



 enhorse / java-interview Public[Code](#) [Issues 9](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) master ▾

...

[java-interview](#) / db.md

KhArtNJava Update db.md

 History 3 contributors 283 lines (194 sloc) | 42 KB

...

[Вопросы для собеседования](#)

Базы данных

- Что такое «база данных»?
- Что такое «система управления базами данных»?
- Что такое «реляционная модель данных»?
- Дайте определение терминам «простой», «составной» (composite), «потенциальный» (candidate) и «альтернативный» (alternate) ключ.
- Что такое «первичный ключ» (primary key)? Каковы критерии его выбора?
- Что такое «внешний ключ» (foreign key)?
- Что такое «нормализация»?
- Какие существуют нормальные формы?
- Что такое «денормализация»? Для чего она применяется?
- Какие существуют типы связей в базе данных? Приведите примеры.
- Что такое «индексы»? Для чего их используют? В чём заключаются их преимущества и недостатки?
- Какие типы индексов существуют?
- В чем отличие между кластерными и некластерными индексами?

- Имеет ли смысл индексировать данные, имеющие небольшое количество возможных значений?
- Когда полное сканирование набора данных выгоднее доступа по индексу?
- Что такое «транзакция»?
- Назовите основные свойства транзакции.
- Какие существуют уровни изолированности транзакций?
- Какие проблемы могут возникать при параллельном доступе с использованием транзакций?

Что такое «база данных»?

База данных — организованный и адаптированный для обработки вычислительной системой набор информации.

[к оглавлению](#)

Что такое «система управления базами данных»?

Система управления базами данных (СУБД) - набор средств общего или специального назначения, обеспечивающий создание, доступ к материалам и управление базой данных.

Основные функции СУБД:

- управление данными
- журнализация изменений данных
- резервное копирование и восстановление данных;
- поддержка языка определения данных и манипулирования ими.

[к оглавлению](#)

Что такое «реляционная модель данных»?

Реляционная модель данных — это логическая модель данных и прикладная теория построения реляционных баз данных.

Реляционная модель данных включает в себя следующие компоненты:

- *Структурный аспект* — данные представляют собой набор отношений.

- *Аспект целостности* — отношения отвечают определенным условиям целостности: уровня домена (типа данных), уровня отношения и уровня базы данных.
- *Аспект обработки (манипулирования)* — поддержка операторов манипулирования отношениями (реляционная алгебра, реляционное исчисление).
- *Нормальная форма* - свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности и определённое как совокупность требований, которым должно удовлетворять отношение.

[к оглавлению](#)

Дайте определение терминам «простой», «составной» (composite), «потенциальный» (candidate) и «альтернативный» (alternate) ключ.

Простой ключ состоит из одного атрибута (поля). **Составной** - из двух и более.

Потенциальный ключ - простой или составной ключ, который уникально идентифицирует каждую запись набора данных. При этом потенциальный ключ должен обладать критерием неизбыточности: при удалении любого из полей набор полей перестаёт уникально идентифицировать запись.

Из множества всех потенциальных ключей набора данных выбирают первичный ключ, все остальные ключи называют **альтернативными**.

[к оглавлению](#)

Что такое «первичный ключ» (primary key)? Каковы критерии его выбора?

Первичный ключ (primary key) в реляционной модели данных один из *потенциальных ключей* отношения, выбранный в качестве основного ключа (ключа по умолчанию).

Если в отношении имеется единственный потенциальный ключ, он является и первичным ключом. Если потенциальных ключей несколько, один из них выбирается в качестве первичного, а другие называют *«альтернативными»*.

В качестве первичного обычно выбирается тот из потенциальных ключей, который наиболее удобен. Поэтому в качестве первичного ключа, как правило, выбирают тот, который имеет наименьший размер (физического хранения) и/или включает наименьшее количество атрибутов. Другой критерий выбора первичного ключа — сохранение его уникальности со временем. Поэтому в качестве первичного ключа стараются выбирать такой потенциальный ключ, который с наибольшей вероятностью никогда не утратит уникальность.

