САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Базы данных

Лекция 7

Представления. Индексы

Агафонов Антон Александрович
к.т.н., доцент кафедры ГИиИБ

Самара

План лекции

- Представления

- Индексы
  - Хеш-индексы
  - Индексы на основе сбалансированного дерева
  - Битовые индексы
  - Пространственные индексы
  - Полнотекстовые индексы

Представление

Представление (**view**) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора **SELECT**, в момент обращения к представлению.

Представление — виртуальная (логическая) таблица, представляющая собой именованный запрос, который будет подставлен как подзапрос при использовании представления.

Преимущества использования представлений

- Возможность гибкой настройки прав доступа к данным за счет того, что права даются не на таблицу, а на представление. Это позволяет назначить права на отдельные строки таблицы или возможность получения не самих данных, а результата каких-то действий над ними.
- Разделение логики хранения данных и программного обеспечения. Можно менять структуру данных, не затрагивая программный код. При изменении схемы БД достаточно создать представления, аналогичные таблицам, к которым раньше обращались приложения. Это удобно, когда нет возможности изменить программный код или к одной базе данных обращаются несколько приложений с различными требованиями к структуре данных.
- Удобство в использовании за счет автоматического выполнения таких действий как доступ к определенной части строк и/или столбцов, получение данных из нескольких таблиц и их преобразование с помощью различных функций.
- Оптимизация или предварительная компиляция запросов, что может повысить производительность выполнения запросов.

Создание представления в MySQL

```
CREATE [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Алгоритм обработки представления:

- **MERGE**: текст запроса к представлению объединяется с текстом запроса самого представления к таблицам БД, результат выполнения объединенного запроса возвращается пользователю.
- **TEMPTABLE**: содержимое представления сохраняется во временную таблицу, которая затем используется для выполнения запроса.
- **UNDEFINED** (значение по умолчанию): MySQL самостоятельно выбирает какой алгоритм использовать при обращении к представлению.

Пример создания представления в MySQL

```
CREATE ALGORITHM=TEMPTABLE
VIEW sales_by_film_category AS

SELECT c.name AS category, SUM(p.amount) AS total_sales
FROM payment p
JOIN rental r ON p.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
GROUP BY c.name
ORDER BY total_sales DESC;
```

Пример использования представления в MySQL

Использование представления

```
SELECT *
FROM sales_by_film_category
ORDER BY category
LIMIT 5;
```

Результат запроса

	category	total_sales
	Action	4375.85
	Animation	4656.30
	Children	3655.55
	Classics	3639.59
	Comedy	4383.58

Пример создания представления в MySQL

```
CREATE ALGORITHM=MERGE
VIEW customer_orders (customer_name, product_name, total_price) AS

SELECT c.first_name, p.product_name, o.product_count * p.price
FROM customers c
JOIN orders o USING (customer_id)
JOIN products p USING (product_id)
WHERE c.customer_id = 1;
```

Использование представления

```
SELECT *
FROM customer_orders
WHERE total_price > 120000;
```

Обновляемые представления

Представление называется обновляемым, если к нему могут быть применимы операторы **UPDATE** и **DELETE** для изменения данных в таблицах, на которых основано представление.

Для того чтобы представление было обновляемым, должны быть выполнены некоторые условия, в т.ч.:

- Отсутствие функций агрегации в представлении.
- Отсутствие следующих выражений в представлении: **DISTINCT**, **GROUP BY**, **HAVING**, **UNION**.
- Отсутствие подзапросов в списке выражения **SELECT**.
- Столбцы представления должны быть простыми ссылками на поля таблиц (а не, например, арифметическими выражениями) и т.д.

Обновляемое представление может допускать добавление данных (**INSERT**), если все поля таблицы-источника, не присутствующие в представлении, имеют значения по умолчанию.

Для представлений, основанных на нескольких таблицах, операция добавления данных работает только в случае, если происходит добавление в единственную реальную таблицу. Удаление данных (**DELETE**) для таких представлений не поддерживается.

Обновляемые представления

При использовании в определении представления конструкции **WITH** [**CASCADED** | **LOCAL**] **CHECK OPTION** все добавляемые или изменяемые строки будут проверяться на соответствие определению представления:

- Изменение данных (**UPDATE**) разрешено только для строк, удовлетворяющих условию **WHERE** в определении представления. Кроме того, новые значения строки также должны удовлетворять условию **WHERE**.
- Добавление данных (**INSERT**) будет происходить, только если новая строка удовлетворяет условию **WHERE** в определении представления.

Ключевые слова **CASCADED** и **LOCAL** определяют глубину проверки для представлений, основанных на других представлениях:

- Для **LOCAL** происходит проверка условия **WHERE** только в собственном определении представления.
- Для **CASCADED** происходит проверка для всех представлений, на которых основано данное представление. Значением по умолчанию является **CASCADED** .

Выборка списка товаров

```
SELECT * FROM products;
```

Результат запроса

	product_id	product_name	manufacturer	product_count	price
	1	iPhone 13	Apple	1	115000.00
	2	iPhone 12	Apple	7	79900.00
	3	Galaxy S22	Samsung	4	88000.00
	4	Galaxy S21	Samsung	3	67000.00
	5	P50	Huawei	1	115000.00
	6	12 Pro	Xiaomi	3	115000.00

Обновляемое представление

