Relational table naming convention

[+173] [5] Andreas

[2011-01-15 23:17:05]

[database database-design coding-style naming-conventions relational-database]

[https://stackoverflow.com/questions/4702728/relational-table-naming-convention]

I'm starting a new project and would like to get my table- and column names right from the start. For example I've always used plural in table names but recently learned singular is correct.

So, if I got a table "user" and then I got products that only the user will have, should the table be named "user_product" or just "product" ? This is a one to many relationship.

And further on, if i would have (for some reason) several product descriptions for each product, would it be "user_product_description" or "product_description" or just "description"? Of course with the right foreign keys set.. Naming it only description would be problematic since i could also have user description or account description or whatever..

What about if i want a pure relational table (many to many) with only two columns, what would this look like? "user_stuff" or maybe something like "rel_user_stuff" ? And if the first one, what would distinguish this from, for example "user_product"?

Any help is highly appreciated and if there is some sort of naming convention standard out there that you guys recommend, feel free to link.

Thanks

(25) (a) The question was asked and answered four years ago. (b) both the question and the selected answer have high votes. (c) We have a naming-conventions tag (d) it may be "opinion-based" for a beginner to answer, but it is a matter of Standards to experienced technical people, whom the seeker is seeking. Nevertheless, reasons are given for each of the many prescriptions. (e) Therefore, by virtue of the evidence, `primarily opinion-based` is patently false. - **PerformanceDBA**

it is a matter of Standards to experienced technical people Or to people who came across the ancient IDEF standards and believe that they are actual Standards. - **gbr**

(1) Further, there are several other Qs re Naming Conventions, all do have a value. Refer Linked in the column on the right.. - **PerformanceDBA**

(1) @gbr. IT people get their current Standards from ISO. IDEF1X was last confirmed by ISO in 2019. - **PerformanceDBA**

[+434] [2011-01-16 00:59:29] PerformanceDBA [✔ACCEPTED]

Table • Name

> *recently learned singular is correct*

Yes. Beware of the heathens. Plural *in the table names* are a sure sign of someone who has not read any of the standard materials and has no knowledge of database theory.

Some of the wonderful things about Standards are:

- they are all integrated with each other
- they work together
- they were written by minds greater than ours, so we do not have to debate them.

The standard table name refers to each **row** in the table, which is used in the all verbiage, not the total content of the table (we know that the `Customer` table contains all the Customers).

Relationship, Verb Phrase

In genuine Relational Databases that have been modelled (as opposed to pre-1970's Record Filing Systems [characterised by `Record IDs` which are implemented in an SQL database container for convenience):

- the tables are the **Subjects** of the database, thus they are *nouns*, again, singular
- the relationships between the tables are the *Actions* that take place between the nouns, thus they are *verbs* (i.e they are not arbitrarily numbered or named)
- that *is* the **Predicate**
- all that can be read directly from the data model (refer my examples at the end)
- (the Predicate for an independent table (the top-most parent in an hierarchy) is that it is independent)
- thus the **Verb Phrase** is carefully chosen, so that it is the most meaningful, and generic terms are avoided (this becomes easier with experience). The Verb Phrase is important during modelling because it assists in resolving the model, ie. clarifying relations, identifying errors, and correcting the table names.

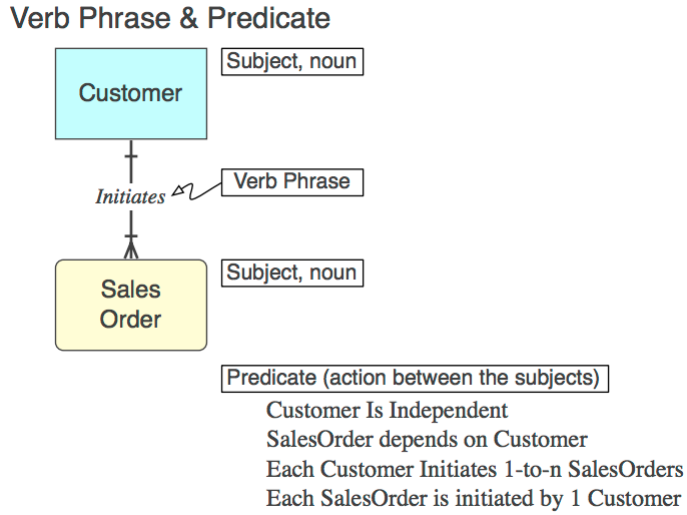


Diagram A [1]

