The **Answer** to a Question:

How to set up Primary Keys in a Relation?

from **Umbra Æternitatis** on StackOverflow exceeded the limits of an answer (StackOverflow is intended for quick answers to simple questions, and database questions and answers tend to be long, as they involve explanations).

This document contains the full answer, rather than breaking it up at the limit.

The answer follows the chronology of the question, and the updates to it. Unfortunately that results in section pertaining to one subject being located in more than one place. Hopefully, that does not interrupt the flow.

A Preliminary

Position

Any practice that is not based on solid theory is not worthy of consideration. I am a strict Relational Model practitioner, with a strong grounding in the theory. The *Relational Model* is based on solid theory, and has never been refuted[1]. There is nothing solid in what passes for "relational theory", I have taken them on, and refuted their notions in their space. Further, Relational Database design is a science, not magic, not art[2], therefore I can provide evidence for any of the propositions or charges that I make. My answers are from that position.

"Relational Theory"

> I wish to know how to correctly set up Primary Keys in a Relation. E.g. we have ER-diagram which contain elements:

If it was an ERD, then you wouldn't be looking at "relations", you would be looking at entities (if the diagram was early) or tables (if it were progressed). "Relations" are a wonderful abstraction which have nothing to do with an implementation. An ERD or a Data Model means an implementation (non-abstract, real) is intended, the intention to the physical leaves the abstract world of theory behind, and enters the physical world, where idiotic abstractions get destroyed.

Further the "theoreticians" who allege to be serving the database space cannot differentiate between base relations and derived relations: while that might be acceptable in the abstract context, it is dead wrong in the implementation context. Eg. base relations are tables, and they need to be Normalised; derived relations are, well, derived, views, of base relations, which by definition are **flattened views** (not "denormalised", which means something slightly different) of base relations. As such, they need not be Normalised.

- But the "theoreticians" try to "normalise" derived relations. And the most damaged two are trying to have the definition of 1NF, that we have had for forty five years, that is fundamental and rock solid, that they themselves have supported, changed, so that their derived relations, which do not need "normalisation", can be classified as "normalised". It would be hilarious if it were not so sad.

Objective Truth

One marvellous quality of objective truth, of science, is that it does not change, it is permanent. It can be relied upon, it must be understood before a practical exercise is undertaken.

Subjective "truth", non-science, changes all the time. It is not worth the reading.

---

1 The are non-science articles, and masses of opinions from those who do not understand the science, yes, but no scientific refutation. Much like pygmies arguing that man cannot fly, it is "true" for them, but not true for mankind, it is based on a complete inability to understand the principle of flight.

2 There is some art in the presentations of high-end practitioners, yes, but that does not make the science an art. It is a science, and only a science, and over and above that, it can be artfully delivered, in models and databases.

Isolation

They live in a world of their own, isolated from the reality of Relational Databases, specifically the Relational Model, and the industry that they allege to serve. In forty five years since the *Relational Model* came out, they have done nothing to progress the *Relational Model* or Relational databases.

- Mind you, they have been progressing all sorts of notions, which are outside the *Relational Model*.
- The progress of the *Relational Model* (completion of what the Neanderthals suggest was "incomplete") has happened solely due to the standardisation (R Brown and others working with Codd, resulting in the IDEF1X Standard for Modelling Relational Databases), and the efforts of high-end SQL vendors and their customers.
- That is the commercial RDBMS vendors, who were already established in the 1980's, not the Non-sql freeware/shareware/vapourware groups of the last decade, who pass off their wares as "sql", which gets you good and glued to their "platform", through non-portable code.

The worst part is, they publish books about their non-relational concepts, and fraudulently label them as "relational". And "professors" blindly "teach" this nonsense, like parrots, without ever understanding either the nonsense, or the Relational Model that it is supposed to explore.

- If you are trying to find answers to some "educational" project, sorry, I cannot provide that, because the "education", as you can see, is totally confused, and has non-relational requirements.
- I can however, provide direct answers to the question, governed by science, the *Relational Model*, the laws of physics, etc.

The point to take from this is, while Relational Theory and Practice were very close after Dr E F Codd published his seminal work, and during the time that the SQL Platforms were developed by the vendors, in the post-Codd era, what passes for "relational theory" is completely divorced from that original Relational Theory.

- I can enumerate the differences, but not here. Note that if you read my posts that touch on this subject, you can gather those particulars, and enumerate them yourself. Or else ask a new question.

B The Question

> I wish to know how to correctly set up Primary Keys in a Relation. E.g. we have ER-diagram which contain elements:

There is no ERD to examine. Ok, in the Update you have an example. Perfect for your questions, because it is a set of user views of the data, and the modelling can now begin. But note, that is not an ERD or a Model. We rely on understanding the data; analysing it; classifying it, not on looking at the data values with a microscope. I realise that that is what you have been taught to do.

> In order to translate it into Relational Model

Yes, that is the stated goal. The word "translate" is incorrect, because the *Relational Model* is not merely a flat or fixed set of criteria that one "satisfies" or fits into (as it is known to the "theoreticians"), it also provides specific **Methods** and **Rules**. Therefore, we will be **Modelling, according to the *Relational Model***.

> we should do some tricks.

We don't need tricks, we use science, and only science. The "theoreticians" and the "professors" who follow them, need tricks, and practice non-science. I can't help in that regard. Further, the tricks they use, are usually to circumvent and subvert the *Relational Model*, so watch out for them.

B.1 Surrogate

> All elements above deal with Primary Keys of relations but they all are Natural Keys - so we can leave them as is or replace with Surrogate Keys.

Well, there it is, your "teacher's" first trick is exposed.

1. Surrogates are physical Record (not row) pointers, they are not logical.
2. There is no such thing as a "surrogate key", the two words contradict each other.
   - A Key has a specific definition in the *Relational Model*, it has to be **made up from the data**. A surrogate isn't made up from the data, it is manufactured, a meaningless number generated by the system. Therefore it is not a Key or a "key".
   - A Key in the *Relational Model* has has a number of Relational qualities, which makes Keys very powerful. Since a surrogate is not a Key, it does not have any of those qualities, it has no Relational power.
   - Therefore, "surrogate" and Key each have specific meanings, and they are quite fine as separate terms, but together, they are self-contradictory, because they are opposites.
   - When people use them term "surrogate key", they naturally expect some, if not all, the qualities of a Key. But they will not obtain any of them. Therefore they are defrauded.
3. The *Relational Model* (the one that the theoreticians know nothing about) has a specific **Access Path Independence Rule**. As long as Relational Keys are used, this rule is maintained. It provides Relational Integrity[3].
   - The use of a surrogate **violates** this rule. The consequence[4] is, Relational Integrity and Relational Navigation[5] are both lost.
   - The consequence of that is, many more joins are required to get at the same data (not less, as the lovers of mythology and magic keep parroting).
   - Therefore surrogates are not permitted, on another, separate count.
4. Since you are in the modelling stage, either conceptual or logical, and Keys are Logical, and surrogates are physical, surrogates should not come into the picture. (They come into the picture, if at all, for consideration, only when the logical model is complete, and the physical model is being considered.) You are nowhere near completion of the Logical, so the introduction of a surrogate should raise a red flag.

   The "teacher", and the author of the "textbook" that he is using, are frauds, on two separate counts:
   - They are introducing a physical field, into the Logical exercise, which should not concern itself with physical aspects of the database.
   - But in so doing, the effect they have is that they establish the surrogate, the physical thing, as a logical thing. Thus they poison the mind.

