# Are You Getting the Best Out of Your MySQL Indexes?

Sheeri Cabral
Senior Database Administrator

**sheeri.cabral@salesforce.com**
**awfief@gmail.com**
**@sheeri**

# Why is the optimizer not doing what I expect?

Lots of MySQL “quirks” around indexing

# KEY vs. INDEX

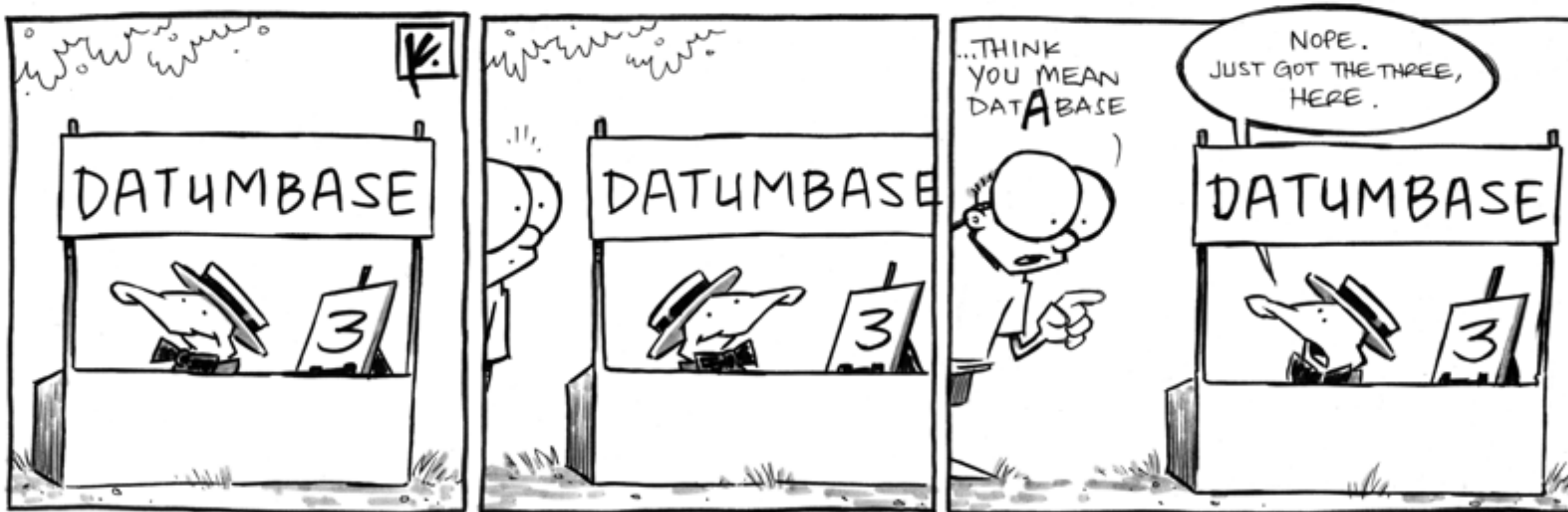
KEY = key constraint

- PRIMARY
- UNIQUE
- FOREIGN

Other indexes are “an implementation detail”

What’s the plural of index?

http://www.sheldoncomics.com/strips/sd120626.png



# More Jargon

Simple

(last_name)

Composite/Compound

(last_name, first_name)

Prefix indexes – an index on (last_name, first_name) is really:

(last_name, first_name)