```
CREATE VIEW apple_products AS
SELECT *
FROM products
WHERE manufacturer = 'apple';
```

Выбор данных из представления

```
SELECT * FROM apple_products;
```

	product_id	product_name	manufacturer	product_count	price
	1	iPhone 13	Apple	1	115000.00
	2	iPhone 12	Apple	7	79900.00

Обновляемое представление

Обновление представления

```
UPDATE apple_products
SET manufacturer = 'Pineapple'
WHERE product_id = 1;
```

Выбор данных из представления

```
SELECT * FROM apple_products;
```

	product_id	product_name	manufacturer	product_count	price
	2	iPhone 12	Apple	7	79900.00

Обновляемое представление с проверкой ограничений

```
CREATE VIEW apple_products_constrained AS
SELECT *
FROM products
WHERE manufacturer = 'apple'
WITH LOCAL CHECK OPTION;
```

Выбор данных из представления

```
SELECT * FROM apple_products_constrained;
```

	product_id	product_name	manufacturer	product_count	price
	1	iPhone 13	Apple	1	115000.00
	2	iPhone 12	Apple	7	79900.00

Обновляемое представление с проверкой ограничений

Обновление представления

```
UPDATE apple_products_constrained
SET manufacturer = 'Pineapple'
WHERE product_id = 1;
```

Сообщение об ошибке:

Error Code: 1369. CHECK OPTION failed 'apple_products_constrained'

Вложенное обновляемое представление с проверкой ограничений

```
CREATE VIEW iphone_products AS
SELECT *
FROM apple_products
WHERE product_name LIKE 'iPhon%'
WITH CASCADED CHECK OPTION;
```

Выбор данных из представления

```
SELECT * FROM iphone_products;
```

	product_id	product_name	manufacturer	product_count	price
	1	iPhone 13	Apple	1	115000.00
	2	iPhone 12	Apple	7	79900.00

Вложенное обновляемое представление с проверкой ограничений

Обновление представления

```
UPDATE iphone_products
SET product_name = 'youPhone'
WHERE product_id = 1;
```

Сообщение об ошибке:
Error Code: 1369. CHECK OPTION failed 'iphone_products'

Обновление представления

