Database Anti-Patterns

Robert Treat

PGCon 2008

Introduction

Robert Treat

DBA, part time postgres ninja

http://www.brighterlamp.org/

OmniTI

Internet Scalability Experts

http://omniti.com/is/hiring

Ground Rules

Reality...

- Talk covers lot of ground
- Not enough time to go into details on every topic
- Plus, we are up against pub time

Ground Rules

So....

- Aim is to familiarize you with concepts
- Give you terms for further research
- Google/Yahoo Are Your Friend

Ground Rules

By the way...

- Questions are good
- Arguments are not

The Grand Scheme

Schema Design

Sketchy Data

Indexes and Constraints

Query Tips

Data Manipulation

Schema Design

Data Types

Defining Data Sets

Normalization

Surrogate Keys

EAV Pattern

Trees

Data Types

- **Just use text**
  - char/varchar/text the same under the hood
  - avoid artificial limits
- **Focus on functions**
  - Phone numbers often require string manipulation
  - Unix timestamp vs. Date arithmetic
- **Minimize typecasts**

Defining Data Sets

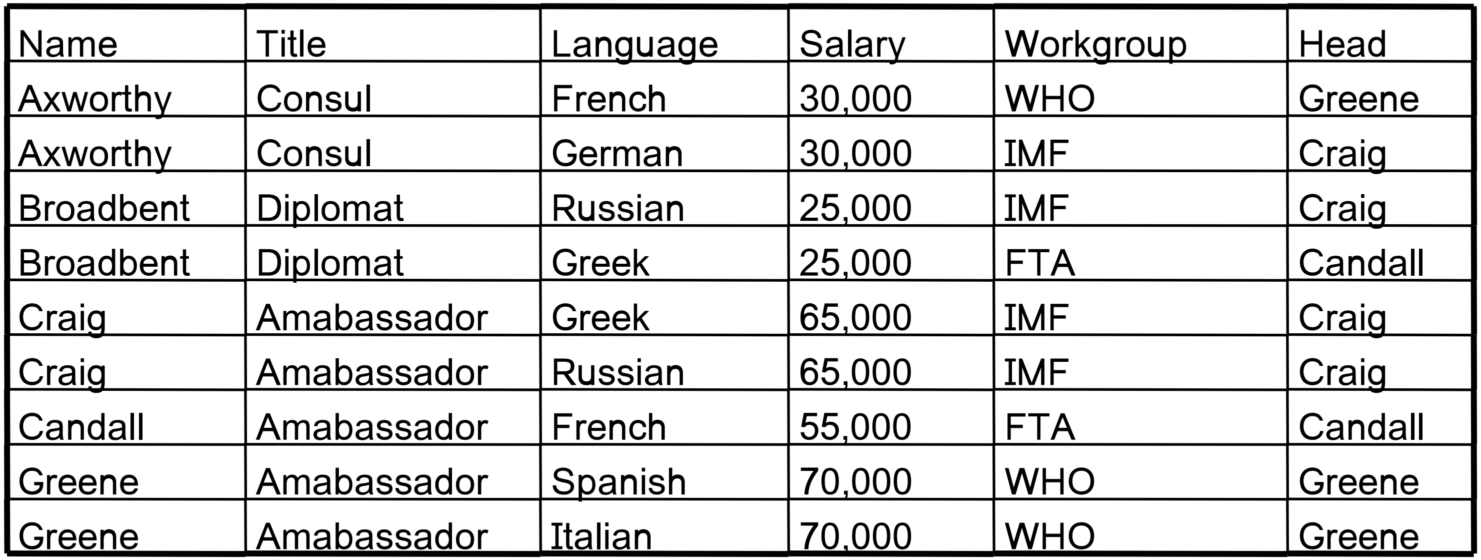
- **Take advantage of strong data typing**
  - CHECK limits input at column level
  - ENUM limits specific values at type level
    - Allows you to define a custom order, provides compact storage
  - DOMAIN defines a data type within constraint boundaries
  - Often outperforms JOIN on lookup tables
  - Allows for simpler schema design
- **Be aware of negative side effects**
  - Changing definitions will involve heavy locks
  - Some changes require table rewrite
  - Corner cases (arrays, functions)

