**SQL**

[*Методичка по всем разделам, автор неизвестен (:*](https://docs.google.com/spreadsheets/d/1owGOr08-YSywk3KVIAdMLX8_tZyvPm9UJWubEeOcgD8/htmlview#)

*вопросы-ответы Хилькевич Игорь 16.04.2020*

Вопросы из “all question”

1. Операторы DDL, DML, TCL, DCL.

2. null в SQL.

3. Временная таблица (temporary table) vs. представления (view).

4. SELECT.

5. Виды join'ов. JOIN vs. подзапросы (sub-query).

6. HAVING vs. WHERE.

7. ORDER BY.

8. GROUP BY (NULL).

9. GROUP BY vs. DISTINCT.

10. Агрегатные функции. COUNT(\*) vs. COUNT({column}).

11. EXISTS.

12. IN, BETWEEN, LIKE.

13. UNION vs. UNIONALL (ограничение на UNION).

14. Ограничения (constraints: primary key, foreign key, check, notnull, unique).

15. Суррогатные ключи.

16. MERGE.

17. DELETE vs. TRUNCATE.

18. Индексы.

19. Хранимая процедура.

20. Триггер.

21. Курсор.

22. DATETIME vs. TIMESTAMP.

23. Транзакции. Принципы ACID.

24. Уровни изолированности транзакций.

25. Нормализация. Нормальные формы. Денормализация.

**Ответы**

1. Операторы DDL, DML, TCL, DCL.

**Какие существуют операторы SQL?**

операторы определения данных (Data Definition Language, DDL):

* CREATE создает объект БД (базу, таблицу, представление, пользователя и т. д.),
* ALTER изменяет объект,
* DROP удаляет объект;

операторы манипуляции данными (Data Manipulation Language, DML):

* SELECT выбирает данные, удовлетворяющие заданным условиям,
* INSERT добавляет новые данные,
* UPDATE изменяет существующие данные,
* DELETE удаляет данные;

операторы определения доступа к данным (Data Control Language, happens-before):

* GRANT предоставляет пользователю (группе) разрешения на определенные операции с объектом,
* REVOKE отзывает ранее выданные разрешения,
* DENY задает запрет, имеющий приоритет над разрешением;

операторы управления транзакциями (Transaction Control Language, TCL):

* COMMIT применяет транзакцию,
* ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции,
* SAVEPOINT разбивает транзакцию на более мелкие.

2. null в SQL.

**Что означает NULL в SQL?**

NULL - специальное значение (псевдозначение), которое может быть записано в поле таблицы базы данных. NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения».

NULL означает отсутствие, неизвестность информации. Значение NULL не является значением в полном смысле слова: по определению оно означает отсутствие значения и не принадлежит ни одному типу данных. Поэтому NULL не равно ни логическому значению FALSE, ни пустой строке, ни 0. При сравнении NULL с любым значением будет получен результат NULL, а не FALSE и не 0. Более того, NULL не равно NULL!

команды: IS NULL, IS NOT NULL

3. Временная таблица (temporary table) vs. представления (view).

**Что такое «временная таблица»? Для чего она используется?**

Временная таблица - это объект базы данных, который хранится и управляется системой базы данных на временной основе. Они могут быть локальными или глобальными. Используется для сохранения результатов вызова хранимой процедуры, уменьшение числа строк при соединениях, агрегирование данных из различных источников или как замена курсоров и параметризованных представлений.

**Что такое «представление» (view) и для чего оно применяется?**

Представление, View - виртуальная таблица, представляющая данные одной или более таблиц альтернативным образом.

В действительности представление – всего лишь результат выполнения оператора SELECT, который хранится в структуре памяти, напоминающей SQL таблицу. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления значительно расширяют возможности управления данными. Это способ дать публичный доступ к некоторой (но не всей) информации в таблице.

4. SELECT.

**Каков общий синтаксис оператора SELECT?**

SELECT - оператор DML SQL, возвращающий набор данных (выборку) из базы данных, удовлетворяющих заданному условию. Имеет следующую структуру:

SELECT

[DISTINCT | DISTINCTROW | ALL]

select\_expression,...

FROM table\_references

[WHERE where\_definition]

[GROUP BY {unsigned\_integer | column | formula}]

[HAVING where\_definition]

[ORDER BY {unsigned\_integer | column | formula} [ASC | DESC], ...]

5. Виды join'ов. JOIN vs. подзапросы (sub-query).

**Что такое JOIN?**

JOIN - оператор языка SQL, который является реализацией операции соединения реляционной алгебры. Предназначен для обеспечения выборки данных из двух таблиц и включения этих данных в один результирующий набор.