```
UPDATE iphone_products
SET manufacturer = 'Pineapple'
WHERE product_id = 1;
```

Сообщение об ошибке:
Error Code: 1369. CHECK OPTION failed 'iphone_products'

Индексирование

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных.

Реляционная таблица может одновременно содержать несколько индексов:

- Индекс первичного ключа – гарантирует, что во входящих в состав первичного ключа столбцах будут храниться уникальные данные.
- Индексы внешних ключей – обеспечивают быстрый поиск записей во внешних таблицах для соединения с этими таблицами.
- Пользовательские индексы – ускоряют поиск данных и могут проверять данные на уникальность.

Индексирование

Возможности:

- ▲ Ускорение упорядочивания строк отношений
- ▲ Ускорение поиска данных по полному или частичному совпадению
- ▲ Ускорение соединения таблиц, связанных по первичному и внешнему ключам
- ▲ Возможность поддержки уникальности данных за счет защиты от ввода повторяющихся значений

Недостатки:

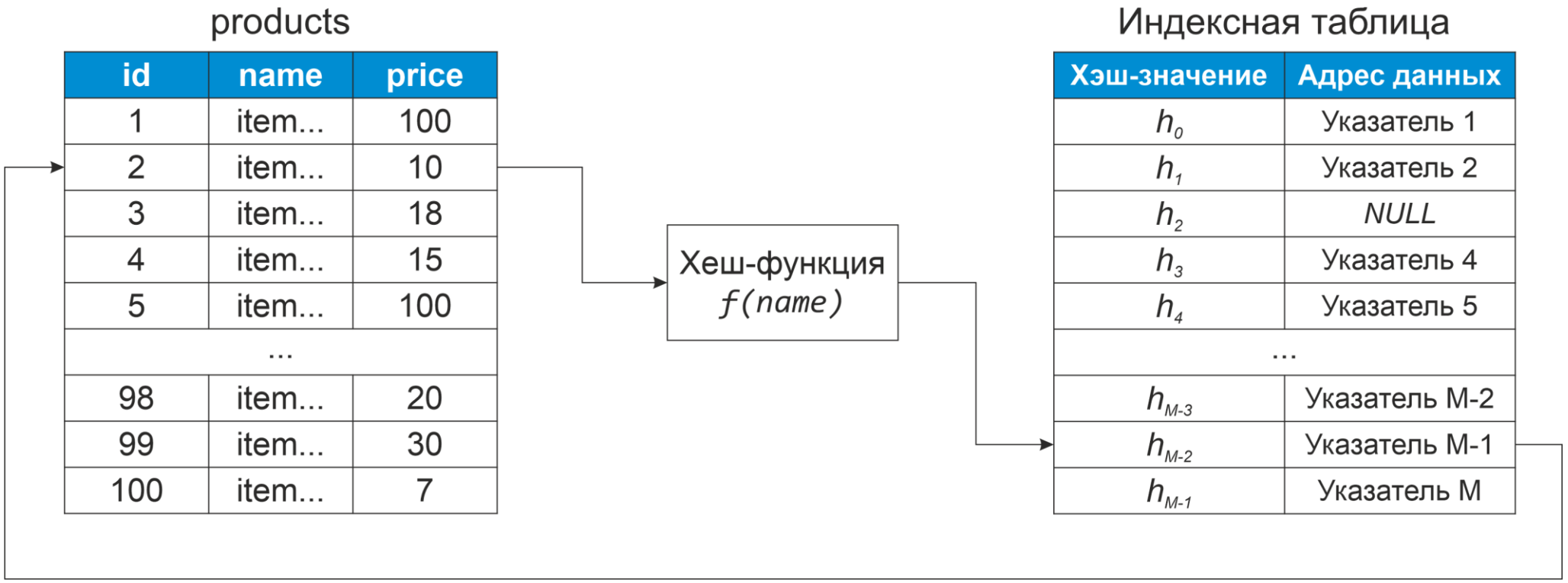
- ▼ Использование дополнительных структур данных, что увеличивает затраты на хранение индексов в памяти
- ▼ Увеличение времени выполнения запросов на добавление, изменение и удаление данных из-за необходимости обновления индекса после выполнения соответствующих запросов

Индексирование

Типы используемых индексов:

- Индексы, базирующиеся на технологии хеширования (hash).
- Индексы на основе сбалансированных B-деревьев (B-tree).
- Пространственные индексы (Spatial grid, R-tree).
- Битовые индексы.
- Полнотекстовые индексы.

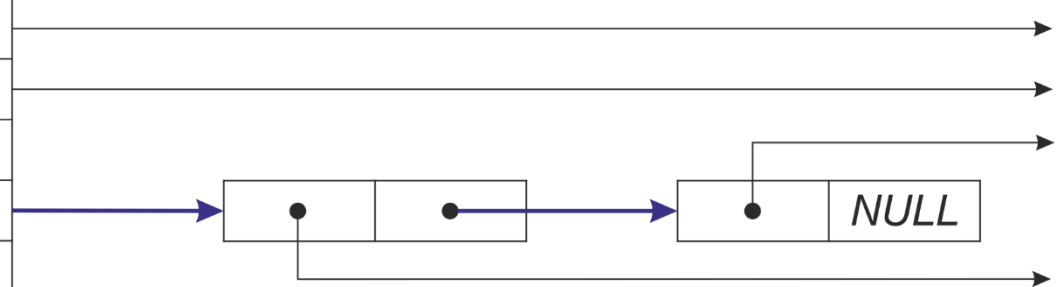
Индексы на основе хеширования



Индексы на основе хеширования

Индексная таблица

Хэш-значение	Адрес данных
h_0	Указатель 1
h_1	Указатель 2
h_2	*NULL*
h_3	Указатель 4