Of course, the relationship is implemented in SQL as a `CONSTRAINT FOREIGN KEY` in the child table (more, later). Here is the **Verb Phrase** (in the model), the **Predicate** that it represents (to be read from the model), and the FK **Constraint Name**:

```
Initiates
Each Customer Initiates 0-to-n SalesOrders
Customer_Initiates_SalesOrder_fk
```

Table • Language

However, *when describing* the table, particularly in technical language such as the Predicates, or other documentation, use singular and plurals as they naturally in the English language. Keeping in mind the table is named for the single row (relation) and the language refers to each derived row (derived relation):

```
Each Customer initiates zero-to-many SalesOrders
```

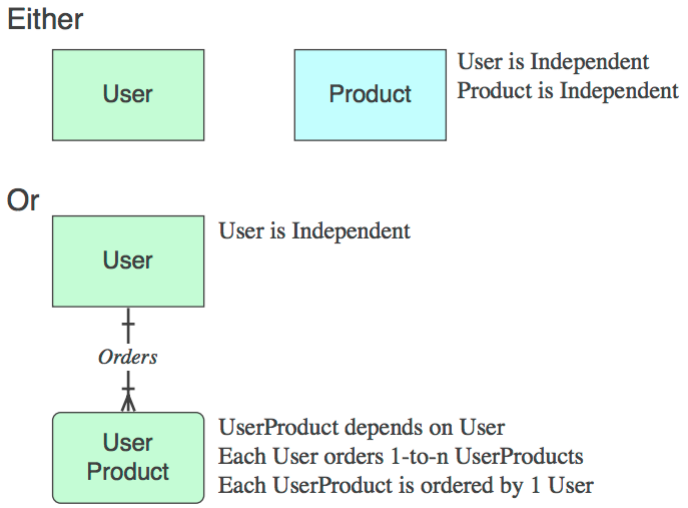
not

```
Customers have zero-to-many SalesOrders
```

> *So, if I got a table "user" and then I got products that only the user will have, should the table be named "user-product" or just "product"? This is a one to many relationship.*

(That is not a naming-convention question; that is a a db design question.) It doesn't matter if `user::product` is 1::n. What matters is whether `product` is a separate entity and whether it is an **Independent Table**, ie. it can exist on its own. Therefore `product`, not `user_product`.

And if `product` exists only in the context of an `user`, ie. it is a **Dependent Table**, therefore `user_product`.



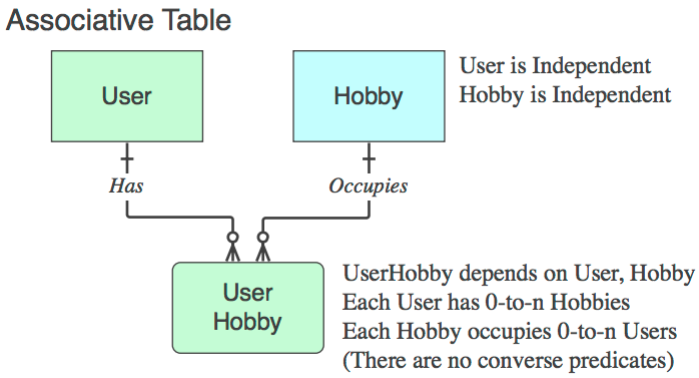
Diagram_B [2]

> *And further on, if i would have (for some reason) several product descriptions for each product, would it be "user-product-description" or "product-description" or just "description"? Of course with the right foreign keys set.. Naming it only description would be problematic since i could also have user description or account description or whatever.*

That's right. Either `user_product_description` xor `product_description` will be correct, based on the above. It is not to differentiate it from other `xxxx_descriptions`, but it is to give the name a sense of where it belongs, the prefix being the parent table.