[к оглавлению](#)

Что такое «внешний ключ» (*foreign key*)?

Внешний ключ (*foreign key*) — подмножество атрибутов некоторого отношения А, значения которых должны совпадать со значениями некоторого потенциального ключа некоторого отношения В.

[к оглавлению](#)

Что такое «нормализация»?

Нормализация - это процесс преобразования отношений базы данных к виду, отвечающему нормальным формам (пошаговый, обратимый процесс замены исходной схемы другой схемой, в которой наборы данных имеют более простую и логичную структуру).

Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в базе данных информации.

[к оглавлению](#)

Какие существуют нормальные формы?

Первая нормальная форма (1NF) - Отношение находится в 1NF, если значения всех его атрибутов атомарны (неделимы).

Вторая нормальная форма (2NF) - Отношение находится в 2NF, если оно находится в 1NF, и при этом все неключевые атрибуты зависят только от ключа целиком, а не от какой-то его части.

Третья нормальная форма (3NF) - Отношение находится в 3NF, если оно находится в 2NF и все неключевые атрибуты не зависят друг от друга.

Четвёртая нормальная форма (4NF) - Отношение находится в 4NF, если оно находится в 3NF и если в нем не содержатся независимые группы атрибутов, между которыми существует отношение «многие-ко-многим».

Пятая нормальная форма (5NF) - Отношение находится в 5NF, когда каждая нетривиальная зависимость соединения в ней определяется потенциальным ключом (ключами) этого отношения.

Шестая нормальная форма (6NF) - Отношение находится в 6NF, когда она удовлетворяет всем нетривиальным зависимостям соединения, т.е. когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6NF, также находится и в 5NF. Введена как обобщение пятой нормальной формы для хронологической базы данных.

Нормальная форма Бойса-Кодда, усиленная 3 нормальной формой (BCNF) - Отношение находится в BCNF, когда каждая её нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ.

Доменно-ключевая нормальная форма (DKNF) - Отношение находится в DKNF, когда каждое наложенное на неё ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данное отношение.

[к оглавлению](#)

Что такое «денормализация»? Для чего она применяется?

Денормализация базы данных — это процесс осознанного приведения базы данных к виду, в котором она не будет соответствовать правилам нормализации. Обычно это необходимо для повышения производительности и скорости извлечения данных, за счет увеличения избыточности данных.

[к оглавлению](#)

Какие существуют типы связей в базе данных? Приведите примеры.

- **Один к одному** - любому значению атрибута A соответствует только одно значение атрибута B, и наоборот.

Каждый университет гарантированно имеет 1-го ректора: *1 университет → 1 ректор*.

- **Один ко многим** - любому значению атрибута A соответствует 0, 1 или несколько значений атрибута B.

В каждом университете есть несколько факультетов: *1 университет → много факультетов*.

- **Многие ко многим** - любому значению атрибута A соответствует 0, 1 или несколько значений атрибута B, и любому значению атрибута B соответствует 0, 1 или несколько значений атрибута A.

1 профессор может преподавать на нескольких факультетах, в то же время на 1-ом факультете может преподавать несколько профессоров: *Несколько профессоров ↔ Несколько факультетов*.

[к оглавлению](#)

Что такое «индексы»? Для чего их используют? В чём заключаются их преимущества и недостатки?

Индекс (index) — объект базы данных, создаваемый с целью повышения производительности выборки данных.

Наборы данных могут иметь большое количество записей, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра набора данных запись за записью может занимать много времени. **Индекс** формируется из значений одного или нескольких полей и указателей на соответствующие записи набора данных, - таким образом, достигается значительный прирост скорости выборки из этих данных.

Преимущества

- ускорение поиска и сортировки по определенному полю или набору полей.

- обеспечение уникальности данных.

Недостатки

- требование дополнительного места на диске и в оперативной памяти и чем больше/длиннее ключ, тем больше размер индекса.
- замедление операций вставки, обновления и удаления записей, поскольку при этом приходится обновлять сами индексы.

