

# Оптимизация запросов

# Индексы

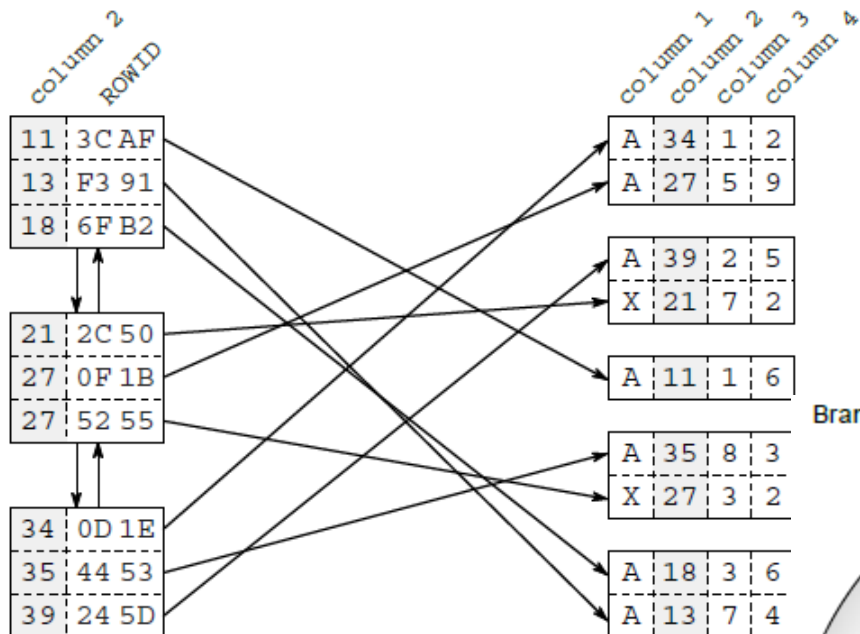
- Количество элементов в индексе в зависимости от глубины дерева при списке в одном узле 4

Глубина	Кол-во элементов
3	64
4	256
5	1,024
6	4,096
7	16,384
8	65,536
9	262,144
10	1,048,576

# Индексы

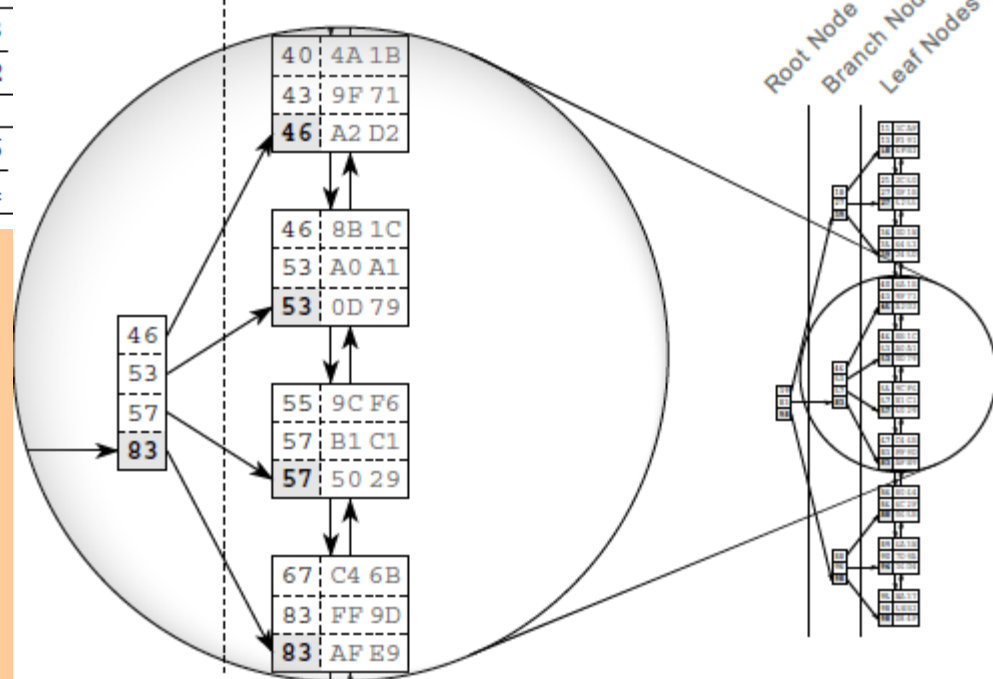
Index Leaf Nodes  
(sorted)