h_4	Указатель 5
…	
h_{M-3}	Указатель M-2
h_{M-2}	Указатель M-1
h_{M-1}	Указатель M



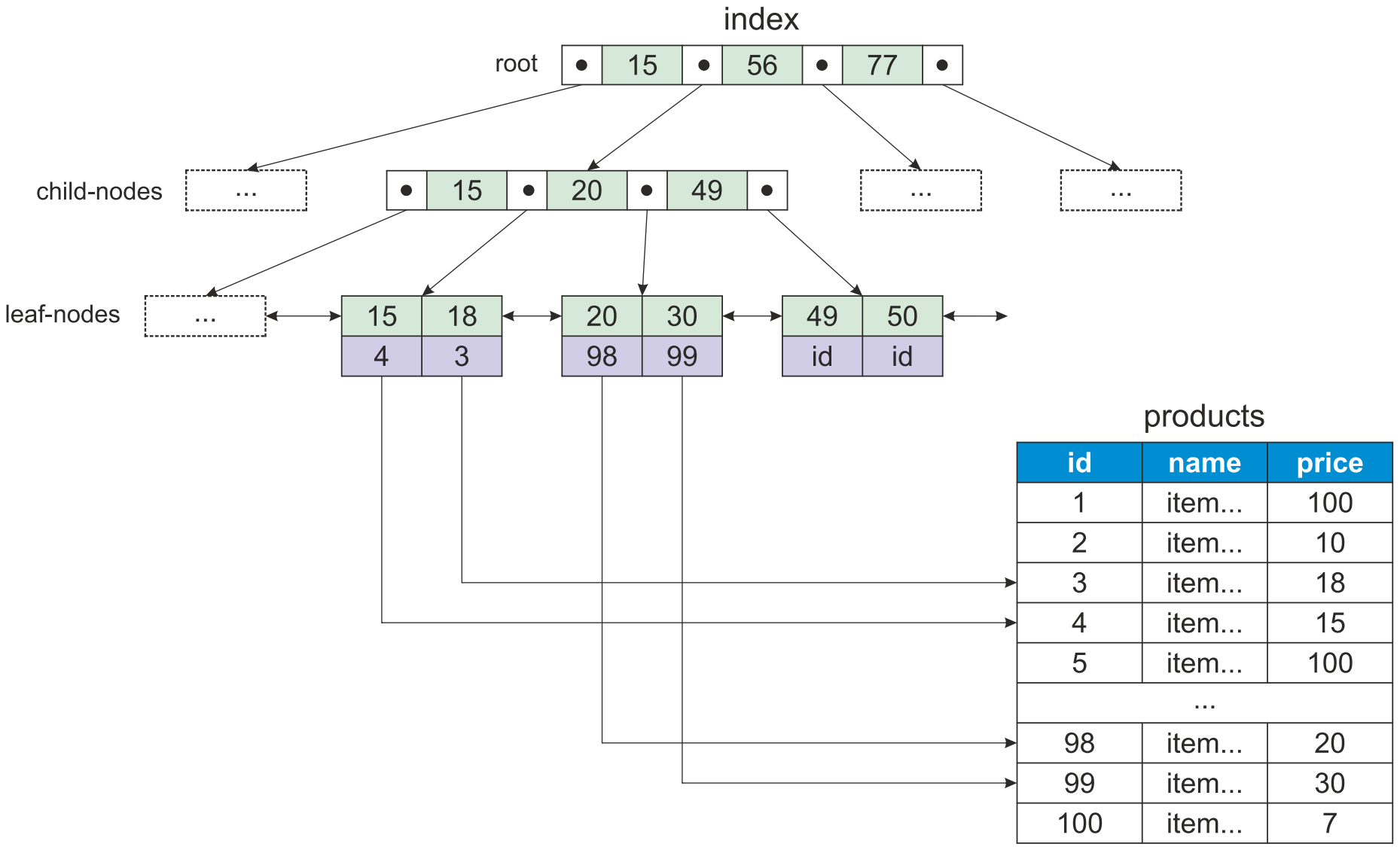
Индексы на основе хеширования

Особенности:

- ▲ Доступ к данным в хеш-индексе очень быстр, если нет большого количества коллизий.

- ▼ Нельзя использовать данные в индексе, чтобы избежать чтения строк.
- ▼ Нельзя использовать для сортировки, поскольку строки в нем не хранятся в отсортированном порядке.
- ▼ Хеш-индексы не поддерживают поиск по частичному ключу, так как хеш-коды вычисляются для всего индексируемого значения.
- ▼ Хеш-индексы поддерживают только сравнения на равенство.
- ▼ Некоторые операции обслуживания индекса могут оказаться медленными, если количество коллизий велико.

Индексы на основе B-деревьев



Индексы на основе B-деревьев

Особенности:

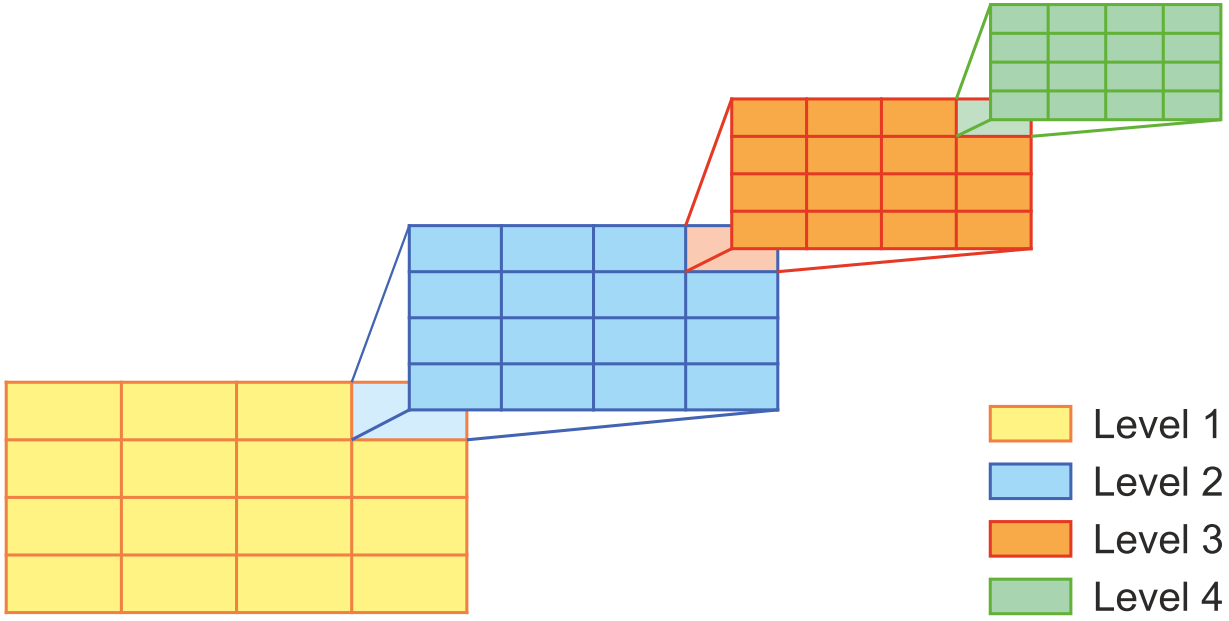
- ▲ Поиск по полному значению;
- ▲ Поиск по самому левому префиксу;
- ▲ Поиск по префиксу столбца;
- ▲ Поиск по диапазону значений;
- ▲ Поиск по полному совпадению одной части и диапазону в другой части;
- ▲ Запросы только по индексу.

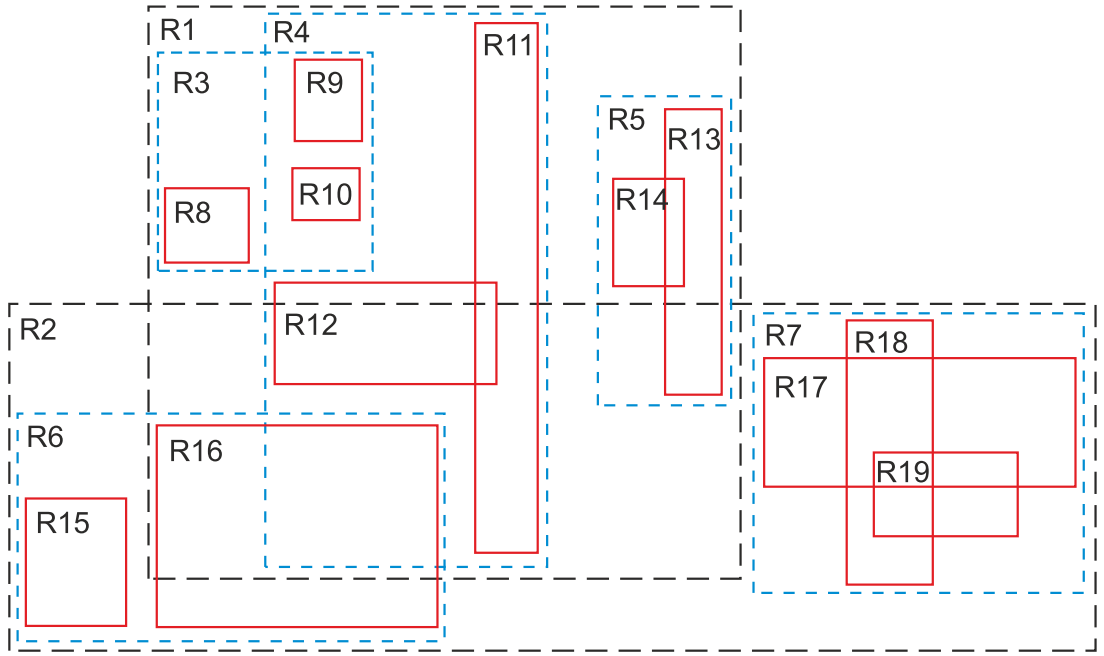
- ▼ Нельзя выполнить поиск без использования левой части ключа;
- ▼ Нельзя пропускать столбцы;
- ▼ Нельзя выполнить оптимизации после поиска в диапазоне;
- ▼ Большие затраты на перестроение индекса.

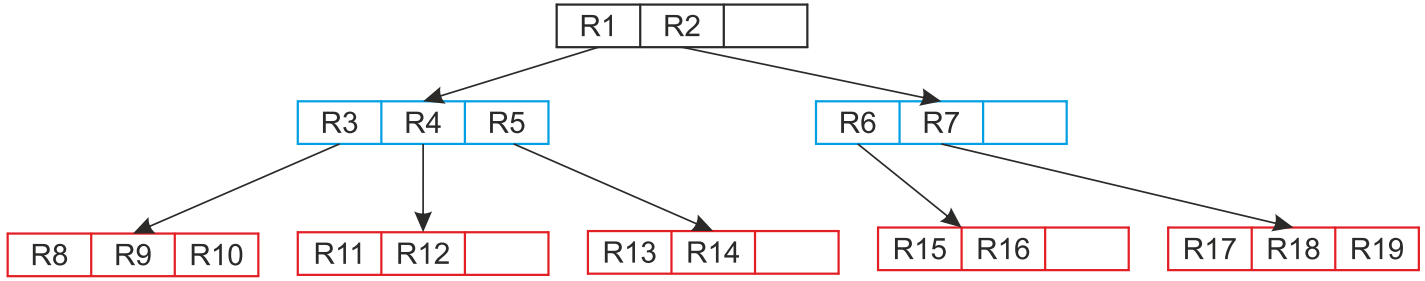
Пространственные индексы. Spatial grid

Пространственные данные представляют сведения о физическом расположении (координатах) и форме геометрических объектов.

Для пространственных типов данных существуют особые методы индексирования на основе сеток (grid-based spatial index: MS SQL Server) и R-деревьев(R-Tree index: MySQL, PostgreSQL).




R1
R4
R11
R3
R9
R5
R13
R10
R14
R8
R12
R2
R7
R18
R17
R6
R16
