INEA00104L: Report #1

Due on Monday, May 5, 2014

Mgr inż. Andrzej Wytyczak-Partyka

inż. Piotr Giedziun

Contents

Introduction

This report was made to present results of work made during third laboratory. Main goal of this lab was to introduce basic optimization topics. Performed tasks focused on indexes usage. Without an index, would begin with the first row and then read through the entire table to find the relevant rows. Indexes allow to improve performance on aspects like WHERE clause, finding MIN/MAX, sorting etc.

Database schema used in this lab was based on previous labs. We were asked to create database for cinema. This topic includes usage of triggers, views and basics of relational databases.

Database

In order to perform given task, we had to add additional field - **theater_id** (type of INT). Figure 1 shows changes made to **ticket** table. Filed **theater_id** is just normal field (it's not liked as a foreign key).

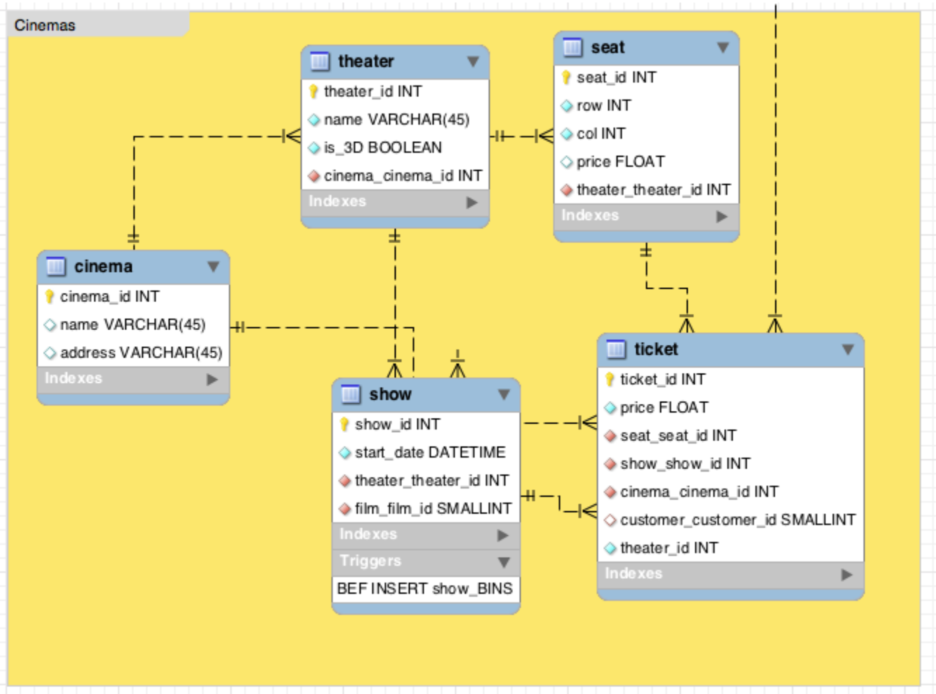


Figure 1: Part of database EER Diagram

Task 1

First task was to create SELECT query with WHERE clause on **ticket.theater_id** field. Another part of the task was to study performance of query WITH/WITHOUT index on **ticket.theater_id**.

Following query adds INDEX on theater_id.

```
CREATE INDEX theater_id_index ON `ticket` (theater_id) USING BTREE;
```

Following query was used to measure performance time.

```
SELECT SQL_NO_CACHE * FROM `ticket` WHERE theater_id = 3;
```

Task 1 continued on next page. . .

Query WITHOUT index on theater_id

First step I did was to use EXPLAIN statement. The EXPLAIN statement provides information about how MySQL executes statements.

```
EXPLAIN SELECT * FROM `ticket` WHERE theater_id = 3;

id,select_type,table,type,possible_keys,key,key_len,ref,rows,Extra
1,SIMPLE,ticket,ALL,NULL,NULL,NULL,NULL,99814,"Using where"
```

In order to obtain results under hood MySQL is looking through all rows. Field ”rows” is equals to number of rows in ticket table. This approach is not optimal, it will be dependent on rows count.