Индексы предпочтительней для:

- Поля-счетчика, чтобы в том числе избежать и повторения значений в этом поле;
- Поля, по которому проводится сортировка данных;
- Полей, по которым часто проводится соединение наборов данных. Поскольку в этом случае данные располагаются в порядке возрастания индекса и соединение происходит значительно быстрее;
- Поля, которое объявлено первичным ключом (primary key);
- Поля, в котором данные выбираются из некоторого диапазона. В этом случае как только будет найдена первая запись с нужным значением, все последующие значения будут расположены рядом.

Использование индексов нецелесообразно для:

- Полей, которые редко используются в запросах;
- Полей, которые содержат всего два или три значения, например: *мужской*, *женский пол* или значения «да», «нет».

[к оглавлению](#)

Какие типы индексов существуют?

По порядку сортировки

- **упорядоченные** — индексы, в которых элементы упорядочены;
- **возрастающие**;
- **убывающие**;
- **неупорядоченные** — индексы, в которых элементы неупорядочены.

По источнику данных

- *индексы по представлению (view)*;
- *индексы по выражениям.*

По воздействию на источник данных

- *кластерный индекс* - при определении в наборе данных физическое расположение данных перестраивается в соответствии со структурой индекса. Логическая структура набора данных в этом случае представляет собой скорее словарь, чем индекс. Данные в словаре физически упорядочены, например по алфавиту. Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными индексами. Увеличение производительности особенно заметно при работе с последовательными данными.
- *некластерный индекс* — наиболее типичные представители семейства индексов. В отличие от кластерных, они не перестраивают физическую структуру набора данных, а лишь организуют ссылки на соответствующие записи. Для идентификации нужной записи в наборе данных некластерный индекс организует специальные указатели, включающие в себя: информацию об идентификационном номере файла, в котором хранится запись; идентификационный номер страницы соответствующих данных; номер искомой записи на соответствующей странице; содержимое столбца.

По структуре

- *B*-деревья*;
- *B+-деревья*;
- *B-деревья*;
- *Хэши*.

По количественному составу

- *простой индекс (индекс с одним ключом)* — строится по одному полю;
- *составной (многоключевой, композитный) индекс* — строится по нескольким полям при этом важен порядок их следования;
- *индекс с включенными столбцами* — некластеризованный индекс, дополнительно содержащий кроме ключевых столбцов еще и неключевые;
- *главный индекс (индекс по первичному ключу)* — это тот индексный ключ, под управлением которого в данный момент находится набор данных. Набор данных не может быть отсортирован по нескольким индексным ключам одновременно. Хотя, если один и тот же набор данных открыт одновременно

в нескольких рабочих областях, то у каждой копии набора данных может быть назначен свой главный индекс.

По характеристике содержимого

- **уникальный индекс** состоит из множества уникальных значений поля;
- **плотный индекс** (NoSQL) — индекс, при котором, каждом документе в индексируемой коллекции соответствует запись в индексе, даже если в документе нет индексируемого поля.
- **разреженный индекс** (NoSQL) — тот, в котором представлены только те документы, для которых индексируемый ключ имеет какое-то определённое значение (существует).
- **пространственный индекс** — оптимизирован для описания географического местоположения. Представляет из себя многоключевой индекс состоящий из широты и долготы.
- **составной пространственный индекс** — индекс, включающий в себя кроме широты и долготы ещё какие-либо мета-данные (например теги). Но географические координаты должны стоять на первом месте.
- **полнотекстовый (инвертированный) индекс** — словарь, в котором перечислены все слова и указано, в каких местах они встречаются. При наличии такого индекса достаточно осуществить поиск нужных слов в нём и тогда сразу же будет получен список документов, в которых они встречаются.
- **хэш-индекс** предполагает хранение не самих значений, а их хэшей, благодаря чему уменьшается размер (а, соответственно, и увеличивается скорость их обработки) индексов из больших полей. Таким образом, при запросах с использованием хэш-индексов, сравниваться будут не искомое со значения поля, а хэш от искомого значения с хэшами полей. Из-за нелинейности хэш-функций данный индекс нельзя сортировать по значению, что приводит к невозможности использования в сравнениях больше/меньше и «is null». Кроме того, так как хэши не уникальны, то для совпадающих хэшей применяются методы разрешения коллизий.
- **битовый индекс (bitmap index)** — метод битовых индексов заключается в создании отдельных битовых карт (последовательностей 0 и 1) для каждого возможного значения столбца, где каждому биту соответствует запись с индексируемым значением, а его значение равное 1 означает, что запись, соответствующая позиции бита содержит индексируемое значение для данного столбца или свойства.
- **обратный индекс (reverse index)** — B-tree индекс, но с реверсированным ключом, используемый в основном для монотонно возрастающих значений

