How to Design Indexes, Really

Bill Karwin, Percona Inc.

It's About Performance

- What's the most frequent recommendation in database performance audits?
- What's the easiest way to speed up SQL queries, *without* schema changes *or* code changes?
- What's a commonly neglected part of database development and maintenance?

indexes.

Indexing Mistakes

Index Shotgun

"Indexes improve performance, so I index *every* column."

Index Shotgun

- Many such indexes are never used.
- Indexes occupy space on disk in memory buffers.
- Each index needs to be modified during `INSERT`, `UPDATE`, `DELETE`.
- Query optimizer considers indexes during every query, more indexes makes this work harder.

Index Aversion

"Indexes have overhead,
so I never create *any* indexes."

Index Aversion

- The *right* indexes are crucial to speed up queries.
- Most workloads are read-heavy, so the net benefit of indexes outweighs the overhead of updating.
- Indexes are compact, so they are a more efficient use of buffer memory.

SQL Aversion

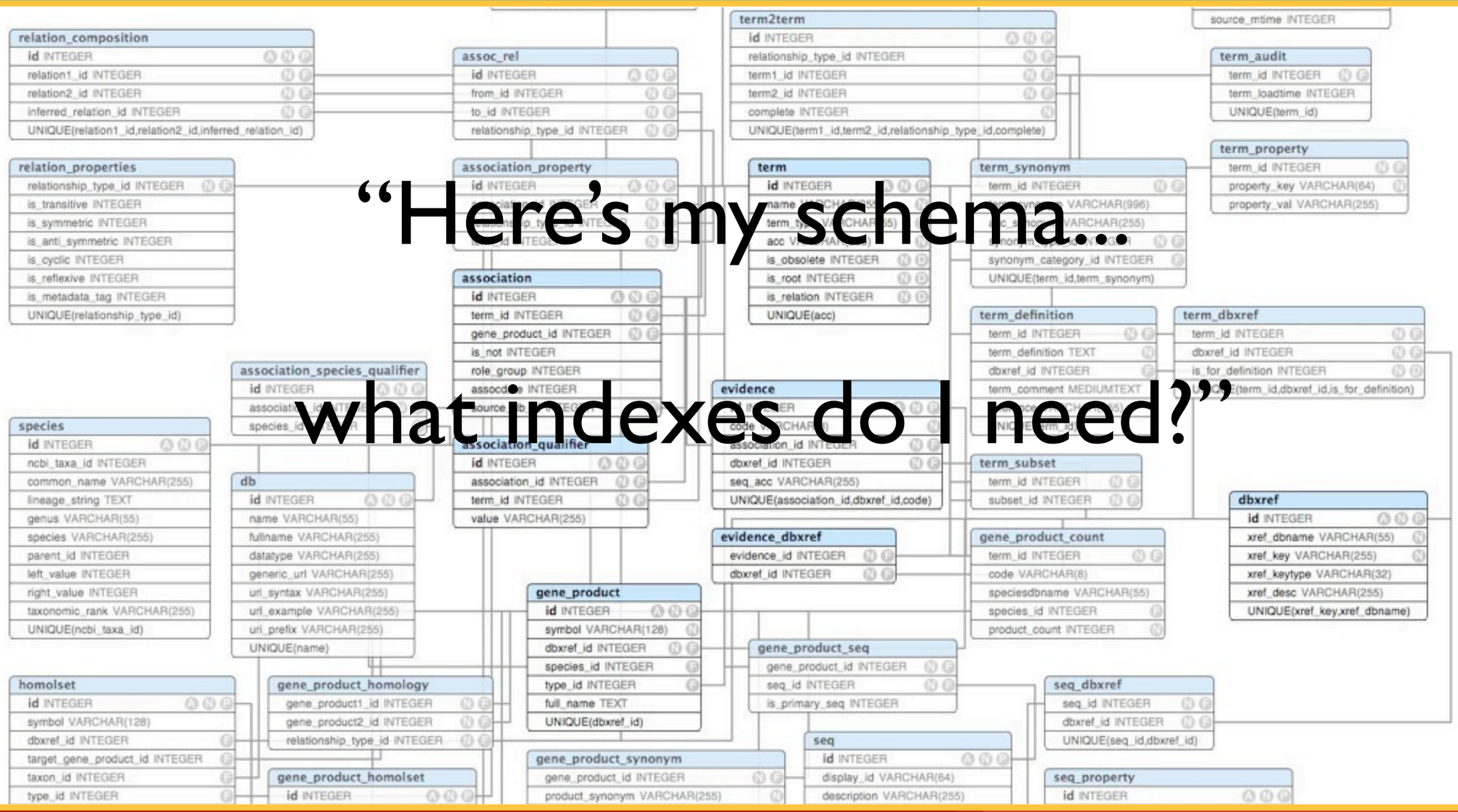
"I use a NoSQL database because it *just works*." *

(* after I create the right indexes.)