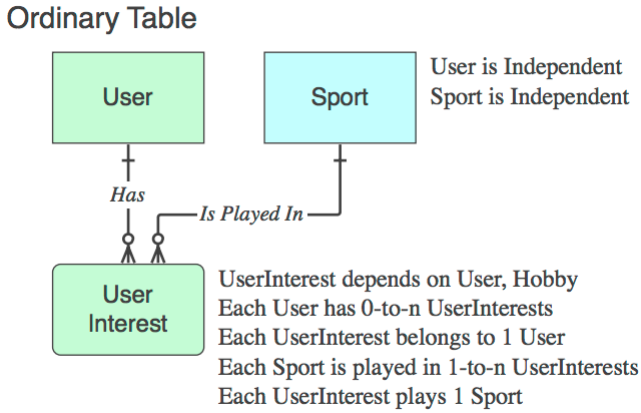
> *What about if i want a pure relational table (many to many) with only two columns, what would this look like? "user-stuff" or maybe something like "rel-user-stuff" ? And if the first one, what would distinguish this from, for example "user-product"?*

1. Hopefully all the tables in the relational database are pure relational, normalised tables. There is no need to identify that in the name (otherwise all the tables will be `rel_something`).
2. If it contains **only** the PKs of the two parents (which resolves the *logical* n::n relationship that does not exist as an entity at the logical level, into a *physical* table), that is an **Associative Table**. Yes, typically the name is a combination of the two parent table names.
    - Note that is such cases the Verb Phrase applies to, and is read as, from parent to parent, ignoring the child table, because its only purpose in life is to relate the two parents.



Diagram_C [3]

- If it is *not* an Associative Table (ie. in addition to the two PKs, it contains data), then name it appropriately, and the Verb Phrases apply to it, not the parent at the end of the relationship.



Diagram_D [4]

3. If you end up with two `user_product` tables, then that is a very loud signal that you have not normalised the data. So go back a few steps and do that, and name the tables accurately and consistently. The names will then resolve themselves.

Naming Convention

> *Any help is highly appreciated and if there is some sort of naming convention standard out there that you guys recommend, feel free to link.*

What you are doing is very important, and it will affect the ease of use and understanding at every level. So it is good to get as much understanding as possible at the outset. The relevance of most of this will not be clear, until you start coding in SQL.

1. **Case** is the first item to address. All caps is unacceptable. Mixed case is normal, especially if the tables are directly accessible by the users. Refer my data models. Note that when the seeker is using some demented NonSQL, that has only lowercase, I give that, in which case I include underscores (as per your examples).

2. Maintain a **data focus**, not an application or usage focus. It is, after all 2011, we have had **Open Architecture** since 1984, and databases are supposed to be independent of the apps that use them.

That way, as they grow, and more than the one app uses them, the naming will remain meaningful, and need no correction. (Databases that are completely embedded in a single app are not databases.) Name the data elements as data, only.

3. Be very considerate, and name tables and columns very **accurately**. Do not use `UpdatedDate` if it is a `DATETIME` datatype, use `UpdatedDtm`. Do not use`_description` if it contains a dosage.

4. It is important to be **consistent** across the database. Do not use `NumProduct` in one place to indicate number of Products and `ItemNo` or `ItemNum`in another place to indicate number of Items. Use `NumSomething` for numbers-of, and `SomethingNo` or `SomethingId` for identifiers, consistently.

5. Do not prefix the column name with a table name or short code, such as `user_first_name`. SQL already provides for the tablename as a qualifier:

```
table_name.column_name  -- notice the dot
```

6. Exceptions:

    - The first exception is for PKs, they need special handling because you code them in joins, all the time, and you want keys to stand out from data columns. Always use `user_id`, never `id`.

        - Note that this is *not* a table name used as a prefix, but a proper descriptive name for the component of the key: `user_id` is the column that identifies an user, not the `id` of the `user` table.
            - (Except of course in record filing systems, where the files are accessed by surrogates and there are no relational keys, there they are one and the same thing).
        - Always use the exact same name for the key column wherever the PK is carried (migrated) as an FK.
        - Therefore the `user_product` table will have an `user_id` as a component of its PK `(user_id, product_no)`.
        - the relevance of this will become clear when you start coding. First, with an `id` on many tables, it is easy get mixed up in SQL coding. Second, anyone other that the initial coder has no idea what he was trying to do. Both of which are easy to prevent, if the key columns are treated as above.