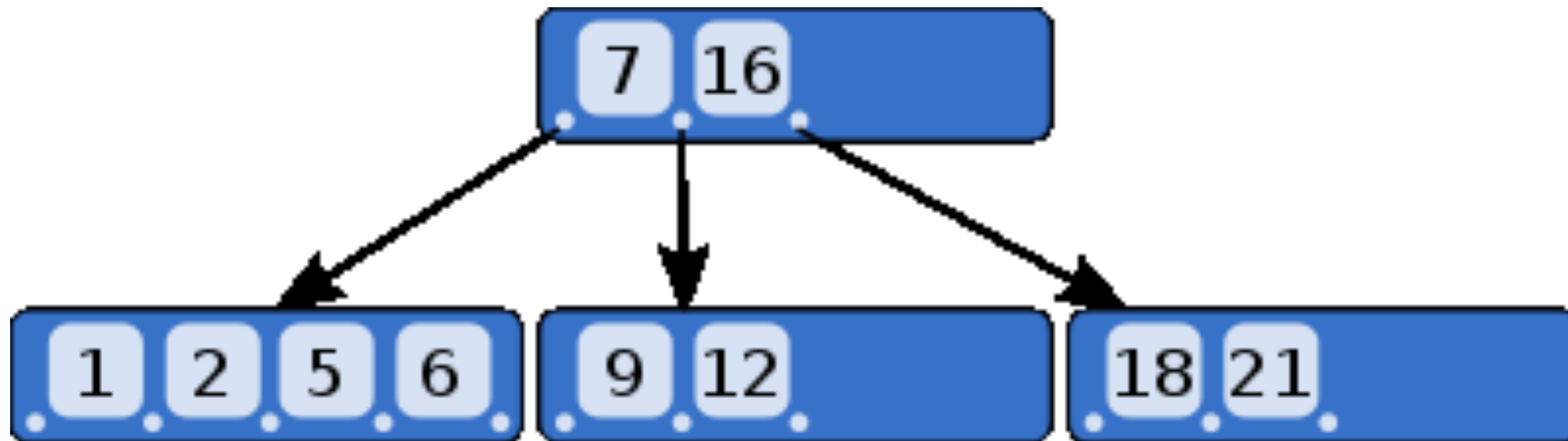
(last_name)

# Data Structures

B-TREE for InnoDB data and indexes (usually)

HASH for MEMORY tables

# B-tree




(Image from Wikipedia)

# B-trees are Excellent for.....

One equality match

- WHERE foo=11

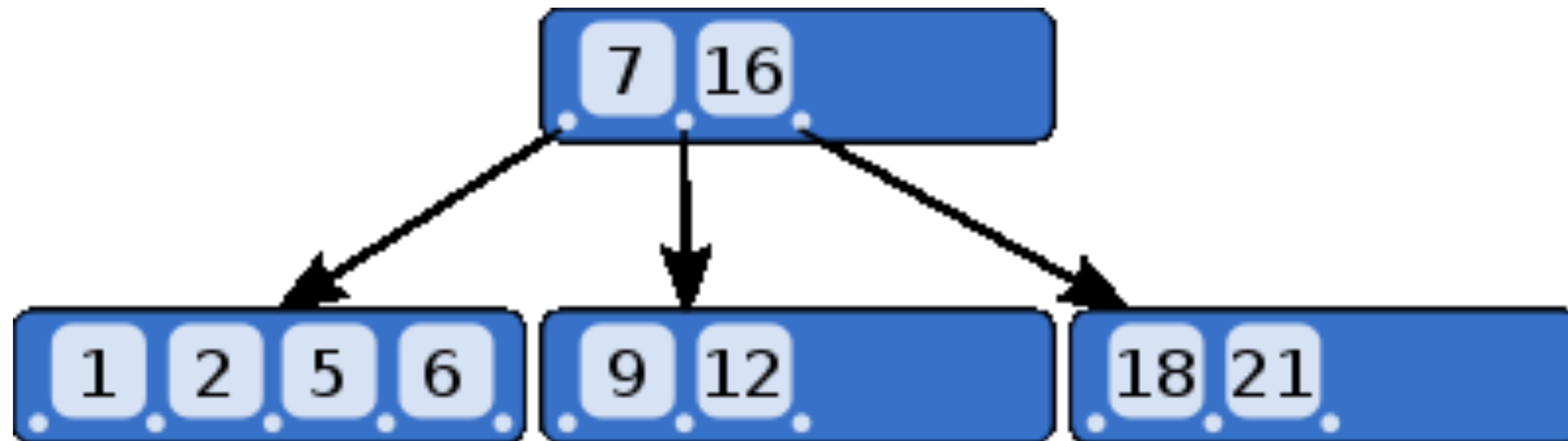
A few equality matches

- WHERE foo IN (5,10,11)

Range searches

WHERE foo BETWEEN 5 and 16

How is it done?