SQL Aversion

- Many non-relational databases require you to define indexes explicitly too.
- The ANSI/ISO SQL standard doesn't specify anything about indexes.
- So if you want to use NoSQL, just use indexes! :-)

Naive Questions



"Here's my schema...

what indexes do I need?"

Naive Questions

- Index choice depends on...
  - What tables and columns you need to query
  - What `JOIN`s you need to perform
  - What `GROUP BY`'s and `ORDER BY`'s you need
- Index design is not *implicit* from table design.

Lesson #1

relational schema design is based on **data**.

but

index design is based on **queries**.

Where Are the Queries?

Slow query log

General query log

Application logging

Tcpdump

~~Binary log~~

~~Process list~~

Collecting Log of Queries

- Enable slow-query log:

```
mysql> SET GLOBAL slow_query_log = ON;
```

- Include *all* queries, not just those that are slow:

```
mysql> SET GLOBAL long_query_time = 0;
```

- Percona Server can log more information:

```
mysql> SET GLOBAL log_slow_verbosity =
  'full';
```

Where to Start?

pt-query-digest

http://www.percona.com/doc/percona-toolkit/2.1/pt-query-digest.html

pt-query-digest

- Analyzes your query log.
- Reports the queries accounting for greatest load.
  - Slow queries.
  - Quick queries that run frequently.
- Outputs a ranked list of top query patterns.

```
$ pt-query-digest \
  /var/lib/mysql/mysql-slow.log \
  > ~/pqd.txt
```

Ranked Queries

```
# Profile
# Rank Query ID           Response time   Calls R/Call  Apdx V/M   Item
# ==== ================== ================ ===== ======= ==== ===== =======
#    1 0xA8D2BBDE7EBE7822 4932.2992 28.8%    78 63.2346 0.00  5.22 SELECT person_info
#    2 0xFE25DAF5DBB71F49 4205.2160 24.6%   130 32.3478 0.00  3.47 SELECT title
#    3 0x70DAC639802CA233 1299.6269  7.6%    14 92.8305 0.00  0.17 SELECT cast_info
#    4 0xE336B880F4FEC4B8 1184.5101  6.9%   294  4.0289 0.36  2.29 SELECT cast_info
#    5 0x60550B93960F1837  905.1648  5.3%    60 15.0861 0.05  1.33 SELECT name
#    6 0xF46D5C09B4E0CA2F  777.2446  4.5% 16340  0.0476 1.00  0.17 SELECT char_name
#    7 0x09FCFFF0E5BC929F  747.4346  4.4%   130  5.7495 0.53  7.69 SELECT name
#    8 0x9433950BE12B9470  744.1755  4.4% 14368  0.0518 1.00  0.18 SELECT name
#    9 0x4DC0E044996DA715  448.5637  2.6%   130  3.4505 0.65  8.31 SELECT title
#   10 0x09FB72D72ED18E93  361.1904  2.1%    78  4.6306 0.28  1.89 SELECT cast_info title
```

Slow Queries

```
# Profile
# Rank Query ID           Response time   Calls R/Call  Apdx V/M   Item
# ==== ================== ================ ===== ======= ==== ===== =======
#    1 0xA8D2BBDE7EBE7822 4932.2992 28.8%    78 63.2346 0.00  5.22 SELECT person_info
#    2 0xFE25DAF5DBB71F49 4205.2160 24.6%   130 32.3478 0.00  3.47 SELECT title
#    3 0x70DAC639802CA233 1299.6269  7.6%    14 92.8305 0.00  0.17 SELECT cast_info
#    4 0xE336B880F4FEC4B8 1184.5101  6.9%   294  4.0289 0.36  2.29 SELECT cast_info
#    5 0x60550B93960F1837  905.1648  5.3%    60 15.0861 0.05  1.33 SELECT name
#    6 0xF46D5C09B4E0CA2F  777.2446  4.5% 16340  0.0476 1.00  0.17 SELECT char_name
#    7 0x09FCFFF0E5BC929F  747.4346  4.4%   130  5.7495 0.53  7.69 SELECT name
#    8 0x9433950BE12B9470  744.1755  4.4% 14368  0.0518 1.00  0.18 SELECT name
#    9 0x4DC0E044996DA715  448.5637  2.6%   130  3.4505 0.65  8.31 SELECT title
#   10 0x09FB72D72ED18E93  361.1904  2.1%    78  4.6306 0.28  1.89 SELECT cast_info title
```