Особенностями операции соединения являются следующее:

в схему таблицы-результата входят столбцы обеих исходных таблиц (таблиц-операндов), то есть схема результата является «сцеплением» схем операндов;

каждая строка таблицы-результата является «сцеплением» строки из одной таблицы-операнда со строкой второй таблицы-операнда;

при необходимости соединения не двух, а нескольких таблиц, операция соединения применяется несколько раз (последовательно).

SELECT

field\_name [,... n]

FROM

Table1

{INNER | {LEFT | RIGHT | FULL} OUTER | CROSS } JOIN

Table2

{ON <condition> | USING (field\_name [,... n])}

**Какие существуют типы JOIN?**

* (INNER) JOIN Результатом объединения таблиц являются записи, общие для левой и правой таблиц. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.
* LEFT (OUTER) JOIN Производит выбор всех записей первой таблицы и соответствующих им записей второй таблицы. Если записи во второй таблице не найдены, то вместо них подставляется пустой результат (NULL). Порядок таблиц для оператора важен, поскольку оператор не является симметричным.
* RIGHT (OUTER) JOIN LEFT JOIN с операндами, расставленными в обратном порядке. Порядок таблиц для оператора важен, поскольку оператор не является симметричным.
* FULL (OUTER) JOIN Результатом объединения таблиц являются все записи, которые присутствуют в таблицах. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.
* CROSS JOIN (декартово произведение) При выборе каждая строка одной таблицы объединяется с каждой строкой второй таблицы, давая тем самым все возможные сочетания строк двух таблиц. Порядок таблиц для оператора не важен, поскольку оператор является симметричным.

**Что лучше использовать JOIN или подзапросы?**

Обычно лучше использовать JOIN, поскольку в большинстве случаев он более понятен и лучше оптимизируется СУБД (но 100% этого гарантировать нельзя). Так же JOIN имеет заметное преимущество над подзапросами в случае, когда список выбора SELECT содержит столбцы более чем из одной таблицы.

Подзапросы лучше использовать в случаях, когда нужно вычислять агрегатные значения и использовать их для сравнений во внешних запросах.

6. HAVING vs. WHERE.

**Для чего используется оператор HAVING?**

HAVING используется для фильтрации результата GROUP BY по заданным логическим условиям.

**В чем различие между операторами HAVING и WHERE?**

HAVING используется как WHERE, но в другой части SQL-выражения и, соответственно, на другой стадии формирования ответа.

7. ORDER BY.

**Для чего используется оператор ORDER BY?**

ORDER BY упорядочивает вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Многочисленные столбцы упорядочиваются один внутри другого. Возможно определять возрастание ASC или убывание DESC для каждого столбца. По умолчанию установлено - возрастание.ORDER BY — необязательный элемент запроса, который отвечает за сортировку таблицы.

Простой пример сортировки по одному столбцу. В данном запросе осуществляется сортировка по городу, который указал клиент:

select \* from Customers

ORDER BY City

По умолчанию сортировка происходит по возрастанию для чисел и в алфавитном порядке для текстовых значений. Если нужна обратная сортировка, то в конструкции ORDER BY после названия столбца надо добавить DESC:

select \* from Customers

order by CustomerID DESC

8. GROUP BY (NULL).

* **Для чего используется оператор GROUP BY?**

GROUP BY используется для агрегации записей результата по заданным признакам-атрибутам.

При использовании GROUP BY обязательно:перечень столбцов, по которым делается разрез, был одинаковым внутри SELECT и внутри GROUP BY,агрегатные функции (SUM, AVG, COUNT, MAX, MIN) должны быть также указаны внутри SELECT с указанием столбца, к которому такая функция применяется.Группировка количества клиентов по городу:select City, count(CustomerID) from CustomersGROUP BY CityГруппировка количества клиентов по стране и городу:select Country, City, count(CustomerID) from CustomersGROUP BY Country, City

**Как GROUP BY обрабатывает значение NULL?**

При использовании GROUP BY все значения NULL считаются равными.

9. GROUP BY vs. DISTINCT.

**В чем разница между операторами GROUP BY и DISTINCT?**

DISTINCT указывает, что для вычислений используются только уникальные значения столбца. NULL считается как отдельное значение. GROUP BY создает отдельную группу для всех возможных значений (включая значение NULL).

Если нужно удалить только дубликаты лучше использовать DISTINCT, GROUP BY лучше использовать для определения групп записей, к которым могут применяться агрегатные функции.

10. Агрегатные функции. COUNT(\*) vs. COUNT({column}).

**Перечислите основные агрегатные функции.**

Агрегатных функции - функции, которые берут группы значений и сводят их к одиночному значению.