There, straight science, pure logic, uncontaminated by insane thinking, and thus immune to the frauds. No surrogates at the Logical stage.

So the final answer to your question:

> All elements above deal with Primary Keys of relations but they all are Natural Keys - so we can leave them as is or replace with Surrogate ~~Key~~ s.

In the conceptual and logical exercise, we deal with Logical Keys only. Physical concepts such as a surrogate are illegal. The replacement of a Logical Key with a physical creature, in the Logical exercise is rejected. Use the Keys you have, which are from the data, and natural.

---

3 Relational Integrity (which the *Relational Model* provides) is distinctly different to Referential Integrity (which SQL provides, and Record Filing Systems might have). If you do not understand this, please open a new question "What is the difference ..." and ping me

4 Breaking any rule has always has undesirable consequences, beyond the act itself.

5 If you do not understand this, please open a new question "What is the Relational Navigation ..." and ping me.

B.2 Surrogate is Not a "Replacement"

There is one more point. The term "replacement" is incorrect. A surrogate is never a replacement or substitute for a Natural Key.

- One of the many qualities that a natural Key provides, is **row uniqueness**, and that too, is demanded in the *Relational Model*, duplicate rows are not permitted.
- Since a surrogate is not a Key to a row (it is a physical pointer to a record), it cannot provide the required row uniqueness. If you do not fully understand what I am saying, please read **this Answer**, from the top to **False Teachers**. Do test the given code exercises.
- Therefore, a surrogate, even if considered, at the physical modelling stage, is always an **additional column and index**. It is not a replacement for a natural Relational Key.
- And conversely, if the surrogate *is* implemented as a replacement, the consequence is duplicate rows, a non-relational file, not a Relational table.

B.3 Case

Case 1

> Key Attribute is a name - so it must be of type CHAR or VARCHAR. Generally names become Key Attributes.

Yes.

Often they are codes (users *do* use codes). Often Codes jump out at you (you have a very good example in your **One More Update**). `{ D | R | B }` would do just as well `{ < | ^ | > }`. This is of course towards the end of the logical model stage, when the model is stable, and one is finalising the Keys and optimising them. For any stage earlier than that, the wide Natural Keys stand.

The idea is to keep it meaningful.

- Keys have meaning (surrogates have no meaning). One of the qualities of a Relational Key is, that that meaning is carried, wherever the Key is migrated as a Foreign Key.
- And as per your example, wherever it is used. Including program code. Writing:

```
IF CrewType = "Backup"  -- meaningful but fixes a value
IF CrewType = 1         -- meaningless
```

is just plain wrong. Because (a) that is not really a Key, and (b) the user may well change the value of that datum from `Backup` to `Reserve`, etc. Never write code that addresses a data value, a descriptor. So the fact is, `Backup` is the *projection* of the Key, the exposition, and the Code is the *actual* Key. That resolves to `CrewType.Name`, and the Key is `CrewTypeCode`.

```
IF CrewTypeCode = "B"   -- Key, meaningful, not fixed
```

While we are on Keys, please note:

1. In the *Relational Model*, we have Primary Keys, Alternate Keys, and Foreign Keys (migrated Primary Keys).
2. We do not have "candidate keys", no such thing is defined in the *Relational Model*. It is something manufactured outside the *Relational Model*. It is therefore non-relational.

   Worse, they are used by people who implement surrogates as "primary keys"[6].
3. A physical consideration[7], but one that should be understood and applied throughout the exercise. When the data is understood and known, the columns will be fixed length. When they are unknown, they might be variable. For Keys, given that they will be indexed, at least on the Primary side, they should never be variable, because that requires unpacking on every access.

Magicians rely on their tricks, to make bunny rabbits look like lions. Scientists do not need them.

---

6 The use the SQL keyword `PRIMARY KEY` does not magically transform a surrogate into a PK. If one follows the *Relational Model*, one (a) determines the possible Keys (no surrogates), and then (b) chooses one as Primary, which (c) means the election is over, therefore (d) the nominated candidates can no longer be called "candidates", the event is history, therefore (e) the remainder, the non-primary Keys, are Alternate Keys.
"Candidate key" is a refusal to conform to the *Relational Model* and nominate a PK, therefore, in and of itself, it is non-relational. Separate to the fact that they have a surrogate as "primary key", which is a second non-relational item.

7 For those non-technical people who believe that no technical knowledge and foresight, no physical considerations at all, should be evaluated during the logical, that's fine, evaluate them at the physical. Since I am not addressing the physical here, I am just making a note for Umbra.

Case 2

> Two (or more) Identifying Relationships become a Composite Primary Key of a relation (which is made of Foreign Keys).

I think you have the right idea, but the wording is incorrect for the generic case.

- That wording is correct for an **Associative Table**, which has two Foreign Keys. Yes, in that case, the two FKs form the PK, which is all that is needed for row uniqueness. Nothing can better that. The addition of a Record ID is superfluous.
- For the generic case, for any table:
  - An **Identifying Relationship**¹ causes the FK (migrated parent PK) to be part of the PK in the child. Hence the name, the parent *Identifies* the child.
  - That makes the child a **Dependent**¹ table, meaning that the child rows can exist only in the context of a parent row. Such tables form the intermediate and leaf nodes in the Data Hierarchies, they are the majority of tables in a Relational database.
  - If the row can exist independently, the table is **Independent**¹. Such tables form the top of each Data Hierarchy, there are very few in a Relational database.
  - A **Non-identifying Relationship**[8] is one where the FK (migrated parent PK), is not used to form the child PK.
  - Compound or Composite Keys are simply made up of more than one column, they are standard fare in Relational databases. Every table except the top of each Data Hierarchy will have a Compound Key. If you do not have any, the database is not Relational.

Please read my **IDEF1X Introduction**carefully.

Case 3

> Identifying Relationship(s) with Weak Key Attribute(s) also become a Composite Primary Key.

The terms *weak* and *strong*, with or without a relationship to *key* is not defined in the *Relational Model*. It is a fiction of the "theoreticians". Thus I cannot answer that question.

- I do note that some of the "theoretical" papers present strong Keys (normal English word, describing the fact that the Key has been established previously) as "weak", and weak "keys" (normal English word, describing the fact that the "key" has not been established previously) as "strong". Such is the nature of schizophrenia.
- Therefore I suspect that it is part and parcel of their evidenced attempt to confuse the science with non-science, and to undermine the *Relational Model*. In the old days, when such people were locked up, humanity was healthly. Now they write books and teach in colleges.

Case 4

> Associative entities usually have two or more Identifying Relationships

Yes. Two is correct.

If you have more than two, then that is not fully Normalised. Codd gives an explicit method to Normalise that, such that there will be two (or more) Associative entities, of two exactly Identifying relationships each.

- "... therefore, all n-ary (more than two) relations ... can be ... and should be, resolved to binary (two) relations."
  (paraphrased for this context)

> so they are to be Junction Relations (Junction Tables).

No. "Junction" relations and "junction" tables are not defined in the *Relational Model*, therefore they are non-relational.

Associative Entities in the logical become Associative Tables in the physical.

---

8 The "theoreticians" do not differentiate Identifying vs Non-identifying, or Dependent vs Independent: all their files are Independent; all their "relationships" between record pointers are Non-identifying. It is a regression to the pre-1970's ISAM Record Filing Systems, devoid of Relational Integrity, power, and speed. Fraudulently labelled as "relational". That is all they understand, that is all they can teach.

B.4 Example

How to set up Primary Keys for Relations in order to handle all above Cases (perhaps some more Cases which I did not mention)?