this query executed a few times,
but each takes over 1 minute

Quick Queries

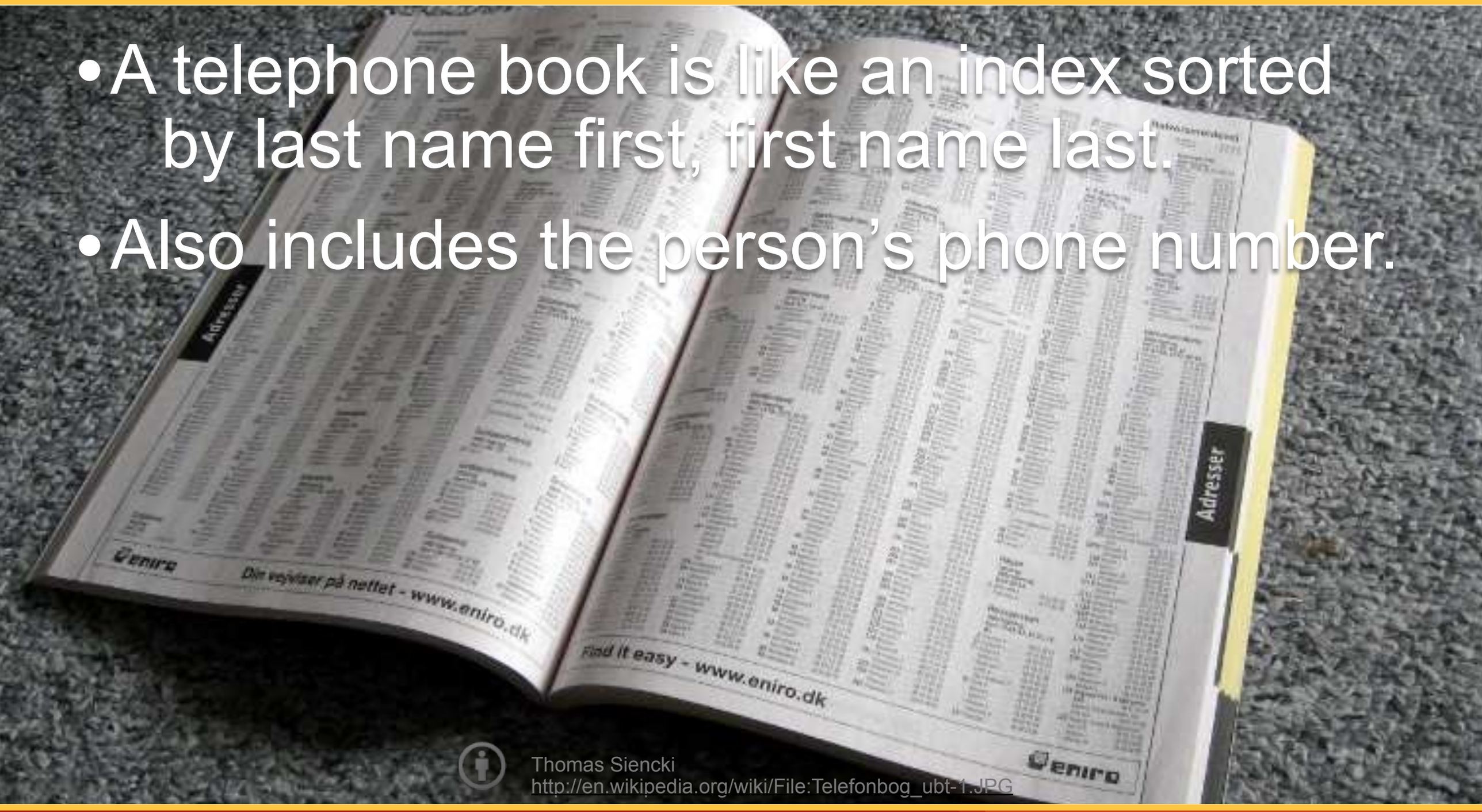
```
# Profile
# Rank Query ID           Response time   Calls R/Call  Apdx V/M   Item
# ==== ================== =============== ===== ======= ==== ===== =======
#    1 0xA8D2BBDE7EBE7822 4932.2992 28.8%    78 63.2346 0.00  5.22 SELECT person_info
#    2 0xFE25DAF5DBB71F49 4205.2160 24.6%   130 32.3478 0.00  3.47 SELECT title
#    3 0x70DAC639802CA233 1299.6269  7.6%    14 92.8305 0.00  0.17 SELECT cast_info
#    4 0xE336B880F4FEC4B8 1184.5101  6.9%   294  4.0289 0.36  2.29 SELECT cast_info
#    5 0x60550B93960F1837  905.1648  5.3%    60 15.0861 0.05  1.33 SELECT name
#    6 0xF46D5C09B4E0CA2F  777.2446  4.5% 16340  0.0476 1.00  0.17 SELECT char_name
#    7 0x09FCFFF0E5BC929F  747.4346  4.4%   130  5.7495 0.53  7.69 SELECT name
#    8 0x9433950BE12B9470  744.1755  4.4% 14368  0.0518 1.00  0.18 SELECT name
#    9 0x4DC0E044996DA715  448.5637  2.6%   130  3.4505 0.65  8.31 SELECT title
#   10 0x09FB72D72ED18E93  361.1904  2.1%    78  4.6306 0.28  1.89 SELECT cast_info title
```

another query executed frequently, each call is quick

Understanding Indexes by Analogy

Telephone Book

- A telephone book is like an index sorted by last name first, first name last.
- Also includes the person's phone number.



