Indexes and DNA

Strategies to scale sites with lots of users and content
The anatomy of table rows
The anatomy of table columns
Defining an index

\{a, b, c\}

\{a, c\}
The anatomy of indexes

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
<th>evil</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Maria</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Mark</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

CREATE TABLE IF NOT EXISTS `people` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(32) NOT NULL,
  `age` smallint(5) unsigned NOT NULL,
  `evil` tinyint(1) unsigned NOT NULL,
  PRIMARY KEY (`id`),
  KEY `name` (`name`, `age`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=4 ;

INSERT INTO `people` (`id`, `name`, `age`, `evil`) VALUES
(1, 'David', 23, 0),
(2, 'Maria', 25, 0),
(3, 'Mark', 19, 0);
What does this index do?

- Allows the following queries to run efficiently
  - SELECT age FROM {people}
  - SELECT * FROM {people} WHERE name LIKE "Mar%"
  - SELECT * FROM {people} ORDER BY name, age
  - SELECT * FROM {people} WHERE name = "Maria" ORDER BY age
How MySQL uses the indexes

mysql> EXPLAIN SELECT * FROM people;
+----+-------------+--------+------+---------------+------+---------+------+------+-------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra |
|----+-------------+--------+------+---------------+------+---------+------+------+-------+
|  1 | SIMPLE      | people | ALL  | NULL          | NULL | NULL    | NULL |    3 |       |
+----+-------------+--------+------+---------------+------+---------+------+------+-------+
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT age FROM people;
+----+-------------+--------+-------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type  | possible_keys | key  | key_len | ref  | rows | Extra       |
|----+-------------+--------+-------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | people | index | NULL          | name | 100     | NULL |    3 | Using index |
+----+-------------+--------+-------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM people WHERE name LIKE "Mar%";
+----+-------------+--------+-------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type  | possible_keys | key  | key_len | ref  | rows | Extra       |
|----+-------------+--------+-------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | people | range | name          | name | 98      | NULL |    1 | Using where |
+----+-------------+--------+-------+---------------+------+---------+------+------+-------------+
1 row in set (0.01 sec)
How MySQL uses the indexes

```
mysql> EXPLAIN SELECT * FROM people ORDER BY name, age;
+----+-------------+--------+-------+---------------+------+---------+------+------+-------+
| id | select_type | table  | type  | possible_keys | key  | key_len | ref  | rows | Extra |
+----+-------------+--------+-------+---------------+------+---------+------+------+-------+
|  1 | SIMPLE      | people | index | NULL          | name | 100     | NULL |    3 |       |
+----+-------------+--------+-------+---------------+------+---------+------+------+-------+
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN SELECT * FROM people ORDER BY name, age;
+----+-------------+--------+-------+---------------+------+---------+------+------+-------+
| id | select_type | table  | type  | possible_keys | key  | key_len | ref  | rows | Extra |
+----+-------------+--------+-------+---------------+------+---------+------+------+-------+
|  1 | SIMPLE      | people | index | NULL          | name | 100     | NULL |    3 |       |
+----+-------------+--------+-------+---------------+------+---------+------+------+-------+
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN SELECT * FROM people WHERE age = 23;
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | people | ALL  | NULL          | NULL | NULL    | NULL |    3 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```
Indexing limitations

\{a, b, c\} \rightarrow A \rightarrow \{a, c\}

\{d, e, f\} \rightarrow E \rightarrow \{e, f\}

Not possible w/o DNA
Pulling together data

```
<table>
<thead>
<tr>
<th>node</th>
<th>node_comment_statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>nid</td>
<td>nid</td>
</tr>
<tr>
<td>title</td>
<td>title</td>
</tr>
<tr>
<td>status</td>
<td>status</td>
</tr>
<tr>
<td>changed</td>
<td>changed</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>last_comment_timestamp</td>
</tr>
</tbody>
</table>
```

“I want all published nodes, ordered by the later of the node’s changed timestamp and the last comment timestamp.”
This is amazingly inefficient.
How MySQL handles the query

1. Looks at indexes
2. Realizes criteria span two tables
3. Picks a table to start from
4. Applies criteria to that table using the index
5. Builds a temporary table
6. Applies remaining criteria to the temporary table
7. Sorts the temporary table
8. Returns results
Enter DNA

Using the Denormalization API to create great indexes
What is DNA?

DNA (The Denormalization API) is a module to consolidate information related to nodes from multiple tables into a single table.
Consolidating node data

- **node**
  - nid
  - title
  - status
  - changed

- **node_comment_statistics**
  - nid
  - last_comment_timestamp

- **latest_changed_nodes_dna**
  - nid
  - last_comment_or_change

Implicit

Explicit
Indexing the consolidated table

\[
\text{latest\_changed\_nodes\_dna}
\]

\[
\text{nid} \quad \text{last\_comment\_or\_change}
\]

\{\text{last\_comment\_or\_change, nid}\}
How MySQL handles this query

1. Looks at indexes
2. Applies criteria to the table using the index
3. Returns results