Propagate Data Changes Without Triggers!

Jonathan S. Katz
NYC PostgreSQL User Group
December 14, 2017

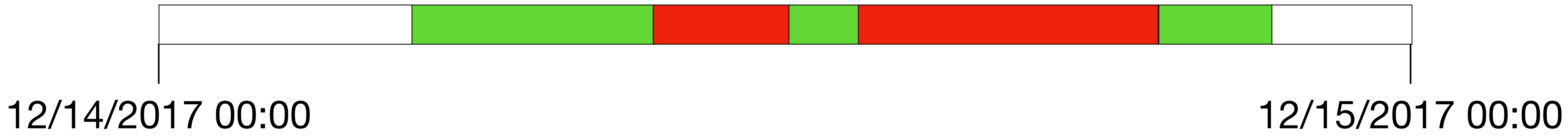
About

- CTO, VenueBook
- Co-Organizer, NYC PostgreSQL User Group (NYCPUG)
- Director, United States PostgreSQL Association
- Volunteer on postgresql.org
- Co-Founder, PGConf US
- @jkatz05

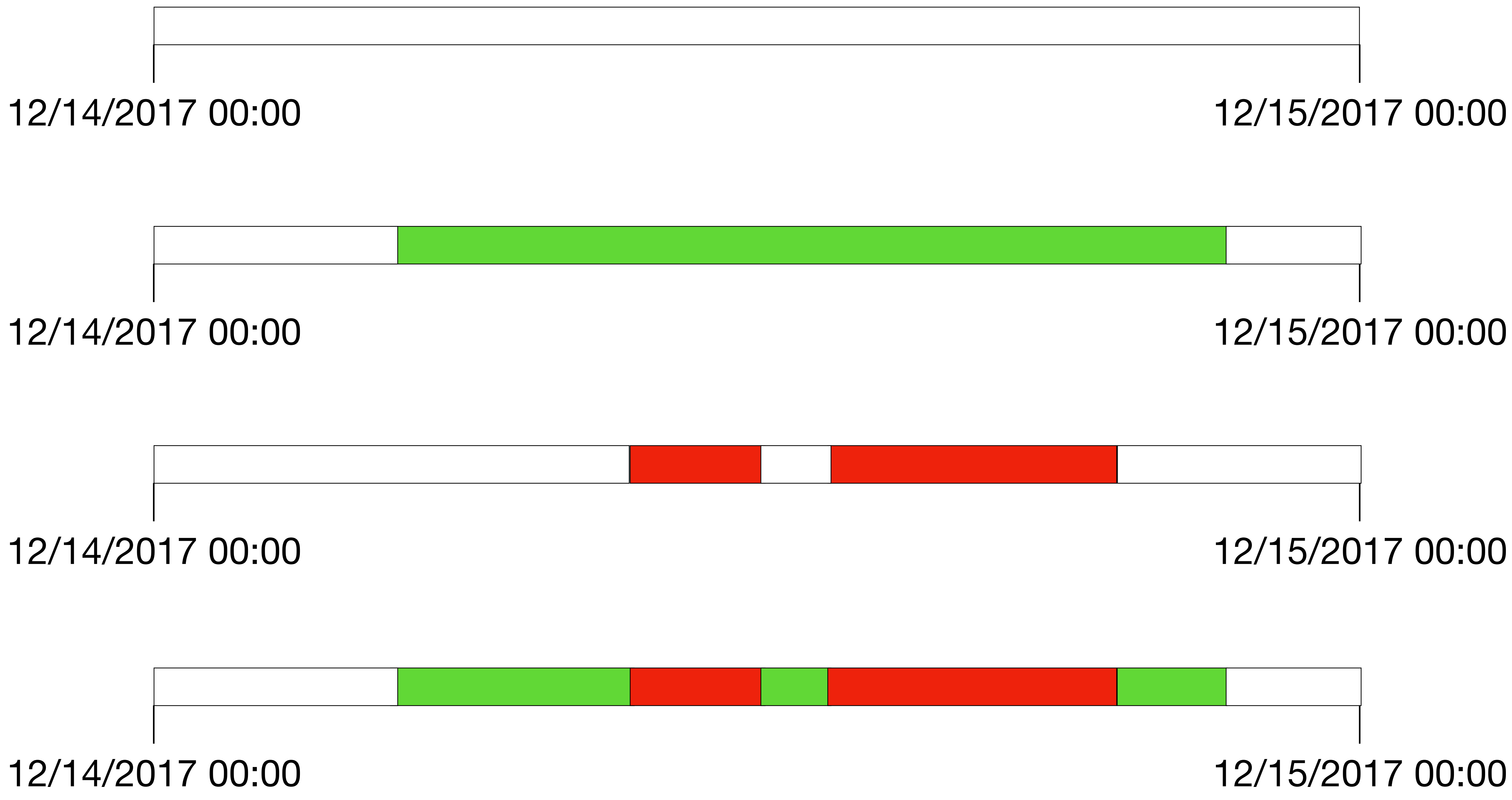
Imagine...

