start_date, end_date, Calculate!

Mastering Dates in SQL

Eric Worden

eric@commandprompt.com

Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

About Me

- I have a strange love of SQL
- 10 years of programming SQL reports for public school and other public sector, to state and federal agencies
- Database Consultant at Command Prompt

Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Why can't life be simple?

- Data types: dates, times, timezones, ranges
- Table and column design
- Future/unknown dates
- Range constraints (overlapping)
- Query performance
- Operate on multiple sets of ranges

Data types: conventional

```
CREATE TABLE reservation
(
    check_in date,
    check_out date
);
INSERT INTO reservation VALUES ('2015-12-01', '2015-12-31');


Select ('2015-12-25' between check_in and check_out) as foo from
reservation;
 foo
-----
 t
```


Data types: Ranges

- A complex datum containing both the start and end dates
- Syntax:
  - `'[2015-12-01, 2015-12-31)'::daterange`
  - `daterange('[2015-12-01, 2015-12-31)')`
  - "[" and ")" mean "inclusive" and "exclusive"

Data types: Ranges

```
CREATE TABLE reservation
(
    stay daterange
);
INSERT INTO reservation VALUES ('[2015-12-01, 2015-12-31)');

select ('2015-12-25'::date <@ stay) as foo from reservation;
 foo
-----
 t
```


Range Operators

Operator	Description
=	equal
<>	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
@>	contains range
@>	contains element
<@	range is contained by
<@	element is contained by
&&	overlap (have points in common)
<<	strictly left of
>>	strictly right of
&<	does not extend to the right of
&>	does not extend to the left of
-|-	is adjacent to
+	union
*	intersection
-	difference

Conventional vs. Ranges

- Which is better?
- Small differences in convenience and performance
- Ranges add very useful data integrity enforcement possibilities
  - Prevent overlapping ranges

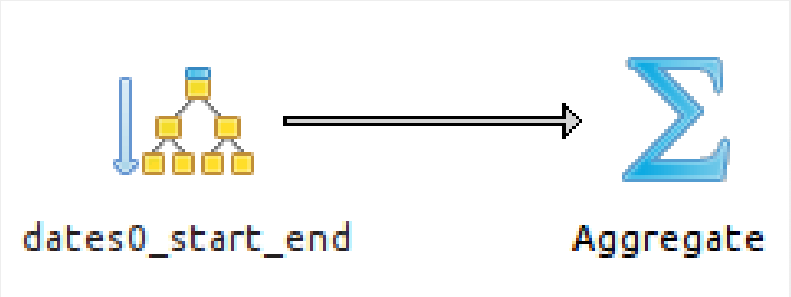
Conventional vs. Ranges Showdown

- Test table, 1 Million rows
- Start randomly in the last 3 years
- End randomly in the next 3 years
- Index (start_date, end_date)
- Index (range)

Equality: conventional

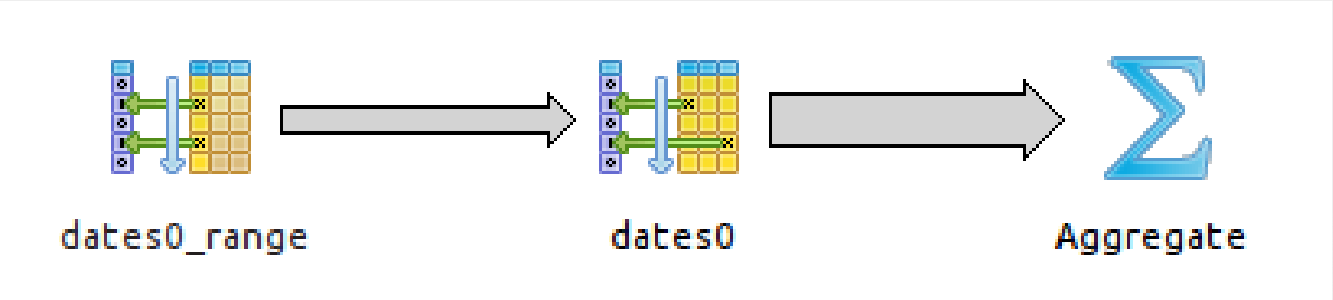
```
select count(*) from test
where
  start_date = current_date
  and end_date = (current_date + 1);
```

