Naming of ID columns in database tables

Asked 16 years, 2 months ago Modified 7 months ago Viewed 76k times

134

I was wondering peoples opinions on the naming of ID columns in database tables.

If I have a table called Invoices with a primary key of an identity column I would call that column InvoiceID so that I would not conflict with other tables and it's obvious what it is.

Where I am workind current they have called all ID columns ID.

So they would do the following:

```
Select
    i.ID
,   il.ID
From
    Invoices i
    Left Join InvoiceLines il
        on i.ID = il.InvoiceID
```

Now, I see a few problems here:
1. You would need to alias the columns on the select
2. ID = InvoiceID does not fit in my brain
3. If you did not alias the tables and referred to InvoiceID is it obvious what table it is on?

What are other peoples thoughts on the topic?

sql naming-conventions

Share Improve this question Follow

asked Oct 16, 2008 at 13:34

Arry
2,002 ● 4 ● 16 ● 22

yiiframework.com/wiki/227/guidelines-for-good-schema-design/… – Real Dreams May 10, 2014 at 9:22

Possible duplicate of Foreign Key naming scheme – philipxy Jul 7, 2019 at 21:57

24 Answers Sorted by: Highest score (default)

187

I always preferred ID to TableName + ID for the id column, and then TableName + ID for a foreign key. That way, all tables have the same name for the id field and there isn't a redundant description. This seems simpler to me because all the tables have the same primary key field name.

As far as joining tables and not knowing which Id field belongs to which table, in my opinion, the query should be written to handle this situation. Where I work, we always prefix the fields we use in a statement with the table/table alias.

Share

Improve this answer

Follow

edited May 24 at 5:42

Pang

10.1k ● 146 ● 85 ● 124

answered Oct 16, 2008 at 13:46

kemiller2002

115k ● 28 ● 199 ● 253

3 I realize an old thread but I agree. Also makes mapping at the dataaccess layer easier: if all "objects" have a Id field that has to be unique then when making methods at the data access layer to do things like delete items you can make them generic since you can receive a list of Ids for anything and know that you just need to delete where Id = blah from that particular table rather than having to keep a mapping of what is unique for each table as well as the table name/program logic name mapping. – Mike Jun 23, 2013 at 3:50

2 Kevin, same as you. for example: user_id, role_id for PKey, and role_user_id for FKey. It is good on a large scale project. Because if all ID field are named to "id", it is too confuse. But i think it is personal preference, somebody think only use "id" are clear. – Cheung Oct 13, 2014 at 7:44

3 This is my preference. Just because cross table queries are more difficult to read doesn't mean the Id column should be changed to make it easier. Just make your aliases more descriptive. – tmutton Mar 9, 2015 at 8:44

1 I suspect that convention of prefixing ID with entity name comes from people using "select * from" for whatever reasons. In an environment tolerant to "select * ", reality usually gets distorted towards supporting bulk selection of everything from everywhere. Does not make much sense if you don't blindly select everything. There are also references to USING and natural joins, which are quite dangerous, and also don't look like an argument to me at all because they effectively prevent you from using role-based foreign key naming convention. – Roman Polunin Feb 16, 2017 at 1:44

1 Thanks to God, you are not creating table in my database. – gotqn Aug 31, 2021 at 13:32

66

There's been a nerd fight about this very thing in my company of late. The advent of LINQ has made the redundant *tablename+ID* pattern even more obviously silly in my eyes. I think most reasonable people will say that if you're hand writing your SQL in such a manner as that you have to specify table names to differentiate *FKs*, then it's not only a savings on typing, but it adds clarity to your SQL to use just the ID in that you can clearly see which is the *PK* and which is the *FK*.

E.g.

```
FROM Employees e
LEFT JOIN Customers c ON e.ID = c.EmployeeID
```

tells me not only that the two are linked, but which is the ***PK*** and which is the ***FK***. Whereas in the old style, you're forced to either look or hope that they were named well.