Implementation

- In MySQL syntax, this would look like the following:

```
CREATE INDEX phone_idx ON TelephoneBook
  (last_name, first_name, phone_number);
```

Simple Searches

- Since the index is pre-sorted, it benefits queries for last name:

```
SELECT * FROM TelephoneBook
WHERE last_name = 'Smith';
```

- By *benefit*, we mean that it helps to narrow down the search quickly, because the entries are already in sorted order.

Compound Searches

- The sort order benefits us further if we search on last name and first name together:

```
SELECT * FROM TelephoneBook
WHERE last_name = 'Smith'
AND first_name = 'John';
```

When the Index Fails

- But if we search only on first_name, the sort order doesn't help:

```
SELECT * FROM TelephoneBook
WHERE first_name = 'John';
```

- The 'John' entries occur throughout the book, in an unpredictable distribution. Anyone's first name can be 'John'. We have to search every page of the book.

Order Matters

- From this, we conclude that the order we define the columns in an index matters.
- An index with the columns declared in the opposite order would benefit a search for first name alone, but then it wouldn't help a search for last name alone.

```
CREATE INDEX phone_idx2 ON TelephoneBook
 (first_name, last_name, phone_number);
```

Both Indexes Can Be Helpful

- If both criteria use equality comparisons, then the order of columns shouldn't matter.
- Either index would benefit a query that searches for a specific value in both columns.

```
SELECT * FROM TelephoneBook
WHERE last_name = 'Smith'
AND first_name = 'John';
```

Range Comparisons

- Searching for multiple names is another case.
- An index still helps us to narrow the search even if we search for all names that match a pattern:

```
SELECT * FROM TelephoneBook
WHERE last_name LIKE 'S%';
```

- The book groups all the names starting with ‘S’ together, so the index can help us to avoid reading the whole book.

Compound Range Comparisons

- If we search by a range of last names, but a specific first name, then the first names are again distributed in an unpredictable way among the matching last names:

```
SELECT * FROM TelephoneBook
WHERE last_name LIKE 'S%'
AND first_name = 'John';
```

- The ‘John’ entries are not grouped together among all ‘S’ names, so we have to search manually every ‘S’ name to find the ones we want.

Order Matters

- From this we see that once we do a range comparison on a column in an index, any criteria we have for subsequent columns cannot benefit from the index.

```
CREATE INDEX phone_idx ON TelephoneBook
  (last_name, first_name, phone_number);
```

this column in the index helps the range search

but then subsequent columns do not help searching

Sorting by Index

- Since the telephone book is pre-sorted, we can rely on this if we need to read through the entries in that order.

```
SELECT * FROM TelephoneBook
WHERE last_name = 'Smith'
ORDER BY first_name;
```

- There's no extra work to do here; we can simply read the entries in the order they are naturally written in the book.
  - So the ORDER BY is a no-op.

When the Index Can’t Help Sorting

- Sorting by any other column is another story.

```
SELECT * FROM TelephoneBook
WHERE last_name = 'Smith'
ORDER BY phone_number;
```

- The book is already ordered by last name and then by first name, but not numerically by phone number.
- To get the result sorted in the manner above, we could copy the ‘Smith’ entries onto sticky notes, then re-order the sticky notes by phone number manually.

Order Matters

- From this, we conclude that an index helps if we sort by the column immediately following the columns in the search criteria.
- But if we need to sort by another column later in the index, or by a column that isn’t in the index at all, we have to do it the hard way.

Index-Only Searches

- Once we find the matching entries, we may need other fields of information:

```
SELECT phone_number
FROM TelephoneBook
WHERE last_name = 'Smith'
AND first_name = 'John';
```

- Because the phone number is included in the index entry, we get it for free. The entry includes all three fields, even if our query didn't search for the phone number explicitly.

When the Index Can't Help

- If we need other information that isn't included in the index, we may need to do extra work.

```
SELECT business_hours
FROM TelephoneBook
WHERE last_name = 'Smith Plumbing';
```

- In this case, we'd have to use the phone number of the business, and call them to ask their hours.
  - That's the extra work.

