When you read a UML diagram/LDM, think about one object of that class and then work around the cardinality.

How do we represent many cardinality in a database? Should we always use a separate table for representing many-many cardinality? And is 1-many cardinality be always represented by 2 tables (I think in this case if the table on the many side of cardinality is a lookup, we should in that case use a 3 table for the association. But if it is a lookup, then any way the association becomes many-many and thus 3rd table is required)?

In 1-1 cardinality association one foreign key should be in either of the tables but you can have foreign keys in both tables if you want to ‘walk back’ from other side as well.

In 1-many cardinality association the table on the ‘many’ side of association will have the foreign key.

There is no need to use inheritance associations in db modelling diagram (discussed down below) as the same concept can be implemented by simple associations and new tables.

If you have a table in which some attributes are sometimes null and others attributes are null the other times, then there may be different types of records in the same table which have some common information. The ‘type’ might not even be explicitly present as a column in the table but something that you have to infer. Each of those types should be represented by a separate table with a foreign key reference from the parent table. That is inheritance. Can this handle multiple inheritance as well? I think so. But in general don’t try to model Objects/inheritance in rdbms. But a properly normalized db (no nulls and no repetition of information) might mirror an object oriented design.

The problem with clare churcher book is that for database modelling she starts talking about inheritance where she should have taken the route of Normalization to achieve similar results

System analysis and design(SAD), NatSoil database: flat file database with a fortran app on top of it🡪ingres🡪oracle🡪access🡪sql server

ERD(with Crow’s notation) vs UML data modelling diagrams(which is extended UML class diagrams for relational db modelling). I could use crow’s foot notation with UML data model diagrams if I wanted to. EA data modelling template provides both Logical data model(LDM) and physical data model support(PDM). LDM is a UML class diagram whereas PDM is a database specific diagram and talks about primary key/foreign keys.

UML class diagrams talk about classes, attributes, operations and association between classes as well as cardinality of the associations. Now there are specialized associations as well (generalize, compose, aggregate, association class, realize, nesting, etc.).

Any type of association (relationship) in db is reflected by having keys from one table in another, i.e. the table at the ‘many’ side of an association. Remember we try to breakdown an association with ‘many’ on both ends by having an association class in between. But not necessarily in the case of code. For example: inheritance (specialization/generelization) is not reflected by having attributes of one class in another

IMO, just start with a simple UML class diagram first (just use associations with cardinality) and then think how those associations would be implemented in either db(by using primary key foreign keys, association tables etc.) or in code(by using compose, generalize, association classes etc.). A UML class from a LDM becomes a class in code or a table in PDM.

- **cardinality** specifies the maximum number of relationships and **ordinality** specifies the absolute minimum number of relationships

-many-to-many association generally need an: association class or a separate table in PDM. The other associations can be represented by setting referential integrity in tables or by setting pointers/references in classes.

-For data retrieval and reporting, attributes used for sorting, grouping and selecting data might be candidates for additional classes.

1. use case: free-format text accounts that describe uses of the system from the point of view of an eventual user. Before you begin database design, you need some use cases about how the system would eventually be used. Database prject use cases revolve around entering/updating data and extracting information based on that data. What are the different questions you want to ask of the data.

2. After uses cases are designed, you need a to start with data model and how it can be used to solve the problems posed in use cases.

3. Questions that need to be asked:

a. what does the user do?

b. what data is involved?

c. what is the main objective of the system?

d. what data is needed to satisfy this objective?

e. what are the input use cases?

f. what is the first data model?

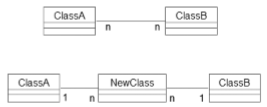
g. what are the output use cases?

4. Use cases should not be too broad as well not too insignificant. Rule of thumb is that the task entailed in use cases should less than 20 minutes of time.

5. Some assumptions need to be made to narrow the scope of the problem and these need to be made clear to the client.

6. Where ever you see 1 as the max cardinality at one end of a relationship, think about historical data and how you might lose it. In these cases if the maximum cardinality of the relationship is 1, it would have to increase and an association class added to maintain the historical records. For LDM just change the cardinality of the association to reflect that we need historical data as well. It is in PDM that we can decide an association class or something else. For example, a one-to-many association might become many-to-many. Here we are talking about max cardinality from each end of the association. You might need an ‘is\_active’ column and active dates (start and end) to prevent hard deletion of data and prior associations.

7. many-to-many association (again we are talking about max cardinality at either end of an association) generally need an association class in code or a separate table in PDM as there is some information about the association that needs to be captured. The other associations can be represented by setting referential integrity in tables or by setting pointers/references in classes.



8. here are the questions based on above discussion:

a. optionality/ordinality: should it be 0 or 1?

b. A cardinality of 1: might it occasionally be 2? Instead of increasing the cardinality, might want to repeat create new object instead.

c. a cardinality of 1: what about historical data?

d. many-many cardinality: are we missing anything? Is there any information that can only be captured in an association class as that info belongs to the relationship?

9. Attribute, class or association (relationship)? If a single attribute is in question, we might want to keep it as a lookup. For data retrieval and reporting, attributes used for sorting, grouping and selecting data might be candidates for additional classes. If you would want to add new related attributes, then also we might want to have a new class either with the same number of attributes or store them as key value pair, also called Entity-Attribute-Value model (EAV) (transpose columns into rows in a new table)

10. If we have a class with two associations with each of which have ‘many’ cardinality on the outside ends, it might be reflective of an issue called fan trap.

11. There should not be multiple paths (associations) in the UML class diagram to get the same piece of information. That reflects redundancy in database.

12. When we have a case where we need simultaneous info from objects from 3 or more classes, we need to introduce a new class connected to all 3 (or more) classes. If we cannot think of an appropriate name, append the names of all the classes it is related to. Is it an association class with more than 2 associations? Looks like.

13. An association (relationship) in db is reflected by having keys from one table in another, i.e. the table at the ‘many’ side of an association. But not necessarily in the case of code. For example: inheritance (specialization/generelization) is not reflected by having attributes of one class in another.

14. Do not think of an association class as a property of the association between classes but as a class that replaces the association between classes so that they are not connected directly now.