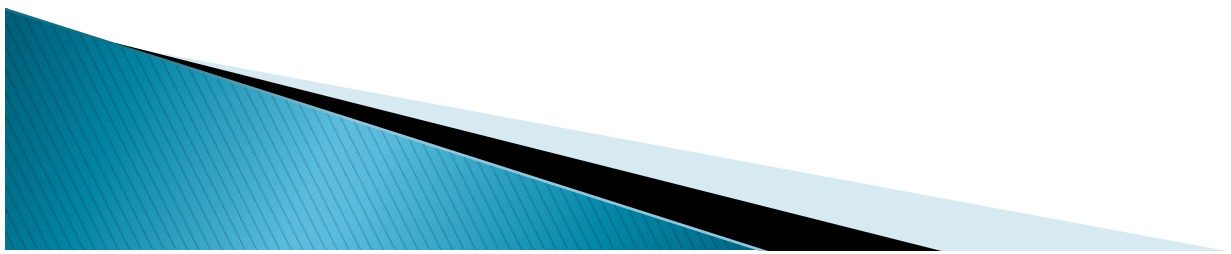
Keys

- A primary key uniquely identifies each record or row.
- A foreign key in one table refers to a primary key in a different table.
  - For example: your “N#” in Albert would be a primary key in the table of students … and a foreign key in a table on class enrollment.



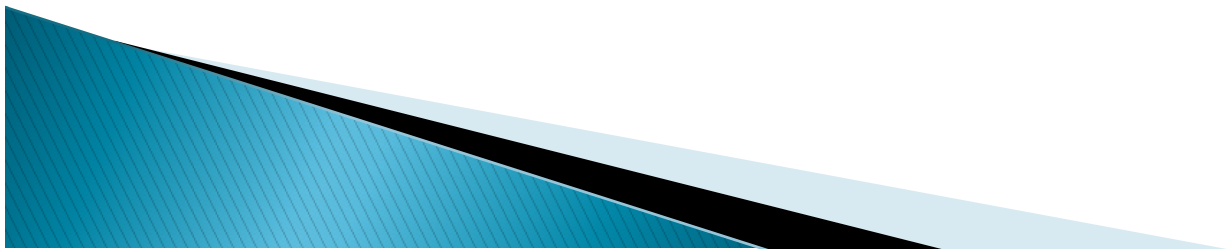
Keys, continued

- A concatenated key (also called a “compound key”) is made up of more than one field.
- A surrogate key (such as your “N#” in Albert) is a randomly or sequentially generated key that is meaningless in and of itself. For example, your netid is not entirely meaningless because it begins with your initials; but the digits in your “N#” do not (to my knowledge) reflect any other information.



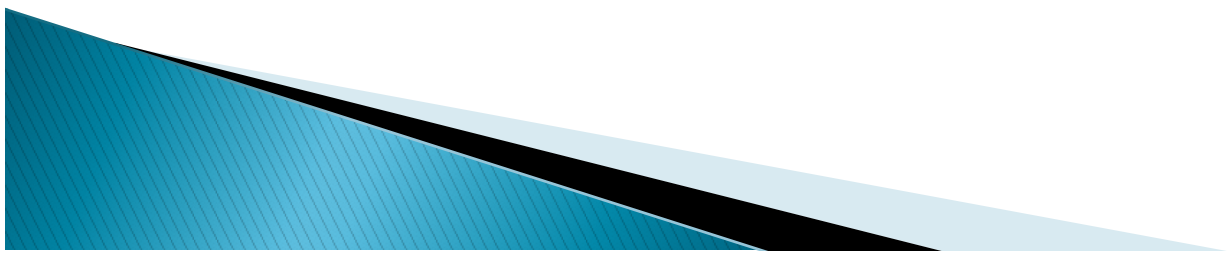
Data Integrity vs. Referential Integrity

- The term “data integrity” refers to working with valid data.
  - For example, in class we used Python to “scrub” data to make it useful. A date such as “05/65” would not make sense. A course number in Albert for a non-existent course should not be permitted. A negative value for a person’s age would not make sense … and so on.



