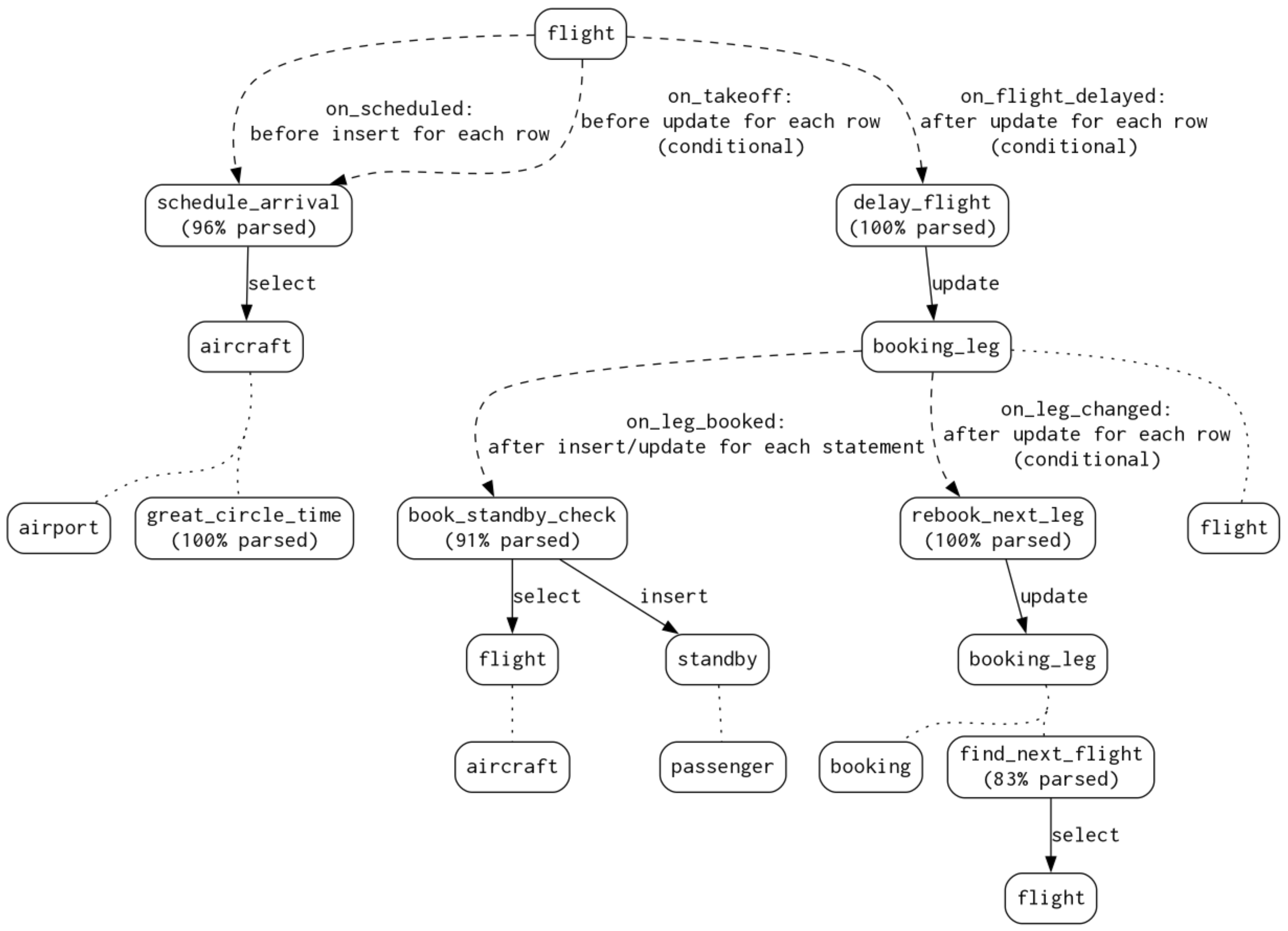
pdot

gitlab.com/dmfay/pdot

Tools for exploring databases

PGConf.EU talk


flight
on_scheduled: before insert for each row
on_takeoff: before update for each row (conditional)
on_flight_delayed: after update for each row (conditional)
schedule_arrival (96% parsed)
delay_flight (100% parsed)
select
update
aircraft
booking_leg
on_leg_booked: after insert/update for each statement
on_leg_changed: after update for each row (conditional)
airport
great_circle_time (100% parsed)
book_standby_check (91% parsed)
rebook_next_leg (100% parsed)
flight
select
insert
update
flight
standby
booking_leg
aircraft
passenger
booking
find_next_flight (83% parsed)
select
flight

Databases are full of graphs, but very few of them are ever visualized: entity-relationship diagrams, sometimes query plans. The rest are left to ad-hoc whiteboard scribbles at best, because formal visual documentation of an evolving schema has historically been a terrible investment. However, diagrams are powerful tools for learning that map naturally to how humans think of and navigate space, and aid focusing attention individually or in shared contexts.

pdot makes database diagramming disposable and therefore useful past the initial design phase. Its primary use so far is as a shell integration which displays graphs directly, allowing the user to “move” their perspective around the database and take in a manageable amount of information at a time. It can also be scripted to produce images for inclusion in documentation.

Currently:

- **Foreign key graph**: ERD-like, or using a simple heuristic to filter for related groups of tables
- **Views**: plot recursive references to an origin relation to see impact of table in schema
- **Functions**: parse bodies and show statement references to relations, recursing into function invocations
- **Triggers** (pictured): diagram recursive trigger cascades as a trigger on the origin table writes to a second, firing another trigger, etc
- **Policies**: parse USING and WITH CHECK to show involved relations and recurse into function invocations
- **Grants**: diagram role inheritance and object permissions
- **GraphViz** (default) or **Mermaid** output
- **ZSH** shell integration

Next:

- There are definitely more graphs out there in Postgres! Please tell me your favorites!
- Getting this into contrib seems like a pretty long shot but I’m interested
- I’m planning to work on an EXPLAIN view next, if I can figure out how to pop $EDITOR
- After that, I’d like to integrate statistics somehow to try to show *why* EXPLAIN is doing what it’s doing, TBD
- Function and policy body parsing uses tree-sitter-sql, a grammar I help maintain. It does not actually support procedural languages, hence the parse ratios above; there are a few ways to tackle this that need evaluation
- I am not very good or experienced with Rust and my code could absolutely be better & faster, please help