Database and SQL (Un)Patterns

PHP Quebec 2008

Lukas Kahwe Smith

Briefly

- Originally from Berlin, Germany
- Now living and Working on Zurich, Switzerland
- Been doing PHP since 2000
- Joined PHP.net in 2002
- Mainly worked on PEAR (MDB2, LiveUser) then
- Recently looking more and more after release management and related tasks on internals
- PHP nice, but I LOVE Ultimate Frisbee

About this talk

- Trying to raise awareness to common issues and challenges and how to solve them
- Glosses over details on most topics (except for a few topics that I like and that are rarely covered in detail)
- Ask if you want details
- However after this talk you will at least know what to put into the search engine of your choice to get an answer

Think about data types

- Most things are not at most 255 chars long, so why default everything to VARCHAR(255)?
- Avoid type casts by choosing data types according to the functions that will be applied to the column
- Avoid FKs on static lookup tables using DOMAIN/ ENUM/CHECK constraints
- Do not default to using surrogate keys, try to keep the schema as self explanatory as possible

Think about how to deal with data that is too long on the app level
Telephone example .. may look like an INT but might require more string functions (like extracting area codes)
Remember that changes require DDL (in MySQL ALTER TABLE requires table rebuild in most cases)
Classic example is Continent-Country-City tables, use 2/3 letter codes instead (meaningful logs)

EAV-(Un)Pattern

- Entity Attribute Value Pattern describes using a single table to store different types of data
- Usually requires using a very wide VARCHAR column
- Prevents usage of more optimal data types, proper indexing and applications of constraints
- Split up into separate tables with as many columns as necessary, create DDL on the fly if necessary

few people hit the limits on columns per table

Normalization

- Basic rules for database schema design
- Default should ALWAYS be 3NF (Normal Form)
- Not following these rules is a bad idea
- Data will be less accessible, changes will need to be done multiple times etc.

Denormalization

- Breaking the rules of Normalization
- Help solve specific performance issues
- Realize that denormalization will help solve one bottleneck by slowing down other parts
- Know what you are trying to solve, benchmark that it solves your problems and profile so that you do not create new problems

indexes to some extend can be considered denormalization

Trees

- Adjacency List Model is the standard
- Materialized Path is the one size fits all
- Nested Sets is the read often, write seldom
- Some RDBMS support native recursive queries using “WITH RECURSIVE” (or “CONNECT BY”)

```
SELECT t1.name name1, t2.name name2, t3.name3 FROM tbl AS t1 LEFT JOIN tbl AS t2 ON t2.parent = t1.id LEFT JOIN tbl AS t3
ON t3.parent = t2.id WHERE t1.name ‘ foo’;
SELECT * FROM tbl WHERE path LIKE ‘/1/23/42/%’ ORDER BY path, name
SELECT * FROM tbl WHERE path LIKE ‘/1/_’ ORDER BY name
SELECT * FROM pers WHERE lft <= 5 AND rgt >= 6
```

Fixed Length Queues

- Maximum of 5 entries in the queue, new entries pop older entries off from the end

```
INSERT INTO q (id, modulo, fruit)
SELECT (COALESCE(MAX(id), -1) + 1),
(COALESCE(MAX(id), -1) + 1) MOD 5, :fruit FROM q
ON DUPLICATE KEY UPDATE id = values(id), fruit = values(fruit);
```

```
REPLACE INTO q (id, modulo, fruit)
SELECT (COALESCE(MAX(id), - 1) + 1),
(COALESCE(MAX(id), - 1) + 1) MOD 5, :fruit FROM q;
```

```
CREATE RULE fruitlimit AS ON
INSERT TO q DO ALSO DELETE FROM q
WHERE id NOT IN (SELECT id FROM q ORDER BY DESC LIMIT 5);
```

first is mysql only, second also works in sqlite, third is postgresql specific

Indexes and Constraints

- Avoid redundant indexes, mind the order of columns
- Remember most constraints imply the existence of a matching index
- Covering Indexes can skip reading table data
- Full Text Indexes are great (just not yet in MySQL)
- Foreign keys off load integrity managing app logic (but can be a pain without tool support)
- CHECK Constraints can be emulated with VIEWs