(например, автоинкрементный идентификатор) в OLTP системах с целью снятия конкуренции за последний листовой блок индекса, т.к. благодаря переворачиванию значения две соседние записи индекса попадают в разные блоки индекса. Он не может использоваться для диапазонного поиска.

- **функциональный индекс**, индекс по вычисляемому полю (*function-based index*) — индекс, ключи которого хранят результат пользовательских функций. Функциональные индексы часто строятся для полей, значения которых проходят предварительную обработку перед сравнением в команде SQL. Например, при сравнении строковых данных без учета регистра символов часто используется функция UPPER. Кроме того, функциональный индекс может помочь реализовать любой другой отсутствующий тип индексов данной СУБД.
- **первичный индекс** — уникальный индекс по полю первичного ключа.
- **вторичный индекс** — индекс по другим полям (кроме поля первичного ключа).
- **XML-индекс** — вырезанное материализованное представление больших двоичных XML-объектов (BLOB) в столбце с типом данных xml.

По механизму обновления

- **полностью перестраиваемый** — при добавлении элемента заново перестраивается весь индекс.
- **пополняемый (балансируемый)** — при добавлении элементов индекс перестраивается частично (например, одна из ветви) и периодически балансируется.

По покрытию индексируемого содержимого

- **полностью покрывающий (полный) индекс** — покрывает всё содержимое индексируемого объекта.
- **частичный индекс (partial index)** — это индекс, построенный на части набора данных, удовлетворяющей определенному условию самого индекса. Данный индекс создан для уменьшения размера индекса.
- **инкрементный (delta) индекс** — индексируется малая часть данных (дельта), как правило, по истечении определённого времени. Используется при интенсивной записи. Например, полный индекс перестраивается раз в сутки, а дельта-индекс строится каждый час. По сути это частичный индекс по временной метке.
- **индекс реального времени (real-time index)** — особый вид инкрементного

индекса, характеризующийся высокой скоростью построения. Предназначен для часто меняющихся данных.

Индексы в кластерных системах

- **глобальный индекс** — индекс по всему содержимому всех сегментов БД (shard).
- **сегментный индекс** — глобальный индекс по полю-сегментируемому ключу (shard key). Используется для быстрого определения сегмента, на котором хранятся данные в процессе маршрутизации запроса в кластере БД.
- **локальный индекс** — индекс по содержимому только одного сегмента БД.

[к оглавлению](#)

В чем отличие между кластерными и некластерными индексами?

Некластерные индексы - данные физически расположены в произвольном порядке, но логически упорядочены согласно индексу. Такой тип индексов подходит для часто изменяемого набора данных.

При кластерном индексировании данные физически упорядочены, что серьезно повышает скорость выборки данных (но только в случае последовательного доступа к данным). Для одного набора данных может быть создан только один кластерный индекс.

[к оглавлению](#)

Имеет ли смысл индексировать данные, имеющие небольшое количество возможных значений?

Примерное правило, которым можно руководствоваться при создании индекса - если объем информации (в байтах) НЕ удовлетворяющей условию выборки меньше, чем размер индекса (в байтах) по данному условию выборки, то в общем случае оптимизация приведет к замедлению выборки.

[к оглавлению](#)

Когда полное сканирование набора данных выгоднее доступа по индексу?

