How I Write SQL, Part 1: Naming Conventions

Launch by Lunch
Databases, DevOps, and Development
RSS
About
Contact
How I Write SQL, Part 1: Naming Conventions
Feb 16 2014
Background
Target Audience
Why Naming Conventions Are Important
Names Are Long Lived
Names Are Contracts
Developer Context Switching
Naming Conventions
Singular Relations
Key Fields
Primary Keys
Foreign Keys
(Prefixes and Suffixes (are bad
Relation Type Prefixes
Application Name Prefixes
Data Type Suffixes
Explicit Naming
Indexes
Constraints
Final Thoughts
Background
".There are only two hard problems in Computer Science: cache invalidation and naming things"
Phil Karlton --
In this post I'll be going into the latter. Specifically, I'll describe naming conventions for database objects, why they are so ,important
.and what you should and shouldn't be doing
.Warning! This is a fairly opinionated post and I welcome feedback from people suggesting alternatives
Target Audience
Our company, JackDB, uses PostgreSQL internally to store persistent state and the naming conventions in this post were written with PostgreSQL in mind. Most of the recommendations should be equally valid for other relational databases such as MySQL, Oracle, or .Microsoft SQL Server
A lot of them will also apply to NoSQL databases, though not everything. For example, the suggestion below to use full english words .goes against the recommended approach for naming fields in MongoDB. When in doubt, find a guide for your specific database type
Why Naming Conventions Are Important
Names Are Long Lived
Data structures are meant to last much longer than application code. Anyone that has worked on a long running system can attest to .that
Well defined data structures and table layouts will outlive any application code. It's not uncommon to see an application completely .rewritten without any changes done to its database schema

id bigint PRIMARY KEY,
full_name text NOT NULL,
birth_date date NOT NULL);

Some guides suggest prefixing the table name in the primary key field name, ie. person_id vs id. The extra prefix is redundant. All field names in non-trivial SQL statements (i.e. those with more than one table) should be explicitly qualified and prefixing as a form of .namespacing field names is a bad idea

Foreign Keys

Foreign key fields should be a combination of the name of the referenced table and the name of the referenced fields. For single .column foreign keys (by far the most common case) this will be something like foo_id

) CREATE TABLE team_member

team_id bigint NOT NULL REFERENCES team(id),
person_id bigint NOT NULL REFERENCES person(id),
((CONSTRAINT team_member_pkey PRIMARY KEY (team_id, person_id;

(Prefixes and Suffixes (are bad

Relation Type Prefixes

Some (older) guidelines suggest naming tables with a TB_ prefix, views with a VW_ prefix, or stored procedures with a SP_ prefix. The rationale being that a programmer reading through some unknown SQL would immediately recognize this and know the object type .based on the name. This is a bad idea

Object names should not include the object type in them. That way you can change it later. A view that is replaced with a table maintains the original contract of a view (ex: you can SELECT from it). A dependent system would not need to be updated after such a .change

I've seen many such systems where at some point a view will become a table. Then you'll end up with code issuing INSERT statements

into vw_foobar. There's even a really powerful feature of PostgreSQL that allows you do define DML rules on views (ie. you can .(INSERT/UPDATE/DELETE from them

.Adding object type prefixes adds extra typing now and extra confusion down the road

Application Name Prefixes

Another (older) suggestion is to have a common prefix for all your database objects. For example, our app "Foobar" would have tables .name Foobar_Users, Foobar_Teams, etc. Again, this is a bad idea

All modern databases support some form of namespacing. For example, in PostgreSQL you can create schemas to group database objects. If you have multiple applications sharing the same database and want to prevent them from clobbering each other, use !schemas instead. That's exactly what they are for

Most people will not even need them though. It's far more common for a database to be used with a single logical data model than .multiple, separate data models. Hence schemas will not be required. If you do need them, it should be fairly obvious

The exception to this rule is if you are developing a database agnostic code base such as a framework or plugin. Supporting multiple .namespacing methods is complicated so many frameworks instead rely on application name prefixing

However, most people develop applications, not plugins or frameworks, and their applications will reside by themselves in a single .type of database. Thus there is no reason to add application name prefixes to all your database objects

