

Введение в реляционные базы данных

Лекция 7: JOIN

Артем Толканев

October 16, 2024

Мы рассматривали процесс нормализации, нужный в том числе для того, что избежать необходимости дублировать информацию.

Затем, мы использовали Теорему Хита

Пусть r - переменная отношения, а A, B, C - непересекающиеся множества атрибутов этой переменной отношения.

Если r удовлетворяет функциональной зависимости $A \rightarrow B$,
то r равна соединению её проекций по атрибутам $r_1(A, B)$ и $r_2(A, C)$.

Чтобы получить изначальные отношения без потери информации

В предыдущих сериях



Берет два отношения и генерирует отношение, которое содержит все возможные пары кортежей из соответствующих отношений для которых выполнены определенные условия

⋈ JOIN

$$R \bowtie_a Q = \{x \oplus y, R^*(x) \cap Q^*(y) \mid (x \in \text{dom}(R)) \text{ and } (y \in \text{dom}(Q)) \text{ and } a(x,y)\}$$

Затем, мы использовали Теорему Хита

Пусть r - переменная отношения, а A, B, C - непересекающиеся множества атрибутов этой переменной отношения.

Если r удовлетворяет функциональной зависимости $A \rightarrow B$, то r равна **соединению** её проекций по атрибутам $r_1(A,B)$ и $r_2(A,C)$.

Чтобы получить изначальные отношения без потери информации

В предыдущих сериях



Берет два отношения и генерирует отношение, которое содержит все возможные пары кортежей из соответствующих отношений для которых выполнены определенные условия

⋈ JOIN

$$R \bowtie_a Q = \{x \oplus y, R^*(x) \cap Q^*(y) \mid (x \in \text{dom}(R)) \text{ and } (y \in \text{dom}(Q)) \text{ and } a(x,y)\}$$

Казалось бы

Для каждой записи $x \in R$ и $y \in Q$, которые подходят под условие $a(x,y)$, соединяем и в одну запись

select

b.name, l.language

from

book b

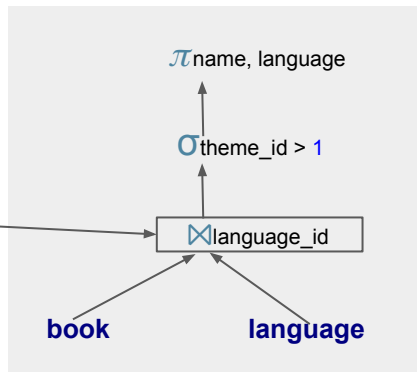
inner join

"language" l

on l.language_id = b.language_id

where

b.theme_id > 1



В предыдущих сериях



Берет два отношения и генерирует отношение, которое содержит все возможные пары кортежей из соответствующих отношений для которых выполнены определенные условия

⋈ JOIN

$$R \bowtie_a Q = \{x \oplus y, R^*(x) \cap Q^*(y) \mid (x \in \text{dom}(R)) \text{ and } (y \in \text{dom}(Q)) \text{ and } a(x,y) \}$$

Казалось бы

Для каждой записей $x \in R$ и $y \in Q$, которые подходят под условие $a(x,y)$, соединяем и в одну запись