Referential Integrity

- Referential integrity means that a value for a foreign key truly finds a match to the related table.
  - For example, referential integrity means that if I get a list of “N-numbers” for all of the students in this class, that every “N-number” refers to a student currently enrolled at NYU.

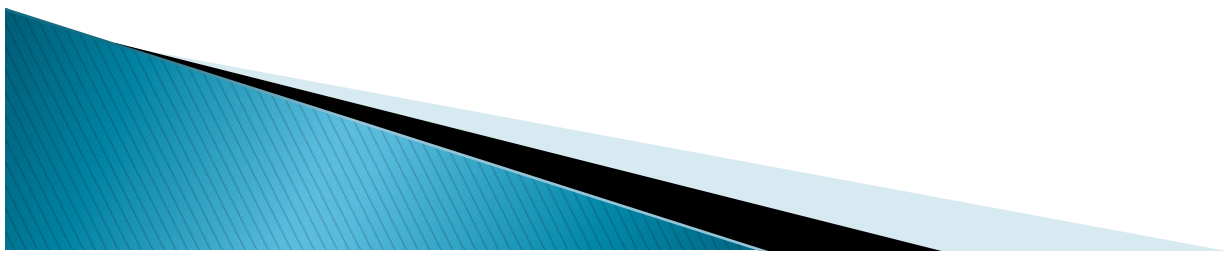


Many:Many Relationships

- There are instances when relationships are effectively Many-to-Many and need to be modeled that way; for example –
  - NYU's "Albert" for tracking grade: every student gets several grades each term, one for every class; every class has multiple students and each student receives a grade. So our intermediate table or joining table in this case is about the grade that each student receives for each class.
  - In the movies: one actress can play in many movies; each movie might employ many actors and actresses. In this case our joining table would include data such as the role that each actor/actress plays in each movie as well as foreign keys to identify both the actor/actress and the movie.
- We will discuss this further in class.

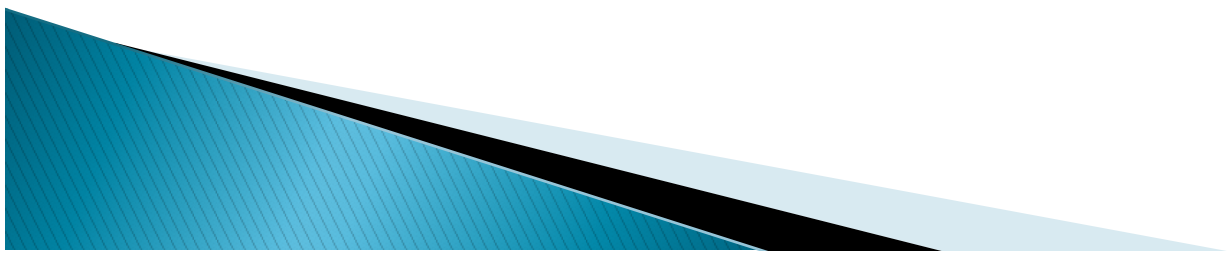
Optionality

- Optionality: Should it be 0 or 1?
- For example – its fine for us to have artists in our database without paintings (as we might sell for them in the future!) but not fine to have paintings in our database without a related artist in the artist's table!