More random tidbits

- "SELECT *" makes apps less readable/maintainable and tends to use disk I/O inefficiently
- Check the affected rows to determine if UPDATE changed rows instead of verifying with a SELECT before
- Over write with ON DUPLICATE KEY UPDATE (REPLACE is a bad idea because it pokes holes in storage pages)
- Do not clean "holes" in columns with surrogate keys
- Most RDBMS have some proprietary syntax for efficient CSV (and XML) Import and Export

SELECT * precludes covering indexes, column level grants
REPLACE does an INSERT and a DELETE on a constraint violation
Surrogate keys are about generating unique id's, also consider UUID()
MySQL: LOAD DATA INFILE PostgreSQL: COPY

GROUP BY

- Avoid Nondeterministic GROUP BY (enable SQL)
- Leverage RDBMS provided special purpose functions to operate on GROUP data (f.e. GROUP_CONCAT())
- Leverage RDBMS provided modifiers to get more details on the groups (f.e. WITH ROLLUP)

SQL_Mode: ONLY_FULL_GROUP_BY to force group by to require that all columns in the select list are either in an aggregate or inside the group by clause
GROUP_CONCAT(country) makes a (comma) separated list of all values in the group
WITH ROLLUP adds a row with your “grand totals” of your aggregates

Groupwise-Max

- Find countries with highest population per continent

```
SELECT name, pop FROM country c1 WHERE pop = (SELECT
MAX(c2.pop) FROM country c2 WHERE c1.continent = c2.continent)
```

```
SELECT name, pop FROM country WHERE ROW(pop, continent) IN
(SELECT MAX(c2.pop), continent FROM country GROUP BY continent)
```

```
SELECT SUBSTRING(MAX(CONCAT(LPAD(pop, 10, ‘0’),name)), 10+1) AS
name, MAX(pop) AS pop FROM country GROUP BY continent
```

Case for CASE

- Think in Sets, not in procedural loops
- Reduce function calls

```
WHERE CASE some_slow_func()
WHEN ‘foo’ THEN 1 WHEN ‘bar’ THEN 1 END
```

- Fold multiple queries into one and get rid of roundtrips between the database and the middleware:

```
UPDATE foo SET
r = (CASE WHEN r > 2 THEN r * 0.90 ELSE r * 1.10 END);
```

Prepared Statements

- In theory they improve performance by having to parse queries only once for any number of executions
- Prepared statements handles are not pooled and therefore only have the lifetime of a single request
- Prepared statements will generate very generic query plans, which can be very suboptimal
- MySQL has severe limitations in prepared statements handling (some of which are fixed in 5.1)
- PDO can emulate prepared statements

```
$db->setAttribute(PDO::ATTR_EMULATE_PREPARES, true);
```
Oracle generates plan optimal for the data provided on the first execute

ORDER BY RANDOM()

- Obvious but slow

ORDER BY RANDOM()

- random_id = rand(1, @max) and COUNT(id) = MAX(id)

WHERE id = :random_id

- Not truly random when distribution is not even, but deals with holes

WHERE id >= :random_id ORDER BY id LIMIT 1

Pivot Table

- Pivot tables also known as cross-tabs or breakdown
- Statistical reports often require for grouping data by one (or multiple) field(s)

```
SELECT name,
COUNT(CASE WHEN gender = ‘m’ THEN id ELSE NULL
END) AS ‘males’,
COUNT(CASE WHEN gender = ‘f’ THEN id ELSE NULL END)
AS ‘females’,
COUNT(*) total
FROM person GROUP BY department
```

get the number of males and females by department

Pivot Table Example

- Adding country and location as additional dimensions

```
SELECT country, loc AS location,
   COUNT(CASE WHEN dept = 'pers' AND gender = 'f' THEN id ELSE NULL END) AS 'pers-f',
   COUNT(CASE WHEN dept = 'pers' AND gender = 'm' THEN id ELSE NULL END) AS 'pers-m',
   COUNT(CASE WHEN dept = 'pers' THEN id ELSE NULL END) AS 'pers',
   COUNT(CASE WHEN dept = 'sales' AND gender = 'f' THEN id ELSE NULL END) AS 'sales-f',
   COUNT(CASE WHEN dept = 'sales' AND gender = 'm' THEN id ELSE NULL END) AS 'sales-m',
   COUNT(CASE WHEN dept = 'sales' THEN id ELSE NULL END) AS 'sales',
   COUNT(CASE WHEN dept = 'dev' AND gender = 'f' THEN id ELSE NULL END) AS 'dev-f',
   COUNT(CASE WHEN dept = 'dev' AND gender = 'm' THEN id ELSE NULL END) AS 'dev-m',
   COUNT(CASE WHEN dept = 'dev'  THEN id ELSE NULL END) AS 'dev',
   COUNT(*) AS total
FROM person
```