select

b.name, l.language

from

book b

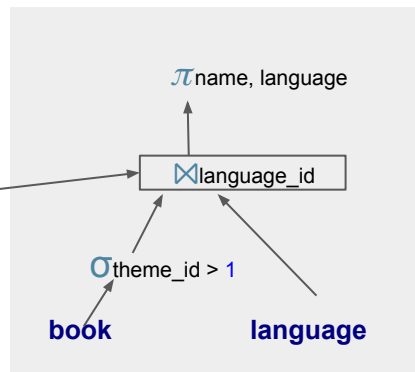
inner join

"language" l

on l.language_id = b.language_id

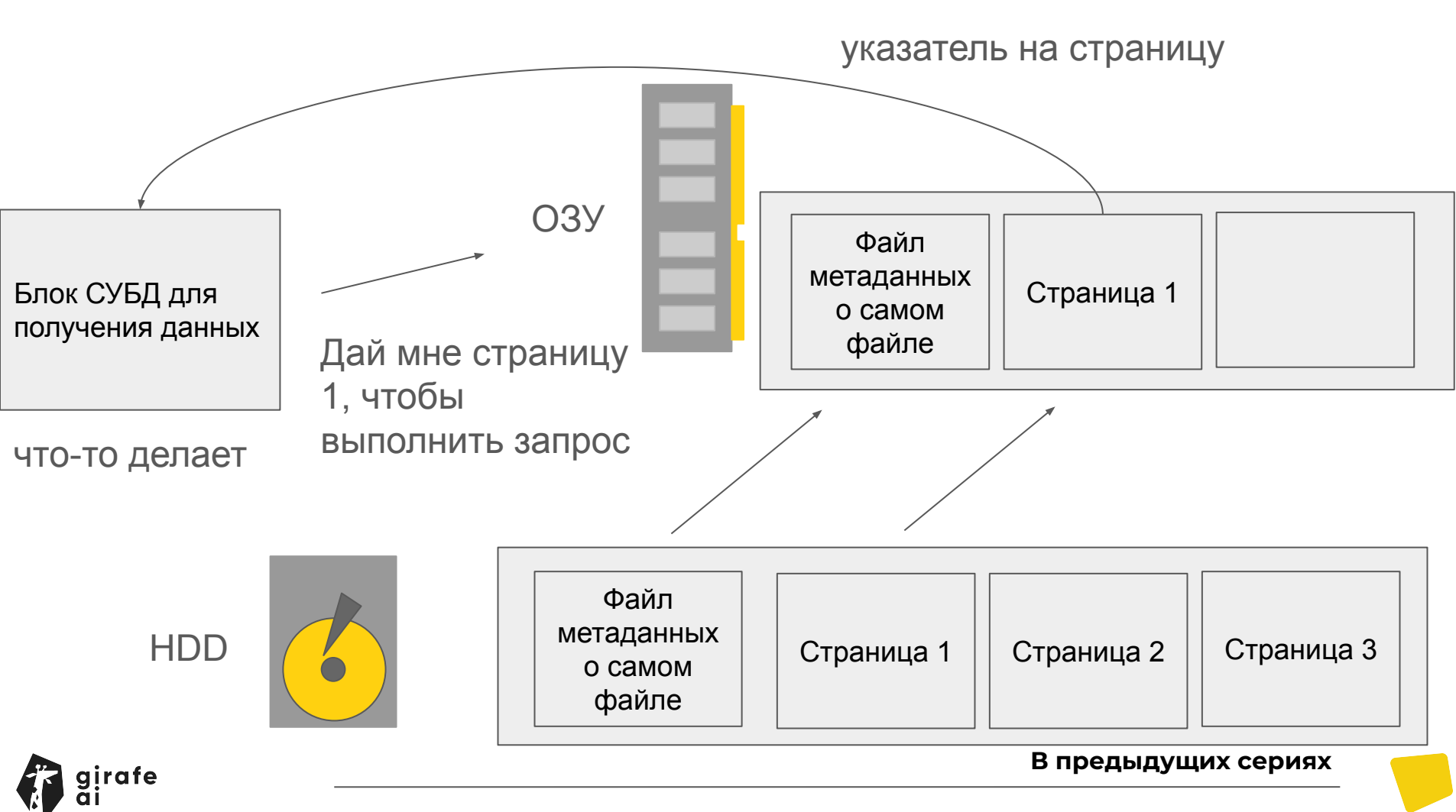
where

b.theme_id > 1



В предыдущих сериях





JOIN

- LOOP JOIN
- MERGE JOIN
- HASH JOIN



LOOP JOIN

```
FOREACH ROW b IN "book":  
  FOREACH ROW l IN "language":  
    IF MATCH expression:  
      OUTPUT
```




```
FOREACH ROW b IN "book":  
  FOREACH ROW l IN "language":  
    IF MATCH expression:  
      OUTPUT
```

```
FOREACH PAGE B IN "book":  
  FOREACH ROW b IN B:  
    FOREACH PAGE L IN "language":  
      FOREACH ROW l IN L:  
        IF MATCH expression:  
          OUTPUT
```

доступ к book + count(b) * доступ language



BLOCK NESTED LOOP JOIN

```
FOREACH ROW b IN "book":  
  FOREACH ROW l IN "language":  
    IF MATCH expression:  
      OUTPUT
```

```
FOREACH PAGE B IN "book":  
  FOREACH ROW b IN B:  
    FOREACH PAGE L IN "language":  
      FOREACH ROW l IN L:  
        IF MATCH expression:  
          OUTPUT
```

```
FOREACH PAGE B IN "book":  
  FOREACH PAGE L IN "language":  
    FOREACH ROW b IN B:  
      FOREACH ROW l IN L:  
        IF MATCH expression:  
          OUTPUT
```



Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables t_1 , t_2 , and t_3 is to be executed using the following

Table	Join Type
t_1	range
t_2	ref
t_3	ALL

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in  $t_1$  matching range {  
  for each row in  $t_2$  matching reference key {  
    for each row in  $t_3$  {  
      if row satisfies join conditions, send to client  
    }  
  }  
}
```

Because the NLI algorithm passes rows one at a time from outer loops to inner loops. it two

its
on



BLOCK NESTED LOOP JOIN

<https://dev.mysql.com/doc/refman/8.4/en/bnl-bka-optimization.html#bnl-optimization>

By default, `block_nested_loop` is on and `batched_key_access` is off. See [Section 10.9.2, "Switchable Optimizations"](#). Optimize [Optimizer Hints for Block Nested-Loop and Batched Key Access Algorithms](#).

For information about semijoin strategies, see [Optimizing IN and EXISTS Subquery Predicates with Semijoin Transformations](#).

Block Nested-Loop Algorithm for Outer Joins and Semijoins

The original implementation of the MySQL BNL algorithm was extended to support outer join and semijoin operations (and was join algorithm; see [Section 10.2.1.4, "Hash Join Optimization"](#)).

When these operations are executed with a join buffer, each row put into the buffer is supplied with a match flag.

If an outer join operation is executed using a join buffer, each row of the table produced by the second operand is checked for a join buffer. When a match is found, a new extended row is formed (the original row plus columns from the second operand) and the remaining join operations. In addition, the match flag of the matched row in the buffer is enabled. After all rows of the table examined, the join buffer is scanned. Each row from the buffer that does not have its match flag enabled is extended by `NULL` on each column in the second operand) and sent for further extensions by the remaining join operations.

The `block_nested_loop` flag of the `optimizer_switch` system variable controls hash joins.

See [Section 10.9.2, "Switchable Optimizations"](#), for more information. Optimizer hints may also be applied; see [Optimizer Hints for Batched Key Access Algorithms](#).

In `EXPLAIN` output, use of BNL for a table is signified when the `Extra` value contains `Using join buffer (Block Nested Loop index, or range)`.

For information about semijoin strategies, see [Optimizing IN and EXISTS Subquery Predicates with Semijoin Transformations](#).

<https://dev.mysql.com/doc/refman/8.4/en/nested-loop-joins.html#nested-loop-join-algorithm>

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables `t1`, `t2`, and `t3` is to be executed using the following

Table	Join Type
t1	range
t2	ref
t3	ALL

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

Because the NLI algorithm passes rows one at a time from outer loops to inner loops, it two

its
on

LOOP JOIN



BLOCK NESTED LOOP JOIN

https://dev.mysql.com/doc/refman/8.4/en/bnl-bka-optimization.html#bnl-optimization

By default, `block_nested_loop` is on and `batched_key_access` is off. See [Section 10.9.2, "Switchable Optimizations"](#). Optimize [Optimizer Hints for Block Nested-Loop and Batched Key Access Algorithms](#).

For information about semijoin strategies, see [Optimizing IN and EXISTS Subquery Predicates with Semijoin Transformations](#).

Block Nested-Loop Algorithm for Outer Joins and Semijoins

The original implementation of the MySQL BNL algorithm was extended to support outer join and semijoin operations (and was join algorithm; see [Section 10.2.1.4, "Hash Join Optimization"](#)).

When these operations are executed with a join buffer, each row put into the buffer is supplied with a match flag.

If an outer join operation is executed using a join buffer, each row of the table produced by the second operand is checked for a join buffer. When a match is found, a new extended row is formed (the original row plus columns from the second operand) and the remaining join operations. In addition, the match flag of the matched row in the buffer is enabled. After all rows of the table examined, the join buffer is scanned. Each row from the buffer that does not have its match flag enabled is extended by `NULL` (each column in the second operand) and sent for further extensions by the remaining join operations.

The `block_nested_loop` flag of the `optimizer_switch` system variable controls hash joins.

See [Section 10.9.2, "Switchable Optimizations"](#), for more information. Optimizer hints may also be applied; see [Optimizer Hints for Batched Key Access Algorithms](#).

In `EXPLAIN` output, use of BNL for a table is signified when the `Extra` value contains `Using join buffer (Block Nested Loop, index, or range)`.

For information about semijoin strategies, see [Optimizing IN and EXISTS Subquery Predicates with Semijoin Transformations](#).

https://dev.mysql.com/doc/refman/8.0/en/nested-loop-joins.html

```
}  
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, it typically reads tables processed in the inner loops many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops are read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all rows in the buffer. This reduces by an order of magnitude the number of times the inner table must be read.