Time: < 1ms



Equality: range

```
select count(*) from test
where
   range = daterange(current_date,current_date + 1);
Time: 85ms
TERRIBLE???
```



Range Indexing

- Range data types require GIST indexes

```
create index ix_range on test using gist (range);
```

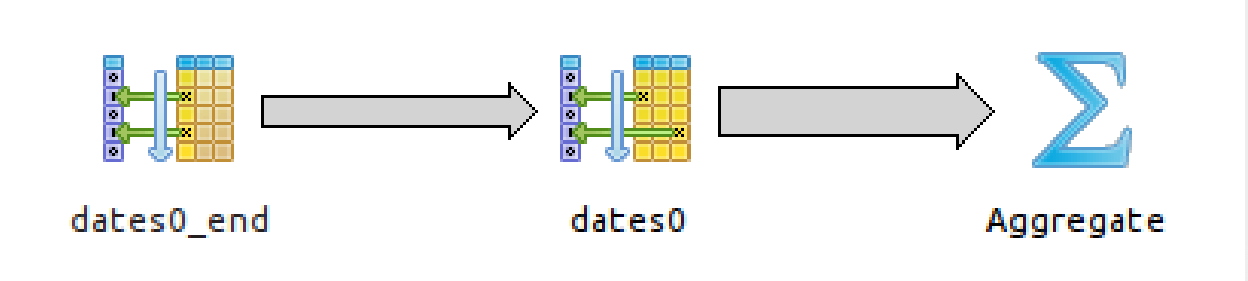
- Lousy at “=”
- Create additional btree index for “=”

```
create index ix_range_btree on test (range);
```


Contains: conventional

```
select count(*) from test

where current_date + 900 between
start_date and end_date;

 count

---------

 8000
```

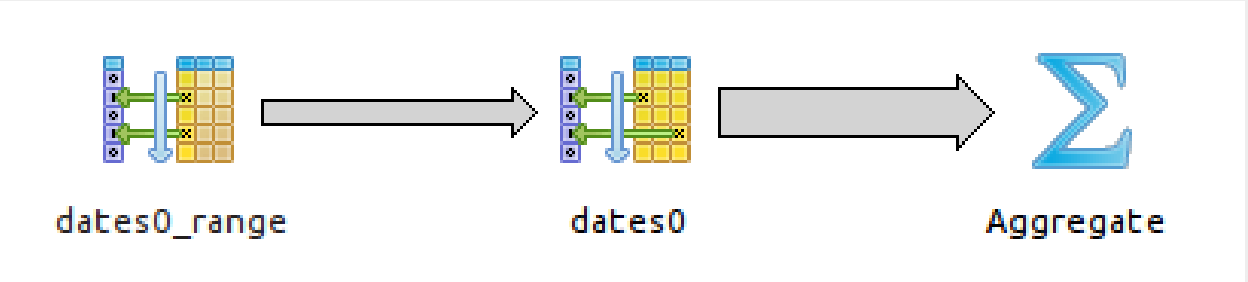
Time: 106ms



Contains: range

```
select count(*) from test
where current_date <@ range;
  count
---------
  8000
```

