Unstructured to Structured –

1. Create a list of entities (all nouns) – Once you have found all nouns mark out the ones that make no sense.
2. Separate the list you made into entities and attributes. – Create a table one column is entities the other column is attributes associated to each entity. Some entities might have multiple attributes for each row. Some might have none for right now.
3. Add additional attributes that make sense. FirstName, LastName, DOB,SSN, etc, etc. PhoneNumber, ID,
4. Identity potential relationships by verbs. Then eliminate the ones that make no sense.
5. Generate more potential relationships by pairing the relationships.

Like this

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entities | Doctors | Medicine | Prescription | Pharmacy |
| Doctors | Consults | Prescribe Medcine | Write precriptions | Contacts |
| Medicine |  | n/a | Inclues | dispenses |
| Prescription |  |  | n/a | Filled by |
| Pharmacy |  |  |  | n/a |

6)Eliminate relationships from table that don’t mean anything. Like doctors consulting other doctors, or doctors contacting the pharmacy. In addition, you can eliminate one where doctors prescribe a medicine and doctors write a prescription no need for two that mean the same thing.

7) Then you list out each entity and number of attributes that go along with each entity. And, say if an attribute is a key attribute, multivariable attribute, compound attribute, or just a regular attribute.

Example: The Doctor entity has x amount of attribute. First name, Last name, ID (which is a key attribute), and the multi-value attribute Specilization(s).

8) Now you must do the relationships descriptions. You get this table from the table above 6. For example, Write, Includes, Filled, (forgot to include patients), but another would be Visits ,would be relationships. So, an example of one would be: The visits relationship is between a Doctor and a Patient. A Patient will visit zero or more Doctors and each Doctor will visit one or more Patients. Assumed that to become a Patient you must visit the Doctor one or more times. Another one would be: The writes relationship is between a Doctor and Prescription. A Doctor will write zero or more Prescriptions. Every prescription must be written by exactly one Doctor. Or Each Prescription is received by a single Patient. Or A Prescription is filled by exactly one Pharmacy. Remember the wording.

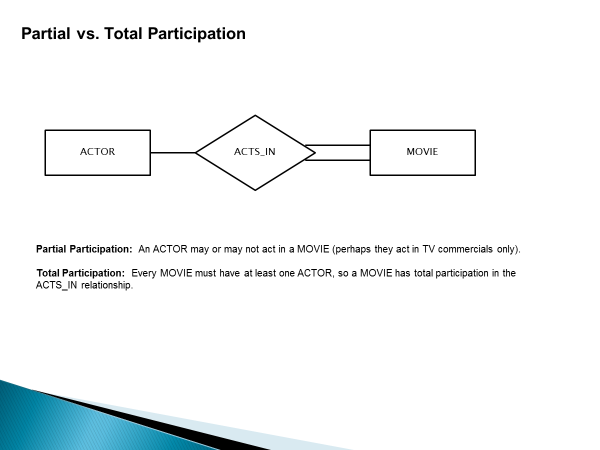
9) Once finished you can now create an ER-Diagram.

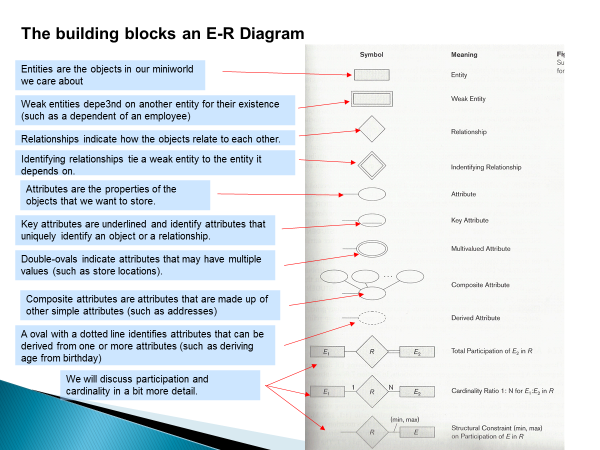
**1:1** – 1 to1

**1:N** – 1 to many

**N:1** – Many to 1

**N:N** – Many to many





* Subclass disjointedness refers to whether or not an entity can belong to more than one subclass in a specialization.
* If an entity can only belong to one of the subclasses, then we say that the specialization leading to the subclasses is disjoint.
* By definition, if a specific attribute in the superclass is used to determine membership in a subclass, then the specialization will be disjoint.
* If an entity can belong to more than one subclass in a specialization, then the specialization will be overlapping.