R19
R15


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

Битовые индексы

Данные

Id	last_name	first_name	gender
1	Орехов	Владимир	Мужской
2	Петровский	Александр	Мужской
3	Кульгина	Оксана	Женский
4	Самойлов	Евгений	Мужской
5	Кузьмина	Ирина	Женский

Битовые маски

value	first-id	bitmask
Женский	1	00101
Мужской	1	11010

Полнотекстовый индекс

Полнотекстовый поиск – поиск по содержимому текстовых документов больших объемов.

Полнотекстовый индекс — словарь, в котором перечислены все слова и указано, в каких местах документа (строках таблицы) встречаются. При наличии такого индекса достаточно осуществить поиск нужных слов в индексе и тогда сразу же будет получен список документов, в которых они встречаются.

Преимущества полнотекстового поиска по сравнению с оператором **`LIKE`**:

- поддержка морфологии;
- выдача результатов по релевантности;
- наличие модификаторов;
- стоп-слова.

Селективность индекса

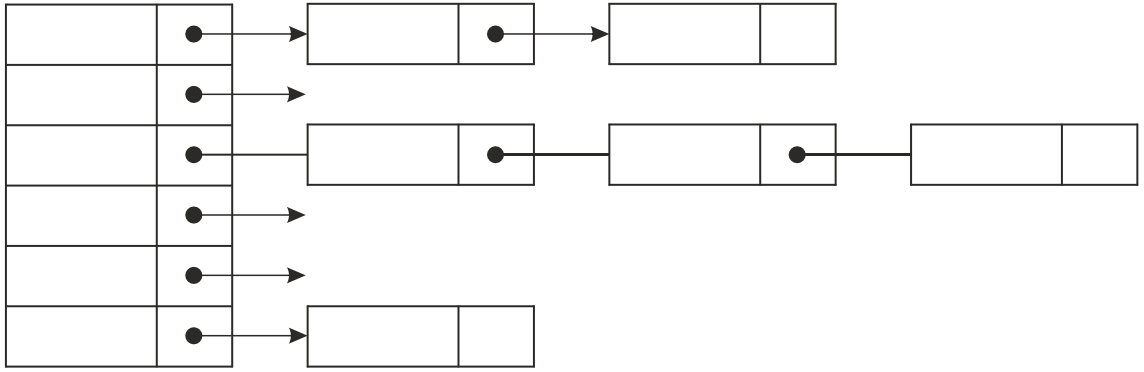
Под **избирательностью** (**селективностью**) индекса понимается число кортежей, которые могут быть выбраны по каждому значению индекса во время поиска.

Формула расчета селективности S:

$$S = \frac{1}{\text{число уникальных значений в столбце(ах)}}$$

Чем меньше результат, тем лучше с точки зрения оптимизатора. Например, в случае с уникальным индексом значение селективности стремится к 0. Здесь каждому значению индекса соответствует единственная запись в таблице, а повторяющихся ключей нет. Если же индекс не уникален и допускает хранение значений-дубликатов, то селективность падает. При самом плохом развитии ситуации (когда все значения ключей одинаковы) селективность стремится к 1.

Лист нижнего уровня B-дерева с цепочками дубликатов



Кластерный индекс