Query WITH index on theater_id

```
EXPLAIN SELECT * FROM `ticket` WHERE theater_id = 3;

id,select_type,table,type,possible_keys,key,key_len,ref,rows,Extra
1,SIMPLE,ticket,ref,theater_id,theater_id,4,const,100,NULL
```

In this case MySQL is looking through only 100 rows. This result was achieved thanks to B-tree construction of index table.

Results

As expected there is a huge spike between execution with and without index on theater_id. For smaller instances (like 1k) execution time was similar. It’s wasn’t the case for 10k+ rows.

Results presented in table.

	1k	10k	100k	1000k
with	0.07s	0.064s	0.066s	0.073s
without	0.09s	0.433s	3.6s	37.02s

Results presented as line chart.

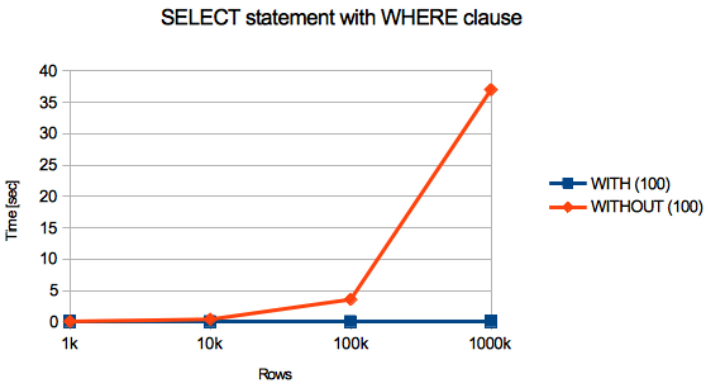


Figure 2: SELECT statement with WHERE clause

Task 2

Second task was also to determine difference between using/not using index on start_date field. This time on DATE field. Date fields behave similar to integer fields, so expected graph should be similar to presented in first task.

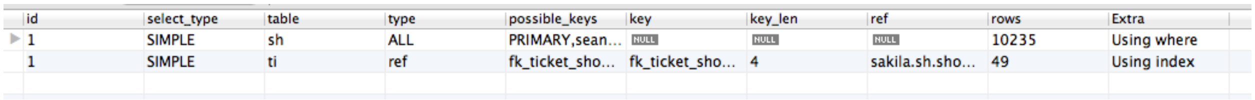
Following query was used to measure performance time.

```
SELECT
    ti . *, sh.start_date
FROM
    `ticket` ti
        INNER JOIN
    `show` sh ON ti.show_show_id = sh.show_id
WHERE
    start_date > now()
        AND start_date < DATE_ADD(now(), INTERVAL 1 YEAR);
```

Query WITHOUT index on start_date

First step I did was to use EXPLAIN statement. The EXPLAIN statement provides information about how MySQL executes statements.

```
EXPLAIN SELECT COUNT(*) FROM `ticket` ti INNER JOIN `show` sh ON
 ti.show_show_id = sh.show_id WHERE start_date > now()
 AND start_date < DATE_ADD(now(), INTERVAL 1 YEAR);
```

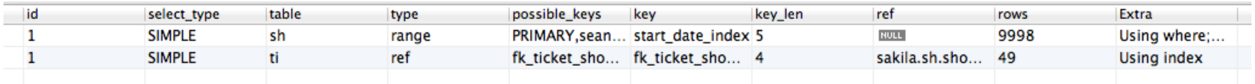


id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	sh	ALL	PRIMARY,sean...	NULL	NULL	NULL	10235	Using where
1	SIMPLE	ti	ref	fk_ticket_sho...	fk_ticket_sho...	4	sakila.sh.sho...	49	Using index

There is nothing more to say here. It's the same as in previous task.

Query WITH index on start_date

```
EXPLAIN SELECT COUNT(*) FROM `ticket` ti INNER JOIN `show` sh ON
 ti.show_show_id = sh.show_id WHERE start_date > now()
 AND start_date < DATE_ADD(now(), INTERVAL 1 YEAR);
```



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	sh	range	PRIMARY,sean...	start_date_index	5	NULL	9998	Using where;...
1	SIMPLE	ti	ref	fk_ticket_sho...	fk_ticket_sho...	4	sakila.sh.sho...	49	Using index

Size of checked records is significantly bigger in comparison to task 1. It's because almost all shows were played in 1 year gap.