Normalization overview

- **Hierarchy of rules for removing redundant data from tables**
  - Helps avoiding INSERT, UPDATE, DELETE anomalies
- **Multiple Normal Forms (NF)**
  - Aim for 3rd NF by default
  - Beyond that can get obscure and not always relevant
- **Denormalize to fix specific performance issues**
  - Balance slow down for INSERT/UPDATE/DELETE with improved performance for SELECT
  - Requires additional logic to handle redundent data

Normalization (1NF)

- **All columns contain only scaler values (not lists of values)**
  - Split Language, Workgroup, Head
  - Name, Language, and Workgroup are now the PK
- **Add all possible permutations?**



Name	Title	Language	Salary	Workgroup	Head
Axworthy	Consul	French	30,000	WHO	Greene
Axworthy	Consul	German	30,000	IMF	Craig
Broadbent	Diplomat	Russian	25,000	IMF	Craig
Broadbent	Diplomat	Greek	25,000	FTA	Candall
Craig	Amabassador	Greek	65,000	IMF	Craig
Craig	Amabassador	Russian	65,000	IMF	Craig
Candall	Amabassador	French	55,000	FTA	Candall
Greene	Amabassador	Spanish	70,000	WHO	Greene
Greene	Amabassador	Italian	70,000	WHO	Greene

Normalization dependence

- **Column A is**
  - Set dependent if its values are limited by another column
  - Functionally dependent if for every possible value in a column, there is one and only one possible value set for the items in a second column
    - Must hold true for all possible values
  - Transitively dependent on another column C if that column is dependent on column B, which in turn is dependent on column C

Normalization (2NF)

- **All non-key columns must be functionally dependent on PK**
  - Title, Salary are not functionally dependent on the Language column
  - Head is set dependent on Workgroup

Name	Language
Axworthy	French
Axworthy	German
Broadbent	Russian
Broadbent	Greek
Craig	Greek
Craig	Russian
Candall	French
Greene	Spanish
Greene	Italian

Name	Title	Salary	Workgroup	Head
Axworthy	Consul	30000	WHO	Greene
Axworthy	Consul	30000	IMF	Craig
Broadbent	Diplomat	25000	IMF	Craig
Broadbent	Diplomat	25000	FTA	Candall
Craig	Amabassador	65000	IMF	Craig
Candall	Amabassador	55000	FTA	Candall
Greene	Amabassador	70000	WHO	Greene

Normalization (3NF)

- **All non-key columns must be directly dependent on PK**
  - Head is only dependent on the Name through the Workgroup column

Name	Title	Salary
Axworthy	Consul	30000
Broadbent	Diplomat	25000
Craig	Amabassador	65000
Candall	Amabassador	55000
Greene	Amabassador	70000

Name	Language
Axworthy	French
Axworthy	German
Broadbent	Russian
Broadbent	Greek
Craig	Greek
Craig	Russian
Candall	French
Greene	Spanish
Greene	Italian

Name	Workgroup
Axworthy	WHO
Axworthy	IMF
Broadbent	IMF
Broadbent	FTA

Workgroup	Head
FTA	Candall
IMF	Craig
FTA	Candall
WHO	Greene

Surrogate Keys

- **Natural Key (NK) is a CK with a natural relationship to that row**
- **Surrogate Key (SK) is an artificially added unique identifier**
  - A lot of ORMs, 3rd party apps, and Martin Fowler love SK
  - Since they are artificial they make queries harder to read and can lead to more joins
    - SELECT city.code, country.code FROM city, country WHERE city.country_id = country.id and city.country = 'EN'
  - Integers do not significantly improve JOIN performance or reduce file I/O for many data sets
  - Can help in making sure the PK is really immutable (just keep them hidden)