- We manage the space at Workbench!
- We have a set of operating hours where we can book events
- Only one event can be booked at a time

The Problem: Availability



The Problem: Availability



Easy, Right?

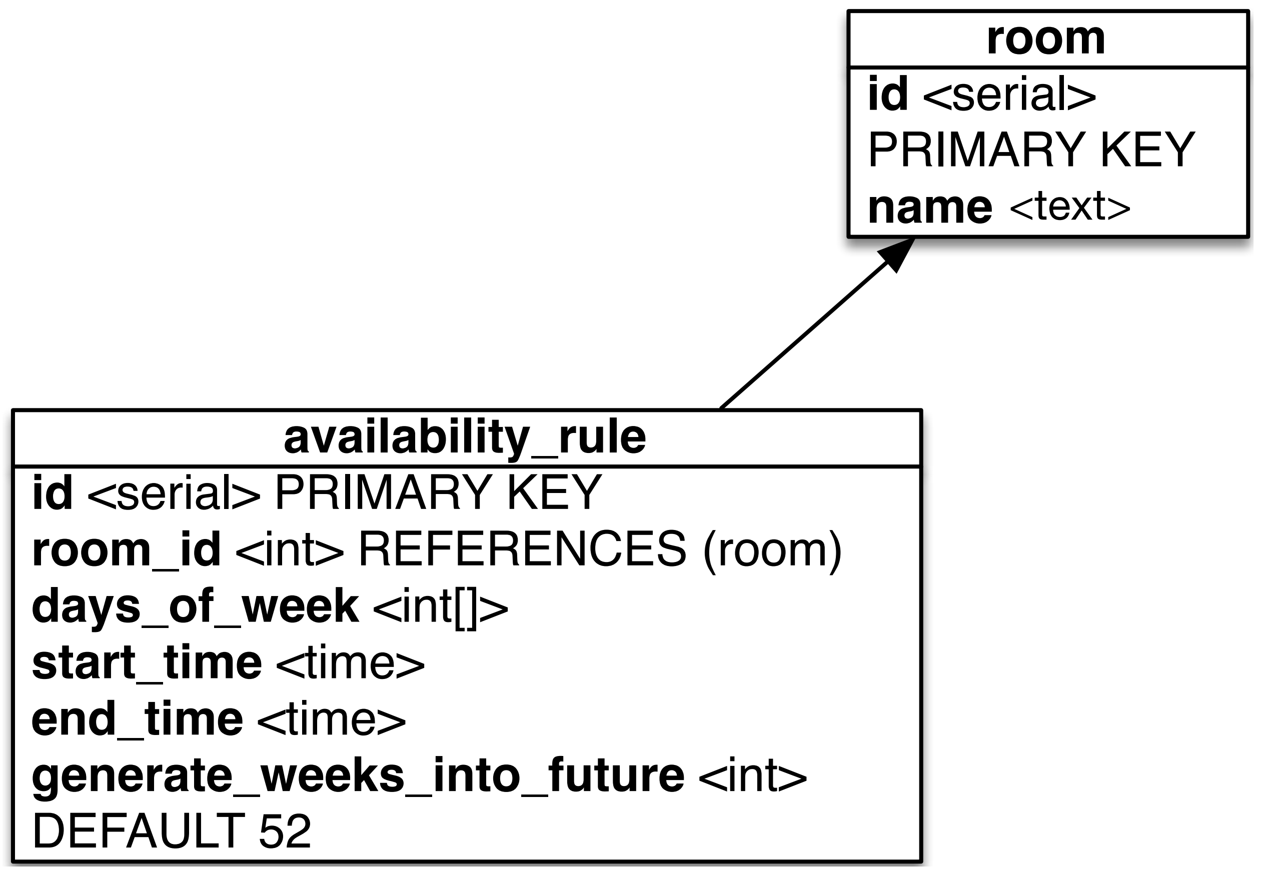
But...