Table  
(not sorted)



Branch Node

Leaf Nodes



Root Node  
Branch Nodes  
Leaf Nodes

# Типы операций поиска в индексе

- INDEX UNIQUE SCAN

Только проход по дереву

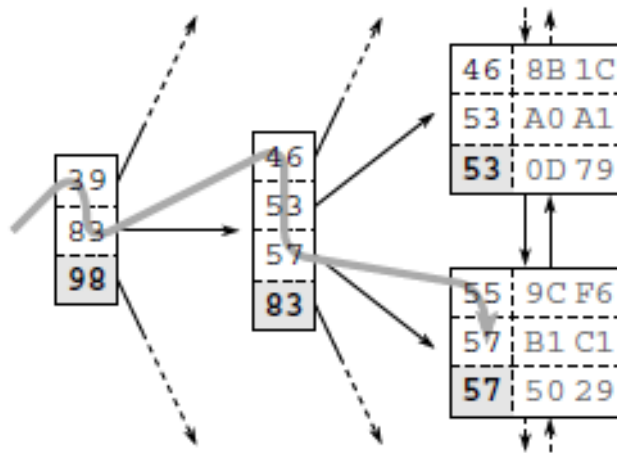
- INDEX RANGE SCAN

Проход по дереву и по списку листа

- TABLE ACCESS BY INDEX ROWID

После поиска (по индексу ) возврат одной строки

# Поиск в неуникальном индексе



# Использование индексов

- CREATE TABLE employees (
  - employee\_id NUMBER NOT NULL,
  - first\_name VARCHAR2(1000) NOT NULL,
  - last\_name VARCHAR2(1000) NOT NULL,
  - date\_of\_birth DATE NOT NULL,
  - phone\_number VARCHAR2(1000) NOT NULL,
  - **CONSTRAINT employees\_pk PRIMARY KEY (employee\_id)**
  - );
- SELECT first\_name, last\_name  
FROM employees  
WHERE **employee\_id = 123**

# Использование индексов

- `SELECT first_name, last_name  
FROM employees  
WHERE employee_id = 123`

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	2
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	2
*2	<b>INDEX UNIQUE SCAN</b>	<b>EMPLOYEES_PK</b>	1	1

Predicate Information (identified by operation id):

2 - access("EMPLOYEE\_ID"=123)

# Использование индексов

- CREATE TABLE employees (
  - employee\_id NUMBER NOT NULL,
  - first\_name VARCHAR2(1000) NOT NULL,
  - last\_name VARCHAR2(1000) NOT NULL,
  - date\_of\_birth DATE NOT NULL,
  - phone\_number VARCHAR2(1000) NOT NULL,
  - **CONSTRAINT employees\_pk PRIMARY KEY (employee\_id)**
- );
- **Alter table** employees add SUBSIDIARY\_ID NUMBER ;
- **Alter table** employees **drop primary key**;
- **Alter table** employees **add primary key** (employee\_id ,SUBSIDIARY\_ID);
- **CREATE UNIQUE INDEX** employee\_pk  
**ON** employees (employee\_id, subsidiary\_id);