Prior to MySQL 8.0.18, this algorithm was applied for equi-joins when no indexes could be used; in MySQL 8.0.18 and later, the hash join optimizer is employed in such cases. Starting with MySQL 8.0.20, the block nested loop is no longer used by MySQL, and a hash join is employed for in all cases where a block nested loop was used previously. See [Section 10.2.1.4, "Hash Join Optimization"](#).

MySQL join buffering has these characteristics:

- Join buffering can be used when the join is of type `ALL` or `index` (in other words, when no possible keys can be used, and a full scan is done on data or index rows, respectively), or `range`. Use of buffering is also applicable to outer joins, as described in [Section 10.2.1.12, "Block Nested-Loop Join Algorithms"](#).
- A join buffer is never allocated for the first nonconstant table, even if it would be of type `ALL` or `index`.
- Only columns of interest to a join are stored in its join buffer, not whole rows.
- The `join_buffer_size` system variable determines the size of each join buffer used to process a query.
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is allocated prior to executing the join and freed after the query is done.

For the example join described previously for the NLJ algorithm (without buffering), the join is done as follows using join buffering:



```
SELECT b.name, b2.language FROM
(
SELECT * FROM book b
) b
FULL OUTER JOIN
(SELECT * FROM "language" l) b2
ON b.language_id <= 1 - b2.language_id
```



MERGE JOIN

sort-merge join

Сортировка

Соединение



ORDER "book", "language" ON language_id

link_b **TO "book"**, link_l **TO "language"**

WHILE link_b **AND** link_l:

IF link_b > link_l: **INCREMENT** link_l

IF link_b < link_l:

INCREMENT link_b

backtrack link_l

ELSE

link_b = link_l:

OUTPUT

INCREMENT link_l

language_id	language
0	русский
1	английский
2	французский
3	немецкий

book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
209	1984	2
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
202	Othello	2
204	Обломов	1

language_id

MERGE JOIN



ORDER "book", "language" ON language_id

link_b TO "book", link_l TO "language"

WHILE link_b AND link_l:

IF link_b > link_l: INCREMENT link_l

IF link_b < link_l:

INCREMENT link_b


backtrack link_l

ELSE

link_b = link_l:


OUTPUT

INCREMENT link_l



language_id	language
0	русский
1	английский
2	французский
3	немецкий

book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
204	Обломов	1
209	1984	2
202	Othello	2



MERGE JOIN



ORDER "book", "language" ON language_id

link_b TO "book", link_l TO "language"

WHILE link_b AND link_l:

IF link_b > link_l: INCREMENT link_l

IF link_b < link_l:

INCREMENT link_b

backtrack link_l

ELSE

link_b = link_l:

OUTPUT

INCREMENT link_l

language_id	language
0	русский
1	английский
2	французский
3	немецкий

book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
204	Обломов	1
209	1984	2
202	Othello	2

book_id	name	language_id	language_id	language
215	Курс аналитической геометрии	1	1	английский

MERGE JOIN



ORDER "book", "language" **ON** language_id

link_b **TO** "book", link_l **TO** "language"

WHILE link_b **AND** link_l:

IF link_b > link_l: **INCREMENT** link_l

IF link_b < link_l:

INCREMENT link_b

 (**GO BACK** link_l)

ELSE

 link_b = link_l:

OUTPUT

INCREMENT link_l

language_id	language
0	русский
1	английский
2	французский
3	немецкий

book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
204	Обломов	1
209	1984	2
202	Othello	2

book_id	name	language_id	language_id	language
215	Курс аналитической геометрии	1	1	английский