How to avoid using Surrogate Keys and in which Cases they're necessary?

How to set up datatypes for Primary Keys?

If a Composite Primary Key has to be passed into Child Relation, shall it be replaced with Surrogate?

Those questions are best handled using an example, and you have given one ...

Your Example

Now we can take up the example given in your **Update** to the question. First, a couple of preliminary issues.

1. The first basic error that you make, both in this and the other question, is that you look at the data, presented in tabular form, and you treat that as a Table. As if that is already Normalised, and fixed.

   - If you mark my words above re this issue, you can see that this is actually an error taught by your "teachers", not something you have invented yourself. Namely, that they cannot tell the difference between a derived relation (does not require Normalisation, because it is derived from base relations) and a base relation (needs to be Normalised). And they are trying to instil that disability in your mind, such that you are as crippled as they are.
   - When the user (or in this case a dumb teacher, or the crippled author who wrote the textbook that he uses) puts some data in front of you, they are not giving you a base relation, they are giving you their perception of the data.
   - It is your job, as the data modeller, to Normalise that data, and to Normalise that jumble of data, the result of which is (a) base relations, that are implemented as tables, which do not look like their jumble, and (b) derived relations, which are *not* base relations, tables, but can be *derived* directly from the base relations, tables.
   - Otherwise you will be implementing their jumble, not a database (let alone a Relational database).

   Therefore, the first task that is required for each of the three representations of data that you have given, is to appreciate that they are perceptions of data, and to Normalise them into base relations.

2. The second basic error that you make, again evidently taught by your "teachers", is to focus on the data *values*, rather than the analysis and classification of data.

   - Data *values* are nice to know, good additional indicators that the classification is correct or incorrect, but they are (a) secondary, and (b) come after the classification of data, (c) therefore the *values* are irrelevant until the classification has been performed.
   - Focussing on the data *values* is a method of avoiding the classification of data, which is essential to the modelling exercise.
   - The post-Codd authors and the teachers who use their books do not understand data classification, so they cannot teach it, they focus on data *values*.

   Codd and I teach the classification of data first, and the verification of that (via data values) second.

   However, in this instance, since you have given data values only, and an user perception of that, I will use the data values as given, much to my reluctance, and form the classifications as we progress.

C.1 Spaceship Normalisation

1. Initial

Appreciate that the data given (falsely, in tabular from) is merely an user perception of the data, not a table, and it needs Normalisation.

Perception of Spaceship Data

ShipName	ShipType
Soyuz TMA-14	Soyuz
Endeavour	Space Shuttle
Soyuz TMA-15M	Soyuz
Atlantis	Space Shuttle
Soyuz TM-31	Soyuz

People who do not have that taught disability can differentiate between base and derived relations. In the Standard for Modelling Relational Database, IDEF1X (which remarkably, the post-Codd authors are ignorant of, and do not use), we have a diagrammatic difference: a dashed outline indicate a view, a derived relation.

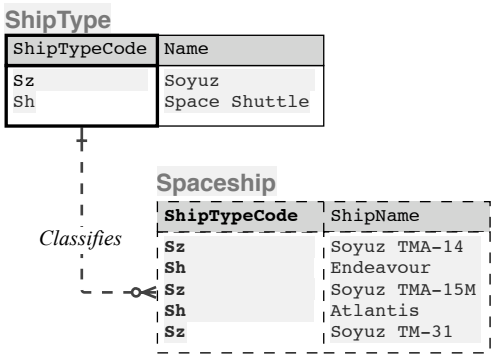
The evidence that it is not Normalised is, we can see that there are repeating groups for `ShipType` and `ShipName`.

ShipType - Foreign Key (but it is not being considered here)

It should be, that is part of Normalisation.

2. Remove Repeating Group A

We remove the repeating group in `ShipType`, we Normalise it into a reference table `ShipType`. One data classification treated correctly.



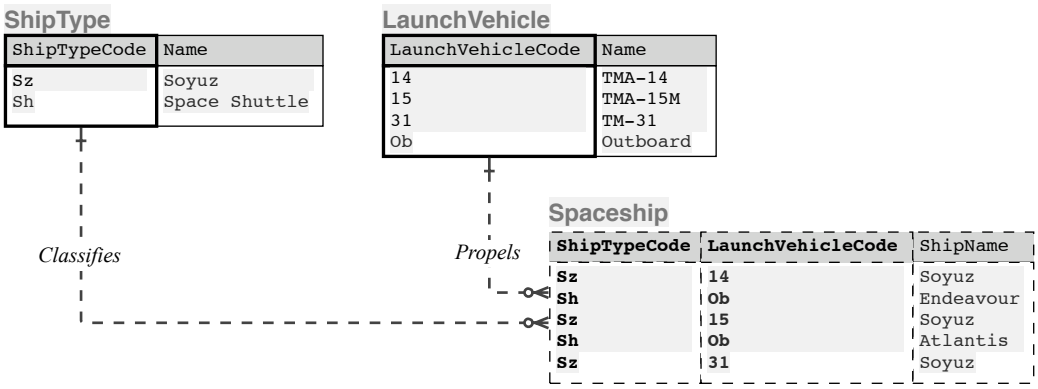
Notice, `ShipType` is now a table, the perception of data remains a derived relation, a view. Since we have not yet determined what the relationship between `ShipType` and the `Spaceship` data is, at this stage it is Non-identifying.

Consistent with my **IDEF1X Introduction**, a bold outline indicates the Primary Key in the table, and bold text indicates a Foreign Key. UML does not come close to representing the richness or precision that IDEF1X affords.

3. Remove Repeating Group B

We can see that there is another series of repeating data in `ShipName`, nicely jumbled with `LaunchVehicleName`. Obviously `LaunchVehicle` is quite separate to `ShipName`, and it has no business being located in `ShipName`.

We remove the repeating group in `ShipName`, we Normalise it into a reference table `LaunchVehicle`. A second data classification treated correctly.

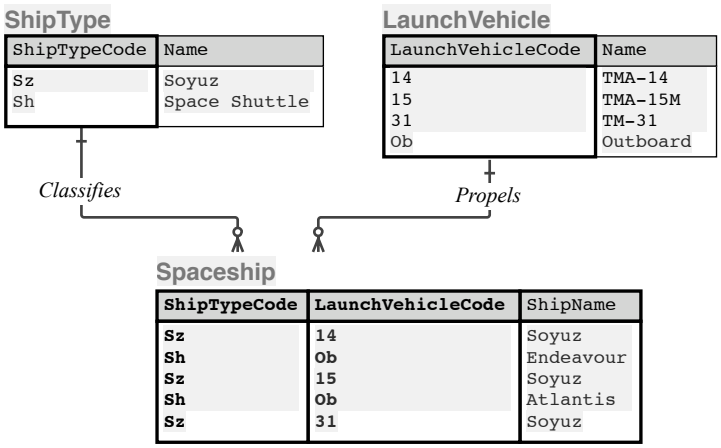


We haven't addressed `Spaceship` directly, yet, so it remains a derived relation, a perception, and the relationship is Non-identifying.

4. Finalise

Now we can address Spaceship directly, the table required to store the Fact that a `Spaceship` exists.

The Russians name their spaceships. But according to your description, which I assume is the assignment requirement, they don't. Due to it being a requirement, although it is a contrived one, we will stick to the latter. In which case, you are right, the **Identifier** for `Spaceship` is a Compound or Composite Key, made up of three things, the two Foreign Keys, plus `ShipName`.