    - The second exception is where there is more than one FK referencing the same parent table table, carried in the child. As per the *Relational Model*, use **Role Names** to differentiate the meaning or usage, eg. `AssemblyCode` and `ComponentCode` for two `PartCodes`. And in that case, do **not** use the undifferentiated `PartCode` for one of them. Be precise.

      Diagram_E [5]

7. **Prefix**
   Where you have more than say 100 tables, prefix the table names with a Subject Area:

   `REF_` for Reference tables
   `OE_` for the Order Entry cluster, etc.

   Only at the physical level, not the logical (it clutters the model).

8. **Suffix**
   Never use suffixes on tables, and always use suffixes on everything else. That means in the logical, normal use of the database, there are no underscores; but on the administrative side, underscores are used as a separator:

`_V` View (with the main `TableName` in front, of course)
`_fk` Foreign Key (the constraint name, not the column name)
`_cac` Cache
`_seg` Segment
`_tr` Transaction (stored proc or function)
`_fn` Function (non-transactional), etc.

The format is the table or FK name, an underscore, and action name, an underscore, and finally the suffix.

This is really important because when the server gives you an error message:

```
_____blah blah blah error on object_name
```

you know exactly what object was violated, and what it was trying to do:

```
_____blah blah blah error on Customer_Add_tr
```

9. **Foreign Keys** (the constraint, not the column). The best naming for a FK is to use the Verb Phrase (minus the "each" and the cardinality).

   ```
   Customer_Initiates_SalesOrder_fk
   Part_Comprises_Component_fk
   Part_IsConsumedIn_Assembly_fk
   ```

   Use the `Parent_Child_fk` sequence, not `Child_Parent_fk` is because (a) it shows up in the correct sort order when you are looking for them and (b) we always know the child involved, what we are guessing at is, which parent. The error message is then delightful:

   ```
   _____Foreign key violation on Vendor_Offers_PartVendor_fk.
   ```

   That works well for people who bother to model their data, where the Verb Phrases have been identified. For the rest, the record filing systems, etc, use `Parent_Child_fk`.

10. Indices are special, so they have a naming convention of their very own, made up of, *in order*, each character position from 1 to 3:

    `U` Unique, or _ for non-unique
    `C` Clustered, or _ for non-clustered
    _ separator

    For the remainder:

    - If the key is one column or a very few columns:
      `_____ColumnNames`
    - If the key is more than a few columns:
      `_____PK` Primary Key (as per model)
      `_____AK[*n*]` Alternate Key (IDEF1X term)

    Note that the table name is *not* required in the index name, because it always shows up as `table_name.index_name.`

    So when `Customer.UC_CustomerId` or `Product.U__AK` appears in an error message, it tells you something meaningful. When you look at the indices on a table, you can differentiate them easily.

11. Find someone qualified and professional and follow them. Look at their designs, and carefully

study the naming conventions they use. Ask them specific questions about anything you do not understand. Conversely, run like hell from anyone who demonstrates little regard for naming conventions or standards. Here's a few to get you started:

- They contain real examples of all the above. Ask questions re naming questions in this thread.
- Of course, the models implement several *other* Standards, beyond naming conventions; you can either ignore those for now, or feel free to ask specific **new questions**.
- They are several pages each, inline image support at Stack Overflow is for the birds, and they do not load consistently on different browsers; so you will have to click the links.
- Note that PDF files have full navigation, so click on the blue glass buttons, or the objects where expansion is identified:
- Readers who are unfamiliar with the Relational Modelling Standard may find the **IDEF1X Notation** [6] helpful.

Order Entry & Inventory [7] with Standard-compliant Addresses

Simple inter-office **Bulletin** [8] system for PHP/MyNonSQL

Sensor Monitoring [9] with full Temporal capability

Answers to Questions

That cannot be reasonably answered in the comment space.

> *Larry Lustig:*
> *... even the most trivial example shows ...*
> *If a Customer has zero-to-many Products and a Product has one-to-many Components and a Component has one-to-many Suppliers and a Supplier sells zero-to-many Components and a SalesRep has one-to-many Customers what are the "natural" names the tables holding Customers, Products, Components, and Suppliers?*

There are two major problems in your comment:

1. You declare your example to be "the most trivial", however, it is anything but. With that sort of contradiction, I am uncertain if you are serious, if technically capable.