Bareword ids

- **Most common with schemas designed around surrogate keys**
  - Makes SQL less obvious to read
    - SELECT id, id, id FROM foo, bar, baz WHERE ...
  - Makes ANSI JOIN syntax more cumbersome
    - JOIN foo USING (bar_id)
    - JOIN foo ON (foo.bar_id = bar.id)
  - Often resort to alias columns to add clarity, scoping
  - Some ORMs really like this (can be overridden)
  - Use verbose id names instead
    - Create table actor (actor_id, full_name text);

Foreign keys

- **DBMS manages relational integrity with FOREIGN KEYs**
  - Ensure that parent row exists in lookup table
    - FOREIGN KEY (parent_id) REFERENCES parent(id)
  - Automatically act on child row when parent row is updated or deleted
    - ON UPDATE CASCADE
    - ON DELETE RESTRICT
    - ON DELETE SET NULL
  - Much safer than having ORM or worse hand maintained code handle this
    - Works on multiple applications, including CLI

Entity Attribute Value Pattern

- **Uses type, name, value to store “anything”**
  - Value type if forces as varchar/text
  - Cannot model constraints (unique, etc.) efficiently
  - Often becomes dumping ground for unrelated data
- **Other options**
  - Seek out proper relational models
    - Advanced SQL (union, subselect, etc.) can help relate tables
    - Generate DDL on the fly (polls)
  - Poor mans EAV
    - Multiple columns for different datatypes
    - Still litters table with NULLS, but indexing will work better
    - Patented (?)

Adjacency Model

- **Text book approach**
  - Each row stores id of parent
  - Root node has no parent
  - Self joins are needed to read more than one depth level in a single query
  - Depth levels to read are hardcoded into the query
    - SELECT t1.name name1, t2.name name2, t3.name name3 FROM tbl t1 LEFT JOIN tbl t2 ON t2.parent_id = t1.id LEFT JOIN tbl t3 ON t3.parent_id = t2.id where t1.name = 'foo';
  - Sub tree can be moved by modifying a single row

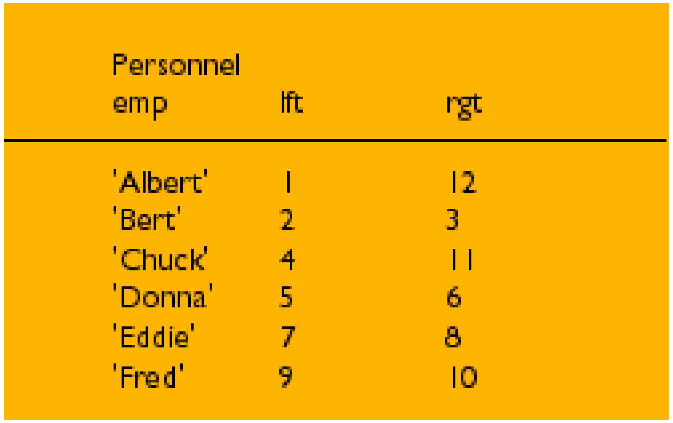
id	parent_id	name
1	NULL	US_HQ
2	1	Europe
3	2	Switzerland
4	2	Germany

Materialized Path

- **Reference parent PK through the full path for each child**
  - Violation of normalization rules
  - No join needed to fetch entire tree as well as vertical or horizontal sub tree's
    - SELECT * FROM tbl ORDER BY path, name
    - SELECT * FROM tbl WHERE path LIKE '1/23/42/%' ORDER BY path, name
    - SELECT * FROM tbl WHERE path LIKE '1/_' ORDER BY name
      - Optionally store a depth column to get rid of the LIKE
      - Optionally use array data type
  - Moving subtrees only requires changes to path column for all rows in the subtree
    - UPDATE tbl SET path = replace(path,'/1/23/42','/1/5/19') WHERE path LIKE '/1/23/42%';
  - Need to know node path