- Availability
    - Just for one day - what about other days?
    - What happens with data in the past?
    - What happens with data in the future?
- Unavailability
    - Ensure no double-bookings
    - Overlapping Events?
- Just one space

🤔

Managing Availability

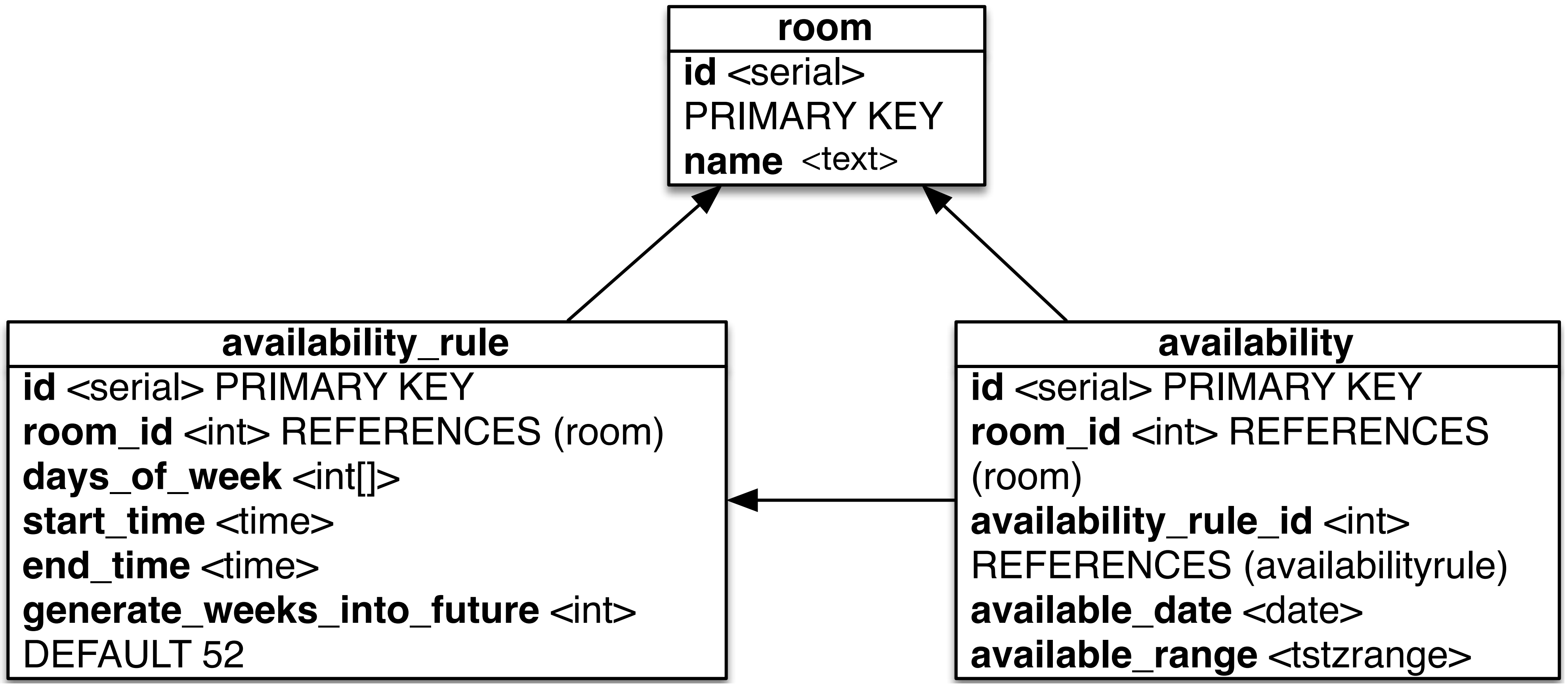
- Can create rules that can generate availability



room
id <serial> PRIMARY KEY
name <text>
availability_rule
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
days_of_week <int[]>
start_time <time>
end_time <time>
generate_weeks_into_future <int> DEFAULT 52

