DBS211
Normalization

What is Normalization?

- Normalization is a series of steps used to evaluate and modify table structures to ensure that every non-key column in every table is directly dependent on the primary key.
- The results of normalization are reduced redundancies, fewer anomalies and improved efficiencies.

2NF - Partial Dependencies

- database already in 1NF
- each attribute in an entity must be wholly dependent on the entire primary key.

- Partial dependencies are relationships where the attribute in focus is not dependent on the entire primary key or is dependent on more than the primary key.

The 2 Variations of Partial Dependencies

An attribute partially dependent on a composite primary key

An attribute dependent on more than one field, but only a single field PK exists

An attribute partially dependent on a composite primary key

Example :A soccer league business rule where teams have more than one player and each player can play on more than one team.

UNF

TEAMS [teamID, teamName, ShirtColour, (PlayerID, PlayerName, PlayerDOB, ShirtNumber)]

1NF TEAM_PLAYERS[teamID, PlayerID, teamName, ShirtColour PlayerName, PlayerDOB, ShirtNumber]

•x**teamName** - is team specific and does not depend on any field other than teamID
•**xShirtColour** - is team specific and depends only on teamID. All players on one team wear the same colour shirt.

•**xPlayerName** - is player specific and depends only on playerID. However the TEAM_PLAYERS entity has a composite key with teamID. PlayerName has no relation with teamID and therefore has a partial dependency with the entity's PK and must be moved.

•**xPlayerDOB** - same situation as PlayerName

•✓**ShirtNumber** - The number on the back of the players shirt will depend on the player, as no two players on the same team wear the same number, however, the player may wear a different number when playing on a different team. Therefore, the ShirtNumber depends on both the teamID and the PlayerID. This matches the composite primary key and is therefore in the right place.

TEAMS [teamID, teamName, ShirtColour]
TEAM_PLAYERS[FK teamID, FK PlayerID, ShirtNumber]
PLAYERS [PlayerID, PlayerName, PlayerDOB]

In this case, the playerID is also left behind in the TEAM_PLAYERS entity in order to maintain the relationship, and will be a FK to the new entity, which now possess playerID as a PK.

Lastly, in order to satisfy 2NF, the solution must also be UNF and 1NF. Checking each attribute in all entities, we can see these are satisfied, and therefore this is a good final 2NF solution.

An attribute dependent on more than one field, but only a single field PK exists

1NF:
TEAMS [teamID, teamName, ShirtColour]
TEAM_PLAYERS[FK teamID, FK PlayerID]
PLAYERS [PlayerID, PlayerName, PlayerDOB, ShirtNumber]
In the case where a player may play on more than one team, then the player number can be different for each team and therefore the shirt number depends on both the player id and the team id.

Players entity, which only contains the playerID, it has a partial dependency with the PK. Therefore it must move to an entity where both the teamID and the playerID are together a composite primary key (the TEAM_PLAYERS entity).

TEAMS [teamID, teamName, ShirtColour]
TEAM_PLAYERS[FK teamID, FK PlayerID, ShirtNumber]
PLAYERS [PlayerID, PlayerName, PlayerDOB]

The Use of Surrogate Keys and Normalization

The surrogate keys are a replacement key for what was the original fields and therefore, those original fields must be used for the dependency analysis. If a surrogate key is introduced too early, then determining partial and transitive dependencies is greatly complicated. **Don't introduce the surrogate keys to replace composite keys until after normalization has been completed.**

For Example: if we were to replace the composite key in TEAM_PLAYERS with an autonumber field it might look like this.

TEAMS [teamID, teamName, ShirtColour]
TEAM_PLAYERS[rosterID, FK teamID, FK PlayerID, ShirtNumber]
PLAYERS [PlayerID, PlayerName, PlayerDOB]

3NF - Transitive Dependencies

Transitive dependencies are dependencies where an attribute depends on another non-pk attribute in the same entity.

CUSTOMERS [CustomerID, CustFName, CustLName, CustPhone, SalesRepID, SaleRepName]

if I change one non-key field, does it force me to change another non-key field. If the answer is yes, you have a transitive dependency.

We move SalesRep details to another entity and only leave behind the ID as a FK to the newly created PK of the new entity.

CUSTOMERS [CustomerID, CustFName, CustLName, CustPhone, FK SalesRepID]

SALESREPS [SalesRepID, SalesRepName]

Exceptions to Normalization

Here are a few areas where normalization goes to far and will cause issues, such as data storage growth or a lack of available information. One of the key examples of this is to do with physical mailing addresses.

CUSTOMERS [cID, name, phone, address1, street, city, province, country, postal_code]

In this case it is not so obvious, but there are several transitive dependencies here.

- Postal Codes are specific to countries and geographical areas
- province is dependent on country
- city is dependent on province
- street is dependent on city
- street house number is dependent on street

This gets extremely complicated and has some very specific requirements. First, let us look at the normalized version of an address.

CUSTOMERS [cID, name, phone, FK postal_code, house_number]
POSTAL_CODE [postal_code, FK streetID]
STREET [streetID, streetName, FK City]
CITY [city, FK provCode]
PROVINCE [provCode, provName, FK CountryCode]
COUNTRY [CountryCode, CountryName]

In true 3NF form, this is what an address would look like. But let us look at more consequences using an example of an online shopping site that sells t-shirts:

Remember that normalization results in FK-PK relationships which require referential integrity, meaning data can not be added as a child record unless the parent record already exists

This level of data storage and maintenance may be alright for very large companies, such as FedEx, but not for most small and medium businesses in the country.

The point here is that, sometimes completing normalization to the extreme is sometimes more costly than it is worth. So in some cases, we bypass normalization for simplicity.