Share

Improve this answer

Follow

edited May 24 at 5:43

Pang

10.1k ● 146 ● 85 ● 124

answered Oct 16, 2008 at 15:30

Echostorm

9,795 ● 8 ● 39 ● 52

3 Except I've never known a single developer in my life to write your example. They instead write: LEFT JOIN Customers as c ON e.ID = c.EmployeeID To me, this is clearer: LEFT JOIN Customers as c ON e.EmployeeID = c.EmployeeID And which item is the foreign key is obvious by the table name. obviously customer.employeeId is a foreign key while employee.employeeId is not. – dallin Mar 6, 2013 at 1:11

8 LOts of people (such as report writers who write very complex sql) and BI specialists doing imports and exports still need to handwrite SQl. The database design has to accomodate them too not just application developers. – HLGEM Oct 7, 2013 at 17:11

2 @dallin I'm pretty sure a lot of developers don't really think about writing it as e.ID = c.EmployeeID or the other way round c.EmployeeID = e.ID. I guess it's just what you are used to. I really have to think what the reason would be to write it in a specific order (other than what you have learned by the person who taught you at the time). And I think Employee.ID is faster to read than Employee.EmployeeID. – SuperDre Aug 12, 2021 at 12:33

37

ID is a SQL Antipattern. See http://www.amazon.com/s/ref=nb_sb_ss_i_1_5?url=search-alias%3Dstripbooks&field-keywords=sql+antipatterns&sprefix=sql+a

If you have many tables with ID as the id, you are making reporting that much more difficult. It obscures meaning and makes complex queries harder to read as well as requiring you to use aliases to differentiate on the report itself.

Further, if someone is foolish enough to use a natural join in a database where they are available, you will join to the wrong records.

If you would like to use the USING syntax that some dbs allow, you cannot if you use ID.

If you use ID, you can easily end up with a mistaken join if you happen to be copying the join syntax (don't tell me that no one ever does this!) and forget to change the alias in the join condition.

So you now have

```
select t1.field1, t2.field2, t3.field3
from table1 t1
join table2 t2 on t1.id = t2.table1id
join table3 t3 on t1.id = t3.table2id
```

when you meant

```
select t1.field1, t2.field2, t3.field3
from table1 t1
join table2 t2 on t1.id = t2.table1id
join table3 t3 on t2.id = t3.table2id
```

If you use tablenameID as the id field, this kind of accidental mistake is far less likely to happen and much easier to find.

Share

Improve this answer

Follow

edited May 24 at 5:46

Pang

10.1k ● 146 ● 85 ● 124

answered Sep 21, 2011 at 17:41

HLGEM

96.4k ● 15 ● 119 ● 189

12 @spencer7593, just because you like ID doesn't mean it isn't an antipattern. It is harder to make mistakes in joins when you have the tablename id because you would get a syntax error immediately. – HLGEM Jul 10, 2012 at 20:15

10 +1 Columns that are synonymous should have the same name - by using the table name prefix you can have a primary key column called table1ID and a foreign key column in another table called table1ID - and you KNOW that they are the same thing. I was taught this 20 years ago and its a practice that doesn't let you down. – amelvin Jul 30, 2012 at 9:18

14 NO! This is the smurf naming convention! – Ross Sep 25, 2012 at 19:31

25 I do not like this convention because it just means you're practically forced to alias all of your tables so that you're not redundantly naming the table, naming the table again, and then adding the id. `join table2 on table1.table1id = table2.table1id`. Your reasoning is ok except that if you use id, the name of the table is in front of the ID anyway. `join table2 on table1.id = table2.table1id` ... which is just as verbose and not redundant, nor forces obscure aliases to prevent the just mentioned redundancy .. which in my opinion are the bane of sql development. – Anther Mar 15, 2013 at 18:35

24 Then, if you have a `Name` field in a table, should you change it to `Table1Name` to avoid the same problems with that field? Should you prefix all columns with the table name for the same reason? That doesn't sound right. – Joanvo Sep 17, 2015 at 7:35

31

We use `InvoiceID`, not `ID`. It makes queries more readable -- when you see `ID` alone it could mean anything, especially when you alias the table to `i`.