Data Type Suffixes

Some guides (again generally older ones), suggest suffixing your column names with the data type of the field. For example, a text field .for a name would be name_tx. There will even be extensive lists to translate from data types to suffixes, text -> tx, date -> dt, etc

id
bigserial
,PRIMARY KEY
email
,text NOT NULL

,first_name text NOT NULL

last_name
,text NOT NULL

;(((CONSTRAINT person_ck_email_lower_case CHECK (email = LOWER(email

;(CREATE INDEX person_ix_first_name_last_name ON person (first_name, last_name

.Explain plans will now be easy to understand. We can clearly see that the index on first name and last name, ie

:person_ix_first_name_last_name, is being used

;;EXPLAIN SELECT * FROM person WHERE first_name = 'alice' AND last_name = 'smith&#39 #=

QUERY PLAN

(Index Scan using person_ix_first_name_last_name on person (cost=0.15..8.17 rows=1 width=72

((Index Cond: ((first_name = 'alice'::text) AND (last_name = 'smith'::text

(rows 2)

Constraints

Similar to indexes, constraints should given descriptive names. This is especially true for check constraints. It's much easier to

.diagnose an errant insert if the check constraint that was violated lets you know the cause

) CREATE TABLE team

id
bigserial
,PRIMARY KEY
name
;(text NOT NULL

) CREATE TABLE team_member

team_id
bigint
REFERENCES
,(team(id
person_id
bigint
REFERENCES
,(person(id

;((CONSTRAINT team_member_pkey PRIMARY KEY (team_id, person_id

.Notice how PostgreSQL does a good job of giving descriptive names to the foreign key constraints

d team_member\ #=

"Table "public.team_member

Column | Type | Modifiers

----------+--------+----------

team_id | bigint | not null

person_id | bigint | not null

:Indexes

(team_member_pkey" PRIMARY KEY, btree (team_id, person_id"

:Foreign-key constraints

(team_member_person_id_fkey" FOREIGN KEY (person_id) REFERENCES person(id"

(team_member_team_id_fkey" FOREIGN KEY (team_id) REFERENCES team(id"

:If we try inserting a row that violates one of these constraints we immediately know the cause just based on the constraint name

;(INSERT INTO team_member(team_id, person_id) VALUES (1234, 5678 <=

"ERROR: insert or update on table "team_member" violates foreign key constraint "team_member_team_id_fkey

."DETAIL: Key (team_id)=(1234) is not present in table "team
Similarly, if we try inserting an email address that is not lower case into the person table created above, we'll get a constraint violation :error that tells us exactly what is wrong
:This insert will work --
;(;INSERT INTO person (email, first_name, last_name) VALUES ('alice@example.com', 'Alice', 'Anderson&#39 <= INSERT 0 1
:This insert will not work --
;(;INSERT INTO person (email, first_name, last_name) VALUES ('bob@EXAMPLE.com', 'Bob', 'Barker&#39 <= "ERROR: new row for relation "person" violates check constraint "person_ck_email_lower_case
.(DETAIL: Failing row contains (2, bob@EXAMPLE.com, Bob, Barker
Final Thoughts
If you're starting a new project then I suggest you follow the conventions outlined here. If you're working on an existing project then .you need to be a bit more careful with any new objects you create
The only thing worse than bad naming conventions is multiple naming conventions. If your existing project already has a standard .approach to naming its database objects then keep using it
!Do you have something to add to this list, a way to improve some of these guidelines, or just think some of these are terrible? Let me know
Posted by Sehrope Sarkuni Feb 16 2014 sql databases postgresql
About
.I'm Sehrope Sarkuni, founder of JackDB. If you'd like to chat about any of these topics feel free to contact me
Recent Posts
Tweet
ANSI? Schmansi!" at PostgresConf 2018"
How I Write SQL, Part 1: Naming Conventions
AWS Tips, Tricks, and Techniques
My blog's tech stack: Pelican powered, Dokku deployed
Encrypting Docker containers on a Virtual Server
Tags
pelican, git, postgresql, security, paas, s3, dokku, sql, aws, crypto, dkim, blog, 2fa, meta, ec2, digitalocean, databases, docker, email, ssh
Follow @sehrope
About
.I'm Sehrope Sarkuni, founder of JackDB. If you'd like to chat about any of these topics feel free to contact me
Recent Posts
ANSI? Schmansi!" at PostgresConf 2018"
How I Write SQL, Part 1: Naming Conventions
AWS Tips, Tricks, and Techniques
My blog's tech stack: Pelican powered, Dokku deployed
Encrypting Docker containers on a Virtual Server
Tags
pelican, git, postgresql, security, paas, s3, dokku, sql, aws, crypto, dkim, blog, 2fa, meta, ec2, digitalocean, databases, docker, email, ssh
Follow @sehrope