RANK()

- SQL 99 Windowing Functions help answer questions like what are the the 3 lowest ages (with ties)

```
SELECT * FROM (SELECT RANK() OVER (ORDER BY age ASC) AS ranking, person_id, person_name, age FROM person) AS foo WHERE ranking <= 3
```

```
SELECT * FROM person AS px WHERE (SELECT COUNT(*) FROM person AS py WHERE py.age < px.age) < 3
```

RANK() not supported in MySQL .. maybe in PostgreSQL 8.4

RANK() MySQL Style

- Find highest five salaries for each department

```
SELECT dep, sal, rank FROM
(SELECT dep, sal,
CASE WHEN @D = dep
THEN @R:=@R=1 ELSE @R:= 1 END rank,
CASE WHEN @D != dep
THEN @D:=dep END AS g
FROM tbl ORDER BY dep, sal)
AS tbl WHERE rank <= 5
```

Optimistic Locking

- Assume that no other transaction modifies the data between independent read and write transactions
- Fail instead of wait on concurrent transactions

```
INSERT INTO addr (id, cntr, street, city) VALUES (nextval('addr')), unix_t(),
'Foo Street', 'Bar Town'); // id = 23, unix_t() = 1179145598

SELECT id, cntr, street, city FROM addr;

UPDATE addr SET street = 'Foo Ave', cntr = unix_t() WHERE id = 23 AND cntr
= 1179145598;
```

- Zero affected rows means a concurrent change occurred between the SELECT and the UPDATE

MVCC

- Multi Version Concurrency Control
- Readers always see a snapshot of the data as it was when the transaction started
- As a result writers do not block readers, which can have surprising effects in transactions
- Use SELECT .. FOR UPDATE to force update LOCK
- Use additional conditions to prevent overlooking conflicting transactions

PostgreSQL does not really delete, instead it marks which transaction is supposed to see which version of a row
Most others (INNODB, Falcon, Maria, Oracle, Firebird, Interbase etc) use a rollback log. Some do it in memory, some optionally on disk, some can automatically resize the rollback log

MVCC Example

Trans #1	Trans #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;	

Add “FOR UPDATE” to the selects to cause locking until the transaction has ended
Add “WHERE 1A is free” to UPDATE and check the affected rows

Nested Transactions

- Most RDBMS do not support nested transactions, which makes it hard to write modular code

```
$dbh->beginNestedTransaction(); # BEGIN TRANS
call_module($dbh)
    $dhh->beginNestedTransaction(); # SET SAVEPOINT 1;
    if ($fail) { $dbh->failNestedTransaction(false); } # set nested transaction as failed
    } else { $dbh->completeNestedTransaction(); } # RELEASE SAVEPOINT 1;
$dbh->completeNestedTransaction(); # COMMIT/ROLLBACK;
if ($dbh->getNestedTransactionError()) { .. }
```

Files in RDBMS

- By default files have no place inside an RDBMS
- However there are exceptions to this rule in order to leverage certain DBMS features: replication, backup, access control, ACID, operating system portability
- Replicate via the DBMS and cache in the File System
- Look into MyBS to get LOB streaming in MySQL

Thank you for listening!
Questions? Comments?

smith@pooteeweet.org
http://pooteeweet.org/files/phpquebec08/sql_un_patterns.pdf

References

- MySQL and PostgreSQL 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://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://www.onlamp.com/pub/a/onlamp/2003/12/04/crosstabs.html
- http://arjen-lentz.livejournal.com/56292.html
- 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
- http://www.nexen.net/images/stories/conferences/mysqlkitchen.mce.pdf.zip
- Suggested topics by Robert Treat and links provided by the #postgresql channel bot
- My own blog http://pooteeweet.org and previous talks linked on the site