Results

This time around I have received almost identical results for both cases. It's because almost all shows were played in 1 year gap.

Results presented in table.

	1k	10k	100k	1000k
with	0.001s	0.003s	0.025s	0.287s
without	0.001s	0.003s	0.026s	0.291s

Task 2 continued on next page. . .

Results presented as line chart.

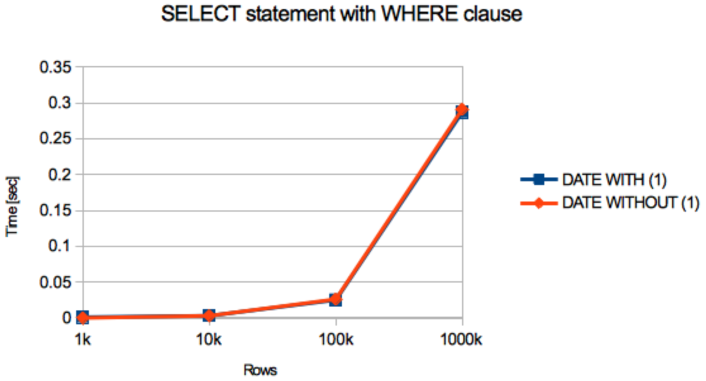


Figure 3: SELECT statement with WHERE clause

Task 3

Third task was also to determine difference between using/not using index on title field. This time on VARCHAR field. In addition this task was divided into two smaller parts (starting with ”the” and having the word ”the”)

Following query was used to measure performance time for starting with ”the”.

```
SELECT SQL_NO_CACHE
    fi.title
FROM
    `ticket` ti
        INNER JOIN
    `show` sh ON sh.show_id = ti.show_show_id
        LEFT JOIN
    `film` fi ON fi.film_id = sh.film_film_id
WHERE
    fi.title LIKE 'the%'
```

Following query was used to measure performance time for having the word ”the”.

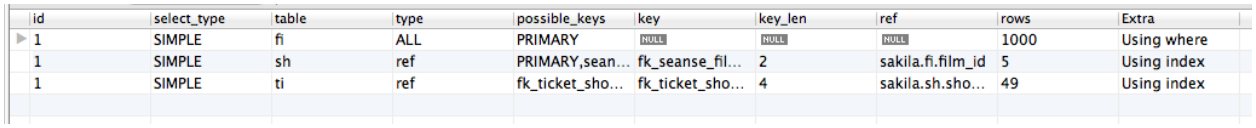
```
SELECT SQL_NO_CACHE
    fi.title
FROM
    `ticket` ti
        INNER JOIN
    `show` sh ON sh.show_id = ti.show_show_id
        LEFT JOIN
    `film` fi ON fi.film_id = sh.film_film_id
WHERE
    fi.title LIKE '%the%'
```

Task 3 continued on next page. . .

Query WITHOUT index on title

First step I did was to use EXPLAIN statement. The EXPLAIN statement provides information about how MySQL executes statements.

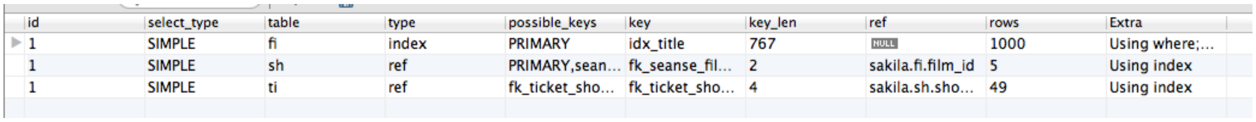
```
EXPLAIN SELECT SQL_NO_CACHE fi.title FROM `ticket` ti INNER JOIN `show` sh ON
sh.show_id = ti.show_show_id LEFT JOIN `film` fi ON fi.film_id = sh.film_film_id
WHERE fi.title LIKE '%the%'
```