Share Improve this answer Follow

answered Oct 16, 2008 at 13:36

Jason Cohen

83k ● 26 ● 110 ● 114

5 Using Id is more appropriate. The column is in the table "invoices" so that should be enough. If you are writing cross table queries you should be using more descriptive aliases. "i" is not enough. Call it "invoices". – tmutton Mar 9, 2015 at 8:39

2 It's not clear that "ID" alone means Invoices. Suppose you've also got foreign keys to Accounts and People. Now which one is "ID?" Isn't it easier in a join, say, to read a.InvoiceID = b.InvoiceID instead of a.ID = b.InvoiceID and not be able to easily debug it. – Jason Cohen Mar 21, 2015 at 22:32

6 @JasonCohen This just means the table alias is junk, not the column name. – Joshua Schlichting Sep 12, 2020 at 22:38

1 But when aliasing tables you still have the tablename, so you know i.ID is from Invoice, and to me i.ID =p.InvoiceID is more readable than i.InvoiceID = p.InvoiceID. @JasonCohen with a.InvoiceID = b.InvoiceID you still have to figure out which one is the foreignkey and which one isn't. And normally if you go combining tables you prefix the specific field, so in your case it would be a.ID (account) or p.ID (product) otherwise the SQL processor can't differentiate it anyway. – SuperDre Aug 12, 2021 at 12:37

1 @NibblyPig It's not about having to type the word invoice, it's about keeping it the same everywhere and universal, when it says i.ID I know i is the main table, if it says i.InvoiceID = y.InvoiceID you'd also have to look up to see which table might be the main table. And yes big blobs of SQL is a pain, but if you don't use aliases I'd rather type Invoice.id = MyOtherTable.InvoiceID then Invoice.InvoiceID = MyOtherTable.InvoiceID. But if you only have 2 tables then you could even use ID = InvoiceID without aliasing. – SuperDre Aug 18, 2021 at 20:11

28

I agree with Keven and a few other people here that the PK for a table should simply be Id and foreign keys list the OtherTable + Id.

However, I wish to add one reason which recently gave more weight to this argument.

In my current position, we are employing the entity framework using POCO generation. Using the standard naming convention of Id, the PK allows for inheritance of a base poco class with validation and such for tables which share a set of common column names. Using the Tablename + Id as the PK for each of these tables destroys the ability to use a base class for these.

Just some food for thought.

Share

Improve this answer

Follow

edited May 24 at 5:51

Pang

10.1k ● 146 ● 85 ● 124

answered Jun 7, 2012 at 14:34

pawnrob

381 ● 3 ● 3

8 +1 for real world example of how generic naming improves code reuse in specific case (that lots of developers will care about since there are millions of .NET devs, and many will use EF). – codingoutloud Nov 21, 2012 at 17:44

Not just EF and the like at my work we have lots of generic methods. So a webservice will have something like List<Result> Save<T>(List<int> Ids); Since we know every table has an Id column that is the primary key we can do things like this with just a simple mapping of C# objects to their tables (something like <Customer, "Customers">, <BillingCode, "BillingCodes"> (or better yet stored proc names), generate the sql on the fly based on the object that is passed and voila no repeated save/delete/edit methods for each type of object. – Mike Jun 23, 2013 at 3:58

Programming by convention is baaaaad IMO. Primary keys are something that exists on the database, not just by convention of being called Id. Relying on the naming being correct is not good. What happens if you have a table with no primary key, or one with a compound key? What happens when you pull a DB managed by another department that doesn't use Id into your project? I would be very wary of relying on the name Id. – NibblyPig Aug 17, 2021 at 10:25

20

It's not really important. You are likely to run into similar problems in all naming conventions.