2. That "trivial" speculation has several gross Normalisation (DB Design) errors.

    - Until you correct those, they are unnatural and abnormal, and they do not make any sense. You might as well name them abnormal_1, abnormal_2, etc.

    - You have "suppliers" who do not supply anything; circular references (illegal, and unnecessary); customers buying products without any commercial instrument (such as Invoice or SalesOrder) as a basis for the purchase (or do customers "own" products?); unresolved many-to-many relationships; etc.

    - Once that is Normalised, and the required tables are identified, their names will become obvious. Naturally.

In any case, I will try to service your query. Which means I will have to add some sense to it, not knowing what you meant, so please bear with me. The gross errors are too many to list, and given the spare specification, I am not confident I have corrected them all.

- I will assume that if the product is made up of components, then the product is an assembly, and the components are used in more than one assembly.

- Further, since "Supplier sells zero-to-many Components", that they do *not* sell products or assemblies, they sell only components.

Speculation vs Normalised Model [10]

In case you are not aware, the difference between square corners (Independent) and round corners (Dependent) is significant, please refer to the IDEF1X Notation link. Likewise the solid lines (Identifying) vs dashed lines (Non-identifying).

> *... what are the "natural" names the tables holding Customers, Products, Components, and Suppliers?*

- Customer
- Product
- Component (Or, AssemblyComponent, for those who realise that one fact identifies the other)
- Supplier

Now that I have resolved the tables, I don't understand your problem. Perhaps you can post a **specific** question.

> *VoteCoffee:*
> *How are you handling the scenario Ronnis posted in his example where multiple relationships exist between 2 tables (user_likes_product, user_bought_product)? I may misunderstand, but this seems to result in duplicate table names using the convention you detailed.*

Assuming there are no Normalisation errors, `User likes Product` is a predicate, not a table. Do not confuse them. Refer to my Answer, where it relates to Subjects, Verbs, and Predicates, and my response to Larry immediately above.

- Each table contains a *set* of Facts (each row is a Fact). Predicates (or propositions), are not Facts, they may or may not be true.
    - The *Relational Model* is based on First Order Predicate Calculus (more commonly known as First Order Logic). A Predicate is a single-clause sentence in simple, precise English, that evaluates to true or false.
    - Further, each table represents, or is the implementation of, **many** Predicates, not one.
- A query is a test of a Predicate (or a number of Predicates, chained together) that results in true (the Fact exists) or false (the Fact does not exist).
- Thus tables should be named, as detailed in my Answer (naming conventions), for the row, the Fact, and the Predicates should be documented (by all means, it is part of the database documentation), but as a separate list of Predicates.
- This is not a suggestion that they are not important. They are very important, but I won't write that up here.
- Quickly, then. Since the *Relational Model* is founded on FOPC, the entire database can be said to be a set of FOPC declarations, a set of Predicates. But (a) there are many types of Predicates, and (b) a table does not represent one Predicate (it is the physical implementation of **many** Predicates, and of different **types** of Predicates).
- Therefore naming the table for "the" Predicate that it "represents" is an absurd concept.
- The "theoreticians" are aware of only a few Predicates, they do not understand that since the *RM* was founded on the FOL, the entire database is a set of Predicates, and of different types.

    - And of course, they choose absurd ones from the few that they do know: `EXISTING_PERSON`; `PERSON_IS_CALLED`. If it were not so sad, it would be hilarious.
    - Note also that the Standard or atomic table name (naming the row) works brilliantly for all the verbiage (including all Predicates attached to the table). Conversely, the idiotic "table represents predicate" name cannot. Which is fine for the "theoreticians", who understand very little about Predicates, but retarded otherwise.

- The Predicates that are relevant to the data model, are expressed **in** the model, they are of two orders.