SQL предоставляет несколько агрегатных функций:

COUNT - производит подсчет записей, удовлетворяющих условию запроса; SUM - вычисляет арифметическую сумму всех значений колонки; AVG - вычисляет среднее арифметическое всех значений; MAX - определяет наибольшее из всех выбранных значений; MIN - определяет наименьшее из всех выбранных значений.

**В чем разница между COUNT(\*) и COUNT({column})?**

COUNT (\*) подсчитывает количество записей в таблице, не игнорируя значение NULL, поскольку эта функция оперирует записями, а не столбцами.

COUNT ({column}) подсчитывает количество значений в {column}. При подсчете количества значений столбца эта форма функции COUNT не принимает во внимание значение NULL.

11. EXISTS.

**Что делает оператор EXISTS?**

EXISTS берет подзапрос, как аргумент, и оценивает его как TRUE, если подзапрос возвращает какие-либо записи и FALSE, если нет.

12. IN, BETWEEN, LIKE.

**Для чего используются операторы IN, BETWEEN, LIKE?**

* IN - определяет набор значений.

SELECT \* FROM Persons WHERE name IN ('Ivan','Petr','Pavel');

* BETWEEN определяет диапазон значений. В отличие от IN, BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку.

SELECT \* FROM Persons WHERE age BETWEEN 20 AND 25;

* LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется чтобы находить подстроки. В качестве условия используются символы шаблонизации (wildkards) - специальные символы, которые могут соответствовать чему-нибудь:

\_ замещает любой одиночный символ. Например, 'b\_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat'.

% замещает последовательность любого числа символов. Например '%p%t' будет соответствовать словам 'put', 'posit', или 'opt', но не 'spite'.

SELECT \* FROM UNIVERSITY WHERE NAME LIKE '%o';

13. UNION vs. UNIONALL (ограничение на UNION).

**Для чего применяется ключевое слово UNION?**

В языке SQL ключевое слово UNION применяется для объединения результатов двух SQL-запросов в единую таблицу, состоящую из схожих записей. Оба запроса должны возвращать одинаковое число столбцов и совместимые типы данных в соответствующих столбцах. Необходимо отметить, что UNION сам по себе не гарантирует порядок записей. Записи из второго запроса могут оказаться в начале, в конце или вообще перемешаться с записями из первого запроса. В случаях, когда требуется определенный порядок, необходимо использовать ORDER BY.

С удалением дублей:

SELECT \* FROM имя\_таблицы1 WHERE условие

UNION SELECT \* FROM имя\_таблицы2 WHERE условие

Без удаления дублей:

SELECT \* FROM имя\_таблицы1 WHERE условие

UNION ALL SELECT \* FROM имя\_таблицы2 WHERE условие

Можно объединять не две таблицы, а три или более:

SELECT \* FROM имя\_таблицы1 WHERE условие

UNION SELECT \* FROM имя\_таблицы2 WHERE условие

UNION SELECT \* FROM имя\_таблицы3 WHERE условие

UNION SELECT \* FROM имя\_таблицы4 WHERE услови

14. Ограничения (constraints: primary key, foreign key, check, notnull,

unique).

**Какие ограничения на целостность данных существуют в SQL?**

* PRIMARY KEY - набор полей (1 или более), значения которых образуют уникальную комбинацию и используются для однозначной идентификации записи в таблице. Для таблицы может быть создано только одно такое ограничение. Данное ограничение используется для обеспечения целостности сущности, которая описана таблицей.
* CHECK используется для ограничения множества значений, которые могут быть помещены в данный столбец. Это ограничение используется для обеспечения целостности предметной области, которую описывают таблицы в базе.
* UNIQUE обеспечивает отсутствие дубликатов в столбце или наборе столбцов.
* FOREIGN KEY защищает от действий, которые могут нарушить связи между таблицами. FOREIGN KEY в одной таблице указывает на PRIMARY KEY в другой. Поэтому данное ограничение нацелено на то, чтобы не было записей FOREIGN KEY, которым не отвечают записи PRIMARY KEY.

**Какие отличия между ограничениями PRIMARY и UNIQUE?**

По умолчанию ограничение PRIMARY создает кластерный индекс на столбце, а UNIQUE - некластерный. Другим отличием является то, что PRIMARY не разрешает NULL записей, в то время как UNIQUE разрешает одну (а в некоторых СУБД несколько) NULL запись.

**Может ли значение в столбце, на который наложено ограничение FOREIGN KEY, равняться NULL?**

Может, если на данный столбец не наложено ограничение NOT NULL.

15. Суррогатные ключи.