It ***is*** important to be consistent so you don't have to look at the table definitions every time you write a query.

Share
Improve this answer
Follow

edited Jul 11, 2023 at 15:10

Rich
6,539 ● 20 ● 36 ● 57

answered Oct 16, 2008 at 13:40

Nir
29.6k ● 11 ● 68 ● 104

17

My preference is also ID for primary key and TableNameID for foreign key. I also like to have a column "name" in most tables where I hold the user readable identifier (i.e. name :-)) of the entry. This structure offers great flexibility in the application itself, I can handle tables in mass, in the same way. This is a *very* powerful thing. Usually an OO software is built on top of the database, but the OO toolset cannot be applied because the db itself does not allow it. Having the columns id and name is still not very good, but it is a step.

> Select
> i.ID , il.ID From Invoices i Left Join InvoiceLines il on i.ID = il.InvoiceID

Why cant I do this?

```
Select
    Invoices.ID
```

```
,   InvoiceLines.ID
From
    Invoices
    Left Join InvoiceLines
        on Invoices.ID = InvoiceLines.InvoiceID
```

In my opinion this is very much readable and simple. Naming variables as i and il is a poor choice in general.

Share

Improve this answer

Follow

edited Sep 21, 2011 at 17:23

answered Sep 21, 2011 at 15:34

bjdodo

361 ● 3 ● 11

13

I just started working in a place that uses only "ID" (in the core tables, referenced by TableNameID in foreign keys), and have already found TWO production problems directly caused by it.

In one case the query used "... where ID in (SELECT ID FROM OtherTable ..." instead of "... where ID in (SELECT TransID FROM OtherTable ...".

Can anyone honestly say that wouldn't have been much easier to spot if full, consistent names were used where the wrong statement would have read "... where TransID in (SELECT OtherTableID from OtherTable ..."? I don't think so.

The other issue occurs when refactoring code. If you use a temp table whereas previously the query went off a core table then the old code reads "... dbo.MyFunction(t.ID) ..." and if that is not changed but "t" now refers to a temp table instead of the core table, you don't even get an error - just erroneous results.

If generating unnecessary errors is a goal (maybe some people don't have enough work?), then this kind of naming convention is great. Otherwise consistent naming is the way to go.

Share Improve this answer Follow

answered Nov 16, 2010 at 22:00

Eric Kassan

578 ● 4 ● 10

4 +1 for real world example of more specific name improving maintainability. – codingoutloud Nov 21, 2012 at 17:41

9

I **personally** prefer (as it has been stated above) the **Table.ID** for the **PK** and **TableID** for the **FK**. Even (please don't shoot me) Microsoft Access recommends this.

HOWEVER, I ALSO know for a fact that some generating tools favor the TableID for PK because they tend to link all column name that contain **'ID'** in the word, *INCLUDING ID!!!*

Even the query designer does this on Microsoft SQL Server (and for each query you create, you end up ripping off all the unnecessary newly created relationships on all tables on column ID)

THUS as Much as my internal OCD hates it, I roll with the **TableID** convention. Let's remember that it's called a Data **BASE**, as it will be the base for hopefully many many many applications to come. And all technologies Should benefit of a well normalized with clear description Schema.

It goes without saying that I DO draw my line when people start using TableName, TableDescription and such. In My opinion, conventions should do the following:

- Table name: Pluralized. Ex. **Employees**
- Table alias: Full table Name, singularized. Ex.

```
SELECT Employee.*, eMail.Address
FROM Employees AS Employee LEFT JOIN eMails as eMail on Employee.eMailID =
eMail.eMailID -- I would sure like it to just have the eMail.ID here.... but
well
```

[Update]