    1. **Unary Predicate**
       The first set is *diagrammatic*, not text: **the notation itself**. These include various Existential; Constraint-oriented; and Descriptor (attributes) Predicates.
        - Of course, that means only those who can 'read' a Standard data model can read those Predicates. Which is why the "theoreticians", who are severely crippled by their text-only mindset, cannot read data models, why they stick to their pre-1984 text-only mindset.
    2. **Binary Predicate**
       The second set is those that form **relationships** between Facts. This is the relation line. The Verb Phrase (detailed above) identifies the Predicate, the *proposition*, that has been implemented (which can be tested via query). One cannot get more explicit than that.
        - Therefore, to one who is fluent in Standard data models, all the Predicates *that are relevant*, are documented in the model. They do not need a separate list of Predicates (but the users, who cannot 'read' everything from the data model, do!).

- Here is a **Data Model** [11], where I have listed the Predicates. I have chosen that example because it shows the Existential, etc, Predicates, as well as the Relationship ones, the only Predicates not listed are the Descriptors. Here, due to the seeker's learning level, I am treating him as an user.

Therefore the event of more than one child table between two parent tables is not a problem, just name them as the Existential Fact re their content, and normalise the names.

The rules I gave for Verb Phrases for relationship names for Associative Tables come into play here. Here is a **Predicate vs Table** [12] discussion, covering all points mentioned, in summary.

For a good short description re the proper use of Predicates and how to use them (which is quite a different context to that of responding to comments here), visit **this answer** [13], and scroll down to the **Predicate** section.

> *Charles Burns:*
> *By sequence, I meant the Oracle-style object purely used to store a number and its next according to some rule (e.g. "add 1"). Since Oracle lacks auto-ID tables, my typical use is to generate unique IDs for table PKs. INSERT INTO foo(id, somedata) VALUES (foo_s.nextval, "data"...)*

Ok, that is what we call a Key or NextKey table. Name it as such. If you have SubjectAreas, use COM_NextKey to indicate it is common across the database.

Btw, that is a very poor method of generating keys. Not scalable at all, but then with Oracle's performance, it is probably "just fine". Further, it indicates that your database is full of surrogates, not relational in those areas. Which means extremely poor performance and lack of integrity.

[1] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Andreas_A.pdf
[2] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Andreas_B.pdf
[3] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Andreas_C.pdf
[4] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Andreas_D.pdf
[5] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Andreas_E.pdf
[6] https://www.softwaregems.com.au/Documents/Documentary%20Examples/IDEF1X%20Notation.pdf
[7] https://www.softwaregems.com.au/Documents/Documentary%20Examples/Order%20DM%20Advanced.pdf
[8] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Andrew/Andrew%202%20DM.pdf
[9] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Mark%20DM.pdf
[10] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Larry%20Lustig.pdf
[11] https://www.softwaregems.com.au/Documents/Student%20Resolutions/dzhu%20Generic%207%20DM.pdf
[12] https://www.softwaregems.com.au/Documents/Student%20Resolutions/Predicate%20vs%20Table.pdf
[13] https://stackoverflow.com/a/29711063/484814