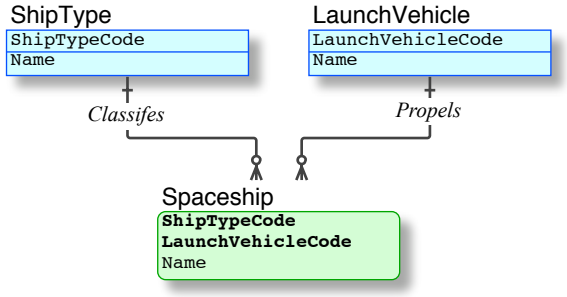


Since the `ShipType` and `LaunchVehicle` Primary Keys are used to form the child Primary Key, they are now **Identifying Relationships**.

Surrogates. First, it is incorrect to consider such physical aspects in the logical model, but since you are considering them, note that a surrogate here does not provide any benefit, it would be a superfluous addition.

5. Model the Data

Now we have progressed to the point where it is worth creating a Logical Data Model.



Note that Identifying Relationships shows **Dependency**. A `Spaceship` does not exist independently; it exists only in the context of (a) a `ShipType`, and (b) a `LaunchVehicle`. It is dependent on both.

6. Construct Predicate

The model exposes the Predicates, and the Predicates are used to verify the Model. It is an important feedback loop, used during the modelling exercise. Therefore, I give the Predicates. The burdensome prefix `Each` for every sentence is omitted.

```
LaunchVehicle is independent
     LaunchVehicle is identified by ( LaunchVehicleCode )
     LaunchVehicle propels 0-to-n Spaceships
     LaunchVehicle is described by ( Name )
ShipType is independent
     ShipType is identified by ( ShipTypeCode )
     ShipType classifies 0-to-n Spaceships
     ShipType is described by ( Name)
Spaceship is dependent on, and identified by, LaunchVehicle
     Spaceship is propelled by 1 LaunchVehicle
     Spaceship is dependent on, and identified by, ShipType
     Spaceship is classified by 1 ShipType
     Spaceship is identified by ( ShipTypeCode, LaunchVehicleCode, Name )
```

As you can see, they verify and confirm that the model is correct (against the requirements given). Note the dependencies expressed in [5] are explicit Predicates.

6.1. Dr E F Codds *Relational Model* is based on First Order Predicate Calculus (commonly called First Order Logic). Thus Predicates are fundamental to Relational Theory, and to the implementation of a Relational Database.

6.2. Note that the post-Codd authors do not understand Relational Theory (they have their own 'relational theory", which is non-relational, as detailed above). They do not understand Predicates, the fundament of its nature. They only know about the most insignificant ones (eg. `SPACESHIP_IS_CALLED`), and they shout those out, as if they were the only Predicates.

- Typically, they make ridiculous claims about what the *Relational Model* can, and cannot do, in terms of semantics.
- All of which are patently false, since (if you understand [6.1]) there is nothing that *cannot* be stated in terms of First Order Logic, There is no such thing as data that is so complex that it can't be declared in FOL.
- The claims therefore prove three things: that the claimants are clueless about the *Relational Model*; clueless about FOL; and thus the claims are really a declaration of the claimants' own intellectual limitations. It has nothing to do with the *Relational Model.*

6.1. The Predicates can be *read* directly from an IDEF1X mode (but not from an UML or other funny diagram).

6.2. But novices cannot *read* all the Predicates, everything that is relevant from an IDEF1X model. The reason being that the post-Codd authors and the teachers who use their books are (a) ignorant of the IDEF1X Standard, and (b) use funny diagrams that do not convey much meaning at all. Thus non-models, which convey little, are commonly presented and used as "models".

6.3. Therefore, when answering questions from novices, I provide all the relevant Predicates, that are implemented in the model, explicitly. Except the insignificant ones, which are Descriptors, the attributes that describe a Relational key, below the PK line, which are more than obvious.

6.4. In this instance, I provide the Descriptors as well. Hopefully, you can see the idiocy of Predicates elevated to table names, such as `SPACESHIP_IS_CALLED`, vs the six Predicates regarding `Spaceship`, given in order of importance.

- `ShipType` is independent, it exists on its own
- `LaunchVehicle` is independent, it exists on its own
- `Spaceship` is dependent, it exists only in the context of a `ShipType` and a `LaunchVehicle`
- `Spaceship` is an Associative table, the Primary Key is the combination of the Primary Keys of `ShipType` and `LaunchVehicle`

7. Derived Relation, View

Now we can prove the data model *even further*, by demonstrating that any and all derived Relations can be projected from the base relations (tables, implemented).

Spaceship_View

ShipTypeCode	ShipType	LaunchVehicleCode	LaunchVehicle	Name
Sz	Soyuz	14	TMA-14	Soyuz
Sh	Space Shuttle	Ob	Outboard	Endeavour
Sz	Soyuz	15	TMA-15M	Soyuz
Sh	Space Shuttle	Ob	Outboard	Atlantis
Sz	Soyuz	31	TM-31	Soyuz

This is the Universal Relation for the three base relations, or tables.

8. Initial Perception is View

Now we can evaluate the Initial Perception of the `Spaceship` Data, against the relevant columns in the Universal Relation.

Inital Perception

ShipName	ShipType
Soyuz TMA-14	Soyuz
Endeavour	Space Shuttle
Soyuz TMA-15M	Soyuz
Atlantis	Space Shuttle
Soyuz TM-31	Soyuz

Normalised

LaunchVehicle	ShipType	Name
TMA-14	Soyuz	Soyuz
Outboard	Space Shuttle	Endeavour
TMA-15M	Soyuz	Soyuz
Outboard	Space Shuttle	Atlantis
TM-31	Soyuz	Soyuz

Of course, since we have Normalised the base relations, the data in the View is Normalised (but flattened, by definition).

- Thus we prove that the data demanded by the user (or dumb teacher or schizophrenic author), presented as a "table", is not such. It is merely the data as perceived by non-technical people, that technical people must (a) differentiate base vs derived relations from, and (b) Normalise the base relations only.
- And we prove that the data demanded, is entirely possible, easy to project, from the Normalised base relations.

C.2 Crew Normalisation

1. **Initial**

   Appreciate that the data given (falsely, in tabular from) is merely an user perception of the data, not a table, and it needs Normalisation.

   Perception of Crew Data

CallSign
Astreus
Altair

   The evidence that it is not Normalised, that it is not Relational is, there is no valid **Identifier**. The *Relational Model* demands that the data can be Identified, in order to form row uniqueness. A `CallSign` that is optional is not a valid Identifier.

2. **Surrogate Fail**

   Now it is obvious to me, that the "teacher" is attempting to force you into a position in which you have to use a surrogate to identify the data, and thus validate the false notion that surrogates are valid.

   That is easily proved false, because (as you know from **this Answer**) surrogates do not provide the **row uniqueness** demanded by the *Relational Model*.

   Crew File

CrewID	CallSign
98765	Astreus
98766	Altair
98767	
54321	Altair
54322	Altair
54323	
54324	Astreus

   As evidenced in the example I have given, the surrogates provide unique Record `IDs`, woo hoo, but provide no uniqueness in the data rows.

   Once again, the surrogate fails. It does not Identify the data, only data can Identify the data

   Once again, if used, the surrogate would be superfluous, providing nothing of value.

3. **Add Identifying Element**

   We need the data to Identify the data. An optional `CallSign` fails to Identify the data. While I would love to stick to the assignment requirement, in this case we can't, because data given in the requirement is grossly inadequate.

   We therefore must add elements to the data, such that we can uniquely Identify it, and differentiate each row from other data rows.