Also, there are some valid posts in this thread about duplicated columns due of the "kind of relationship" or role. Example, if a Store has an **EmployeeID**, that tells me squat. So I sometimes do something like **Store.EmployeeID_Manager**. Sure it's a bit larger but at leas people won't go crazy trying to find *table ManagerID*, or what *EmployeeID* is doing there. When querying is WHERE I would simplify it as: SELECT EmployeeID_Manager as ManagerID FROM Store

Share

Improve this answer

Follow

edited Aug 14, 2013 at 21:55

answered Aug 14, 2013 at 21:45

percebus

847 ● 1 ● 10 ● 21

1 I think your point is good for database's beauty and didactic purposes, but to functionality I think it is a problem. Plural table names promotes inconsistency between tables and PK Id <-> FK IdTable creates diferences on the name of the same thing. At the same time, something like `User.UserId` is kinda weird to type in programming. – Machado Aug 9, 2017 at 3:17

For the sake of simplicity most people name the column on the table ID. If it has a foreign key reference on another table, then they explicity call it InvoiceID (to use your

7

example) in the case of joins, you are aliasing the table anyway so the explicit inv.ID is still simpler than inv.InvoiceID

Share Improve this answer Follow

answered Oct 16, 2008 at 13:38

Michael Brown

9,143 ● 1 ● 32 ● 37

4

There are lots of answers on this already, but I wanted to add two major things that I haven't seen above:

- Customers coming to you for support.

Many times a customer or user or even dev of another department have hit a snag and have contacted us saying they're having a problem doing an operation. We ask them what record they're having a problem with. Now, the data they see on the screen, e.g. a grid with customer name, number of orders, destination etc is an aggregate of many tables. They say they've having trouble with id 83. There's no way to know what id that is, which table it is, if it's just called 'id'.

Namely, a row of data does not give any indication which table it is from. Unless you happen to know the schema of your database well, which is rarely the case on complex systems or non-greenfield systems you've been told to take over, you don't know what id=83 means even if you have more data like name, address, etc (which might not even be in the same table!).

This id could be coming from a grid, or it could be coming from an error in your API, or a faulty query dumping the error message to the screen, or to a log file.

Often a developer just dumps 'ID' into a column and forgets about it, and often DBs have many similar tables like Invoice, InvoiceGrouping, InvoicePlan and the ID could be for any of them. In frustration you look in the code to see which one it is, and see that they've called it Id on the model as well, so you then have to dig into how the model for the page was constructed. I cannot count how many times I've had to do this to figure out what an Id is. It's a lot. Sometimes you have to dig out a SPROC as well that just returns 'Id' as a header. Nightmare.

- Log files are easier when it's clear what went wrong

Often SQL can give pretty crappy error messages. "Could not insert item with ID 83, column would be truncated" or something like that is very hard to debug. Often error messages are not very helpful, but usually the thing that broke will make a vague attempt to tell you what record was broken by just dumping out the primary key name and the value. If it's "ID" then it doesn't really help at all.

This is just two things that I didn't feel were mentioned in the other answers.

I also think that a lot of comments are 'if you program in X way then this isn't an issue', and I think the points above (and other points on this question) are valid specifically because of the way people program and because they don't have the time, energy, budget and foresight to program in perfect logging and error handling or change engrained habits of quick SQL and code writing.

Share Improve this answer Follow

answered Aug 17, 2021 at 10:34

NibblyPig

52.9k ● 75 ● 216 ● 374

3

Coming at this from the perspective of a formal data dictionary, I would name the data element `invoice_ID`. Generally, a data element name will be unique in the data dictionary and ideally will have the same name throughout, though sometimes additional qualifying terms may be required based on context e.g. the data element named `employee_ID` could be used twice in the org chart and therefore qualified as `supervisor_employee_ID` and `subordinate_employee_ID` respectively.

Obviously, naming conventions are subjective and a matter of style. I've find ISO/IEC 11179 guidelines to be a useful starting point.