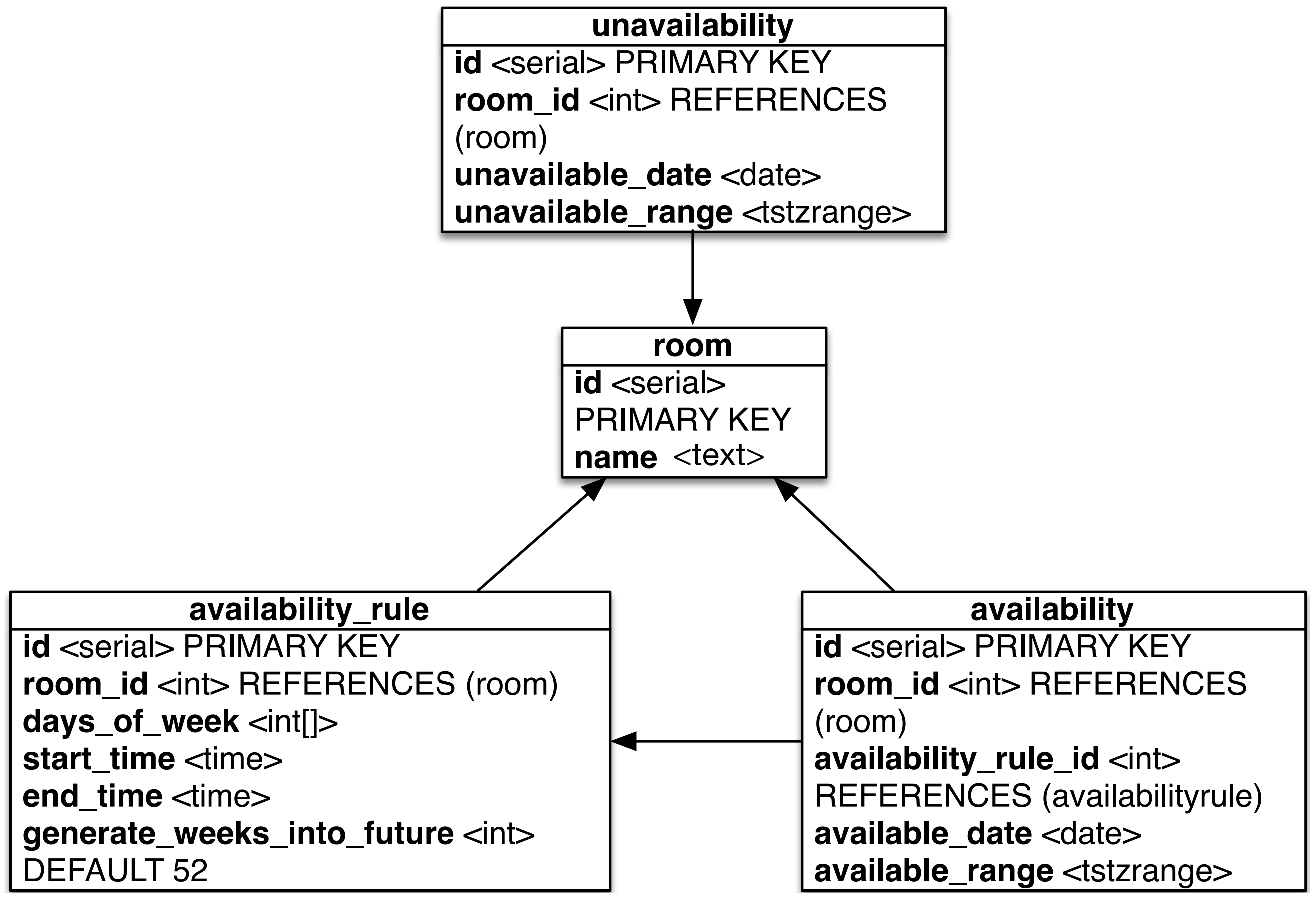
Managing Availability

- The rules can then determines what the availability is for a given date


room
id <serial> PRIMARY KEY
name <text>
availability_rule
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
days_of_week <int[]>
start_time <time>
end_time <time>
generate_weeks_into_future <int> DEFAULT 52
availability
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
availability_rule_id <int> REFERENCES (availabilityrule)
available_date <date>
available_range <tstzrange>