   Now, in the real world, `Crew` is a subset of `Person`, and all `Persons` have a first and last name, which is commonly used to Identify `Persons`. The idea of `Crew`, especially astronauts, who are famous, not having a first and last name is not reasonable.

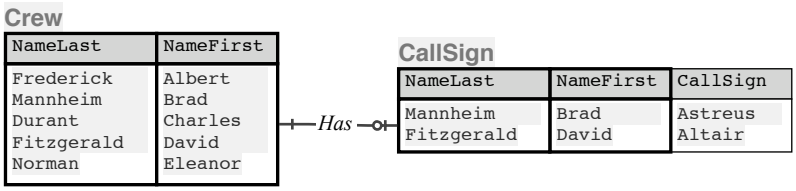
   Demanded Crew Data

NameLast	NameFirst	CallSign
Frederick	Albert	
Mannheim	Brad	Astreus
Durant	Charles	
Fitzgerald	David	Altair
Norman	Eleanor	

   Thus we add a `NameLast` and `NameFirst` to the perception of `Crew` data, in order to be able to Identify `Crew`, and to provide uniqueness for each `Crew` data row.

4. Normalise the Data

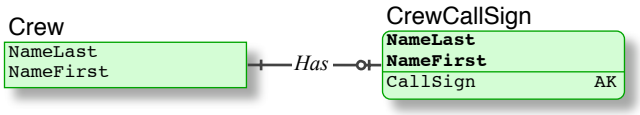
Retaining the contrivance that `CallSign` is optional, for the assignment, even though the notion of a `Crew` member without one is absurd.



The data is easily Normalised into a `Crew` table, and a `CallSign` table for the optional column.

5. Model the Data

The model is easy enough to construct. `Crew` is a Fact, and `CrewCallSign` is a separate and optional Fact.



This exposes the issue that the `CallSign` column does not have uniqueness, and it should have (the requirement given by the "teacher" is inadequate). We implement that as an Alternate Key. Now when we call a `Crew` member by `CallSign`, we get one and only one `Crew`, instead of some unknown number.

6. Construct Predicate

The model exposes the Predicates, and the Predicates are used to verify the Model. The model [5] is further confirmed.

```
Crew is independent
      Crew is identified by ( NameLast, NameFirst )
      Crew has 0-or-1 CrewCallSign
CrewCallSign is dependent on, and identified by, Crew
      CrewCallSign belongs to 1 Crew
      CrewCallSign is primarily   identified by ( NameLast, NameFirst )
      CrewCallSign is alternately identified by ( CallSign )
```

- `Crew` exists independently, eg. of `Spaceship`.
- `CrewCallSign` is dependent on `Crew`, ie. it exists only in the context of a `Crew` row.

C.3 Flight Normalisation

1. **Initial**

   Appreciate that the data given (falsely, in tabular from) is merely an user perception of the data, not a table, and it needs Normalisation.

Perception of Flight Data

ShipName	FlightName	FlightNum
Soyuz TM-31	NULL	1
Atlantis	STS-117	28
Soyuz TMA-14	NULL	1
Endeavour	STS-126	22
Soyuz TMA-15M	NULL	1
Endeavour	STS-111	18
Atlantis	STS-122	29

   The evidence that it is not Normalised, that it is not Relational is, the data values are in un-normalised form, that we have since Normalised under **Spaceship Normalisation**.

   Further, users do not use `NULL`. If they see `NULL` in a report, it confuses them, and then the `NULL` has to be explained. We can take it that the requirement is, American `Flights` are named, but Russian `Flights` are not. Unlike the "teacher", we do not have to present confusing information to the user.

2. **Surrogate Fail**

   Now it is obvious to me, the evidence is, once again, that the "teacher" is attempting to force you into a position in which you have to use a surrogate to identify the data (the `NULL FlightName`, and the `FlightNum` is not unique), and thus validate the false notion that surrogates are valid.

   That is easily proved false, because (as you know from **this Answer**) surrogates do not provide the **row uniqueness** demanded by the *Relational Model*.

Flight File

FlightID	ShipName	FlightName
6661	Soyuz TM-31	NULL
6662	Soyuz TM-31	NULL
6663	Soyuz TM-31	NULL
6664	Endeavour	STS-126
6665	Endeavour	STS-126
6666	Endeavour	STS-111
6667	Endeavour	STS-111

   As evidenced in the example I have given, the surrogates provide unique Record `IDs`, woo hoo, but provide no uniqueness in the data rows.

   Once again, the surrogate fails. It does not Identify the data, only data can Identify the data

   Once again, if used, the surrogate would be superfluous, providing nothing of value.

3. **Flight is a Dependent Fact**

   We need the data to Identify the data. An optional `FlightName` causes the given data to be inadequate in terms of data uniqueness.

   - We can't address your description "`ShipName`, `FlightName` are a Composite PK", because the basis of the existence of a `Flight` is incorrect.

   While I would love to stick to the assignment requirement, in this case we can't, because data given in the requirement is grossly inadequate, and it has ulterior motives.

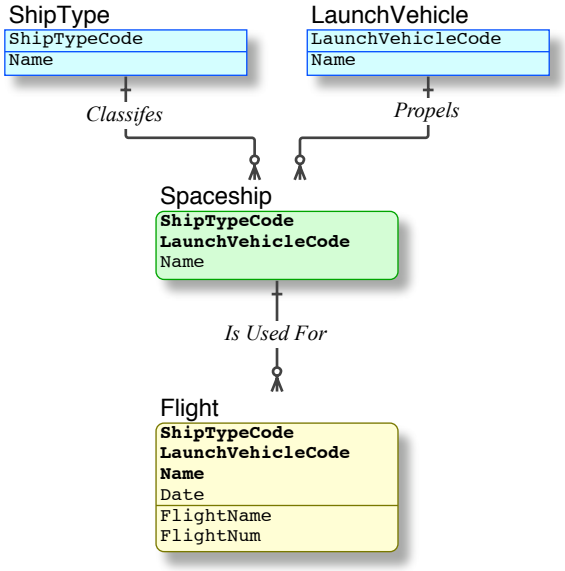
   Here we don't need to add elements to the data, such that we can uniquely Identify it, here we only need to assert that the Fact of a `Flight` is not independent (they would be, if a surrogate is forced, but such an assertion is false).

   `Flights` are not independent. They cannot occur without a `Spaceship`. The truth is, the Fact of a `Flight` is dependent on the Fact of a `Spaceship`, it simply cannot occur without one.

   Now. in the real world of ISS flight, a `Spaceship` can take flight at most once on any given day. Given that a `Flight` is Dependent on `Spaceship`, and nothing else, the Identifier for `Flight` is therefore the `Spaceship` Primary Key, plus Date.

   When that Identifier is used, the nullability of `FlightName`, and the non-uniqueness of `FlightNum`, are demoted to non-issues, plain attributes, because they have no relevance pertaining to the Identifier. The trickery used to force you into a non-relational filing systems is puncture.

4. Model the Data



I won't take you through all the steps, since they are adequately explained in the previous examples above, and you can execute them yourself.

