

# Not just UNIQUE

Jeff Davis  
Truviso, Inc.

# Why is UNIQUE so unique?

- Only constraint where two tuples can conflict with each other
  - That is, the existence of one tuple precludes others from existing
- Effectively a predicate lock on a very simple predicate
- Special code to enforce unique constraint in the BTree code – doesn't work GiST, etc.

# Motivation for More General Constraints

- PERIOD data type can represent a period of time:
  - <http://pgfoundry.org/projects/temporal>
- Definite beginning and end time, e.g., the period of time during which a professor is teaching in a classroom
- But two professors can't teach in the same classroom at the same time
- So periods of time cannot overlap

# Digression: Temporal Data

- Managing temporal data effectively is important and a very common problem.
- Hopefully you saw Scott Bailey's talk earlier – if not, wait for the video.
- Enforcing non-overlapping constraints is one of the few pieces that requires good support in the core PostgreSQL engine.

# Non-Overlapping Constraint

- Very commonly known as a “schedule conflict”
- How do you specify a non-overlapping constraint in PostgreSQL currently?
- Any ideas?

# Non-Overlapping Constraint

- Very commonly known as a “schedule conflict”
- How do you specify a non-overlapping constraint in PostgreSQL currently?
- Any ideas?
- Most people don't enforce in DBMS
- Maybe with a trigger that takes a full table lock?
  - Horrible performance

# Non-Overlapping Constraint

- If the constraint is not enforced by the database...
- ...then it will be enforced when two professors each believe they have reserved the same room
- A duel?
- Probably a less desirable constraint enforcement mechanism than a friendly error from the DBMS

# Operator Exclusion Constraints

- New feature
- Working on getting it accepted for PostgreSQL 8.5
- Offers more general constraint enforcement mechanism

# Example

```
CREATE TABLE reservation
(
  room          TEXT,
  professor     TEXT,
  during        PERIOD,
  EXCLUSION    USING gist
  (room        CHECK WITH =,
   during      CHECK WITH &&)
);
```

# Example

```
CREATE TABLE reservation
(
  room          TEXT,
  professor     TEXT,
  during        PERIOD,
  EXCLUSION    USING gist
  ( room        CHECK WITH =,
    during      CHECK WITH && )
);
```

Can be arbitrary expression of fields in table.

# Example

```
CREATE TABLE reservation
(
  room          TEXT,
  professor     TEXT,
  during        PERIOD,
  EXCLUSION    USING gist
  (room        CHECK WITH =,
   during      CHECK WITH && )
);
```

Exclusion operator. In this case, “overlaps”.

# Example

```
CREATE TABLE reservation
(
  room          TEXT,
  professor     TEXT,
  during        PERIOD,
  EXCLUSION    USING gist
    (room       CHECK WITH =,
     during     CHECK WITH &&)
);
```

Type of index to build and use for enforcement.

# Operator Detects Conflicts

- The operator is used to search for conflicts
- Should return TRUE when two values conflict
- Should return **TRUE** when two values conflict
- So the “overlaps” operator (“&&”) would be used to enforce the constraint that no two tuples contain overlapping values

# Back to UNIQUE

- If you specify all operators as “=”, the semantics are identical to UNIQUE
- Performance slightly worse, because one additional index search is required
- But can be used with GiST

# UNUNIQUE

- If you specify all operators as “<>”, then constraint is the opposite of unique: all values must be the same!
- However, won't work for types that don't have a search strategy for “<>”.
- Not obviously useful, but it's there.

# Multi-Column Constraints

- ... EXCLUSION USING gist  
(a CHECK WITH =,  
b CHECK WITH &&) ...
- Tuple1 conflicts with Tuple2 if and only if:
  - Tuple1.a = Tuple2.a AND
  - Tuple1.b && Tuple2.b
- Otherwise, both tuples can appear in the table.

# Extra Capabilities

- Support for predicates (WHERE)
  - Constraint on a subset of the table
- Support for arbitrary expressions
  - ... EXCLUSION ((t::circle)  
CHECK WITH &&) ...
- Can use other tablespaces and index parameters, similar to UNIQUE.
- Deferrable (work in progress)
- GIN support (work in progress)

# Future Work

- Multiple constraints can use the same index
- `UNIQUE(a, b)` and `UNIQUE(a, c)` can both use an index on `(a, b, c)`
- Depending on selectivity of “a”, may perform much better than two separate indexes

# Conclusion

- Constraints are always enforced
- Sometimes by the DBMS (cheap), sometimes by real life (expensive)
- The very simple, very common “schedule conflict” constraint is almost impossible to enforce with most DBMSs
- Let's make it easy, scalable, and flexible.
- “Operator Exclusion Constraints” in 8.5