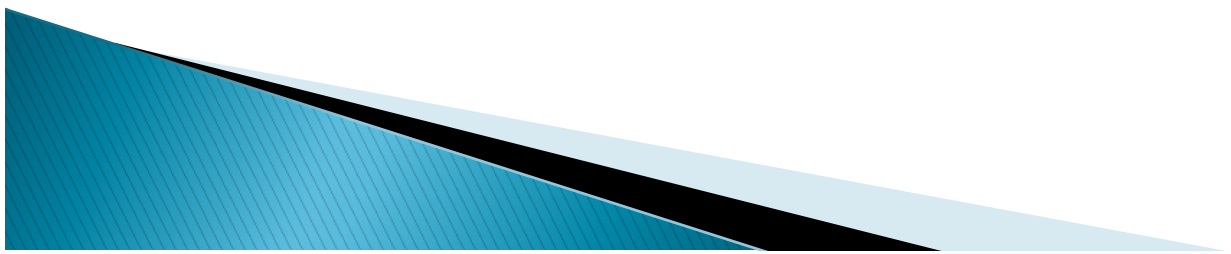
Cardinality

- Cardinality of 1: Might it occasionally be 2?
- If there are many cases, we do need to create a separate table and use a Foreign Key field value to relate back to the original table.
- However, if there are very few cases, one might not re-design the database for it.



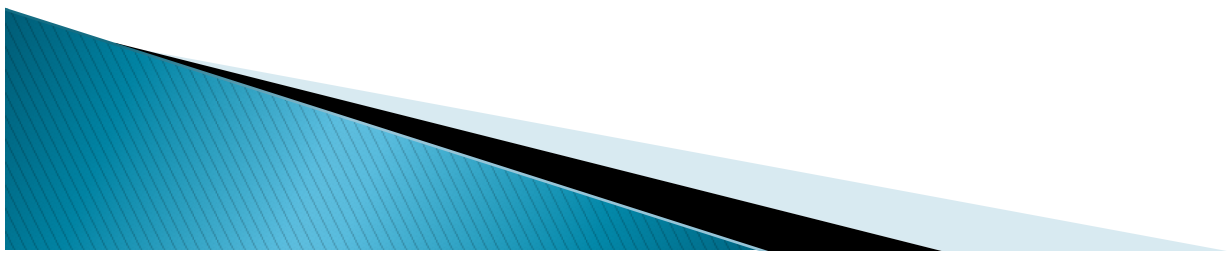
Cardinality of 1 continued

- Cardinality of 1: What about historical data?
- Managing historical data will often require related tables. For example, there could be multiple transactions for the same object or person in a transactions.
- Historical data requires in these cases that we keep the transactions in a separate table.
  - For example a person's medical records should be kept in a table separate from the information about the person
  - For example, sales of a valuable painting over time should be kept in a separate transactions table.



Cascading Updates and Deletes

- A cascading delete would keep all related records "in synch" in the event that a record is deleted:
  - If a bank customer passes away, a cascading delete would remove all of his/her accounts from the table of active accounts. (However, in this case, the data would normally be archived rather than deleted.)
- In a cascading update, related data would be updated but this is a rare occurrence. *(Question for class discussion: Why??)*



Choosing Data types in a database:

- Character types – for example:
  - strings and textual information
  - zip codes, social security numbers, phone numbers and similar data would be characters or text rather than numbers
  - URLs can be stored as text and later activated as hyperlinks in the application
  - filenames (e.g. “cat.jpg”) can be stored as text and then rendered as images in the webpages with appropriate programming in the application
- Integer data types
- Floating point data types
- Dates
- Boolean values
- It is worth reviewing the documentation to see the many options that you have: http://dev.mysql.com/doc/refman/5.1/en/data-types.html

Summary of design topics for class discussion:

- Select appropriate data-types.
- Consider data validation and constraints on the values: make sure that the data are reasonable.
- Sorts and indexes: plan your tables to be sure to accommodate ordering needs (the way the data would be sorted).
- Define and ascertain the information that you will need to track for all of the calculations that might be needed.
- Working with keys:
  - Identify primary and foreign keys and relationships that they support.
  - Identify concatenated keys (also called “compound keys”) and surrogate keys.
- Referential Integrity
  - Insure referential integrity.
  - Identify any needs for cascading updates and/or cascading deletes.
- Document the relationships among the tables:
  - 1: many
  - many: many