Columns Matter

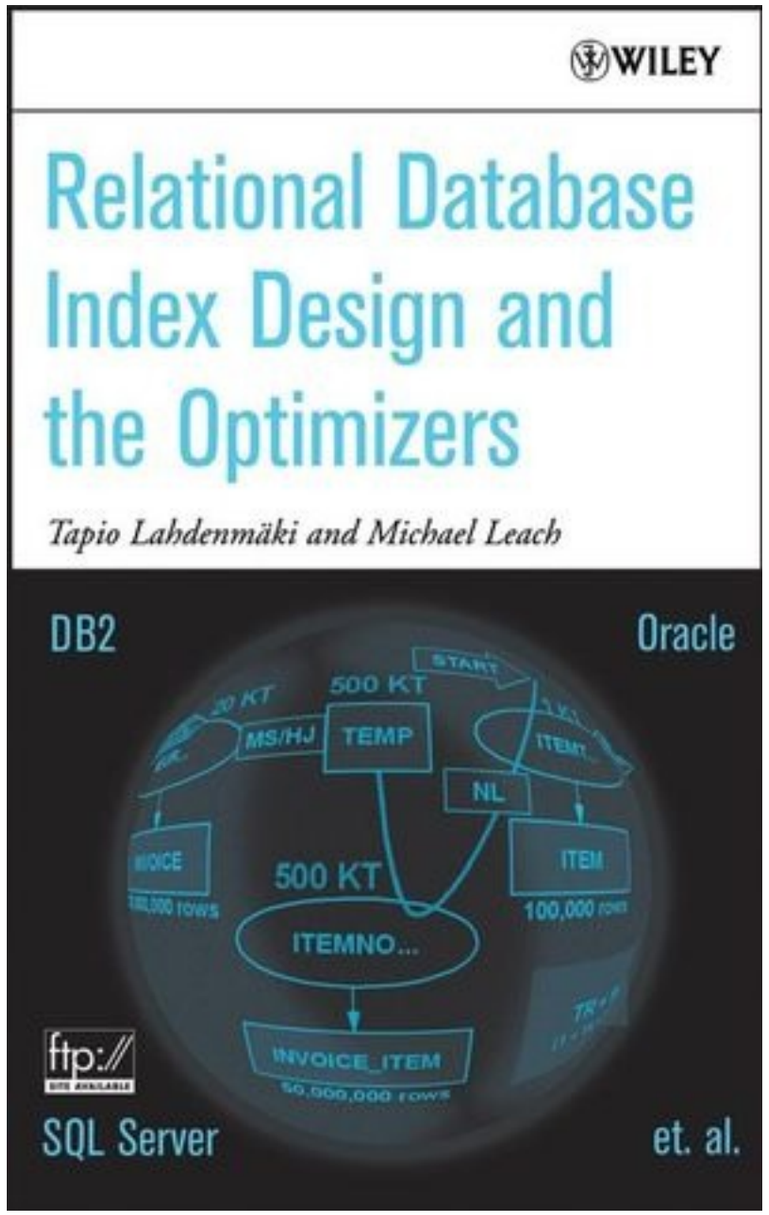
- From this, we conclude that putting columns in an index can be very important for certain queries, even if that query doesn't use the columns for searching or sorting.
- This is called a *covering index*.

Lesson #2

rate your indexes using the star system.

The Star System

- Rate an index with respect to a given query by three “stars” which describe effectiveness.
- This technique is covered in *Relational Database Index Design and the Optimizers* by Tapio Lahdenmäki and Michael Leach.
  - http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471719994.html



The Star System

★ **First star:**
Rows referenced by your query are grouped together in the index.

★ **Second star:**
Rows referenced by your query are ordered in the index the way you want them.

★ **Third star:**
The index contains all columns referenced by your query (covering index).

Look at this Terrible Query

```
mysql> EXPLAIN SELECT person_id,
  person_role_id FROM cast_info
  WHERE movie_id = 91280 AND role_id = 1
  ORDER BY nr_order ASC\G

           id: 1
  select_type: SIMPLE
        table: cast_info
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 24008907
        Extra: Using where; Using filesort
```

scans whole table

uses no index

scans a lot of rows

sorts the hard way

★ First Star

★ Pick all columns from *equality* predicates.
Define the index with these columns first.

```
SELECT person_id, person_role_id
FROM cast_info
WHERE movie_id = 91280 AND role_id = 1
ORDER BY nr_order ASC;
```

equality predicates

```
ALTER TABLE cast_info ADD INDEX
  (movie_id, role_id);
```

★ Second Star

★ Add the column(s) in the `GROUP BY` or `ORDER BY` clause, if the query has one.

```
SELECT person_id, person_role_id
FROM cast_info
WHERE movie_id = 91280 AND role_id = 1
ORDER BY nr_order ASC;
```

sorting column

```
ALTER TABLE cast_info ADD INDEX
 (movie_id, role_id, nr_order);
```

★ Third Star

★ Add any remaining columns referenced in the `SELECT` list.

```
SELECT person_id, person_role_id
FROM cast_info
WHERE movie_id = 91280 AND role_id = 1
ORDER BY nr_order ASC;
```

select-list columns

```
ALTER TABLE cast_info ADD INDEX
  (movie_id, role_id, nr_order,
  person_id, person_role_id);
```