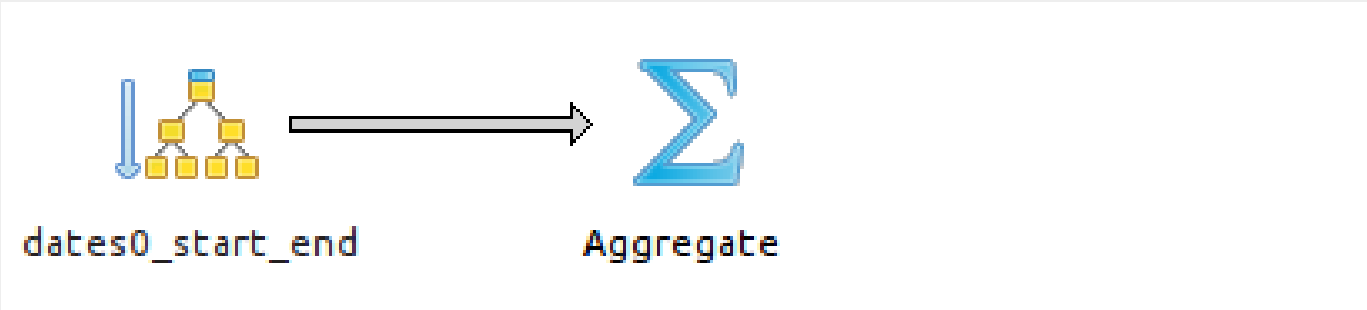
Time: 70ms



= lower bound: Conventional

- "lower" bound means the start date

```
select count(*) from test
where start_date = current_date;
Time: <1ms
```



lower(), upper()

- `lower('[1999-12-31,2000-02-01)') = '1999-12-31'`
- `upper('[1999-12-31,2000-02-01)') = '2000-01-31'`
- `upper('[1999-12-31,2000-02-01]') = '2000-02-01'`

= lower bound: Range

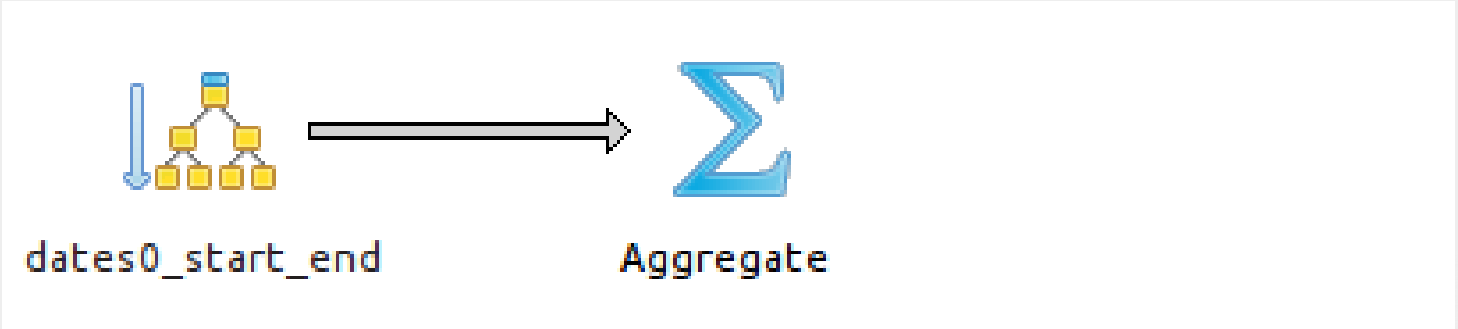
```
select count(*) from vdates0
where lower(range) = current_date;
Time: 3000ms
TERRIBLE??
```

Index Range Members

- Use a Postgresql “expression” index

```
create index lower_range on test
(lower(range));
```

```
Time: <1ms
```



Data model: single or dual columns

start_date	end_date
2015-01-01	2015-01-02
2015-09-01	2015-09-02
2015-10-01	

-VS-

pair_id	the_date	date_type
0	2015-01-01	start
0	2015-01-02	end
1	2015-09-01	start
1	2015-09-02	end
2	2015-10-01	start

Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Single Column Integrity Fail