Managing Availability

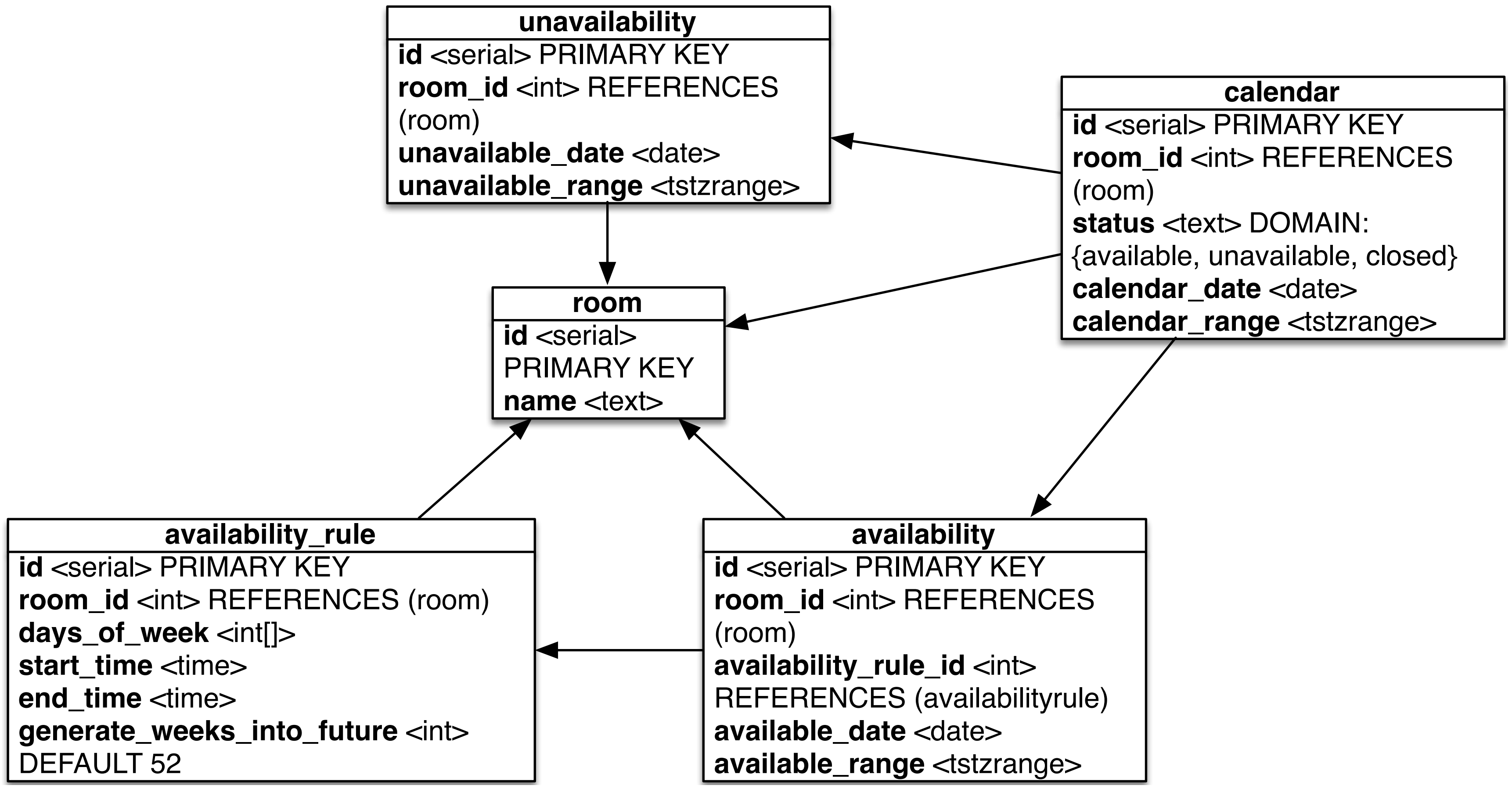
- We need to know when a room is being used



unavailability
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
unavailable_date <date>
unavailable_range <tstzrange>
room
id <serial> PRIMARY KEY
name <text>
availability_rule
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
days_of_week <int[]>
start_time <time>
end_time <time>
generate_weeks_into_future <int> DEFAULT 52
availability
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
availability_rule_id <int> REFERENCES (availabilityrule)
available_date <date>
available_range <tstzrange>