5. Construct Predicate

The model exposes the Predicates, and the Predicates are used to verify the Model. The model [4] is further confirmed.

```
LaunchVehicle is independent
     LaunchVehicle is identified by ( LaunchVehicleCode )
     LaunchVehicle propels 0-to-n Spaceships
     LaunchVehicle is described by ( Name)
ShipType is independent
     ShipType is identified by ( ShipTypeCode )
     ShipType classifies 0-to-n Spaceships
     ShipType is described by ( Name )
Spaceship is dependent on, and identified by, LaunchVehicle
     Spaceship is propelled by 1 LaunchVehicle
     Spaceship is dependent on, and identified by, ShipType
     Spaceship is classified by 1 ShipType
     Spaceship is identified by ( ShipTypeCode, LaunchVehicleCode, Name )
     Spaceship is used for 0-to-n Flights
Flight is dependent on, and identified by, Spaceship
     Flight uses 1 Spaceship
     Flight is identified by ( ShipTypeCode, LaunchVehicleCode, Name, Date )
     Flight is described by ( FlightName, FlightNum )
```

6. Derived Relation, View

Now we can prove the data model *even further*, by demonstrating that any and all derived Relations can be projected from the base relations (tables, implemented).

Flight_View

ShipType	LaunchVehicle	Name	FlightName	FlightNum
Soyuz	TM-31	Soyuz		1
Space Shuttle	Outboard	Atlantis	STS-117	28
Soyuz	TMA-14	Soyuz		1
Space Shuttle	Outboard	Endeavour	STS-126	22
Soyuz	TMA-15	Soyuz		1
Space Shuttle	Outboard	Endeavour	STS-111	18
Space Shuttle	Outboard	Atlantis	STS-122	29

This is the `Flight` View for the `Flight` base relation, or table, minus the Code columns.

7. Initial Perception is View

Now we can evaluate the Initial Perception of the `Flight` Data, against the relevant columns in the `Flight`. Of course, since we have Normalised the base relations, the data in the View is Normalised (but flattened, by definition).

- Thus we prove that the data demanded by the user , is not such. It is merely the data as perceived by non-technical people, that technical people must (a) differentiate base vs derived relations from, and (b) Normalise the base relations only.
- And we prove that the data demanded, is entirely possible, easy to project, from the Normalised base relations.

C.4 Role Normalisation

1. **Initial**

   Appreciate that the data given (falsely, in tabular from) is merely an user perception of the data, not a table, and it needs Normalisation.

Perception of Designation Data

ShipName	FlightName	CrewId	CrewType
Soyuz TM-15M	NULL	4243	Deliver
Soyuz TM-15M	NULL	4243	Return
Soyuz TM-15M	NULL	4445	Backup
Soyuz TM-16M	NULL	4344	Deliver
Soyuz TM-17M	NULL	4445	Deliver
Soyuz TM-18M	NULL	4344	Return
Endeavour	STS-111	55	Deliver
Endeavour	STS-112	44	Return
Endeavour	STS-113	55	Return

   The evidence that it is not Normalised, that it is not Relational is, the data values are in un-normalised form, that we have since Normalised under **Spaceship Normalisation**, **Crew Normalisation**, and **Flight Normalisation**.

   Further, Designation and `CrewAssignment` data are nicely jumbled into one perception or view. Let's deal with one thing at a time, in a scientific manner, with the dependencies in the correct order.

2. **Remove Repeating Group C**

   We can see that there is another series of repeating data in `CrewType`. Further, the label `CrewType` is not accurate, it does not indicate a Type of `Crew` (such as a Qualification), it indicates a role that is assigned for the `Crew`, for each `Flight` (separate to their Qualification). Let's label it accurately, as `Role`.

   Note the careful choice of words: design and assign are verbs; role is a **noun**. The tables are nouns, the relationships are the **actions** between the nouns, hence the **Verb Phrase**.

Role

RoleCode	Name
B	Backup
D	Deliver
R	Return

Designates

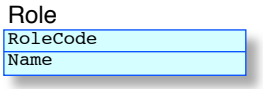
Perception of Designation Data

ShipName	FlightName	CrewId	RoleCode	Role
Soyuz TM-15M	NULL	4243	D	Deliver
Soyuz TM-15M	NULL	4243	R	Return
Soyuz TM-15M	NULL	4445	B	Backup
Soyuz TM-16M	NULL	4344	D	Deliver
Soyuz TM-17M	NULL	4445	D	Deliver
Soyuz TM-18M	NULL	4344	R	Return
Endeavour	STS-111	55	D	Deliver
Endeavour	STS-112	44	R	Return
Endeavour	STS-113	55	R	Return

   We remove the repeating group in the Designation perception, we Normalise it into a reference table Designation. A third data classification treated correctly.

   You have done that, intuitively, in your **One More Update**. I have given the precise steps, and corrected the label.

3. **Model the Data**



   Modelling that could not be simpler.

4. **Construct Predicate**

   The Predicates too, are simple.

```
Role is independent
     Role is identified by ( RoleCode )
     Role is described by ( Name)
```

   The model [3] is further confirmed.

C.5 CrewAssignment Normalisation

1. **Initial**

   Appreciate that the data given (falsely, in tabular from) is merely an user perception of the data, not a table, and it needs Normalisation.

   Now that we have subtracted Role from the Designation perception, which was all jumbled (**Role Normalisation.1**), we have just the data that relates to the assignment of the crew, that remains for Normalisation. Responding to that jumble would (a) validate it, and (b) take much more time to resolve.

   Rather than dealing with that scrambled perception, given that we have already Normalised and verified most of the data concerned, on that basis, we can simply continue our Normalisation, for the remaining un-normalised data.

2. **CrewAssignment is a Dependent Fact**

   Once again, here we only need to assert that the Fact of a `CrewAssignment` is not independent (it would be, if a surrogate is forced, but such an assertion is false).

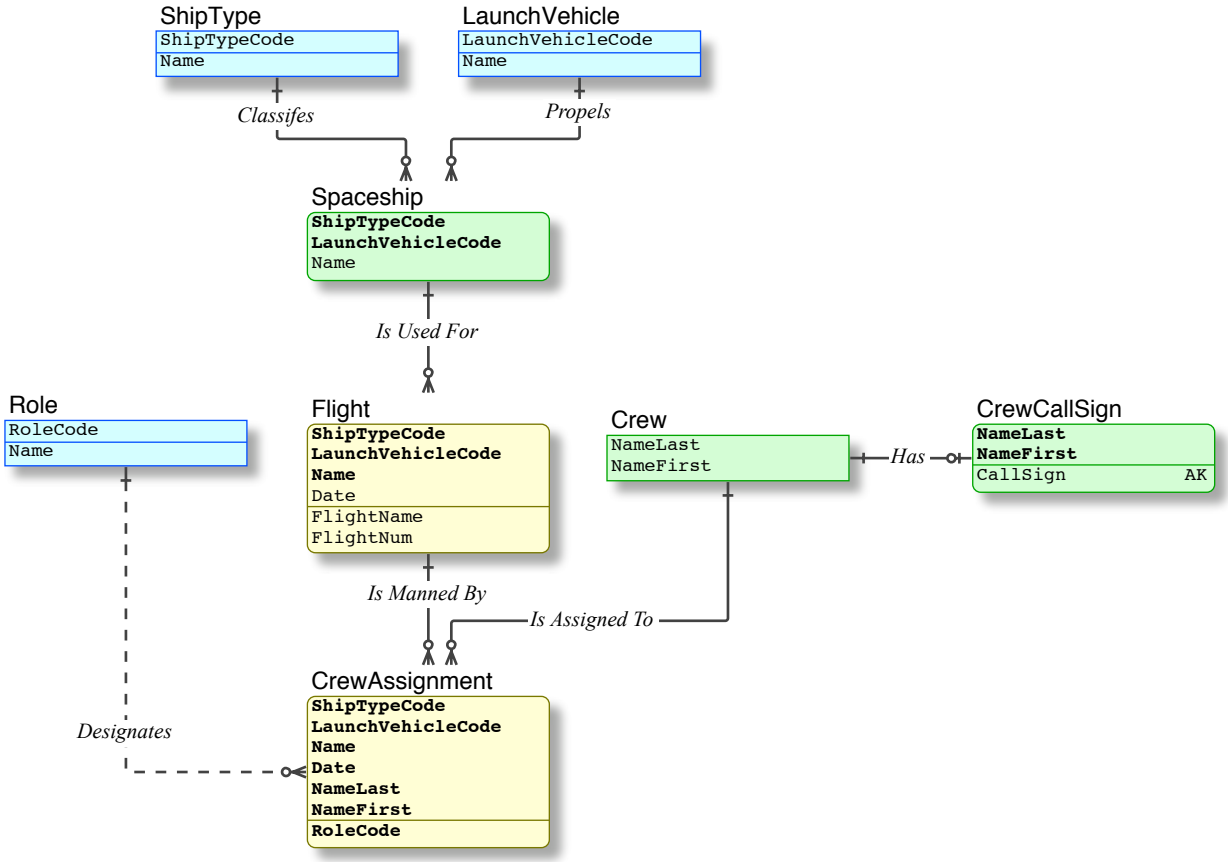
   `CrewAssignment` is not independent. The truth is, the Fact of a `CrewAssignment` is dependent on the Fact of a `Flight` having been scheduled, it should not occur without one.