(Image from Wikipedia)

# Composite Indexes

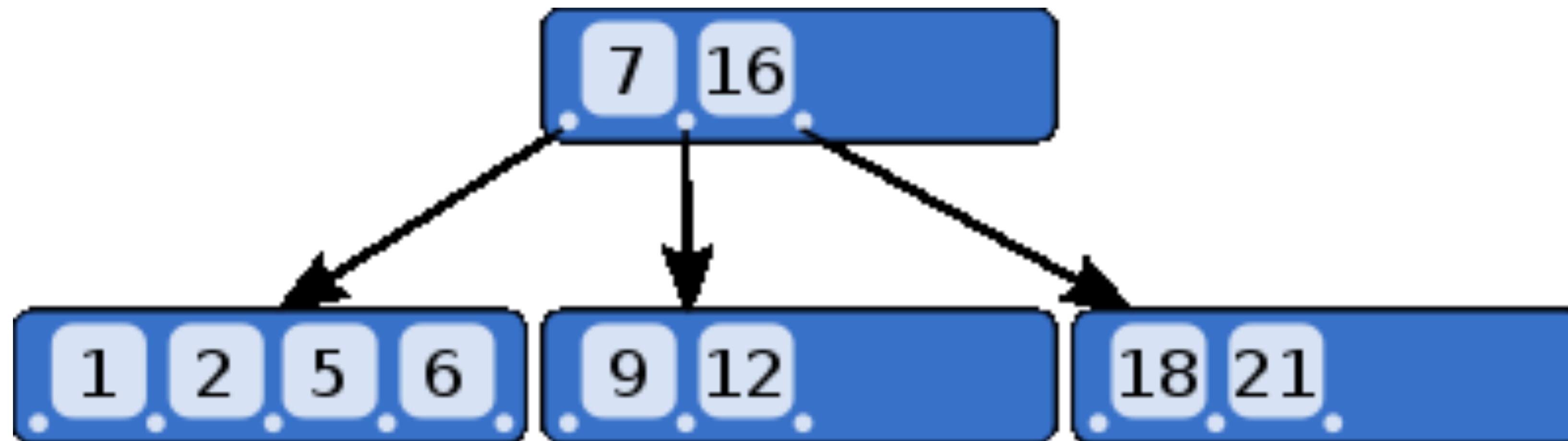
Index on (last_name, first_name)

- Like a dictionary index
- Used to find last  names beginning with “g”
- Not used to find first names beginning with “g”

Perfectly reasonable to have 2 indexes:

(last_name, first_name)

(first_name)




(Image from Wikipedia)

# Composite Indexes

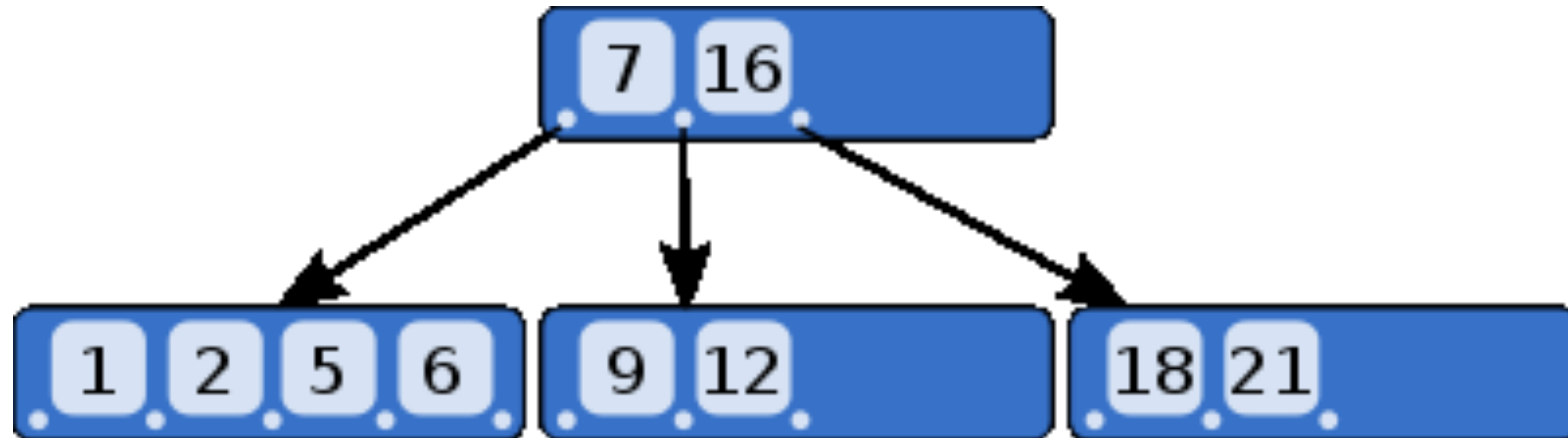
Index on (last_name,first_name,middle_name)