SELECT L2.Name

FROM L2 LEFT JOIN L3 ON

L2.StudentID = L3.StudentID

WHERE L3.StudentID IS NULL

SELECT AppointmentDate, AppointmentTime, Mechanics.FirstName + ' '+ Mechanics.LastName AS 'Mechanic' , ServiceName,'$' + CAST( SUM(Services.Cost) AS varchar(10)) AS 'Total Cost',Customers.FirstName + ' ' + Customers.LastName AS 'Customer'

FROM Customers JOIN ServiceAppointments ON Customers.ID = ServiceAppointments.CustomerID

JOIN AppointmentServices ON AppointmentServices.AppointmentID = ServiceAppointments.AppointmentID

JOIN Services ON Services.ServiceID = AppointmentServices.ServiceID JOIN Mechanics ON Mechanics.ID = ServiceAppointments.MechanicID

GROUP BY AppointmentDate,AppointmentTime,Mechanics.FirstName,Mechanics.LastName,ServiceName,Customers.FirstName,Customers.LastName

ORDER BY Mechanic DESC, AppointmentTime ASC

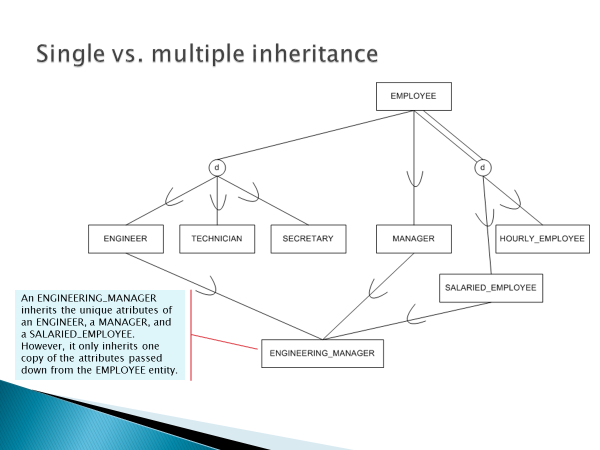
SELECT \*

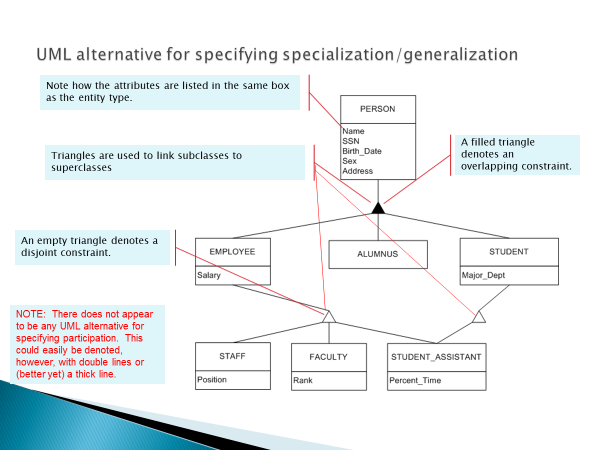
FROM TennisPlayers JOIN Registration ON

TennisPlayers.ID = Registration.PlayerID

JOIN Tournaments ON

Tournaments.ID = Registration.TournamentID





The process by which we efficiently organize data to achieve these goals: Eliminating redundancy Ensuring data is stored in the correct table Eliminating need for restructuring database when data is added. Five levels of normal form each must be met in order to be normalized. Although the third form is alright for databases. Relationships are created between tables using primary and foreign keys.

Stored Procedure- adv. Faster than direct queries since they are precompiled on database, does not require any java coding to test, seperates business logic from application code. Disadv. – less flexible, requires effort in database as well as in the application code.

Direct Query – adv. Very flexible, does not require any database changes. Disadv. May be slower than precompiled stored procedures, sensitive to database changes, intermingles database logic and application logic.

* **public ResultSet getCustomerByID(String customerID){**
* //Returns a ResultSet containing at most one row of data about a customer
* StringBuffer buf = **new StringBuffer();**
* buf.append("SELECT \* FROM Customers WHERE Customers.ID =");
* buf.append(customerID);
* **return executeQuery(buf.toString());**
* }