- End date without start date

pair_id	the_date	date_type
4	2015-04-01	end

- End date precedes start date

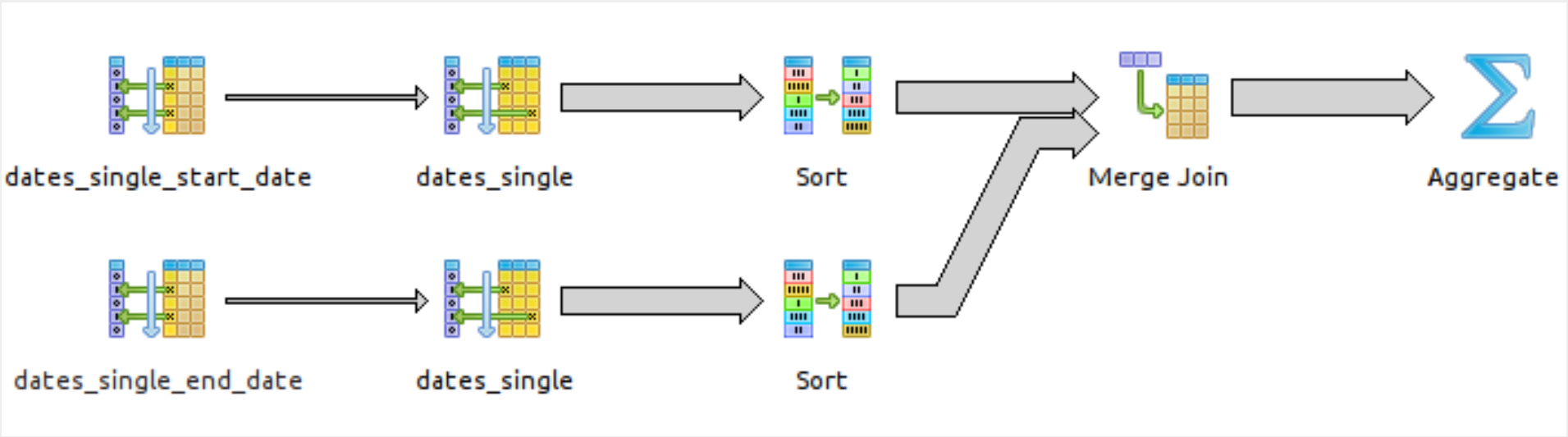
pair_id	the_date	date_type
3	1900-01-01	end
3	2015-01-02	start

Single Column Managability Fail

- Find pairs that start today and end tomorrow:

```
select count(*)
from
    (
        select 1
        from
            dates_single starts
            join dates_single ends
                on ends.pair_id = starts.pair_id
        where
            starts.date_type = 'start'
            and ends.date_type = 'end'
    ) dates
where start_date = current_date and end_date = (current_date + 1);
```

Single Column Performance Fail



Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Unknown end date

- *null*, future constant, or *infinity*?
- Example future constant: 3000-01-01
- *infinity* is a special Postgres value
  - `(current_date < 'infinity'::date) = true`
- Oracle, MySQL have different behavior
- It matters a little, not a lot, but people argue about it

ANSI null means "unknown"

- `Select count(*) from test where end_date > current_date;`
  - *null* end dates: 0
  - *infinity* end dates: 75000

*OOPS

Date “Dimension” Table

- Dates are finite (practically speaking)
- Executing date_part() will probably miss the index
- Date tables can be huge
- Therefore, precalculating and indexing values can **improve report performance**

Date Dimension Table

the_date	year	month	day	dow	mo_name	quarter
2015-02-27	2015	2	27	5	February	1
2015-02-28	2015	2	28	6	February	1
2015-03-01	2015	3	1	7	March	2

- "the_date" is the primary key
- Other tables implicitly relate to this table – foreign keys not necessary

Date Dimension Table

- Index the heck out of it
- No updates = no I/O penalty
- Materialize columns, don't rely on expression indexes
- create index all_dates_year on all_dates (year);
- create index all_dates_month on all_dates (month);
- create index all_dates_dow on all_dates (dow);
- create index all_dates_day on all_dates (day);

Dimension Table Showdown

- Run test on 1 million row table
- Query all “Friday the 13th in 2015
- Plans vary on different Postgres versions, but times are similar

Dimension Table Showdown: No dimension table

```
select start_date from test
where
    start_date between '2015-01-01'
       and '2015-12-31'
    and extract(day from start_date) = 13
    and extract(dow from start_date) = 5;
```