Functions as:

(last_name,first_name,middle_name)

(last_name,first_name)

(last_name)




(Image from Wikipedia)

# MySQL Uses Indexes For....

Matching a WHERE clause

Eliminating rows (filtering)

- Perhaps for a JOIN

Resolving MIN() or MAX() values

Eliminate the need for sorting

Help with grouping

Everything – covering index

# MySQL Ignores Indexes For....

Functions

- date(timstamp_col)='2017_05_07'

Queries and JOINs if the fields are not the same

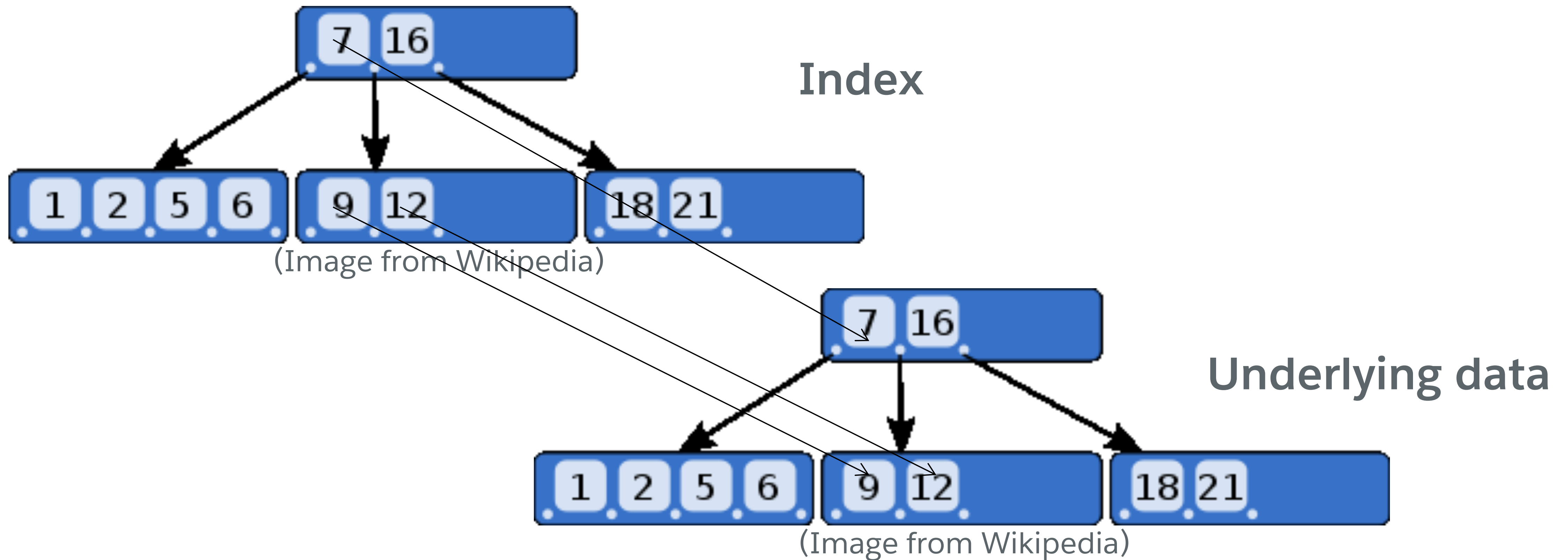
- date_col=ts_col
- date_col='2017_05_07 00:00:00'

# MySQL Ignores Indexes For....

"Too much data"

SELECT * FROM tbl WHERE foo in (7,9,12)

- 6 lookups vs. walking the 10-node tree.



(Image from Wikipedia)

(Image from Wikipedia)

# “Too Much” Data

How much is “too much”?