Полное сканирование производится многоблочным чтением. Сканирование по индексу - одноблочным. Также, при доступе по индексу сначала идет сканирование самого индекса, а затем чтение блоков из набора данных. Число блоков, которые надо при этом прочитать из набора зависит от фактора кластеризации. Если суммарная стоимость всех необходимых одноблочных чтений больше стоимости полного сканирования многоблочным чтением, то полное сканирование выгоднее, и оно выбирается оптимизатором.

Таким образом, полное сканирование выбирается при слабой селективности предикатов запроса и/или слабой кластеризации данных, либо в случае очень маленьких наборов данных.

[к оглавлению](#)

Что такое «транзакция»?

Транзакция - это воздействие на базу данных, переводящее её из одного целостного состояния в другое и выражаемое в изменении данных, хранящихся в базе данных.

[к оглавлению](#)

Назовите основные свойства транзакции.

Атомарность (atomicity) гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной.

Согласованность (consistency). Транзакция, достигая своего нормального завершения и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных.

Изолированность (isolation). Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат.

Долговечность (durability). Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбой в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу.

[к оглавлению](#)

Какие существуют уровни изолированности транзакций?

В порядке увеличения изолированности транзакций и, соответственно, надёжности работы с данными:

- **Чтение неподтверждённых данных (грязное чтение) (read uncommitted, dirty read)** — чтение незафиксированных изменений как своей транзакции, так и параллельных транзакций. Нет гарантии, что данные, изменённые другими транзакциями, не будут в любой момент изменены в результате их отката, поэтому такое чтение является потенциальным источником ошибок. Невозможны потерянные изменения, возможны неповторяемое чтение и фантомы.
- **Чтение подтверждённых данных (read committed)** — чтение всех изменений своей транзакции и зафиксированных изменений параллельных транзакций. Потерянные изменения и грязное чтение не допускается, возможны неповторяемое чтение и фантомы.
- **Повторяемость чтения (repeatable read, snapshot)** — чтение всех изменений своей транзакции, любые изменения, внесённые параллельными транзакциями после начала своей, недоступны. Потерянные изменения, грязное и неповторяемое чтение невозможны, возможны фантомы.
- **Упорядочиваемость (serializable)** — результат параллельного выполнения сериализуемой транзакции с другими транзакциями должен быть логически эквивалентен результату их какого-либо последовательного выполнения. Проблемы синхронизации не возникают.

[к оглавлению](#)

Какие проблемы могут возникать при параллельном доступе с использованием транзакций?

При параллельном выполнении транзакций возможны следующие проблемы:

- **Потерянное обновление (lost update)** — при одновременном изменении одного блока данных разными транзакциями одно из изменений теряется;
- **«Грязное» чтение (dirty read)** — чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится);
- **Неповторяющееся чтение (non-repeatable read)** — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;
- **Фантомное чтение (phantom reads)** — одна транзакция в ходе своего выполнения несколько раз выбирает множество записей по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет или удаляет записи или изменяет столбцы некоторых записей, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества записей. Предположим, имеется две транзакции, открытые различными приложениями, в которых выполнены следующие SQL-операторы:

Транзакция 1	Транзакция 2
	SELECT SUM(f2) FROM tbl1;
INSERT INTO tbl1 (f1,f2) VALUES (15,20);	
COMMIT;	
	SELECT SUM(f2) FROM tbl1;

В транзакции 2 выполняется SQL-оператор, использующий все значения поля f2. Затем в транзакции 1 выполняется вставка новой строки, приводящая к тому, что повторное выполнение SQL-оператора в транзакции 2 выдаст другой результат. Такая ситуация называется чтением фантома (фантомным чтением). От неповторяющегося чтения оно отличается тем, что результат повторного обращения к данным изменился не из-за изменения/удаления самих этих данных, а из-за появления новых (фантомных) данных.

[к оглавлению](#)

Источники

- [Википедия](#)

- tokarchuk.ru
- [Quizful](#)

[Вопросы для собеседования](#)