Nested Set

- **Store left and right node number for each node**
  - Start counting up from one left of the root node while moving around the outer edges
  - Very fast read performance for full tree
  - Very slow write performance



Personnel

emp	lft	rgt
'Albert'	1	12
'Bert'	2	3
'Chuck'	4	11
'Donna'	5	6
'Eddie'	7	8
'Fred'	9	10

Table 2: Nested Set Model

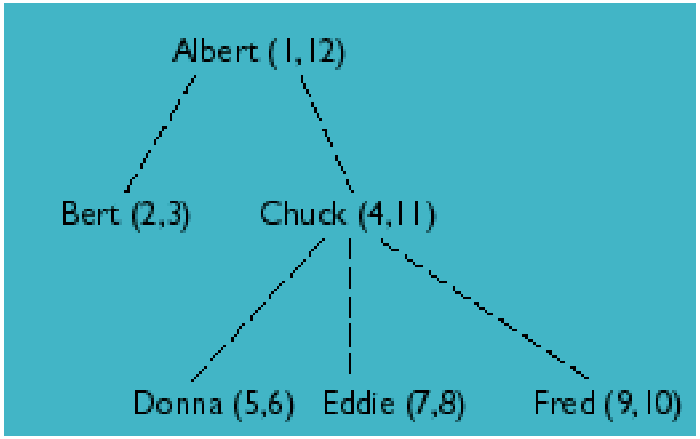


Figure 1: Directed graph.

Nested Set

- **Some example queries**
  - Get the entire path to Dylan
    - SELECT * FROM pers WHERE lft <=5 and right >=6
  - Get all leaf nodes
    - SELECT * FROM pers WHERE rgt – lft = 1
  - Get subtrees starting attached to Emma
    - SELECT * FROM pers WHERE lft > 4 and right < 11
- Changes to the tree require updating a lot of rows
  - Need to know left and right node number
  - Cannot be hand maintained
  - Results in meaningless numbers inside queries when examing log files

The Grand Scheme

Schema Design

Sketchy Data

Indexes and Constraints

Query Tips

Data Manipulation

Sketchy Data

Complex Data Structures

Images in the database

NIH Definitions

Complex Data Structures

- **Some data structures are inefficient to normalize**
  - Configurations that can have an arbitrary structure
  - Large numbers of optional fields that suggest EAV
- **Use XML**
  - If data is sometimes queried
  - If structure / data needs to be validated
- **Use serialized strings**
  - If there is no intention to ever query inside the data
    - Make sure data does not fit inside the code or configuration file that can be managed inside an SCM

Images in the database

- **Many good reasons for storing LOB in the database**
  - Replication
  - Backups
  - Access control
  - Transactions
  - OS Portability
- **Use mod_rewrite to cache public images on the filesystem**
  - mod_rewrite points missing images to a script with the name as a parameter
  - Script pulls image from database
    - If the image is public it is cached in the filesystem
  - Script returns image

Use standard definitions

- **Often data has been designed in a standard way**
  - Country Code
  - Email address
  - Zip Code
  - VIN
  - SEX (ISO 5218)
- **Helps eliminate short-sightedness**
- **Increases commonality across projects**

The Grand Scheme

Schema Design

Sketchy Data

Indexes and Constraints

Query Tips

Data Manipulation

Indexes and Constraints

Over indexing

Covering indexes

Foreign Keys

Full Text Indexing

Over indexing

- **Indexes must be updated when data changes occur**
  - INSERT/UPDATE/DELETE all touch indexes
  - Some like it HOT, pg_stat_all_tables
- **BitMap vs. Multi-Column Indexes**
  - Combine index on (a) and (b) in memory
  - Index on (x,y,z) implies index on (x) and (x,y)
- **Make sure indexes are used**
  - pg_stat_all_indexes

Covering indexes

- **Creating indexes to avoid accessing data in the table**
  - TOAST makes this less necessary
  - Visibility information stored in the table