Names Are Contracts
Database objects are referenced by their names, thus object names are part of the contract for an object. In a way you can consider .your database table and column names to be the API to your data model
Once they are set, changing them may break dependent applications. This is all the more reason to name things properly before the .first use
Developer Context Switching
Having consistent naming conventions across your data model means that developers will need to spend less time looking up the names of tables, views, and columns. Writing and debugging SQL is easier when you know that person_id must be a foreign key to the .id field of the person table
Naming Conventions

Avoid quotes. If you have to quote an identifier then you should rename it. Quoted identifiers are a serious pain. Writing SQL by hand .using quoted identifiers is frustrating and writing dynamic SQL that involves quoted identifiers is even more frustrating
.This also means that you should never include whitespace in identifier names
."Ex: Avoid using names like "FirstName" or "All Employees
Lowercase. Identifiers should be written entirely in lower case. This includes tables, views, column, and everything else too. Mixed case identifier names means that every usage of the identifier will need to be quoted in double quotes (which we already said are not .(allowed
."Ex: Use first_name, not "First_Name
Data types are not names. Database object names, particularly column names, should be a noun describing the field or object. Avoid .using words that are just data types such as text or timestamp. The latter is particularly bad as it provides zero context
.(Underscores separate words. Object name that are comprised of multiple words should be separated by underscores (ie. snake case
.Ex: Use word_count or team_member_id, not wordcount or wordCount
Full words, not abbreviations. Object names should be full English words. In general avoid abbreviations, especially if they're just the type that removes vowels. Most SQL databases support at least 30-character names which should be more than enough for a couple .English words. PostgreSQL supports up to 63-character for identifiers
.Ex: Use middle_name, not mid_nm
"Use common abbreviations. For a few long words the abbreviation is both more common than the word itself. "Internationalization .and "localization" are the two that come up most often as i18n and l10n respectively. In these cases use the abbreviation
.If you're in doubt, use the full English word. It should be obvious where the abbreviation makes sense
Avoid reserved words. Avoid using any word that is considered a reserved word in the database that you are using. There aren't that many of them so it's not too much effort to pick a different word. Depending on the context, reserved words may require quoting. This .means sometimes you'll write "user" and sometimes just user
.Another benefit of avoiding reserved words is that less-than-intelligent editor syntax highlighting won't erroneously highlight them
.Ex: Avoid using words like user, lock, or table
.Here are the list of reserved words for PostgreSQL, MySQL, Oracle, and MSSQL
Singular Relations
Tables, views, and other relations that hold data should have singular names, not plural. This means our tables and views would .be named team, not teams
.Rather than going into the relational algebra explanation of why this is correct I'll give a few practical reasons
?They're Consistent. It's possible to have a relation that holds a single row. Is it still plural
.They're unambiguous. Using only singular names means you don't need to determine how to pluralize nouns
?Ex: Does a "Person" object go into a "Persons" relation or a "People" one? How about an "Octopus" object? Octopuses? Octopi ?Octopodes
Straightforward 4GL Translation. Singular names allow you to directly translate from 4GL objects to database relations. You may .need to remove some underscores and switch to camel case but the name translation will always be straight forward
.Ex: team_member unambigously becomes the class TeamMember in Java or the variable team_member in Python
Key Fields
Primary Keys
Single column primary key fields should be named id. It's short, simple, and unambiguous. This means that when you're writing .SQL you don't have to remember the names of the fields to join on
) CREATE TABLE person
!This is a bad idea
.Field data types can change. A date field could become a timestamp, an int could become a bigint or numeric
Explicit Naming
Some database commands that create database objects do not require you specify a name. An object name will be generated either randomly (ex: fk239nxvknvsdvi) or via a formula (ex: foobar_ix_1). Unless you know exactly how a name will be generated and you are .happy with it, you should be explicitly specifying names
This also includes names generated by ORMs. Many ORMs default to creating indexes and constraints with long gibberish generated names. The couple minutes of time savings in the short run are not worth the head ache in remembering what fkas9dfnksdfnks refers .to in the long run
Indexes

Indexes should be explicitly named and include both the table name and the column name(s) indexed. Including the column names make it much easier to read through SQL explain plans. If an index is named foobar_ix1 then you would need to look up what columns that index covers to understand if it is being used correctly.

```
CREATE TABLE person (
```