# Использование индексов

```
SELECT first_name, last_name  
FROM employees  
WHERE employee_id = 123  
AND subsidiary_id = 30;
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	2
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	2
*2	INDEX UNIQUE SCAN	EMPLOYEES_PK	1	1

Predicate Information (identified by operation id):

2 - access("EMPLOYEE\_ID"=123 AND "SUBSIDIARY\_ID"=30)

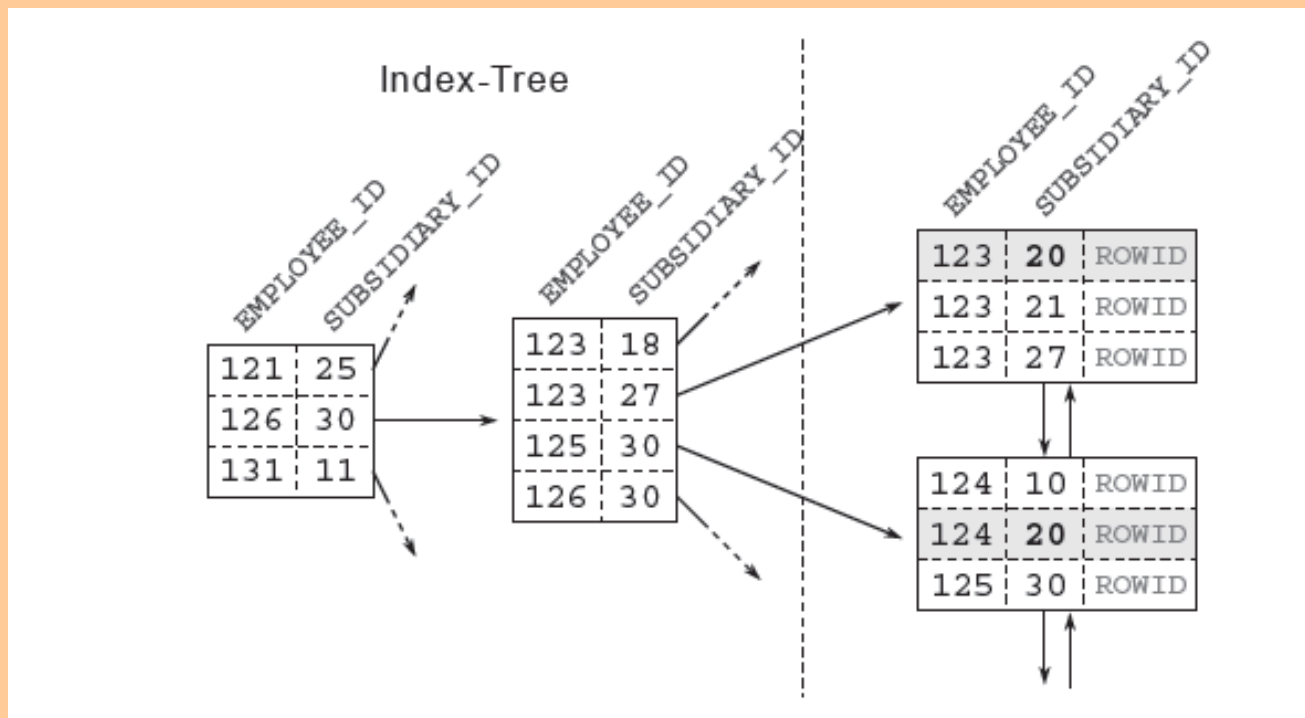
```
SELECT first_name, last_name  
FROM employees  
WHERE subsidiary_id = 20;
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		106	478
*1	TABLE ACCESS FULL	EMPLOYEES	106	478

Predicate Information (identified by operation id):

1 - filter("SUBSIDIARY\_ID"=20)

# Составной индекс



# Разница между индексами

`SELECT` first\_name, last\_name `FROM` employees `WHERE` subsidiary\_id = 20;

`CREATE UNIQUE INDEX` employee\_pk  
`ON` employees (**employee\_id**, **subsidiary\_id**);

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		106	478
* 1	TABLE ACCESS FULL	EMPLOYEES	106	478

Predicate Information (identified by operation id):

1 - filter("SUBSIDIARY\_ID"=20)

`CREATE UNIQUE INDEX` EMPLOYEES\_PK  
`ON` EMPLOYEES (**SUBSIDIARY\_ID**, **EMPLOYEE\_ID**);

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		106	75
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	106	75
*2	INDEX RANGE SCAN	EMPLOYEE_PK	106	2

Predicate Information (identified by operation id):

2 - access("SUBSIDIARY\_ID"=20)

# Влияния на другие запросы

- `SELECT first_name, last_name, subsidiary_id,  
phone_number  
FROM employees  
WHERE last_name = 'Иванов'  
AND subsidiary_id = 30;`

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	30
*1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	30
*2	INDEX RANGE SCAN	EMPLOYEES_PK	40	2

Predicate Information (identified by operation id):

- 1 - filter("LAST\_NAME"='Иванов')
- 2 - access("SUBSIDIARY\_ID"=30)

- `SELECT /*+ NO_INDEX(EMPLOYEES EMPLOYEE_PK) */`
- `first_name, last_name, subsidiary_id, phone_number`
- `FROM employees`
- `WHERE last_name = 'Иванов'`
- `AND subsidiary_id = 30;`

-----				
Id	Operation	Name	Rows	Cost
-----				
0	SELECT STATEMENT		1	<b>477</b>
* 1	<b>TABLE ACCESS FULL</b>	EMPLOYEES	1	477
-----				

Predicate Information (identified by operation id):

-----  
 1 - filter("LAST\_NAME"='Иванов' AND "SUBSIDIARY\_ID"=30)

Скан полный, но выполняется быстрее. Почему?