- Entire index scanned
- Time: 134ms

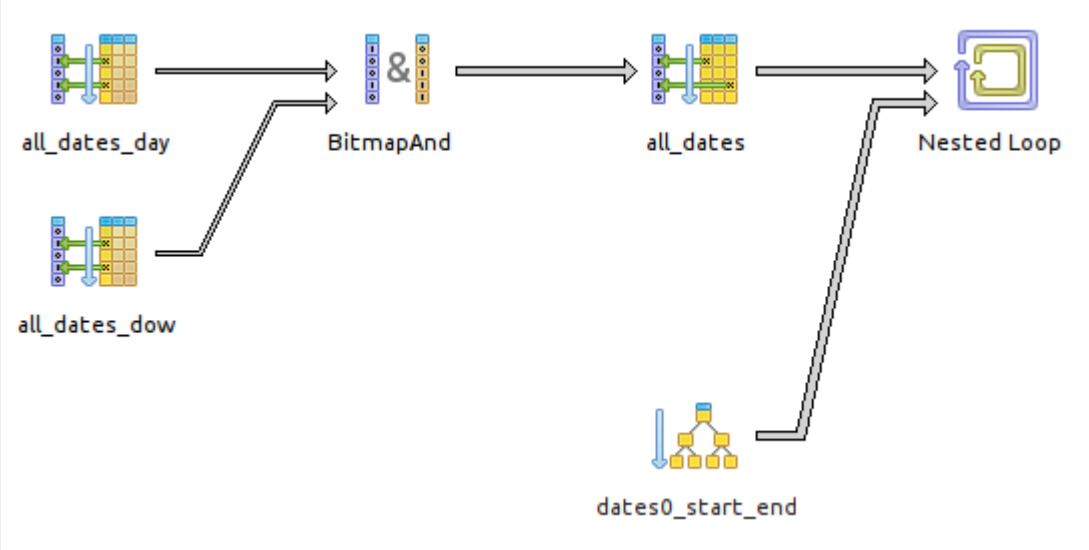
Dimension Table Showdown: Expression indexes

```
select test.start_date
from
    test
    join all_dates ad
        on test.start_date = ad.the_date
where
    extract(year from ad.date) = 2015
    and extract(dow from ad.date) = 5
    and extract(day from ad.date) = 13;
```

Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Dimension Table Showdown: Expression indexes

- Planner has fewer statistics on indexes than columns
- Time: 6ms

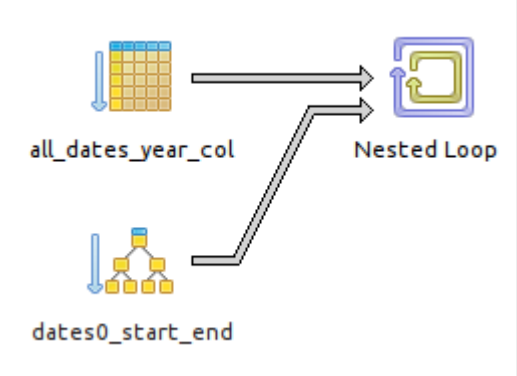


Dimension Table Showdown: Dimension columns

```
select test.start_date
from
    test
    join all_dates ad
        on test.start_date = ad.the_date
where
    ad.year = 2015
    and ad.dow = 5
    and ad.day = 13;
```


Dimension Table Showdown: Dimension columns

- Planner knows exactly what to expect
- Time: 4ms



More Data Integrity

- Catban.com, Free Trial! Eliminate cats from your Facebook feed!
- Catban.com wants to prevent freeloaders who take serial free trials

Facebook ID	trial_start	next_trial_after
grouchygrouch	2015-02-01	2016-02-01
cathatin	2015-03-25	2016-03-25
cathatin	2015-04-25	2016-04-25

Data integrity: exclusion constraint

- Same concept as "unique" constraint, but more flexibility with operators
- With GIST-indexed constraint on range column

```
alter table freetrials add constraint
ex_range exclude using gist

(facebook_id with =, range with &&);
```