(1) **Moderator Note** I've cleaned up the comments here, there were far too many off-topic arguments. If you feel the need to continue this discussion, then take it to chat. - **Taryn**
Thank you. I did not realise that I should move comments such as "This is the kind of answer I just wish I could star" into the Answer. I will conduct myself accordingly. There were also a lot of unacceptable comments, which should not be allowed on SO. Evidence that such persons should be banned, but no chance of that. Thanks again. - **PerformanceDBA**
(2) @ChrisF. (a) Thank you very much for the explanation, I didn't understand the previous moderator actions. (b) Is it possible for you to re-open the question, the attempt to close it is obviously an error (see my comment on the question). Otherwise SO continues to lose good questions/answers, and new ones replace them. The Help states "We don't like to lose great answers!". Thanks. - **PerformanceDBA**
(22) Can you provide reference to any of these "Standards"? Currently this is just a very well written personal opinion. - **Shane Courtrille**
The edit in response to VoteCoffee does not answer their question. - **Noumenon**
I like plural table names instead of singular names. In my opinion (for example in Hibernate) an entity (which store a row from table) should be singular but the table for it should be plural. - **Nagy Attila**
(3) I have read much of the standard materials, and I am fairly versed in relational database theory. I know the arguments of both the singular and plural stance when it comes to naming relations, yet I, too, respectfully disagree with the former. As in most cases, naming conventions should be more about consistency than dogmatism. Nobody working with database queries is going to be confused about whether a relation can hold several tuples or not, just because it's named in the singular or the plural. - **Daniel Saner**
(1) @Noumenon. Is this what you have not found: `UserProductPurchase` and `UserProductPreference` ? - **PerformanceDBA**
(1) @ShaneCourtrille. This answer is for those who can tell the difference between a bus in New York city vs a bus in China. And **why** the two buses are different. Explaining what a Standard is, to someone who argues against Standards, is beyond the scope of Q&A in SO. You might feel better if you view the Answer as **Convention** (as per the original Question). - **PerformanceDBA**
(1) @PerformanceDBA That is correct. I did not click through to the second PDF since the link text "Predicate vs Table discussion" does not appear to be specific to this question. - **Noumenon**
(1) @PerformanceDBA this is a masterpiece compilation across multiple textbooks and work experience... thank you for taking the time to share. I was wondering how you would recommend prefixing the name of a table that associates other tables but has its own attributes. In my case, I am tabulating a subset of waterways that have a specific regulatory classification as well as other land and asset features associated with the waterway. This table currently has five foreign keys to these tables along with a dozen attributes we track. - **JackedUpDBA**
(1) @JackedUpDBA. if Predicate vs Table does not answer your question, write to me. - **PerformanceDBA**
This is factually incorrect: "sure sign of someone who has not read any of the standard materials and has no knowledge of database theory". Well-educated individuals are able to recognize when something is a matter of taste. - **Jonathan B.**
@JonathanB. Educated individuals know that Science is about reality; objective truth. It does not care about subjective notions; or taste. Those who pervert science first make a truth subjective, then they reframe it as something that it is not. They may redefine the term "fact", and propose that objective truth is taste. - **PerformanceDBA**

1

[+20] [2011-01-15 23:25:06] Jonathan Leffler

There is no 'correct' about singular vs plural - it is mostly a matter of taste.

It depends in part on your focus. If you think of the table as a unit, it holds 'plurals' (because it holds many rows - so a plural name is appropriate). If you think of the table name as identifying a row in a table, you'll prefer 'singular'. This means your SQL will be thought of as working on one row from the table. That's OK, though it is usually an oversimplification; SQL works on sets (more or less). However, we can go with singular for the answers to this question.

1. Since you'll probably need a table 'user', another 'product', and the third to connect users to products, then you need a table 'user_product'.
2. Since the description applies to a product, you would use 'product_description'. Unless each user names each product for themselves...
3. The 'user_product' table is (or could be) an example of a table with a product ID and a user ID and not much else. You name the two-attribute tables in the same general way: 'user_stuff'. Decorative prefixes like 'rel_' don't really help. You'll see some people using 't_' in front of each table name, for instance. That is not a lot of help.

When you say "and the third to connect users". Do you mean a third table? Why should I need a third table when having a one to many relation (users have many products) ? Would you recommend using user_product instead of UserProduct by the way? - **Andreas**

My answer is predicated on there being a table listing products that the system knows about. There should also be a table listing the users the system knows about. And since more than one user can (under my hypothesis) be associated with a particular product, then there is a third table that could be named 'user_product' (or 'product_user'). If you really have just two tables, so each user's products are unique to that user and never used by anyone else, then (a) you have an unusual scenario, and (b) you only need two tables - you don't need the 'product' table I hypothesized. - **Jonathan Leffler**

Sorry, I should have used a better example than products. I meant it in a way that the product is unique to a user. So with this cleared, i assume the description table should be "user_product_description" since it's also unique for the user/product.. I know see what a horrible example i took with products :) Thank you - **Andreas**

@Andreas: it is often hard to choose good examples, and one of the problems is people's preconceptions about what a product table would contain. However, given your clarification, then 'user', 'user_product', and 'user_product_description' seem appropriate as table names. - **Jonathan Leffler**

2

[+20] [2011-01-16 00:15:18] Ronnis

Singular vs. Plural: Pick one and stick with it.