MERGE JOIN



```

ORDER "book", "language" ON language_id
link_b TO "book", link_l TO "language"
WHILE link_b AND link_l:
  IF link_b > link_l: INCREMENT link_l
  IF link_b < link_l:
    INCREMENT link_b
    (GO BACK link_l )
  ELSE
    link_b = link_l:
    OUTPUT
    INCREMENT link_l

```



language_id	language
0	русский
1	английский
2	французский
3	немецкий


book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
204	Обломов	1
209	1984	2
202	Othello	2

book_id	name	language_id	language_id	language
215	Курс аналитической геометрии	1	1	английский
210	Мартин Иден	1	1	английский

MERGE JOIN



Становится очень грустно,
когда значение атрибута, по
которому происходит
соединение, сильно
повторяется



language_id	language
0	русский
1	английский
2	французский
3	немецкий


book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
204	Обломов	1
209	1984	2
202	Othello	2

book_id	name	language_id	language_id	language
215	Курс аналитической геометрии	1	1	английский
210	Мартин Иден	1	1	английский

MERGE JOIN



Становится очень грустно, когда значение атрибута, по которому происходит соединение, сильно повторяется
(на сам MERGE уходит что-то между BL и B + L)



language_id	language
0	русский
1	английский
2	французский
3	немецкий

book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
204	Обломов	1
209	1984	2
202	Othello	2

book_id	name	language_id	language_id	language
215	Курс аналитической геометрии	1	1	английский
210	Мартин Иден	1	1	английский

MERGE JOIN



Становится очень грустно,
когда значение атрибута, по
которому происходит
соединение, сильно
повторяется

Когда это полезно?
Когда изначально таблицы
отсортированы по атрибуту, который
используется в объединении



HASH JOIN

Берем одну таблицу и на основе атрибута создаем хэш-таблицу на основе какой-то хеш-функции

Сканируем вторую таблицу и используем хеш-функцию на каждой строке



book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
209	1984	2
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
202	Othello	2
204	Обломов	1

```
FOREACH ROW b IN "book":
  PUT h(b) IN hash_table
```



hash_table	...
0x00	
0x01	



language_id	language
0	русский
1	английский
2	французский
3	немецкий

```
FOREACH ROW l IN "language":
  IF h(l) IN hash_table:
    IF MATCH EXPRESSION :
      OUTPUT
```



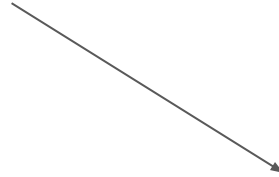
hash_table	...
0x00	
0x01	

HASH JOIN



book_id	name	language_id
215	Курс аналитической геометрии	1
210	Мартин Иден	1
205	Капитанская Дочка	1
200	Война и Мир 1,2 том	1
214	Русские сказки	1
211	Сердца Трех	1
203	Курс аналитической геометрии	1
206	Общая физика	1
209	1984	2
213	Три Сестры	1
201	Война и Мир 3,4 том	1
208	Анна Каренина	1
212	Белый Клык	1
207	Дубровский	1
202	Othello	2
204	Обломов	1
....	f1gf12g1	9999999999

```
FOREACH ROW b IN "book":
    PUT h(b) IN hash_table_B
```



hash_table_B	...
0x00	
0x01	



language_id	language
0	русский
1	английский
2	французский
3	немецкий
999999	1f

```
FOREACH ROW l IN "language":
    PUT h(l) IN hash_table_L
```



hash_table_L	...
0x00	
0x01	

HASH JOIN



```
FOREACH ROW h(b) IN hash_table_B:
  FOREACH ROW h(l) IN hash_table_L:
    IF h(l) = h(b):
      IF MATCH EXPRESSION :
        OUTPUT
```

hash_table_B	...
0x00	
0x01	

hash_table_L	...
0x00	
0x01	



```

FOREACH ROW h2(h(b)) IN hash_table_B2:
  FOREACH ROW h2(h(l)) IN hash_table_L2:
    IF h2(h(b)) = h2(h(l)):
      IF MATCH EXPRESSION :
        OUTPUT

```

hash_table_B	...
0x00	
0x01	

hash_table_L	...
0x00	
0x01	

```

FOREACH ROW h(b) IN hash_table_B:
  PUT h2(h(b)) IN hash_table_B2

```

```

FOREACH ROW h(l) IN hash_table_L:
  PUT h2(h(l)) IN hash_table_L2

```

hash_table_B2	...
0x00	
0x01	

hash_table_L2	...
0x00	
0x01	