Data integrity: foreign column constraint

- Prevent paid subscriber from cancelling subscription and starting free trial

Facebook ID	paid_sub_start	paid_sub_end
felineterror	2015-02-01	2016-02-01

Facebook ID	trial_start	next_trial_after
felineterror	2015-03-01	2016-03-01

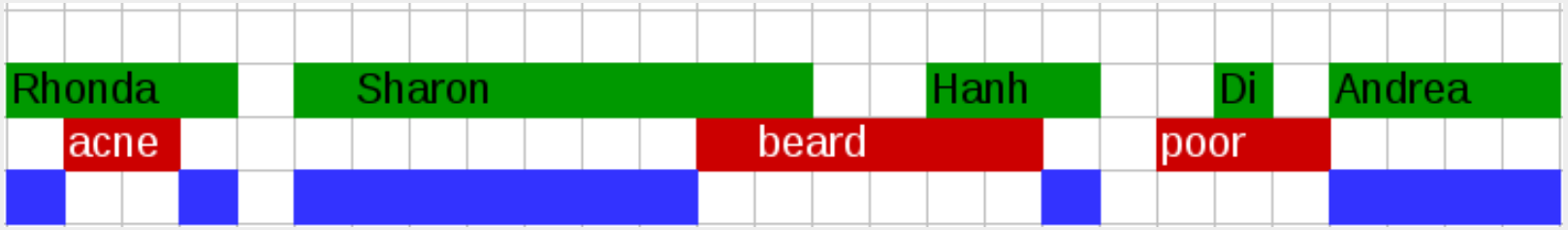
Data integrity: foreign column constraint

- Postgresql is out of tricks: write a trigger or an in-app check

```
CREATE TRIGGER free_trial_insert
    FOR EACH ROW
    EXECUTE PROCEDURE freeloader();
```


Sets of ranges

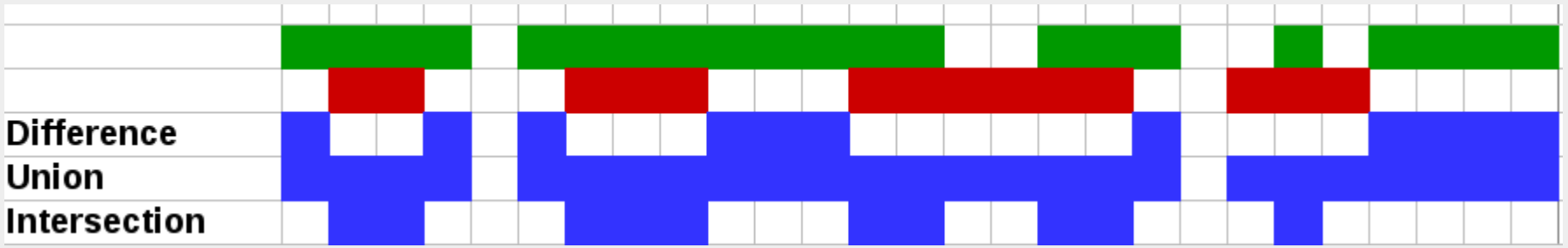
- Compare two sets of ranges to each other
- Example*: dating during my 20's



*For illustrative purposes, the large red “stupid” range is omitted

Set Operations

- Difference
- Union
- Intersection



Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

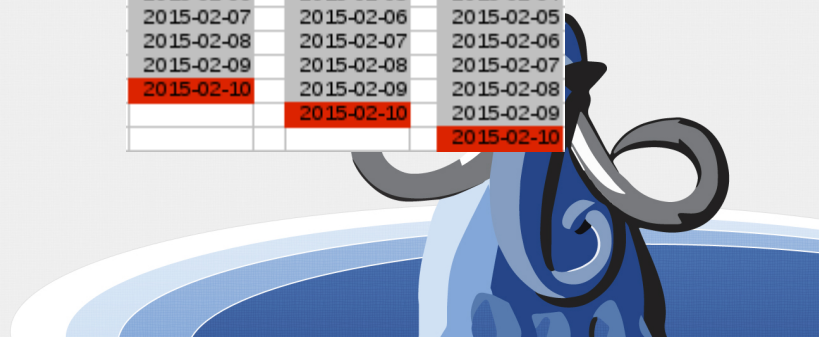
Set Operations: Web Programmer Solution

- If
  - Loop
    - If
      - Loop
        - Case
          - Foo
          - Bar
    - Else
      - Loop
  - If
    - Loop
- Else
  - Where was I??