# После обновления статистики

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	680
*1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	680
*2	INDEX RANGE SCAN	EMPLOYEES_PK	1000	4

Без Индекса

Predicate Information (identified by operation id):

- 1 - filter("LAST\_NAME"='Иванов')
- 2 - access("SUBSIDIARY\_ID"=30)

CREATE INDEX emp\_name ON employees (last\_name);

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	3
* 1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	3
* 2	INDEX RANGE SCAN	EMP_NAME	1	1

С Индексом

Predicate Information (identified by operation id):

- 1 - filter("SUBSIDIARY\_ID"=30)
- 2 - access("LAST\_NAME"='Иванов')

# Функции и арифметика

- SELECT first\_name, last\_name, phone\_number FROM employees WHERE **UPPER(last\_name) = UPPER('Иванов');**

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		10	477
* 1	TABLE ACCESS FULL	EMPLOYEES	10	477

Predicate Information (identified by operation id):

1 - filter(UPPER("LAST\_NAME")='Иванов')

CREATE INDEX emp\_up\_name ON employees (**UPPER(last\_name)**);

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	3
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	3
*2	INDEX RANGE SCAN	EMP_UP_NAME	1	1

Predicate Information (identified by operation id):

2 - access(UPPER("LAST\_NAME")='Иванов')

# Как посмотреть план выполнения?

- {**EXPLAIN** | **DESCRIBE** | **DESC**}  
*tbl\_name* [*col\_name* | *wild*]
- {**EXPLAIN** | **DESCRIBE** | **DESC**} [*explain\_type*]  
{*explainable\_stmt* | **FOR CONNECTION** *connection\_id*}
- *explain\_type*: { **FORMAT** = *format\_name* }
- *format\_name*: { TRADITIONAL | JSON | TREE }
- *explainable\_stmt*:  
{ **SELECT** statement |  
**DELETE** statement |  
**INSERT** statement |  
**REPLACE** statement |  
**UPDATE** statement }



# Пример explain

18 • `explain select * from student where id_st=1;`

<

Result Grid | Filter Rows:  | Export: | Wrap Cell Content: [IA](#)

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	student	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL

19 • `explain select * from student where id_st=(select max(id_st) from student);`

Result Grid | Filter Rows:  | Export: | Wrap Cell Content: [IA](#)

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	PRIMARY	student	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
	2	SUBQUERY	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Select tables optimized away

21 • `explain select * from student where id_gr in(select max(id_gr) from st_group);`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	student	NULL	ALL	NULL	NULL	NULL	NULL	1	100.00	Using where
	2	DEPENDENT SUBQUERY	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Select tables optimized away

create index ind1 on student (id\_gr);

22 • `explain select * from student where id_gr in(select max(id_gr) from st_group);`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	student	NULL	ALL	NULL	NULL	NULL	NULL	1	100.00	Using where
	2	DEPENDENT SUBQUERY	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Select tables optimized away

26 • `explain select * from student inner join st_group on st_group.id_gr=student.id_gr`  
 27 `where surname like 'Иванов%';`

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	student	NULL	ALL	ind1	NULL	NULL	NULL	5	20.00	Using where
	1	SIMPLE	st_group	NULL	eq_ref	PRIMARY	PRIMARY	4	uni.student.id_gr	1	100.00	NULL

# EXPLAIN Выходные столбцы

Столбец	Имя в JSON	Назначение
<a href="#">id</a>	select_id	Идентификатор SELECT
<a href="#">select type</a>	None	Тип SELECTа
<a href="#">table</a>	table_name	Таблица выходной строки
<a href="#">partitions</a>	partitions	The matching partitions
<a href="#">type</a>	access_type	Тип соединения
<a href="#">possible keys</a>	possible_keys	Возможные индексы
<a href="#">key</a>	key	Выбранный индекс
<a href="#">key len</a>	key_length	Длина выбранного ключа
<a href="#">ref</a>	ref	Столбец сравниваемый с индексом
<a href="#">rows</a>	rows	Оценочное количество возвращенных строк
<a href="#">filtered</a>	filtered	Процент строк отфильтрованных по условию
<a href="#">Extra</a>	None	Дополнительная информация