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	fi	ALL	PRIMARY	NULL	NULL	NULL	1000	Using where
1	SIMPLE	sh	ref	PRIMARY,sean…	fk_seanse_fil…	2	sakila.fi.film_id	5	Using index
1	SIMPLE	ti	ref	fk_ticket_sho…	fk_ticket_sho…	4	sakila.sh.sho…	49	Using index

Query WITH index on title

```
EXPLAIN SELECT SQL_NO_CACHE fi.title FROM `ticket` ti INNER JOIN `show` sh ON
sh.show_id = ti.show_show_id LEFT JOIN `film` fi ON fi.film_id = sh.film_film_id
WHERE fi.title LIKE '%the%'
```



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	fi	index	PRIMARY	idx_title	767	NULL	1000	Using where;…
1	SIMPLE	sh	ref	PRIMARY,sean…	fk_seanse_fil…	2	sakila.fi.film_id	5	Using index
1	SIMPLE	ti	ref	fk_ticket_sho…	fk_ticket_sho…	4	sakila.sh.sho…	49	Using index

Results

In my case film database in negligible compared to ticket size. This fact have big impact on result graph. There is no major difference for table with 100 rows with/without indexes.
In this case results were impacted by ticket table size (amount of inner joins).

Results presented as line chart.

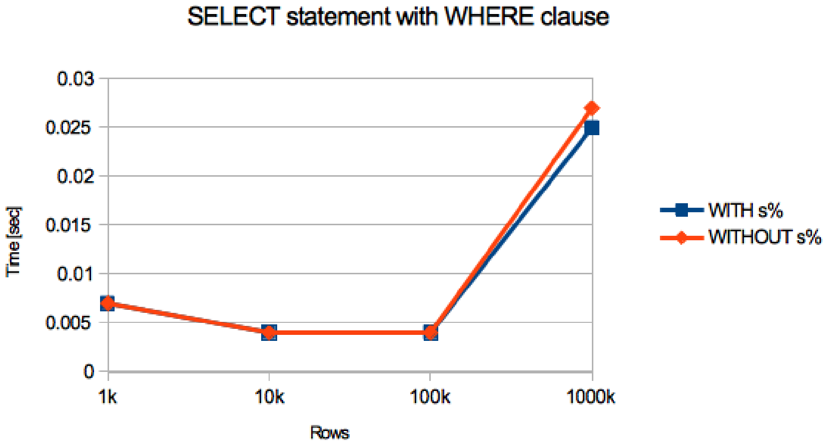


Figure 4: SELECT statement with WHERE clause

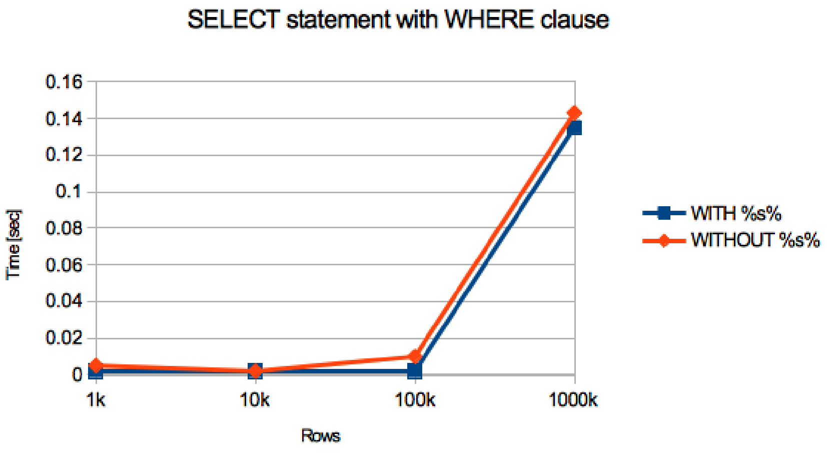
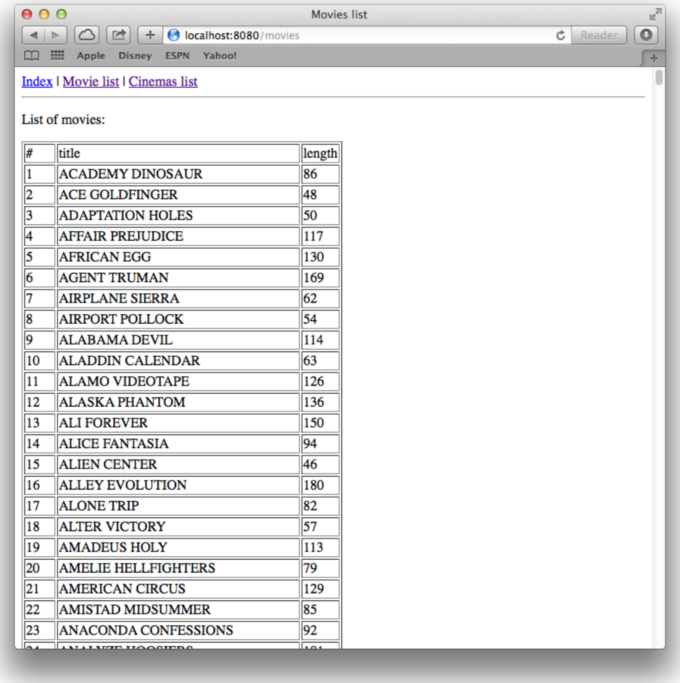