- About 15-25%

Can you be more specific?

- Not really, here’s how it’s calculated……

# Metadata!

Approximate in InnoDB

- ANALYZE TABLE does a few dives into the data
  - Default is 8 pages
  - Pages are 16K by default
  - 128K samples
- Configurable by innodb_stats_sample_pages (5.5), innodb_stats_transient_sample_pages (5.7)
- Fast (locking), but not accurate
- OPTIMIZE TABLE slow, locking, but accurate

Used for rows read and join calculations

Cardinality/Selectivity

Average “value group” size

# Cardinality

To choose an index, the MySQL optimizer uses metadata

- Cardinality (# elements in the set). Lower = more duplicates.
  - A yes/no field has a cardinality of 2
  - A unique field that has 100 rows populated has a cardinality of 100.
- Total # rows
  - use an index, or full table scan?

For a composite index, pick the order wisely

- Better to eliminate as much as possible as early as possible
- Higher cardinality items first are great

# Value Group

Consider a value group: Body parts

10 fingers, 2 eyes = 12 body parts

Average “value group” size is 6

- Not accurate for eyes nor fingers
- “Looking for eyes isn’t ‘too much data, there’s only 2 values in that group (left eye, right eye)’
- But optimizer uses 6, not 2
    - May tip the query into “too much data” territory

# Composite Indexes and sorting

E.g. (first_name, last_name)

Can be sorted ASC or DESC

- Actually....DESC is ignored until MySQL 8.0
- Slower to traverse the tree backwards

But not (ASC, DESC) or (DESC, ASC)

- Until MySQL 8.0

So a query may use an index but still need an extra pass for sorting/ordering

# Prefix Indexing

For VARCHAR/CHAR/TEXT, but not FULLTEXT

767 bytes on InnoDB by default

- Beware of charset!
- Up to 3072 bytes (3 Kb) with innodb_large_prefix
  - Innodb_file_per_table=TRUE
  - Innodb_file_format=barracuda
- HOWEVER – make your index prefix as short as possible
- Optimizer won’t use an index on the_text(5) for a query WHERE the_text LIKE ’abc%’
  - https://www.percona.com/blog/2017/04/11/correct-index-choices-for-equality-like-query-optimization/

# GROUP BY and Sorting

GROUP BY also sorts by default

This may add an extra data pass
- e.g. “filesort” in EXPLAIN output

To avoid this, ORDER BY NULL

Ideal: ALWAYS use EXPLICIT ORDER BY for each GROUP BY
- Self-documenting query

Bug 30477 (Aug 2007) https://bugs.mysql.com/bug.php?id=30477
- "Relying on implicit GROUP BY sorting in MySQL 5.6 is deprecated. To achieve a specific sort order of grouped results, it is preferable to use an explicit ORDER BY clause. GROUP BY sorting is a MySQL extension that may change in a future release; for example, to make it possible for the optimizer to order groupings in whatever manner it deems most efficient and to avoid the sorting overhead."

# NULL and Equality

## Which statements return true?

IF (NULL=NULL)

- Not true

IF (x=NULL)

- Not true

IF (x<=>NULL)

- May be true if x is NULL

If the optimizer sees IF (field=NULL), MySQL immediately returns false

- It knows that value will NEVER be true

Use IS NULL or IS NOT NULL or <=>

# NULL-Safe Equality

## Which statements return true?

IF (x<=>NULL)

- May be true if x is NULL

No NULL-safe inequality operator

- Use IF !(x<=>NULL)

# NULL and Value Groups

Let’s say “body parts” is nullable

We have 2 eyes, 10 fingers, and 3 NULL values

How many value groups? What’s the average value group size?

# NULL and Value Groups

## Configurable, if need be...

innodb_stats_method

- **nulls_equal (default)**
- Single value group for all NULLs
- 15 values, 3 groups = average value group size of 5
  - Still not accurate for fingers or eyes!