Кластерный индекс — это такой способ организации индекса, при котором значения индекса хранятся вместе с данными, им соответствующими. И индексы, и данные при такой организации упорядочены.

В MySQL (для движка InnoDB) кластерный индекс существует для всех таблиц и определяется автоматически одним из трёх способов:

- Если в таблице определен первичный ключ, то он и будет являться кластерным индексом.
- Иначе, если в таблице есть уникальные индексы, то кластерным будет первый из них.
- Иначе InnoDB самостоятельно создаёт скрытое поле с суррогатным ID размером в 6 байт, которое и будет кластерным индексом.

Рекомендации по созданию индексов

- В первую очередь следует индексировать столбцы отношений, которые наиболее часто используются в запросах на выборку данных.
- Последовательность столбцов в составном индексе должна соответствовать последовательности столбцов, следующих в инструкции **SELECT** сразу после директивы **ORDER BY** или **GROUP BY**.
- Наиболее эффективны индексы, накладываемые на столбцы, в которых не хранятся повторяющиеся значения.

- Важно помнить, что индексы предполагают дополнительные операции записи на диск. При каждом обновлении или добавлении данных в таблицу, происходит также обновление данных в индексе.
- Создавать следует только необходимые индексы, чтобы не расходовать зря ресурсы сервера.
- Не имеет смысла создавать индексы на таблицах, число записей в которых меньше нескольких тысяч. Для таких таблиц выигрыш от использования индекса будет почти незаметен.
- Не следует создавать индексы заранее. Индексы должны устанавливаться под форму и тип нагрузки работающей системы.

Создание / удаление индекса в MySQL

Создание индекса:

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
    USING {BTREE | HASH}
```