Set Operations: Insane SQL Solution

- Dimensions date table
- where [not] in
- lead(), lag() window functions
- Accurate, elegant, but horrible performance

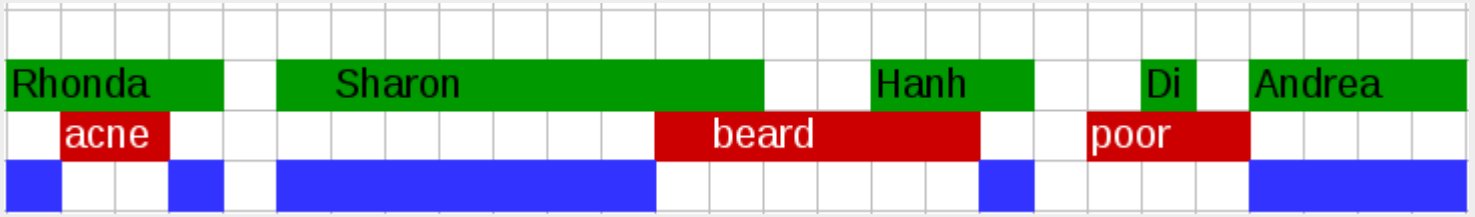
2015-01-01		
2015-01-02	2015-01-01	
2015-01-03	2015-01-02	2015-01-01
2015-01-04	2015-01-03	2015-01-02
2015-01-05	2015-01-04	2015-01-03
2015-01-06	2015-01-05	2015-01-04
2015-01-07	2015-01-06	2015-01-05
2015-01-08	2015-01-07	2015-01-06
2015-01-09	2015-01-08	2015-01-07
2015-01-10	2015-01-09	2015-01-08
2015-01-11	2015-01-10	2015-01-09
2015-01-12	2015-01-11	2015-01-10
2015-01-13	2015-01-12	2015-01-11
2015-01-14	2015-01-13	2015-01-12
2015-01-15	2015-01-14	2015-01-13
2015-01-16	2015-01-15	2015-01-14
2015-01-17	2015-01-16	2015-01-15
2015-01-18	2015-01-17	2015-01-16
2015-01-19	2015-01-18	2015-01-17
2015-01-20	2015-01-19	2015-01-18
2015-01-21	2015-01-20	2015-01-19
2015-01-22	2015-01-21	2015-01-20
2015-01-23	2015-01-22	2015-01-21
2015-01-24	2015-01-23	2015-01-22
2015-01-25	2015-01-24	2015-01-23
2015-01-26	2015-01-25	2015-01-24
2015-01-27	2015-01-26	2015-01-25
2015-01-28	2015-01-27	2015-01-26
2015-01-29	2015-01-28	2015-01-27
2015-01-30	2015-01-29	2015-01-28
2015-01-31	2015-01-30	2015-01-29
2015-02-01	2015-01-31	2015-01-30
2015-02-02	2015-02-01	2015-01-31
2015-02-03	2015-02-02	2015-02-01
2015-02-04	2015-02-03	2015-02-02
2015-02-05	2015-02-04	2015-02-03
2015-02-06	2015-02-05	2015-02-04
2015-02-07	2015-02-06	2015-02-05
2015-02-08	2015-02-07	2015-02-06
2015-02-09	2015-02-08	2015-02-07
2015-02-10	2015-02-09	2015-02-08
	2015-02-10	2015-02-09
		2015-02-10



Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Set Operations: Smart Solution

- For start_date is (green start) not in red,
  - End date is (red start) in green
- etcetera...



Set Operations: PgChronos

- Create extension pgchronos;
-
```
CREATE FUNCTION difference(
  dr1  IN daterange[ ],
  dr2  IN daterange[ ]
) RETURNS daterange[ ] AS...
```

-
```
CREATE OPERATOR - (
  PROCEDURE = difference,
  LEFTARG = daterange[],
  RIGHTARG = daterange[]
);
```

Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Set Operations: PgChronos Example

```
Select unnest(
    (select array_agg(range) from women) -
    (select array_agg(range) from excuses));

    dating_women
-------------------------
 [2015-01-02,2015-01-06)
 [2015-01-07,2015-01-21)
 [2015-01-22,2015-01-29)
```

Command Prompt, Inc.
http://www.commandprompt.com/
@cmdpromptinc

Connect

- Eric Worden
- eric@commandprompt.com
- Whatcom Postgresql
- https://github.com/worden341/pgchronos