# type

- eq\_ref

Проход только по В-дереву (только первичный уник)

- ref, range

Проход по В-дереву и по списку в листе (диапазон)

- index

Просматривает весь индекс

- ALL

Просматривает всю таблицу на диске

# Другие столбцы

- Using Index ( “Extra” )

Не просматривается таблица, т.к. все необходимое есть в индексе

- PRIMARY ( “key” / “possible\_keys” )

PRIMARY – имя автоматически сгенерированного индекса на первичный ключ

- using filesort ( “Extra” )

Явная операция сортировки на диске или в памяти – ресурсоемкое т.к материализация

- implicit: no “using filesort” ( “Extra” )

При использовании limit если нет using filesort- то pipeline

# Подробности explain

```
explain select * from student where id_st=(select max(id_st) from student);  
SHOW WARNINGS ;
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	student	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
	2	SUBQUERY	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	Select tables optimized away

	Level	Code	Message
▶	Note	1003	/*select#1 */select '7' AS `id_st`,`gr` AS `surname`,`gr1` AS `name`,`gr` AS `patronym`,`7` AS `id_gr` from `uni`.`student` where 1

# SHOW WARNINGS

- **<auto\_key>** Автоматически сгенерированный ключ для временной таблицы.
- **<cache>(expr)** Выражение (например, скалярный подзапрос) выполняется один раз, и полученное значение сохраняется в памяти для последующего использования. Для результатов, состоящих из нескольких значений, можно создать временную таблицу, и вместо этого будет **<temporary table>** .
- **<exists>** (фрагмент запроса) Предикат подзапроса преобразуется в предикат EXISTS, а подзапрос преобразуется, чтобы его можно было использовать вместе с предикатом EXISTS.
- **<in\_optimizer>** (фрагмент запроса) Это внутренний объект оптимизатора без значения пользователя.
- **<index\_lookup>** (фрагмент запроса) Фрагмент запроса обрабатывается с использованием поиска по индексу для поиска подходящих строк.
- **<if>** (условие, expr1, expr2) Если условие истинно, оцените как expr1, иначе expr2.
- **<is\_not\_null\_test>** (выражение) Тест для проверки того, что выражение не оценивается как NULL.
- **<materialize>** (фрагмент запроса) используется материализация подзапроса. `Материализовался-subquery`.col\_name Ссылка на столбец col\_name во внутренней временной таблице, материализованной для хранения результата оценки подзапроса.
- **<primary\_index\_lookup>** (фрагмент запроса) Фрагмент запроса обрабатывается с использованием поиска первичного ключа для поиска подходящих строк.
- **<Ref\_null\_helper>** (expr1) Это внутренний объект оптимизатора без значения пользователя.
- **/ \* select # N \* / select\_stmt** SELECT связан со строкой в расширенном выводе EXPLAIN со значением id, равным N.
- **outer\_tables semi join (inner\_tables)** Операция полусоединения. inner\_tables показывает таблицы, которые не были извлечены.
- **<temporary table>** Это представляет собой внутреннюю временную таблицу, созданную для кэширования промежуточного результата.

# Анализ статистики таблицы

- `ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]  
TABLE tbl_name [, tbl_name] ...`
- `ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]  
TABLE tbl_name UPDATE HISTOGRAM ON  
col_name [, col_name] ... [WITH N BUCKETS]`
- 