*Суррогатный ключ — понятие теории реляционных баз данных. Это дополнительное служебное поле, добавленное к уже имеющимся информационным полям таблицы, единственное предназначение которого — служить первичным ключом.*

**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 перестает уникально идентифицировать запись.

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

**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.

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

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

16. MERGE.

**Что делает оператор MERGE?**

MERGE позволяет осуществить слияние данных одной таблицы с данными другой таблицы. При слиянии таблиц проверяется условие, и если оно истинно, то выполняется UPDATE, а если нет - INSERT. При этом изменять поля таблицы в секции UPDATE, по которым идет связывание двух таблиц, нельзя.

17. DELETE vs. TRUNCATE.

**В чем отличие между операторами DELETE и TRUNCATE?**

DELETE - оператор DML, удаляет записи из таблицы, которые удовлетворяют критерию WHERE при этом задействуются триггеры, ограничения и т.д.

TRUNCATE - DDL оператор (удаляет таблицу и создает ее заново. Причем если на эту таблицу есть ссылки FOREIGN KEY или таблица используется в репликации, то пересоздать такую таблицу не получится).

18. Индексы.

**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 путем 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».

**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 нелинейнойсти 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 определенного 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.

**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 выборок данных (но только в случае последовательного доступа к данным). Для одного набора данных может быть создан только один кластерный индекс.

**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.

**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 зароса и/или слабой кластеризации данных, либо в случае очень маленьких наборов данных.

**Как создать индекс?**

Индекс можно создать либо с помощью выражения CREATE INDEX:

CREATE INDEX index\_name ON table\_name (column\_name)

либо указав ограничение целостности в виде уникального UNIQUE или первичного PRIMARY ключа в операторе создания таблицы CREATE TABLE.

19. Хранимая процедура.

**Что такое «хранимая процедура»?**

Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных. В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным) и в дальнейшем её обработка осуществляется быстрее.

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных. В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным) и в дальнейшем её обработка осуществляется быстрее.

20. Триггер.

**Что такое «триггер»?**

Триггер (trigger) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением, удалением или изменением данных в заданной таблице реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически и все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события) или AFTER (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись. Кроме того, триггеры могут быть привязаны не к таблице, а к представлению (VIEW). В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова BEFORE и AFTER влияют лишь на последовательность вызова триггеров, так как собственно событие (удаление, вставка или обновление) не происходит.

21. Курсор.

**Что такое «курсор»?**

Курсор — это объект базы данных, который позволяет приложениям работать с записями «по-одной», а не сразу с множеством, как это делается в обычных SQL командах.

Порядок работы с курсором такой:

1. Определить курсор (DECLARE)
2. Открыть курсор (OPEN)
3. Получить запись из курсора (FETCH)
4. Обработать запись...
5. Закрыть курсор (CLOSE)
6. Удалить ссылку курсора (DEALLOCATE). Когда удаляется последняя ссылка курсора, SQL освобождает структуры данных, составляющие курсор.

22. DATETIME vs. TIMESTAMP.

**Опишите разницу типов данных DATETIME и TIMESTAMP.**

DATETIME предназначен для хранения целого числа: YYYYMMDDHHMMSS. И это время не зависит от временной зоны настроенной на сервере. Размер: 8 байт

TIMESTAMP хранит значение равное количеству секунд, прошедших с полуночи 1 января 1970 года по усреднённому времени Гринвича. При получении из базы отображается с учётом часового пояса. Размер: 4 байта

23. Транзакции. Принципы ACID.

**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.

**Назовите основные свойства транзакции.**

1. Атомарность (atomicity) гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной.
2. Согласованность (consistency). Транзакция, достигающая своего нормального завершения и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных.
3. Изолированность (isolation). Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на ее результат.
4. Долговечность (durability). Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохраненными после возвращения системы в работу.

24. Уровни изолированности транзакций.

**Какие существуют уровни изолированности транзакций?**

В порядке увеличения изолированности транзакций и, соответственно, надежности 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, измененные 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 подтвержденных 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, внесенные 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.

**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 измененных транзакцией, которая впоследствии не подтвердится (откатится);
* Неповторяющееся чтение (non-repeatable read) — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются измененными;
* 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.

25. Нормализация. Нормальные формы. Денормализация.

**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 объема базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в базе данных информации.

**Какие существуют нормальные формы?**

* Первая нормальная форма (1NF) - Отношение находится в 1NF, если значения всех его атрибутов атомарны (неделимы).В каждой ячейке таблицы должно находиться только 1 значение.
* 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 нормальная форма (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 нее ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данное отношение.

**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.

Обязательные задачи SQL <https://habr.com/ru/post/181033/>