For the DBMS, I see tables as collections of entites (except those that only ever contain one row e.g. cofig table, table of constants, etc) e.g. the table where my `employee_ID` is the key would be named `Personnel`. So straight away the `TableNameID` convention doesn't work for me.

I've seen the `TableName.ID=PK TableNameID=FK` style used on large data models and have to say I find it slightly confusing: I much prefer an identifier's name be the same throughout i.e. does not change name based on which table it happens to appear in. Something to note is the aforementioned style seems to be used in the shops which add an `IDENTITY` (auto-increment) column to *every* table while shunning natural and compound keys in foreign keys. Those shops tend not to have formal data dictionaries nor build from data models. Again, this is merely a question of style and one to which I don't personally subscribe. So ultimately, it's not for me.

All that said, I can see a case for sometimes dropping the qualifier from the column name when the table's name provides a context for doing so e.g. the element named `employee_last_name` may become simply `last_name` in the `Personnel` table. The rationale here is that the domain is 'people's last names' and is more likely to be `UNION`ed with `last_name` columns *from* other tables rather than be used as a foreign key *in* another table, but then again... I might just change my mind, sometimes you can never tell. That's the thing: data modelling is part art, part science.

Share

Improve this answer

Follow

edited Oct 17, 2008 at 7:34

answered Oct 16, 2008 at 19:32

onedaywhen

57k ● 12 ● 102 ● 142

1 This naming convention may have been relevant in the past, but modern programming would discourage names that are not exactly as they exist on the domain entity, especially from a code first approach. – Ryan Naccarato Feb 21, 2023 at 20:03

2

My vote is for InvoiceID for the table ID. I also use the same naming convention when it's used as a foreign key and use intelligent alias names in the queries.

```
Select Invoice.InvoiceID, Lines.InvoiceLine, Customer.OrgName
From Invoices Invoice
Join InvoiceLines Lines on Lines.InvoiceID = Invoice.InvoiceID
Join Customers Customer on Customer.CustomerID = Invoice.CustomerID
```

Sure, it's longer than some other examples. But smile. This is for posterity and someday, some poor junior coder is going to have to alter your masterpiece. In this example there is no ambiguity and as additional tables get added to the query, you'll be grateful for the verbosity.

Share Improve this answer Follow

answered Oct 16, 2008 at 15:32

Rob Allen

17.7k ● 6 ● 53 ● 70

2

FWIW, our new standard (which changes, uh, I mean "evolves", with every new project) is:

- Lower case database field names
- Uppercase table names
- Use underscores to separate words in the field name - convert these to Pascal case in code.
- `pk_` prefix means primary key
- `_id` suffix means an integer, auto-increment ID
- `fk_` prefix means foreign key (no suffix necessary)
- `_VW` suffix for views
- `is_` prefix for booleans

So, a table named NAMES might have the fields `pk_name_id, first_name, last_name, is_alive,` and `fk_company` and a view called `LIVING_CUSTOMERS_VW`,

defined like:

```
SELECT first_name, last_name
FROM CONTACT.NAMES
WHERE (is_alive = 'True')
```

As others have said, though, just about any scheme will work as long as it is consistent and doesn't unnecessarily obfuscate your meanings.

Share Improve this answer Follow

answered Oct 17, 2008 at 19:16

CMPalmer

8,623 ● 5 ● 42 ● 50

1

I definitely agree with including the table name in the ID field name, for exactly the reasons you give. Generally, this is the only field where I would include the table name.

Share Improve this answer Follow

answered Oct 16, 2008 at 13:39

DOK

32.8k ● 8 ● 63 ● 93

1

I do hate the plain id name. I strongly prefer to always use the invoice_id or a variant thereof. I always know which table is the authoritative table for the id when I need to, but this confuses me

```
SELECT * from Invoice inv, InvoiceLine inv_l where
inv_l.InvoiceID = inv.ID
SELECT * from Invoice inv, InvoiceLine inv_l where
inv_l.ID = inv.InvoiceLineID
SELECT * from Invoice inv, InvoiceLine inv_l where
inv_l.ID = inv.InvoiceID
SELECT * from Invoice inv, InvoiceLine inv_l where
inv_l.InvoiceLineID = inv.ID
```

What's worst of all is the mix you mention, totally confusing. I've had to work with a database where almost always it was foo_id except in one of the most used ids. That was total hell.