Удаление индекса:

```
DROP INDEX index_name ON tbl_name
```

Выполнение запроса без индекса

Выборка данных с информацией об артистах из БД discogs

```
SELECT ARTIST_ID, NAME, REALNAME, PROFILE, DATA_QUALITY
FROM artist
LIMIT 5;
```

	ARTIST_ID	NAME	REALNAME	PROFILE	DATA_QUALITY
	1	The Persuader	Jesper Dahlbäck		Needs Vote
	2	Mr. James Barth …	Cari Lekebusch …		Correct
	3	Josh Wink	Joshua Winkelman	After forming…	Needs Vote
	4	Johannes Heil	Johannes Heil	Electronic music producer...	Needs Vote
	5	Heiko Laux	Heiko Laux	German DJ and producer…	Needs Vote

```
SELECT COUNT(*) FROM artist;
```

	COUNT(*)
	6432265

Выполнение запроса без индекса

Выборка данных с информацией об артисте из БД discogs

```
SELECT * FROM artist
WHERE name = 'Linkin Park';
```

	ARTIST_ID	NAME	REALNAME	PROFILE	DATA_QUALITY
	40029	Linkin Park		Alternative Rock / Modern Rock band...	Needs Vote

Action output:
Duration / Fetch: **3.437** sec / 0.000 sec

Анализ запроса

```
EXPLAIN SELECT * FROM artist
WHERE name = 'Linkin Park';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	SIMPLE	artist	NULL	ALL	NULL	NULL	NULL	NULL	6289718	10.00	Using where

- **select_type**: тип запроса
- **table**: к какой таблице относится анализируемая строка
- **partition**: какой секции соответствуют данные из запроса. NULL, если секционирование таблицы не используется
- **type**: информация о том, каким образом MySQL выбирает данные из таблицы. ALL означает полное сканирование таблицы
- **possible_keys**: какие индексы могут использоваться для запроса
- **key**: индекс, который оптимизатор MySQL решил фактически использовать
- **key_len**: длина выбранного ключа (индекса) в байтах
- **ref**: показывает, какие столбцы или константы сравниваются с указанным в key индексом
- **rows**: (приблизительное) количество строк, которое, по мнению MySQL, будет прочитано
- **filtered**: оценка доли от общего количества числа просмотренных строк, которые вернет MySQL

Анализ запроса

```
EXPLAIN ANALYZE SELECT * FROM artist
WHERE name = 'Linkin Park';
```

```
 -> Filter: (artist.`NAME` = 'Linkin Park')  (cost=664370.03 rows=628972)
(actual time=31.022..3875.829 rows=1 loops=1)
     -> Table scan on artist  (cost=664370.03 rows=6289718) (actual
time=1.916..3436.848 rows=6432265 loops=1)
```

В дополнение к стоимости и количеству строк можно увидеть фактическое время (в мс) получения первой строки и фактическое время получения всех строк, которые выводятся в формате

```
actual time={время получения первой строки}..{время получения всех строк}
```

Также выводится значение `rows`, которое указывает на фактическое количество прочитанных строк. Значение `loops` — это количество циклов, которые будут выполнены для соединения с внешней таблицей (выше по дереву).

Создание индекса

```
CREATE INDEX idx_artist_name ON artist (NAME ASC);
```

Выборка данных с информацией об артисте из БД discogs

```
SELECT * FROM artist
WHERE name = 'Linkin Park';
```

	ARTIST_ID	NAME	REALNAME	PROFILE	DATA_QUALITY
	40029	Linkin Park		Alternative Rock / Modern Rock band...	Needs Vote

Action output:
Duration / Fetch: 0.000 sec / 0.000 sec

Анализ запроса

```
EXPLAIN SELECT * FROM artist
WHERE name = 'Linkin Park';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	SIMPLE	artist	NULL	ref	idx_artist_name	idx_artist_name	1022	const	1	100.0	

- **type**: ref – поиск по индексу, в результате которого возвращаются все строки, соответствующие единственному заданному значению
- **ref**: const – значения в индексе сравниваются с константой

Анализ запроса

```
EXPLAIN ANALYZE SELECT * FROM artist
WHERE name = 'Linkin Park';
```

```
 -> Index lookup on artist using idx_artist_name (NAME='Linkin Park')
(cost=1.10 rows=1) (actual time=0.027..0.030 rows=1 loops=1)
```

Выполнение запроса с использованием индекса

Выборка данных с информацией об артистах из БД discogs

```
SELECT * FROM artist
WHERE name LIKE 'Linkin%';
```

	ARTIST_ID	NAME	REALNAME	PROFILE	DATA_QUALITY
	4266594	Linkin			Needs Major Changes
	5805598	Linkin Gewargis			Needs Major Changes
	5381201	Linkin Hsu			Needs Major Changes
	40029	Linkin Park		Alternative Rock / Modern Rock band...	Needs Vote
	…	…	…	…	…

Action output:
Duration / Fetch: 0.000 sec / 0.000 sec

Анализ запроса

```
EXPLAIN SELECT * FROM artist
WHERE name LIKE 'Linkin%';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	SIMPLE	artist	NULL	ref	idx_artist_name	idx_artist_name	1022	NULL	10	100.0	Using index condition

```
EXPLAIN ANALYZE SELECT * FROM artist
WHERE name LIKE 'Linkin%';
```

```
-> Index range scan on artist using idx_artist_name over
('Linkin    ' <= NAME <= 'Linkin...'), with index condition: (artist.`NAME`
like 'Linkin%')  (cost=13.01 rows=10) (actual time=0.028..0.164 rows=10
loops=1)
```

Выполнение запроса с использованием индекса

Выборка данных с информацией об артистах из БД discogs

```
SELECT * FROM artist
WHERE name LIKE '%Linkin%';
```

	ARTIST_ID	NAME	REALNAME	PROFILE	DATA_QUALITY
	40029	Linkin Park		Alternative Rock / Modern Rock band...	Needs Vote
	606729	Weird Blinking Lights			Correct
	881065	Les Clinkingbeard			Needs Vote
	1836724	Tony Linkin		Management with [a2834476].	Needs Vote
	…	…	…	…	…

Action output:
Duration / Fetch: 4.062 sec / 0.000 sec

Анализ запроса

```
EXPLAIN SELECT * FROM artist
WHERE name LIKE '%Linkin%';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	SIMPLE	artist	NULL	ALL	NULL	NULL	NULL	NULL	6289718	11.11	Using where

```
EXPLAIN ANALYZE SELECT * FROM artist
WHERE name LIKE '%Linkin%';
```

```
-> Filter: (artist.`NAME` like '%Linkin%')  (cost=664382.83 rows=698788) (actual
time=31.699..4312.486 rows=36 loops=1)
    -> Table scan on artist  (cost=664382.83 rows=6289718) (actual
time=2.488..3309.592 rows=6432265 loops=1)
```