- `ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]  
TABLE tbl_name DROP HISTOGRAM ON col_name  
[, col_name] ...`



# Пример анализа статистики таблицы

28 • `ANALYZE table student;`

<



Result Grid |  Filter Rows:  | Export: 

	Table	Op	Msg_type	Msg_text
▶	uni.student	analyze	status	OK

29 ✖ `ANALYZE table student UPDATE HISTOGRAM ON id_gr;`

<




Result Grid |  Filter Rows:  | Export:  | Wrap Cell Content: 

	Table	Op	Msg_type	Msg_text
▶	uni.student	histogram	status	Histogram statistics created for column 'id_gr'.

Оператор/ компонент WITH

# Поддержка оператора With

	BigQuery	Db2 (LUW)	MariaDB	MySQL	Oracle DB	PostgreSQL	SQL Server	SQLite
With clause	✓ <sub>0</sub>	✓	✓	✓	✓	✓	✓	✓
With recursive clause	✗	✓ <sub>1</sub>	✓	✓	✓ <sub>2</sub>	✓	✓ <sub>2</sub>	✓
Over (...)	✓	✓	✓	✓	✓	✓	✓	✓
from ... for system_time ...	✓ <sub>4</sub>	✓ <sub>3</sub>	✓ <sub>3</sub>	✗	✗ <sub>5</sub>	✗	✓ <sub>3</sub>	✗

<sup>0</sup> Without column names: WITH name AS (SELECT...)

<sup>1</sup> Without keyword recursive. No join in recursive branch—use comma-join (,).

<sup>2</sup> Without keyword recursive.

<sup>3</sup> Some minor omissions and variations.

<sup>4</sup> Without DDL (automatically). Only FOR SYSTEM\_TIME AS OF.

<sup>5</sup> Alternative syntax. E.g. no for system\_time.

# Поддержка различных особенностей

	BigQuery	Db2 (LUW)	MariaDB	MySQL	Oracle DB	PostgreSQL	SQL Server	SQLite
with on top-level	✓ <sub>0</sub>	✓	✓	✓	✓	✓	✓	✓
with in subqueries	✓ <sub>2</sub>	✓ <sub>1</sub>	✓ <sub>1</sub>	✓	✓	✓ <sub>1</sub>	✓	✓
insert ... with ... select	✓ <sub>0</sub>	✓	✓	✓	✓	✗ <sub>3</sub>	✗ <sub>3</sub>	✓ <sub>3</sub>
with masks schema objects	✓	✓	✓	✓	✓	✓	✓	✓
with doesn't imply recursive	✓	✗	✓	✓	✗	✓	✗	✗
views bypass with	✓	✓	✓	✓	✓	✓	✓	✗
qualified names bypass with	✓	✗	✓	✓	✗	✓	✓	✓

<sup>0</sup> Without column names: WITH name AS (SELECT...)

<sup>1</sup> CTE in subquery cannot see global CTEs.

<sup>2</sup> Without column names: WITH name AS (SELECT...) CTE in subquery cannot see global CTEs.

<sup>3</sup> Supports proprietary syntax with ... insert ... select

# Синтаксис

- WITH query\_name (column\_name1, ...) AS
- (SELECT ...)
- 
- SELECT ...

- WITH query\_name1 AS (
  - SELECT ...
  - )
  - , query\_name2 AS (
    - SELECT ...
    - FROM query\_name1
    - ...
    - )
  - SELECT ...

# Пример использования with

- `select st_group.id_gr,  
number_gr, count(id_st) as cnt  
from st_group join student on  
student.id_gr=st_group.id_gr  
group by st_group.id_gr,  
number_gr  
having count(id_st)=  
(select max(cnt) from  
(select st_group.id_gr,  
count(id_st) as cnt from  
st_group join student on  
student.id_gr=st_group.id_gr  
group by st_group.id_gr )q1);`
- `with q1 as  
(select st_group.id_gr,  
number_gr, count(id_st) as  
cnt from st_group join  
student on  
student.id_gr=st_group.id_gr  
group by st_group.id_gr,  
number_gr)  
select q1.id_gr,  
q1.number_gr, cnt from q1  
where cnt=  
(select max(cnt) from q1);`

# Рекурсивный with

- **Recursive Common Table Expressions CTE**

```
1  SELECT ...      -- return initial row set
2  UNION ALL
3  SELECT ...      -- return additional row sets
```

```
1  WITH RECURSIVE cte (n) AS
2  (
3    SELECT 1
4    UNION ALL
5    SELECT n + 1 FROM cte WHERE n < 5
6  )
7  SELECT * FROM cte;
```



# Рекурсивный with