Foreign key indexing

- **Foreign Keys ensures integrity between *two* relations**
  - Indexes automatically created on PRIMARY KEY
  - Indexes not created for child relations
  - Watch out for type mismatches (int/bigint, text/varchar)

Full text indexing

- **Add search engine style functionality to DBMS**
  - LIKE '%foo%' and LIKE '%foo' cannot use index
  - Regex searching has similar issues
  - Built-in tsearch functionality in 8.3+
    - GIN, expensive to update, very fast for searching
    - GIST, cheaper to update, not as fast for searching
- **Database Specific Syntax**

The Grand Scheme

Schema Design

Sketchy Data

Indexes and Constraints

Query Tips

Data Manipulation

Query Tips

SELECT *

Optimizating

Case for CASE

ORDER BY random()

GROUP BY

Ranking

SELECT *

- **Self-documentation is lost**
  - Which columns are needed with SELECT * ?
- **Breaks contract between database and application**
  - Changing tables in database should break dependencies
- **Hurts I/O performance**
  - SELECT * must read/send all columns
- **Useful for CLI (examples)**
- **Do not use it in production**

Premature optimization

- **Using surrogate keys or denormalization without**
  - Seeing real world specific bottleneck
  - Understanding what will slow down as a result
- **Using fancy non-standard features unless necessary**
  - I'm looking at you arrays!
- **Thinking too much about future scalability problems**

Forgetting about optimization

- **Testing performance on unrealistic data**
  - Test on expected data size
  - Test on expected data distribution
  - Many benchmark tools have data generators included
- **Not thinking about scalability beforhand**
  - This one is a fine balance, it gets easier with experience
  - Don't be afraid to draw upon outside experts if the expectation is to grow up quick

Case for CASE

- **Cut down on function calls**
  - WHERE some_slow_func() = 'foo' OR some_slow_func() = 'bar'
  - WHERE CASE some_slow_func() WHEN 'foo' THEN 1 WHEN 'bar' THEN 2 END
- **Fold multiple queries into one**
  - **Foreach ($rows as $id => $row)**
    - If (..) UPDATE foo set r * 0.90 WHERE id = $id
    - Else UPDATE foo set r * 1.10 WHERE id = $id
  - UPDATE foo SET r = (CASE WHEN r > 2 THEN r * .90 ELSE r * 1.10 END);

ORDER BY random()

- **ORDER BY random()**
  - Obvious but slow
- **>= random() limit 1**
  - Faster, but has distribution issues
- **Plpgsql functions / aggregates**
  - Not a drop in replacement

GROUP BY

- **All non-aggregate columns in SELECT/ORDER BY must be in GROUP BY**
  - **SQL Standard / Oracle only require unique column**
  - **MySQL GROUP BY is non-deterministic (ie. Broken), but allows standard syntax**
- **Can be used as an optimization hack**
  - Select distinct(name) from users (unique/sort)
  - Select name from users group by name (hashaggregate)

GROUP BY with aggregates

Rollup values into a single column

```
CREATE AGGREGATE array_accum (anyelement)
(
    sfunc = array_append,
    stype = anyarray,
    initcond = '{}'
);
```

```
pagila=# select country_id, array_accum(city) from city
pagila-# group by country_id having count(city) > 1 limit 5;
country_id |                          array_accum
------------+-------------------------------------------------------------------------
          2 | {Skikda,Bchar,Batna}
          4 | {Namibe,Benguela}
          6 | {"Vicente Lpez",Tandil,"Santa F","San Miguel de Tucumn","Almirante Brown"}
          9 | {Salzburg,Linz,Graz}
         10 | {Sumqayit,Baku}
```