Figure 5: SELECT statement with WHERE clause

Webpage

Part of laboratory preparation was to create website to handle requests. Prepared site is able to display movies list, cinemas list, view upcoming show and book seats.

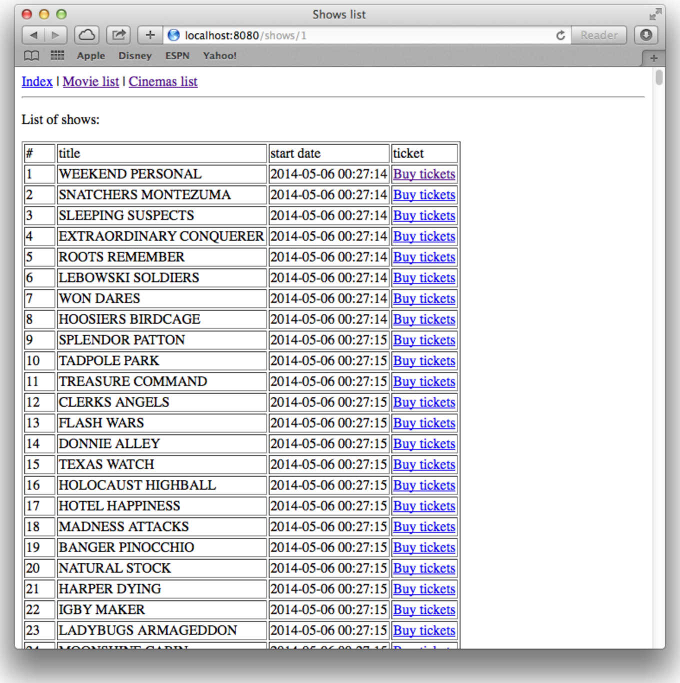


Index | Movie list | Cinemas list

List of movies:

#	title	length
1	ACADEMY DINOSAUR	86
2	ACE GOLDFINGER	48
3	ADAPTATION HOLES	50
4	AFFAIR PREJUDICE	117
5	AFRICAN EGG	130
6	AGENT TRUMAN	169
7	AIRPLANE SIERRA	62
8	AIRPORT POLLOCK	54
9	ALABAMA DEVIL	114
10	ALADDIN CALENDAR	63
11	ALAMO VIDEOTAPE	126
12	ALASKA PHANTOM	136
13	ALI FOREVER	150
14	ALICE FANTASIA	94
15	ALIEN CENTER	46
16	ALLEY EVOLUTION	180
17	ALONE TRIP	82
18	ALTER VICTORY	57
19	AMADEUS HOLY	113
20	AMELIE HELLFIGHTERS	79
21	AMERICAN CIRCUS	129
22	AMISTAD MIDSUMMER	85
23	ANACONDA CONFESSIONS	92