Explain That

```
mysql> EXPLAIN SELECT person_id,
  person_role_id FROM cast_info
  WHERE movie_id = 91280 AND role_id = 1
  ORDER BY nr_order ASC\G

           id: 1
  select_type: SIMPLE
        table: cast_info
         type: ref
possible_keys: movie_id
          key: movie_id
      key_len: 8
          ref: const,const
         rows: 57
        Extra: Using where; Using index
```

★1: *effective index*

★2: *no filesort*

★3: *covering index*

Complications

- Can't achieve First-Star indexes when:
  - `WHERE` clause has more complex expressions including `OR` predicates (*disjunction*).

```
SELECT * FROM TelephoneBook
WHERE last_name = 'Smith'
OR first_name = 'John'
```

disjunction

Complications

- Can’t achieve Second-Star indexes when:
  - `WHERE` clause includes *range* predicates (>, <, `!=`, `NOT IN`, `BETWEEN`, `LIKE`, etc.).
  - Query includes both `GROUP BY` and `ORDER BY` referencing different columns.
  - Query must sort by multiple columns over different tables (can’t make a single index span tables).
  - Query includes `ORDER BY` in a different order than the order in which rows are accessed.

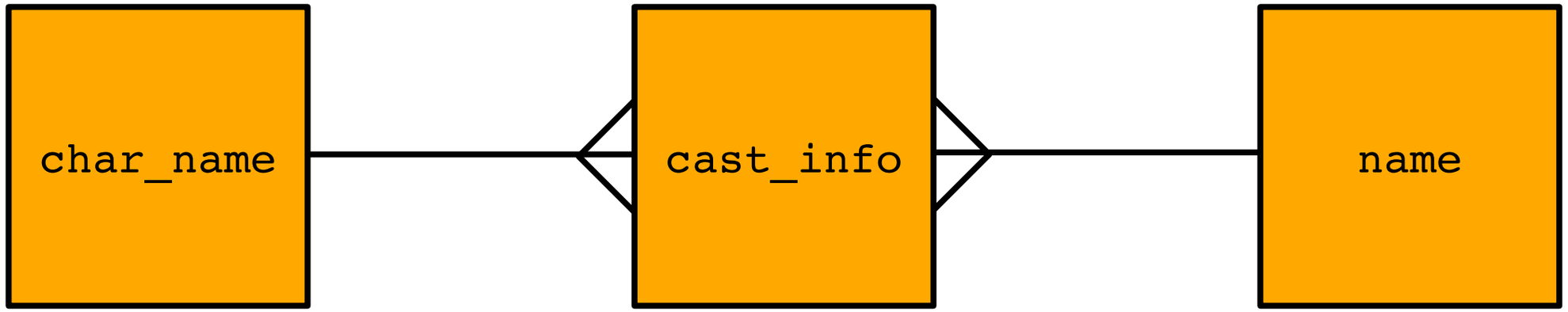
Complications

- Can’t achieve Third-Star indexes when:
  - Query requires more columns than the maximum of 16 columns per index.
  - Query requires `BLOB`, `TEXT`, or `VARCHAR` columns longer than the maximum of 1000 bytes per index (utf8 counts three bytes per character).

Example Query

Example Query

```
mysql> EXPLAIN SELECT STRAIGHT_JOIN n.*
  FROM char_name AS c
  INNER JOIN cast_info AS i
     ON c.id=i.person_role_id
  INNER JOIN name AS n
     ON i.person_id=n.id
  WHERE c.name = 'James Bond'
  ORDER BY n.name\G
```



Not Optimized

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: c
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2402968
        Extra: Using where; Using temporary; Using filesort
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: i
         type: index
possible_keys: person_id
          key: person_id
      key_len: 9
          ref: NULL
         rows: 21676661
        Extra: Using where; Using index; Using join buffer
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: n
        type: eq_ref
```

2.4 million × 21.7 million
= 52 trillion comparisons

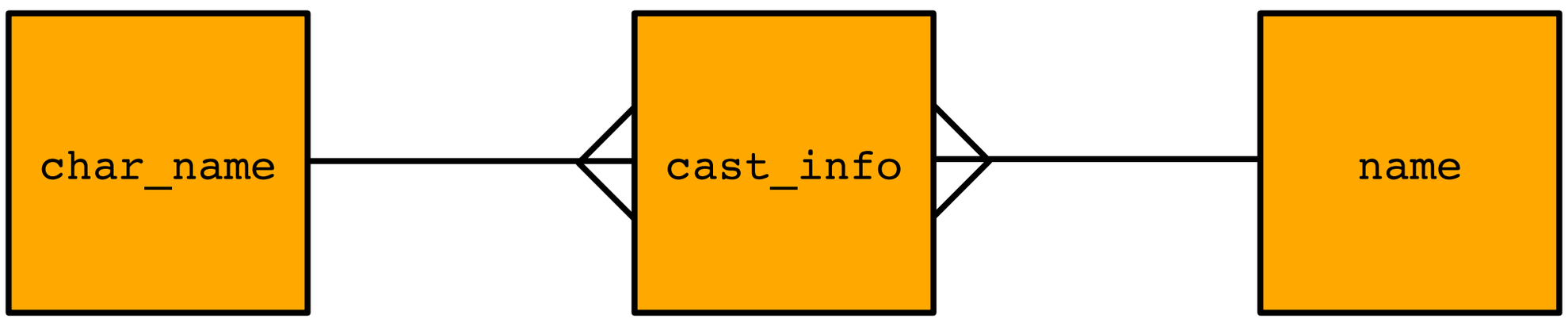
Add Indexes

★ First-star index:

```
mysql> ALTER TABLE char_name
  ADD INDEX n (name(20));
```

★ Third-star index:

```
mysql> ALTER TABLE cast_info
  ADD INDEX pr_p (person_role_id, person_id);
```



Optimized

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: c
         type: ref
possible_keys: PRIMARY,n
          key: n
      key_len: 62
          ref: const
         rows: 1
        Extra: Using where; Using temporary; Using filesort
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: i
         type: ref
possible_keys: pr_p
          key: pr_p
      key_len: 5
          ref: imdb.c.id
         rows: 108383
        Extra: Using where; Using index
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: n
         type: eq_ref
```

improved by a factor of 480 million!

Why Not Second Star?

- Not possible to create a second-star index here.
- The order of the join column isn't necessarily the same as the order of the sort column:

```
+----------+-------------------------+
| id       | name                    |
+----------+-------------------------+
|   127817 | Connery, Sean           |
|   201693 | Lazenby, George         |
|   213877 | Moore, Roger            |
|   228614 | Dalton, Timothy         |
|   581060 | Brosnan, Pierce         |
|   707448 | Craig, Daniel           |
+----------+-------------------------+
```

Lesson #3

tidy up indexes you don't need.

Indexes You Don't Need?

create indexes that help your queries

and

drop indexes that don't help your queries

Redundant Indexes

pt-duplicate-key-checker

http://www.percona.com/doc/percona-toolkit/2.1/pt-duplicate-key-checker.html

Redundant Indexes

```
mysql> create table foo (a int, b int,
  key (a), key (b), key (a,b));

$ pt-duplicate-key-checker h=localhost

# ############################################################
# test.foo
# ############################################################

# a is a left-prefix of a_2
# Key definitions:
#   KEY `a` (`a`),
#   KEY `a_2` (`a`,`b`)
# Column types:
#    `a` int(11) default null
#    `b` int(11) default null
# To remove this duplicate index, execute:
ALTER TABLE `test`.`foo` DROP INDEX `a`;
```

Unused Indexes

pt-index-usage

http://www.percona.com/doc/percona-toolkit/2.1/pt-index-usage.html

Unused Indexes

```
$ sudo pt-index-usage /var/lib/mysql/mysql-slow.log
/var/lib/mysql/mysql-slow.log:  12% 01:07 remain
/var/lib/mysql/mysql-slow.log:  25% 00:57 remain
/var/lib/mysql/mysql-slow.log:  37% 00:49 remain
/var/lib/mysql/mysql-slow.log:  49% 00:41 remain
/var/lib/mysql/mysql-slow.log:  61% 00:31 remain
/var/lib/mysql/mysql-slow.log:  73% 00:21 remain
/var/lib/mysql/mysql-slow.log:  84% 00:12 remain
/var/lib/mysql/mysql-slow.log:  97% 00:02 remain

. . .
ALTER TABLE `imdb`.`cast_info` DROP KEY `movie_id`;
  -- type:non-unique
. . .
```

Index Usage Statistics

Index Statistics

in Percona Server

http://www.percona.com/doc/percona-server/5.5/diagnostics/user_stats.html

Index Usage Statistics

- `INFORMATION_SCHEMA.INDEX_STATISTICS`
- Enable with:

```
mysql> SET GLOBAL userstat = ON;
```

- Statistics are cleared when you restart mysqld.
- Negligible performance impact:
  - http://www.mysqlperformanceblog.com/2012/06/02/how-expensive-is-user_statistics/

Index Usage Statistics

- Find indexes that were never used:

```
mysql> SELECT CONCAT(
       'ALTER TABLE `', s.table_schema, '`.`', s.table_name,
       ' DROP KEY `', s.index_name, '`;') AS ddl
  FROM INFORMATION_SCHEMA.STATISTICS AS s
  LEFT OUTER JOIN INFORMATION_SCHEMA.INDEX_STATISTICS AS i
       USING (table_schema, table_name, index_name)
  WHERE s.table_schema = 'imdb' AND i.rows_read IS NULL;

+-----------------------------------------------------------+
| ddl                                                       |
+-----------------------------------------------------------+
| ALTER TABLE `imdb`.`cast_info DROP KEY `movie_id`;        |
| ALTER TABLE `imdb`.`cast_info DROP KEY `person_id`;       |
| ALTER TABLE `imdb`.`comp_cast_type DROP KEY `kind`;       |
| ALTER TABLE `imdb`.`company_type DROP KEY `kind`;         |
. . .
```

Lesson #4

make it a habit.