Выполнение запроса без индекса

Выборка данных с информацией о релизах из БД discogs

```
SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
LIMIT 5;
```

	RELEASE_ID	TITLE	COUNTRY	RELEASED	NOTES
	1	Stockholm	Sweden	1999-03-01	The song titles are the names of six of Stockholm…
	2	Knockin' Boots Vol 2 Of 2	Sweden	1998-06-01	All joints recorded in NYC (Dec.97).
	3	Profound Sounds Vol. 1	US	1999-07-13	1: Track title is given as "D2" (which is the side...
	4	Moving Cities	US	1999-11-02	US release that is missing the track "A Day To Go...
	5	Flowerhead	Germany	1995-01-15	Rather Interesting presents: Datacide "Flowe…

```
SELECT COUNT(*) FROM `release`;
```

	COUNT(*)
	11530540

Выполнение запроса без индекса

Выборка данных с информацией о релизах по стране и году

```
SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE COUNTRY = 'Russia' and YEAR(RELEASED) = 2010;
```

	RELEASE_ID	TITLE	COUNTRY	RELEASED	NOTES
	5858308	Live Demo'10	Russia	2017-02-23	Issued in paper envelopes with photos and inse…
	6004701	Пожалей Моё Желе	Russia	2017-12-15	
	6152802	Апостроф	Russia	2017-02-15	Pressing info - 395 records total.74 - Black Vinyl …
	6153124	Апостроф	Russia	2017-02-15	Pressing info - 395 records total.74 - Black Vinyl…
	…	…	…	…	…

Action output:
Duration / Fetch: **5.360** sec / 0.156 sec

Создание индекса

```
CREATE INDEX idx_country_released ON `release` (COUNTRY, RELEASED);
```

Выборка данных с информацией о релизах

```
EXPLAIN ANALYZE SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE COUNTRY = 'Russia' and YEAR(RELEASED) = 2010;
```

```
-> Index lookup on release using idx_country_released (COUNTRY='Russia'), with
index condition: (year(`release`.RELEASED) = 2017)
(cost=442716.80 rows=446216) (actual time=38.979..1017.941 rows=10358 loops=1)
```

Создание функционального индекса

```
CREATE INDEX idx_country_year_released
ON `release` (COUNTRY, (YEAR(RELEASED)));
```

Выборка данных с информацией о релизах

```
EXPLAIN ANALYZE SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE COUNTRY = 'Russia' and YEAR(RELEASED) = 2010;
```

```
 -> Index lookup on release using idx_country_year_released
(COUNTRY='Russia', year(RELEASED)=2017)
(cost=21359.70 rows=19420) (actual time=0.303..684.493 rows=10358 loops=1)
```


Анализ запроса

```
EXPLAIN SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE COUNTRY = 'Greenland';
```

	id	select_type	table	partitions	type	possible_keys	key	…
	1	SIMPLE	release	NULL	ALL	idx_country_released, idx_country_year_released	idx_country_released	…

```
EXPLAIN ANALYZE SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE COUNTRY = 'Greenland';
```

```
 -> Index lookup on release using idx_country_released (COUNTRY='Greenland')
(cost=259.46 rows=236) (actual time=20.518..39.410 rows=236 loops=1)
```

Анализ запроса

```
EXPLAIN SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE YEAR(RELEASED) = 2010;
```

	id	select_type	table	partitions	type	possible_keys	key	…
	1	SIMPLE	release	NULL	ALL	NULL	NULL	…

```
EXPLAIN ANALYZE SELECT RELEASE_ID, TITLE, COUNTRY, RELEASED, NOTES
FROM `release`
WHERE YEAR(RELEASED) = 2010;
```

```
 -> Filter: (year(RELEASED) = 2010)  (cost=1341009.83 rows=1208116) (actual
time=135.507..6510.040 rows=247652 loops=1)
     -> Table scan on release  (cost=1341009.83 rows=12081161) (actual
time=0.694..6093.392 rows=11530540 loops=1)
```

Полнотекстовый поиск в MySQL выполняется с помощью конструкции
`MATCH(fields) … AGAINST(words)`

```
CREATE FULLTEXT INDEX idx_fulltext_released ON `release` (NOTES);
```

Выборка данных с информацией о релизе по описанию

```
SELECT RELEASE_ID, NOTES
FROM `release`
WHERE MATCH (NOTES) AGAINST ('"Linkin Park" "Hybrid Theory"');
```

	RELEASE_ID	NOTES
	9775825	(C) 2002 Printed in USATracks 1 to 6 from [url=https://www.discogs.com/Hybrid-Theory-Hybrid-Theory/release/1129272]Hybrid Theory EP[/url]Tracks 7 to 15 live from Orlando, FL aug, 16 2000Tracks 16 to 21 live from KROQ Almost Christmas dec, 16 2000Track 22...
	1882259	An official collection of favourites released through the Linkin Park fan club. The CD includes a Linkin Park Underground one month membership redemption card and a free download of a bonus track.Tracks 3 and 8 originally appear on the Hybrid Theory EP...
	…	…

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

БЛАГОДАРЮ ЗА ВНИМАНИЕ

Агафонов А.А.
к.т.н., доцент кафедры ГИиИБ