# Community standup notes

## Intro

* PG talk, but for non-PG people too
* What is PG?
  + Open source, free (but enterprise versions, HyperScale, paid support...). Mention Citus, AWS Redshift.
  + Unix culture, but Windows too.
* EF generally same feature-set, passes same specification tests.
* But not 100% the same: PG is case-sensitive, SQL Server is insensitive; quoting rules are different, types are different. TEXT (vs. varchar/nvarchar, max, not max)
* ADO and EF
* Perf

## Just types

* Inet
  + Show DbType.String parameter vs. Object parameter\
  + Show returning ID
* PostgreSQL extensions
* Mention postgis, new ltree support...

## Arrays

* Just as a data type, cool.
  + string[] on SQL Server fails
  + Many-to-many. Simple query with simple include, no filter yet. Heavy, inefficient. We want NoSQL.
  + Back to string[], with CSV value converter, including some seeding. Works, but does not support (efficiently) querying for all blogs with a given tag.
  + Remove the value converter, switch to Npgsql. Look at the seeding logs.
* Querying, efficiency, indexes
  + Add Where with Contains: containment (WHERE b."Tags" @> ARRAY['bar']::text[]).
  + Switch to db pane, do EXPLAIN - seq scan.
  + Switch back and add an index on tags. Doesn't work! Why?
  + Load 50k rows, still doesn't work!
  + GIN index.
  + How about searching for blogs with any of several tags? .Where(b => b.Tags.Any(t => new[] { "foo", "bar" }.Contains(t)))
  + EXPLAIN
* Contains, IN, constant expansion
  + Go back to SQL Server (uncomment value converter, comment weird PG queries)
  + Show Contains over the blog name, with a constant array - all good.
  + Change to parameter - constant expansion. Explain.
  + Query cache explosion
  + Incomplete caching within EF Core internally
  + Switch back to Npgsql (leave value converter which will cause GIN index to bomb). Show ANY.
  + Add an index, do again. Note that seeding takes much more time (another index). Note that it's like a bunch of ORs.

SUMMARIZE: different indexes are used by different operation types; the EF provider translates to the right SQL so that indexes can be used.

* Alternative related entity loading (single/split), code sample below. Mention Emill.

## Composite

* Show SQL
* Fully nestable! Arrays of composites too!
* Supported at the ADO.NET level, not at the EF Core level (though could be)

## JSON

* PG has two JSON types. SQL Server stores only as text, so:
  + JSON correctness isn't enforced
  + Documents need to be reparsed (e.g. in order to project something out)
  + Indexing/lookup can be limited
* Show simple string mapping: Blog with Details, [Column(TypeName = "jsonb")].
* Create Details class, with Orders etc.
* Seed a single Blog with some details/orders, see in the database
* Add a simple query: .Where(b => b.Details.Views > 5)
* Complex query: .Where(b => b.Details.Orders.Length > 5)
* Containment query: .Where(b => EF.Functions.JsonContains(b.Details, new Details { Views = 8 })
* This is indexed
* Support is still somewhat limited (e.g. partial updates)

Talk about JSON in EF Core, owned entity types

## Other stuff

* Native enum
* Ranges (NodaTime DateInterval)
* NodaTime?