```
WITH RECURSIVE cte AS  
(  
    SELECT 1 AS n, CAST('abc' AS CHAR(20)) AS str  
    UNION ALL  
    SELECT n + 1, CONCAT(str, str) FROM cte WHERE n < 3  
)  
SELECT * FROM cte;
```

Result Grid   Filter Rows:		
	n	str
▶	1	abc
	2	abcbc
	3	abcbcabcbc

# Ограничения в рекурсивных подзапросах

- В рекурсивных подзапросах применяются некоторые синтаксические ограничения:
- Рекурсивная часть SELECT не должна содержать этих конструкций:
- Агрегатные функции, такие как SUM ()
- Оконные функции
- GROUP BY
- ORDER BY
- DISTINCT
- До MySQL 8.0.19 рекурсивная часть SELECT рекурсивного CTE также не могла использовать предложение LIMIT. Это ограничение снято в MySQL 8.0.19, и теперь в таких случаях поддерживается LIMIT вместе с дополнительным предложением OFFSET. Эффект на результирующий набор такой же, как при использовании LIMIT во внешнем SELECT, но также более эффективен, поскольку его использование с рекурсивным SELECT останавливает генерацию строк, как только будет создано их запрошенное количество.
- Эти ограничения не применяются к нерекурсивной части SELECT рекурсивного CTE. Запрет на DISTINCT распространяется только на членов UNION; UNION DISTINCT разрешен.
- Рекурсивная часть SELECT должна ссылаться на CTE только один раз и только в своем предложении FROM, а не в каком-либо подзапросе. Он может ссылаться на таблицы, отличные от CTE, и объединять их с

# WITH

```
explain
with q1 as(select st_group.id_gr,number_gr,
count(id_st) as cnt
from st_group
join student on student.id_gr=st_group.id_gr
group by st_group.id_gr,number_gr)
select q1.id_gr,q1.number_gr,cnt from q1
where cnt=(select max(cnt) from q1);
```




	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	PRIMARY	<derived2>	NULL	ref	<auto_key0>	<auto_key0>	8	const	1	100.00	Using where
	3	SUBQUERY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	3	100.00	NULL
	2	DERIVED	st_group	NULL	ALL	PRIMARY	NULL	NULL	NULL	3	100.00	Using temporary
	2	DERIVED	student	NULL	ref	id_gr_st_1	id_gr_st_1	5	uni.st_group.id_gr	1	100.00	Using index

# Подзапрос

```
explain select st_group.id_gr,number_gr,
count(id_st) as cnt
from st_group join student
on student.id_gr=st_group.id_gr
group by st_group.id_gr,number_gr
having count(id_st)=
(select max(cnt) from
(select st_group.id_gr,number_gr,count(id_st) as cnt
from st_group join student
on student.id_gr=st_group.id_gr
group by st_group.id_gr,number_gr)q1);
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:									
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	PRIMARY	st_group	NULL	ALL	PRIMARY	NULL	NULL	NULL	3	100.00	Using temporary
	1	PRIMARY	student	NULL	ref	id_gr_st_1	id_gr_st_1	5	uni.st_group.id_gr	1	100.00	Using index
	2	SUBQUERY	<derived3>	NULL	ALL	NULL	NULL	NULL	NULL	3	100.00	NULL
	3	DERIVED	st_group	NULL	ALL	PRIMARY	NULL	NULL	NULL	3	100.00	Using temporary
	3	DERIVED	student	NULL	ref	id_gr_st_1	id_gr_st_1	5	uni.st_group.id_gr	1	100.00	Using index

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	PRIMARY	<derived2>	NULL	ref	<auto_key0>	<auto_key0>	8	const	1	100.00	Using where
	3	SUBQUERY	<derived2>	NULL	ALL	NULL	NULL	NULL	NULL	3	100.00	NULL
	2	DERIVED	st_group	NULL	ALL	PRIMARY	NULL	NULL	NULL	3	100.00	Using temporary
	2	DERIVED	student	NULL	ref	id_gr_st_1	id_gr_st_1	5	uni.st_group.id_gr	1	100.00	Using index

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
►	1	PRIMARY	st_group	NULL	ALL	PRIMARY	NULL	NULL	NULL	3	100.00	Using temporary
	1	PRIMARY	student	NULL	ref	id_gr_st_1	id_gr_st_1	5	uni.st_group.id_gr	1	100.00	Using index
	2	SUBQUERY	<derived3>	NULL	ALL	NULL	NULL	NULL	NULL	3	100.00	NULL
	3	DERIVED	st_group	NULL	ALL	PRIMARY	NULL	NULL	NULL	3	100.00	Using temporary
	3	DERIVED	student	NULL	ref	id_gr_st_1	id_gr_st_1	5	uni.st_group.id_gr	1	100.00	Using index

# Литература