Figure 6: Movies list

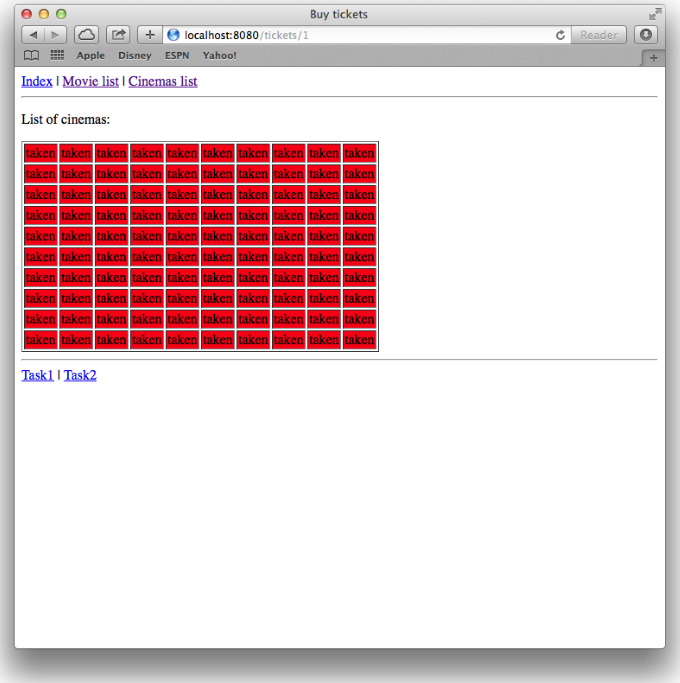


Index | Movie list | Cinemas list

List of shows:

#	title	start date	ticket
1	WEEKEND PERSONAL	2014-05-06 00:27:14	Buy tickets
2	SNATCHERS MONTEZUMA	2014-05-06 00:27:14	Buy tickets
3	SLEEPING SUSPECTS	2014-05-06 00:27:14	Buy tickets
4	EXTRAORDINARY CONQUERER	2014-05-06 00:27:14	Buy tickets
5	ROOTS REMEMBER	2014-05-06 00:27:14	Buy tickets
6	LEBOWSKI SOLDIERS	2014-05-06 00:27:14	Buy tickets
7	WON DARES	2014-05-06 00:27:14	Buy tickets
8	HOOSIERS BIRDCAGE	2014-05-06 00:27:14	Buy tickets
9	SPLENDOR PATTON	2014-05-06 00:27:15	Buy tickets
10	TADPOLE PARK	2014-05-06 00:27:15	Buy tickets
11	TREASURE COMMAND	2014-05-06 00:27:15	Buy tickets
12	CLERKS ANGELS	2014-05-06 00:27:15	Buy tickets
13	FLASH WARS	2014-05-06 00:27:15	Buy tickets
14	DONNIE ALLEY	2014-05-06 00:27:15	Buy tickets
15	TEXAS WATCH	2014-05-06 00:27:15	Buy tickets
16	HOLOCAUST HIGHBALL	2014-05-06 00:27:15	Buy tickets
17	HOTEL HAPPINESS	2014-05-06 00:27:15	Buy tickets
18	MADNESS ATTACKS	2014-05-06 00:27:15	Buy tickets
19	BANGER PINOCCHIO	2014-05-06 00:27:15	Buy tickets
20	NATURAL STOCK	2014-05-06 00:27:15	Buy tickets
21	HARPER DYING	2014-05-06 00:27:15	Buy tickets
22	IGBY MAKER	2014-05-06 00:27:15	Buy tickets
23	LADYBUGS ARMAGEDDON	2014-05-06 00:27:15	Buy tickets
24	MOONSHINE CABIN	2014-05-06 00:27:15	Buy tickets

Figure 7: Shows list



Index | Movie list | Cinemas list

List of cinemas:

taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken
taken	taken	taken	taken	taken	taken	taken	taken	taken	taken

Task1 | Task2

Figure 8: Seats for selected show

Summary

- All indexes in our tasks (data types - INT, VARCHAR, DATA) were stored in B-trees.
- For those task, where number of searches records were about 10k+ there were huge performance improvement for those with index.
- It's always good practice to use indexes while creating database model.
- Indexes should be created on fields used in WHERE clause, finding MIN/MAX and sorting.