Columns shouldn't be prefixed/suffixed/infixed or in anyway fixed with references to the fact that it is a column. The same goes for tables. Don't name tables EMPLOYEE_T or TBL_EMPLOYEES because the second it is replaced with a view, things get really confusing.

Don't embed type information in names, such as "vc_firstname" for varchar, or "flavour_enum". Also don't embed constraints in column names, such as "department_fk" or "employee_pk".

Actually, the only good thing about *fixes I can think of, is that you can use reserved words like `where_t`, `tbl_order`, `user_vw`. Of course, in those examples, using plural would have solved the issue :)

Don't name all keys "ID". Keys refering to the same thing, should have the same name in all tables. The user id column could be called USER_ID in the user table and all tables referencing the user. The only time it is renamed is when different users are playing different roles, such as Message(sender_user_id, receiver_user_id). This really helps when dealing with larger queries.

Regarding CaSe:

```
thisiswhatithinkofalllowercapscolumnnames.

ALLUPPERCAPSISNOTBETTERBECAUSEITFEELSLIKESOMEONEISSCREAMINGATME.

CamelCaseIsMarginallyBetterButItStillTakesTimeToParse.

i_recommend_sticking_with_lower_case_and_underscore
```

In general it is better to name "mapping tables" to match the relation it describes rather than the names of the referenced tables. A user can have any number of relations to products: `user_likes_product`, `user_bought_product`, `user_wants_to_buy_product`.

(6) I prefer looking at underscore. But I prefer typing camelCase. There is something about the underscore... no matter how much I practice, I'm forced to stop and look at the keyboard. - **Lord Tydus**
@Ronnis, would you please elaborate on ""Don't name all keys "ID". Keys refering to the same thing, should have the same name in all tables."" ? - **Travis**
@Travis, sure I could, but that entire paragraph is an elaboration? - **Ronnis**
I guess my question is about the benefits of naming a (non-differentiated role) synthetic primary key `{table_name}_id` rather than just `id`, since the column will only ever be referred to with the table name prefixed as a qualifier, e.g. `table_name.id`. *For context, I'm operating in an ecosystem where join syntax of the form `table_a JOIN table_b ON table_b_id_column` is not supported; I have to do `table_a JOIN table_b ON table_b.id_column = table_a.table_b_id_column`.* - **Travis**
For me this is about clarity and the *logical* data model. If I use a number sequence for USER_ID and COMPANY_ID, some of those values will be the same of course. But the 123 from USER_ID is not the same as 123 from COMPANY_ID, because their values are drawn from difference *domains*. That way it makes sense to name them differently. - **Ronnis**
3

[+4] [2011-01-15 23:27:09] amelvin

Plurals aren't bad as long as they are used consistently - but singular is my preference.

I would dispense with underscores unless you want to outline a many-to-many relationship; and use an initial capital because it helps distinguish things in ORMs.

But there are many naming conventions, so if you want to use underscores that's OK as long as its done consistently.

So:

```
User

UserProduct (it is a users products after all)
```

If only one user can have any product then

```
UserProductDescription
```

But if the product is shared by users:

```
ProductDescription
```

If you save your underscores for many-to-many relationships you can do something like:

```
UserProduct_Stuff
```

to form a M-to-M between UserProduct and Stuff - not sure from the question the exact nature of the many-to-many required.

I like this, seems like a good way of doing it. The only thing I'm wondering about here is, since I "should" save the

underscore for many to many, i "have" to use upper case naming of tables. I'm not sure why but somehow I've learned that one shouldn't use that for table names, only for columns... I probably heard it from the same person that said plural is wrong though. - **Andreas**
@Andreas You don't need to use upper case for tables, just capitalize the first letter of the distinct words. - **amelvin**

4

[+2] [2011-01-15 23:32:37] Ozzy

There is not more correct to use singular than plural form, where have you heard that? I would rather say that plural form is more common for naming database tables...and in my opinion also more logic. The table most often contain more than one row ;) In a conceptual model though the names of the entities are often in singular.

About your question, if 'Product' and 'ProductDescription' are concepts with an identity (i.e. entities) in your model I would simply call the tables 'Products' and 'ProductDescriptions'. For tables that are used in order to implement a many-to-many relationship I most often use the naming convention "SideA2SideB", for example "Student2Course".

5