- nulls_unequal
- Every NULL is a different value group
- 15 values, 5 groups = average value group size of 3
  - Better for eyes, but a coincidence

- nulls_ignored
- No value group for NULLs
- 12 values, 2 groups = average value group size of 6

# How keys and indexes work

# PRIMARY and UNIQUE keys

## PRIMARY key

- Row identifier
- Can be simple or composite
- Cannot be NULL
- Disk order = PRIMARY key order

## UNIQUE key

- Row identifier
- Can be simple or composite
- Can be NULL

# FOREIGN key constraints

Parent/child relationship

- E.g. customer is a parent of a row in payments
- FOREIGN KEY (parents.customer_id) REFERENCES (customer.id)

Cascading updates/deletes

# Surrogate vs. Meaningful FOREIGN key constraints

Usually in a “lookup table” scenario

- e.g. when a customer table refers to a status table
- Compare:

| customer_id | status_id |
|---|---|
| 121 | 1 |
| 122 | 2 |
| 125 | 1 |

| status_id | status |
|---|---|
| 1 | free |
| 2 | paid |
| 3 | disabled |

- With:

| customer_id | status |
|---|---|
| 121 | free |
| 122 | paid |
| 125 | free |

| status |
|---|
| free |
| paid |
| disabled |

Tradeoff:
Longer to lookup/write

Faster to read - no join needed to lookup value

# FULLTEXT indexes

CHAR/VARCHAR/TEXT types

MATCH (text1, text2, ...) AGAINST ('textToMatch')

- In WHERE clause
- In SELECT clause – to see relevance
- Can do both with no penalty

Case sensitivity depends on collation

Accent mark respect depends on collation

# FULLTEXT indexes

Not on partitioned tables

In a composite FULLTEXT index, all fields must have the same charset and collation

Cannot use % or _ as a wildcard

AGAINST can only take expressions that are constant

- E.g., not a table field, cannot do MATCH (body) AGAINST (author)

# FULLTEXT indexes Silent Failures

## Innodb_ft_min_token_size

- Default: 3
- Possible: 0-16

## Innodb_ft_max_token_size

- Default: 84
- Possible: 10-84

## ngram_token_size

- Languages with no word boundaries, like Chinese
- Default: 2
- Possible: 1-10

# FULLTEXT Indexes Silent Failures

Stopword list

- Any word that appears in >50% of rows
- Changeable
- Static, or table
- innodb_ft_server_stopword_table
- Need to rebuild the FULLTEXT index when this changes
  - ALTER TABLE DROP INDEX...., ADD INDEX
  - Or OPTIMIZE TABLE (but that rebuilds all indexes)

# FULLTEXT indexes IN NATURAL LANGUAGE MODE

“test” will also match “tests” and “testing”

Uses stopword list

Ordered by relevance automatically

SELECT id, MATCH (textField) AGAINST (‘textToMatch’)

Can use quotes for exact phrase (minus word breaks)

All fields in the index must be used

FULLTEXT index on (title, body) does NOT give you (title) NOR (body)

# FULLTEXT indexes IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION

Automatic relevance feedback (aka blind query expansion)

Usually for short phrases, e.g. “database”

Will also match words that appear in docs with “database”

- e.g. “mysql”, “oracle”, “open source”, “tutorial”
- WITH QUERY EXPANSION will find docs with the related words but NOT “database”

Performs the search twice

# FULLTEXT indexes IN BOOLEAN MODE

+ requires presence to match (AND)

- requires absence to match (NOT)

~ adds negative relevance

< and > changes priority

OR is implied by no operator

* is a wildcard

Examples:

- '+apple +(>turnover <strudel)'
  - Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.
- 'apple*'
  - Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

# FULLTEXT indexes IN BOOLEAN MODE

Does NOT follow 50% stopword rule

Still uses stopword list

DOES NOT order by relevance automatically

- Manual has relevance calculation
  - https://dev.mysql.com/doc/refman/5.7/en/fulltext-boolean.html

@NUMBER– word distance between all search terms

- MATCH(col1) AGAINST('"word1 word2 word3" @8' IN BOOLEAN MODE)

Subexpressions with ()

# SPATIAL indexes
# for geometric data
# (out of scope)

# Knowing All This

Use EXPLAIN EXTENDED=JSON

- EXPLAIN talk soon

If you are not getting the results you expect......

Check your expectations and MySQL's quirks

Or....file a bug at [https://bugs.mysql.com](https://bugs.mysql.com)

Questions?
Feedback?
sheeri.com
@sheeri