Ranking

- **SQL 99 “windowing functions”**
  - Supported in Oracle, DB2 (doesn't look good for 8.4)
  - SELECT * FROM (SELECT RANK() OVER (ORDER BY age ASC) as ranking, person_id, person_name, age, FROM person) as foo WHERE ranking <=3
    - Find people with three lowest ages (include ties)
- **Alternatively use a self JOIN**
  - SELECT * FROM person AS px WHERE
(SELECT count(*) FROM person AS py
WHERE py.age < px.age) < 3
- **Find rank of a user by score**
  - SELECT count(*)+1 as rank FROM points WHERE score > (SELECT score FROM points WHERE id < :id)

The Grand Scheme

Schema Design

Sketchy Data

Indexes and Constraints

Query Tips

Data Manipulation

Data Manipulation

SQL injection

Affected Rows

MVCC

Surrogate Key Generation

CSV import/export

SQL injection

- **Always quote!**
  - Quote, validate, filter all data and identifiers from external resources
    - You can't trust external sources
    - Think about refactoring
  - Use real libraries (no addslash(), regex-fu)
- **Schema path injection**
  - Postgres allows you to modify schema path, operators, datatypes
    - Make = = <>

Affected Rows

- **Check affected rows to avoid SELECT before data change**
  - If affected rows after update / delete > 0
    - Something was modified
- **INSERT / UPDATE give RETURNING clause**
- **Good ORM supports UPDATE/DELETE without SELECT**

MVCC problems

- **MVCC prevents readers from being blocked**
  - Readers get a snapshot of the data valid at the start of their transaction
  - Can lead to issues with concurrent transactions

Transaction #1	Transaction #2	Comments
BEGIN TRANS;		
	BEGIN TRANS;	
SELECT FLIGHT 23		Seat 1A available
UPDATE FLIGHT 23		Book 1A
	SELECT FLIGHT 23	Seat 1A available
COMMIT		
	UPDATE FLIGHT 23	Book 1A
	COMMIT	

MVCC solutions

- **Add checks into UPDATE**
  - SET customer = 'foo' WHERE flight = '23' and seat = '1A' and customer IS NULL
  - Look at affected rows to check for concurrent updates
- **Use FOR UPDATE to aquire a lock in transaction**
  - SELECT seat FROM seats WHERE flight = '23' AND customer IS NULL FOR UPDATE
  - Disables benefits MVCC for the SELECT

Surrogate Key Generation

- **SERIAL type is facade over sequences**
  - Watch initializations when doing deployments, dump/restore
- **Don't clean up “holes”**
  - Point of surrogate key is to ensure uniqueness, not that they are sequential or in any particular order
- **Alternative generators**
  - UUID()
  - Timestamp
    - Watch out for multiple INSERTs per millisecond

Bulk Import / Export

- **Wrap multiple INSERT in a transaction**
- **Use multi-values INSERT syntax**
- **COPY TO/FROM**
  - Supports copy from select
  - Specific syntax to handle CSV data
- **Disable constraints**
  - Alter table disable trigger
- **Drop / Create indexes**

The End

Thanks: Lukas Smith, http://www.pooteeweet.org/

Other References:

- PostgreSQL, MySQL, Oracle online documentation
- SQL Performance Tuning by Peter Gulutzan and Trudy Plazer
- http://jan.kneschke.de/projects/mysql
- http://decipherinfosys.wordpress.com/2007/01/29/name-value-pair-design/
- http://parseerror.com/sql/select*isevil.html
- http://www.xaprb.com/blog/2007/01/11/how-to-implement-a-queue-in-sql/
- http://people.planetpostgresql.org/greg/index.php?/archives/89-Implementing-a-queue-in-SQL-Postgres-version.html
- http://arjen-lentz.livejournal.com/56292.html
- http://www.depesz.com/index.php/2007/09/16/my-thoughts-on-getting-random-row/
- http://forums.mysql.com/read.php?32,65494,89649#msg-89649
- http://troels.arvin.dk/db/rdbms/
- http://www.intelligententerprise.com/001020/celko.jhtml?_requestid=1266295
- http://archives.postgresql.org/pgsql-hackers/2006-01/msg00414.php