Review Queries Regularly

- Repeat the query analysis periodically:
  - As new application code introduces new queries.
  - As the volume of data grows, making trivial queries more expensive.
  - As site traffic changes, making some queries more frequent.

Review Queries Regularly

- Track query types you have reviewed before:
  - `pt-query-digest --review`
    Saves query types to a table, so you can add notes to them, mark them as reviewed, omit them from future reports.
  - `pt-query-digest --review-history`
    Saves query performance statistics to a table, so you can analyze trends in the data over time.

Review Indexes Regularly

- Run `pt-duplicate-key-checker` periodically:
  - After schema changes.
  - You can run this on a development or test instance.

Review Indexes Regularly

- Run `pt-index-usage` or review `INDEX_STATISTICS` periodically:
  - On a regular schedule (e.g. weekly).
  - Especially after application code changes.

Lessons Redux

identify queries.

rate your indexes.

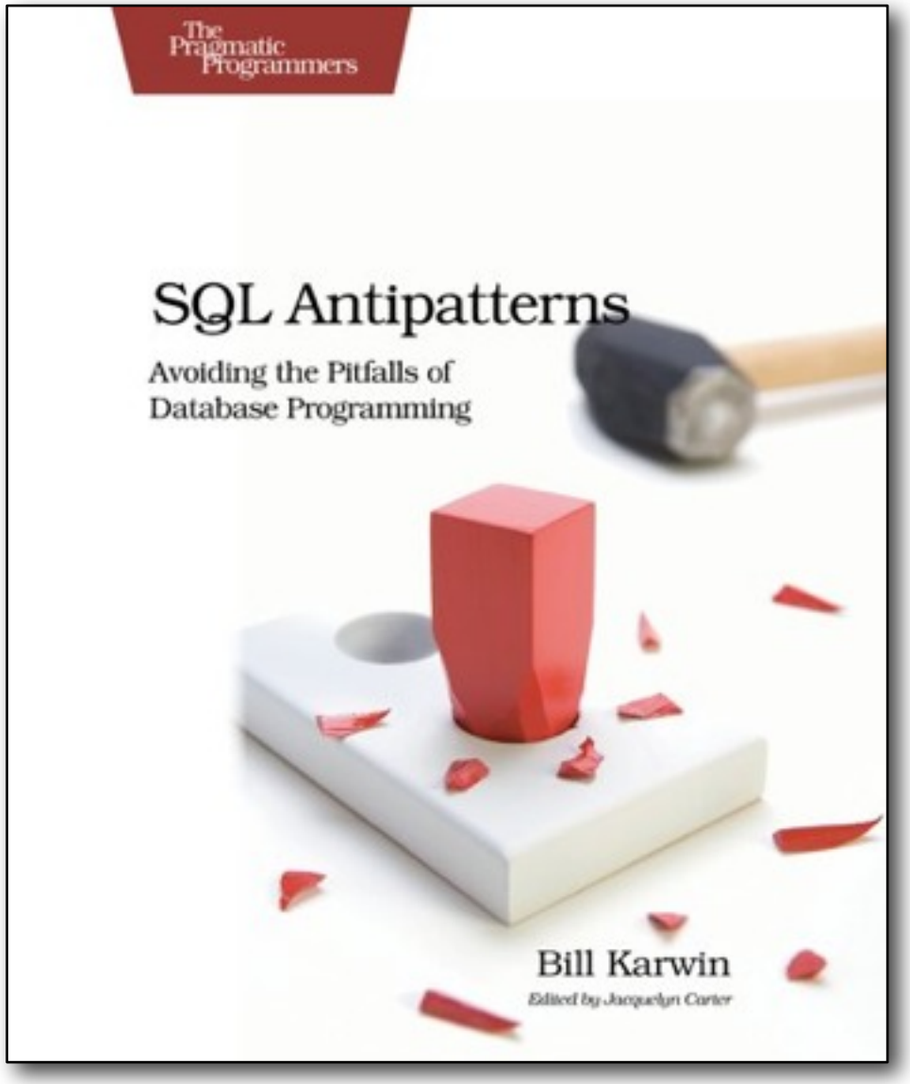
tidy up.

make it a habit.

London, December 3-4, 2012
Santa Clara, April 22-25, 2013

www.percona.com/live

SQL Antipatterns



http://www.pragprog.com/titles/bksqla/

License and Copyright