Managing Availability

- And we can have a calendar set up for quick lookups



unavailability
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
unavailable_date <date>
unavailable_range <tstzrange>
calendar
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
status <text> DOMAIN: {available, unavailable, closed}
calendar_date <date>
calendar_range <tstzrange>
room
id <serial> PRIMARY KEY
name <text>
availability_rule
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
days_of_week <int[]>
start_time <time>
end_time <time>
generate_weeks_into_future <int> DEFAULT 52
availability
id <serial> PRIMARY KEY
room_id <int> REFERENCES (room)
availability_rule_id <int> REFERENCES (availabilityrule)
available_date <date>
available_range <tstzrange>

Semi-out-of-scope but...

- GiST vs SP-GiST on the tstzrange types

Keeping the Calendar in Sync

- Triggers!
  - Triggers can fire BEFORE and AFTER write operations [INSERT, UPDATE, DELETE]
  - Triggers must successfully execute before transaction commits

Demo #1: The Setup

Demo #2: Basic Management

Demo #2 Lessons

- [Test your live demos before running them, and you will have much success!]
- availability_rule inserts took some time, ~500ms
  - availability: INSERT 52
  - calendar: INSERT 52 from nontrivial function
- Updates on individual availability / unavailability are not too painful
- Lookups are faaaaaaaast

Demo #3:
Go Big or Go Home

Demo #3 Lessons

- Even with only 100 more rooms with a few set of rules, rule generation time increased 30%
- Lookups are still lightning fast!

Logical Decoding

- Added in PostgreSQL 9.4
- Replays all logical changes made to the database
    - Create a logical replication slot in your database
    - Only one receiver can consume changes from one slot at a time
    - Slot keeps track of last change that was read by a receiver
        - If receiver disconnects, slot will ensure database holds changes until receiver reconnects

Logical Decoding Out of the Box

- A logical replication slot has a name and a decoder
    - PostgreSQL comes with the "test" decoder
    - Have to write a custom parser to read changes from test decoder

Decoder Examples

- wal2json: https://github.com/eulerto/wal2json
- jsoncdc: https://github.com/posix4e/jsoncdc

Driver Support

- C: libpq
  - pg_recvlogical
- PostgreSQL functions
- Python: psycopg2 - version 2.7
- JDBC: version 42

Demo #4: Prerequisites

- wal2json
- In postgresql.conf (requires restart):
  - wal_level = logical
  - max_wal_senders = 2
  - max_replication_slots = 2
- In pg_hba.conf, use these DEVELOPMENT ONLY settings (requires reload):
  - local   replication   jkatz   trust
- In the databases streaming changes, run:
  - `SELECT * FROM pg_create_logical_replication_slot('calendar', 'wal2json');`
- **ONLY WORKS ON TABLES WITH PRIMARY KEYS**

Demo #4:
Watch the Changes Fly By

Demo #4 Lessons

- Every change in the database is streamed
- Need to be aware of the logical decoding format

Thoughts

- We know it takes time to regenerate calendar
- Want to ensure change propagates, but want to make sure user has great experience

Demo #5:
Calendar Streaming Changes

Demo #5 Lessons

- Logical decoding allows the bulk inserts to occur significantly faster from a transactional view
- DELETEs are tricky if you need to do anything other than using the PRIMARY KEY
- Based on implementation, changes applied serially
  - Potential bottleneck for long running queries
  - Use a distributed streaming tool like Kafka to perform follow-up queries

Conclusions

- Triggers will keep your data in sync but can have significant performance overhead
- Utilizing a logical replication slot can eliminate trigger overhead and transfer the computational load elsewhere
- Not a panacea: still need to use good architectural patterns!
- We also inadvertently covered a lot of other PostgreSQL goodies!
    - Range types
    - Recursive queries
    - generate_series
    - LATERAL

Thank You! Questions?

@jkatz05