Share

Improve this answer

Follow

edited Oct 16, 2008 at 13:44

answered Oct 16, 2008 at 13:38

Vinko Vrsalovic

340k ● 55 ● 340 ● 373

3 I have read the word "invoice" too many times in this post. It looks funny now – Kevin Oct 16, 2008 at 14:23

2 Ugh, implicit joins! I want to tear my eyeballs out looking at that. – HLGEM Jan 12, 2012 at 16:21

1

I think you can use anything for the "ID" as long as you're consistent. Including the table name is important to. I would suggest using a modeling tool like Erwin to enforce the naming conventions and standards so when writing queries it's easy to understand the relationships that may exist between tables.

What I mean by the first statement is, instead of ID you can use something else like 'recno'. So then this table would have a PK of invoice_recno and so on.

Cheers, Ben

Share Improve this answer Follow

answered Oct 16, 2008 at 15:12

Ben Sullins

0

For the column name in the database, I'd use "InvoiceID".

If I copy the fields into a unnamed struct via LINQ, I may name it "ID" there, if it's the only ID in the structure.

If the column is NOT going to be used in a foreign key, so that it's only used to uniquely identify a row for edit editing or deletion, I'll name it "PK".

Share Improve this answer Follow

answered Oct 16, 2008 at 13:43

James Curran

103k ● 37 ● 185 ● 262

0

If you give each key a unique name, e.g. "invoices.invoice_id" instead of "invoices.id", then you can use the "natural join" and "using" operators with no worries. E.g.

```
SELECT * FROM invoices NATURAL JOIN invoice_lines
SELECT * FROM invoices JOIN invoice_lines USING (invoice_id)
```

instead of

```
SELECT * from invoices JOIN invoice_lines
    ON invoices.id = invoice_lines.invoice_id
```

SQL is verbose enough without making it more verbose.

Share Improve this answer Follow

answered Oct 16, 2008 at 14:33

Steven Huwig

20.7k ● 9 ● 58 ● 80

Do you know if SQL Server supports natural join? – Arry Oct 16, 2008 at 15:01

I don't think that it does. According to connect.microsoft.com/SQLServer/feedback/… it appears that the syntax is slated to be added in some version after SQL Server 2005. I know it works in PostgreSQL and in Oracle. – Steven Huwig Oct 16, 2008 at 15:19

7 Never, never, never use natural join. If one table has a Description field when you write the query you're fine. If later, someone adds a description field to the other table, you'll start joining on that as well and completely break. – Mark Brady Oct 17, 2008 at 13:50

1 heh, that sounds like the sound of real-life experience :) – dland Oct 17, 2008 at 21:32

I would only use natural join for ad hoc queries. – Steven Huwig Oct 18, 2008 at 1:09

0

What I do to keep things consistent for myself (where a table has a single column primary key used as the ID) is to name the primary key of the table `Table_pk`. Anywhere I have a foreign key pointing to that tables primary key, I call the column `PrimaryKeyTable_fk`. That way I know that if I have a `Customer_pk` in my Customer table and a `Customer_fk` in my Order table, I know that the Order table is referring to an entry in the Customer table.

To me, this makes sense especially for joins where I think it reads easier.

```
SELECT *
FROM Customer AS c
    INNER JOIN Order AS c ON c.Customer_pk = o.Customer_fk
```

Share Improve this answer Follow

answered Oct 17, 2008 at 18:49

Ian Andrews

106 ● 8

-1

I prefer DomainName || 'ID'. (i.e. DomainName + ID)

DomainName is often, but not always, the same as TableName.