   Given that `CrewAssignment` is Dependent on `Flight`, the Identifier for `CrewAssignment` is therefore the `Flight` Primary Key, plus something.

   `CrewAssignment` is dependent on `Crew` as well. The truth is, the Fact of a `CrewAssignment` is dependent on the Fact of a `Crew` having been established, it should not occur without one.

   Given that `CrewAssignment` is Dependent on `Crew`, the Identifier for `CrewAssignment` is therefore the `Crew` Primary Key, plus something.

3. **Model the Data**



   Since `CrewAssignment` is dependent on both `Flight`, and `Crew`, it is an **Associative Table**.

   As such, the Primary Key is the combination of the Primary Keys of the two parent tables.

   A surrogate cannot *replace* the Primary Key, which provides **row uniqueness**, and thus cannot be removed. A surrogate would add nothing, it would be superfluous.

4. Construct Predicate

The model exposes the Predicates, and the Predicates are used to verify the Model. The model [3] is further confirmed.

```
Crew is independent
      Crew is identified by ( NameLast, NameFirst )
      Crew has 0-or-1 CrewCallSign
      Crew is assigned to 0-to-n CrewAssignments
CrewCallSign is dependent on, and identified by, Crew
      CrewCallSign belongs to 1 Crew
      CrewCallSign is primarily   identified by ( NameLast, NameFirst )
      CrewCallSign is alternately identified by ( CallSign )
Role is independent
      Role is identified by ( RoleCode )
      Role is described by ( Name)
      Role designates 0-to-n CrewAssignments
LaunchVehicle is independent
      LaunchVehicle is identified by ( LaunchVehicleCode )
      LaunchVehicle propels 0-to-n Spaceships
      LaunchVehicle is described by ( Name )
ShipType is independent
      ShipType is identified by ( ShipTypeCode )
      ShipType classifies 0-to-n Spaceships
      ShipType is described by ( Name )
Spaceship is dependent on, and identified by, LaunchVehicle
      Spaceship is propelled by 1 LaunchVehicle
      Spaceship is dependent on, and identified by, ShipType
      Spaceship is classified by 1 ShipType
      Spaceship is identified by ( ShipTypeCode, LaunchVehicleCode, Name )
      Spaceship is used for 0-to-n Flights
Flight is dependent on, and identified by, Spaceship
      Flight uses 1 Spaceship
      Flight is identified by ( ShipTypeCode, LaunchVehicleCode, Name, Date )
      Flight is described by ( FlightName, FlightNum )
      Flight is manned by 0-to-n CrewAssignments
CrewAssignment is dependent on, and identified by, Flight
      CrewAssignment is a manning of 1 Flight
     CrewAssignment is dependent on, and identified by, Crew
      CrewAssignment is an assignment of 1 Crew
      CrewAssignment is a designation of 1 Role
      CrewAssignment is identified by ( ShipTypeCode, LaunchVehicleCode, Name, Date,
      NameLast, NameFirst )
```

- `CrewAssignment` is an Associative table, dependent on `Flight`, and on `Crew`
- `CrewAssignment` is Identified by `Flight` PK and `Crew` PK
- `CrewAssignment.Role` describes `CrewAssignment`.
-

5. Derived Relation, View

Now we can prove the data model *even further*, by demonstrating that any and all derived Relations can be projected from the base relations (tables, implemented).

CrewAssignment_View

ShipType	LaunchVehicle	Name	FlightName	NameLast	NameFirst	Role
Soyuz	TM-15M	Soyuz		Pppppp	Ppp	Deliver
Soyuz	TM-15M	Soyuz		Pppppp	Ppp	Return
Soyuz	TM-15M	Soyuz		Qqqqqq	Qqq	Backup
Soyuz	TM-16M	Soyuz		Rrrrrr	Rrr	Deliver
Soyuz	TM-17M	Soyuz		Qqqqqq	Qqq	Deliver
Soyuz	TM-18M	Soyuz		Rrrrrr	Rrr	Return
Space Shuttle	Outboard	Endeavour	STS-111	Ssssss	Sss	Deliver
Space Shuttle	Outboard	Endeavour	STS-112	Tttttt	Ttt	Return
Space Shuttle	Outboard	Endeavour	STS-113	Ssssss	Sss	Return

This is the projection or derived relation or `CrewAssignment` View for the `CrewAssignment` base relation, or table, minus the Code columns.

6. Initial Perception is View

Now we can evaluate the Initial Perception of the Designation Data, against the relevant columns in the `CrewAssignment` View. Of course, since we have Normalised the base relations, the data in the View is Normalised (but flattened, by definition).

- Thus we prove that the data demanded by the user, presented as a table, is not such. It is merely the data as perceived by non-technical people, that technical people must (a) differentiate base vs derived relations from, and (b) Normalise the base relations only.
- And we prove that the data demanded, is entirely possible, easy to project, from the Normalised base relations.

D.1 Question • First Set

In the progression above, I trust that I have explained, using your example, step by step, the essential elements, the scientific considerations, from the *Relational Model*, as well as giving the method, that relate to your first set of questions:

> I wish to know how to correctly set up Primary Keys in a [Base] Relation. E.g. we have ER-diagram [and example] which contain elements:
>
> 1. Key attributes
> 2. Weak key attributes
> 3. Identifying [and Non-identifying] relationships
> 4. Associative entities
>
> ... can we leave them as is or ~~replace with~~ [add] Surrogate ~~Key~~ s.
>
> How to set up Primary Keys for Relations in order to handle all above Cases (perhaps some more Cases which I did not mention)?
>
> How to avoid using Surrogate Keys and in which Cases they're necessary?

Further, I specifically gave the differentiation re base vs derived relations, such that the usual confusion of post-Codd authors and the "teachers" that follow them, is eliminated.

Relational Key vs Surrogate

I have provided complete coverage of this issue, at each step, and that should suffice to eliminate discussion about same.

Note that in science, truth is objective, not subjective. Thus there is no argument to be had, no consensus to be reached. Those who argue about it, and those who seek consensus, by the evidence that I have provided here, are simply ignorant of the science, and particularly of the *Relational Model*. When I am engaged, I don't argue, I simply provide education, which eliminates the argument.