The problem with ID all by itself is that it doesn't scale upwards. Once you have about 200 tables, each with a first column named ID, the data begins to look all alike. If you always qualify ID with the table name, that helps a little, but not that much.

DomainName & ID can be used to name foreign keys as well as primary keys. When foriegn keys are named after the column that they reference, that can be of mnemonic assistance. Formally, tying the name of a foreign key to the key it references is not necessary, since the referential integrity constrain will establish the reference. But it's awfully handy when it comes to reading queries and updates.

Occasionally, DomainName || 'ID' can't be used, because there would be two columns in the same table with the same name. Example: Employees.EmployeeID and Employees.SupervisorID. In those cases, I use RoleName || 'ID', as in the example.

Last but not least, I use natural keys rather than synthetic keys when possible. There are situations where natural keys are unavailable or untrustworthy, but there are plenty of situations where the natural key is the right choice. In those cases, I let the natural key take on the name it would naturally have. This name often doesn't even have the letters, 'ID' in it. Example: OrderNo where No is an abbreviation for "Number".

Share Improve this answer Follow

answered Oct 17, 2008 at 10:57

Walter Mitty
18.9k ● 2 ● 31 ● 59

-1

See the Interakt site's naming conventions for a well thought out system of naming tables and columns. The method makes use of a suffix for each table (`_prd` for a product table, or `_ctg` for a category table) and appends that to each column in a given table. So the identity column for the products table would be `id_prd` and is therefore unique in the database.

They go one step further to help with understanding the foreign keys: The foreign key in the product table that refers to the category table would be `idctg_prd` so that it is obvious to which table it belong (`_prd` suffix) and to which table it refers (category).

Advantages are that there is no ambiguity with the identity columns in different tables, and that you can tell at a glance which columns a query is referring to by the column names.

Share Improve this answer Follow

answered Oct 17, 2008 at 14:36

flamingLogos
6,011 ● 4 ● 39 ● 45

-2

For each table I choose a tree letter shorthand(e.g. Employees => Emp)

That way a numeric autonumber primary key becomes **nkEmp**.

It is short, unique in the entire database and I know exactly its properties at a glance.

I keep the same names in SQL and all languages I use (mostly C#, Javascript, VB6).

Share Improve this answer Follow

answered Oct 17, 2008 at 13:58

pkario

2,220 ● 6 ● 28 ● 30

-3

You could use the following naming convention. It has its flaws but it solves your particular problems.

1. Use short (3-4 characters) nicknames for the table names, i.e. Invoice - `inv`, InvoiceLines - `invl`
2. Name the columns in the table using those nicknames, i.e. `inv_id`, `invl_id`
3. For the reference columns use `invl_inv_id` for the names.

this way you could say

```
SELECT * FROM Invoice LEFT JOIN InvoiceLines ON inv_id = invl_inv_id
```

Share Improve this answer Follow

answered Oct 16, 2008 at 13:39

Ilya Kochetov

18.4k ● 6 ● 47 ● 62

6 ick! I'd vote against using short nicknames for tables (or any other object). With nicknames, you'll never know exactly what the short name is. Remember, there are many way to spell it wrong; there's only one way to spell it right. – James Curran Oct 16, 2008 at 13:45

2 James, I disagree. If you have a short name that isn't descriptive and you can't remember what it is, then you've picked the wrong name or don't understand the naming convention that someone else chose. – kemiller2002 Oct 16, 2008 at 13:49

3 Use aliases to achieve the same effect. select * from Invoice inv left join InvoiceLines invl on inv.ID = invl.InvoiceID – yfeldblum Oct 16, 2008 at 14:57

3 NO no no no. alias the table with a abrievation in the query. But the table name should be a full. – Mesh Oct 17, 2008 at 14:43

2 Why are so many programmers lazy and seem the answer to everything is to type the least possible just because its too hard to type a bit more. – mP. Oct 14, 2009 at 5:55