Date; Darwen; Fagin; Fowler; Ambler; Kimball; Celko; Abitebul; Hull; Vianu, and their followers, such as Hidders; Köhler; and the hundreds of "professors", have a lot to answer for. Anyone who understands what I have written above can use the material to prove each and every one of them to be (a) ignorant of the science, (b) ignorant of the *Relational Model* that they allege to be promoting, and (c) and dead wrong.

By science and straight-forward logic (above), the facts are:

1. Relational keys provide:
   - Relational Integrity (as distinct from Referential integrity). This item alone, once understood, kills any argument.
   - Relational Power (`JOIN` power at the level of your question, meaning fewer `JOINs` to get at the data, in stark contradiction to the magical mystical myths that state [but do not prove] that Relational Keys result in more `JOINs`),
   - Relational Speed, that cannot be obtained by other methods. Two to three orders of magnitude faster than the Record ID systems it replaced.
2. Record Filing Systems, typified by:
   - Record `IDs` as "primary key" in every file
   - Every file is independent (dependencies are not supported)
   - Every relationship is Non-identifying (there are no Relational Keys),

have none of that. No Relational Integrity (it does have Referential Integrity of records, but not of rows), no Relational Power due to the violation of the **Access Path Independence Rule** (more `JOINs` on every access), and no Relational Speed (it has the speed of pre-1970's ISAM filing systems).

The knee-jerk implementation of surrogates on every file remains dead wrong. Further (in addition to the above), it cripples the modelling exercise, and thus prevents the genuine modelling of Facts (the database is a collection of Facts), the result being not only a Record Filing system, but one where the files are not Normalised to Facts. The consequence is massive data duplication, and an RFS that is difficult to use, to write code for.

If that evidence is not enough, if you [or anyone else] want further proof, please ask a new question, and ping me.

Please note, that is not to say that surrogates do not have any use. It is not a simplistic black-or-white issue. They do have an use. Codd spent considerable time examining them, and refuting them (those papers are fraudulently used to portray that Codd validated surrogates in generic terms). But the valid use of surrogates is far beyond the question, and the detail covered in the answer.

D.2 Question • Second Set

How to set up datatypes for Primary Keys

Good question. I will answer in terms of the Standard, much like the Standard for Naming of components in a Relational Database.

You should set up, and use, a set of Datatypes to cover all the Domains (*Relational* term) in the database. These are the Building Blocks of your database. Domain definition is the first level of Constraint on data. Conversely, there should not be a column in the database that is a raw Datatype.

Your carefully chosen Datatypes should be used in all SQL code as well. This prevents entire category of errors. One worth mentioning is:

- Datatype Mismatch. Easily the most common error, often hard to determine, and the easiest to prevent. The Datatypes are defined once, in the central location where they belong, the database, and used by everything that uses the database.

Primary Keys, and indeed the components of a Primary Key, are not simple attributes. They have qualities that plain attributes do not have. Further they are used in ways that attributes are not (eg. for `JOINs`). Thus they qualify for special treatment.

- Each component of a Primary Key should have a private Datatype defined for it.
- This Datatype should be used wherever the Primary Key is referenced (ie. migrated as a Foreign Key).
- Further, note that this documents the fact that for Relational keys, every reference of a Primary Key (ie. migrated as a Foreign Key) is directly related to the Primary Key.
- **Hint**: Direct `JOIN`, without navigating the intervening tables.

If a Composite Primary Key has to be passed into Child Relation, shall it be replaced with Surrogate?

No. Answered in detail above.

Advantages and disadvantages of using Surrogate ~~Key~~ s in my view:

Advantages

They're compact (usually of type INT) and are sometimes good replacement for Composite Keys

Compact is an irrelevant consideration. We no longer work in Kilobytes and Megabytes, we work in Gigabytes. The power of machines is no longer measured in Megahertz, it is measured in Gigahertz. SQL platforms (not counting the NONsql s/w fraudulently using the name of the SQL Standard without complying with it) are more than capable of handling Relational Keys.

- In case you are not aware, the SQL platforms are heavily optimised for Relational Database access. Even little old myNONsql provides a Clustered Index, at least up at the Big House.
- No optimisation is possible for surrogates, for Record Filing Systems.

If you want a Relational Database, ie. Relational Integrity; Relational Power; Relational Speed, then you need composite Keys. That is all there is to it. The consideration is irrelevant

If you are happy with a Record Filing System, by all means, use one, and then you don't need Relational Keys, use surrogates.

They're illustrative when they're in Foreign Keys

First, they are not illustrative, in any way, shape, or form. Relational Keys are illustrative, especially when they are Foreign Keys.

- In the case that your definition of *illustrative* is different to mine, then in Foreign Keys, they are no more nor less *illustrative* than Relational Keys.

Second, you are contradicting your statement (below), that they are meaningless. I agree that they are meaningless numbers.

They're painlessly indexed

Relational Keys suffer no more, and no less pain, in being indexed. On SQL platforms.

Disadvantages

They're numbers and meaningless. E.g. I wish to fill up ~~Junction~~ [Associative] Table in my Interface Application - so I will be left no other choice but to relate just numbers

They're confusing

That is correct. Numbers have no meaning. Users hate them. Developers love them in simple examples (starting from 1, such as in questions on SO), but once they are faced with 6- and 8-digit numbers, they too, hate them, and yearn for meaningful codes and strings.

Relational Keys have meaning, and carry that meaning wherever they are migrated.

They're redundant

If you understand that a surrogate is not a *replacement* for a compound Key, then yes, they are redundant.

When used correctly (which is rare, and the condition is beyond the scope of this question), no, they are not redundant.

If you don't understand that a surrogate is not a *replacement* for a compound Key, then you will be flooded with duplicate rows, and surrogates are not redundant, they are the single column "primary key" that guarantees the flood.

The **Two Main Disadvantages**, that make them untenable for Relational Database, that silence any argument, have not been mentioned.

1. **Loss of Relational Integrity**

   Surrogates are physical pointers to Record `IDs`, which means they have "integrity" re Record `IDs`, but not the content of those records, the rows. Thet have no Relational Integrity[3].

   Relational Keys are Logical, they form the logical structure of the database, they have meaning and that meaning is carried wherever the Key is carried. In so doing, they maintain context and logical Integrity between the rows (regardless of records).

2. **Loss of Relational Power**

   Loss of `JOIN` power. Due to violation of the **Access Path Independence Rule**, which is a requirement of the *Relational Model*.

   Every file between any two distal files that are queried, and that are related, have to be navigated, in order to "link" the two subject files.

   Whereas in a Relational Database, the two tables can be `JOINed` directly, without navigating the intervening tables. The Logical key *is* the Navigator.

As for setting up datatypes - there must be more tricks as well as setting up Primary Keys as whole.

No tricks from me. I have given the requirement for Datatypes for Compound Primary Keys above.

But treating Compound Primary Keys as a single unit, no, that is dead wrong. In each instance that it is located (ie. including their migration, as Foreign Keys), you must be able to address each component of the Key, directly and simply. You have that now. Do not complicate it.

Therefore to place it in a complex Datatype, is wrong.

- Complex datatypes have an use, for external services, such as `XML_EXTRACT`. They have no use within the database.

Please study carefully, and comment, etc. If there is anything that is not crystal clear, or that needs expansion, please ask.

If you have trouble locating Codd's papers, go to my profile, and email me.

I will respond to your comments as time permits.