SQL

часть II

Хранимые процедуры

Хранимая процедура – это последовательность компилированных операторов Transact-SQL, выполняемых в виде пакета и хранящихся в системной базе данных SQL Server.

Хранимые процедуры

- это набор операторов SQL и другие логические конструкции, например операторы If и WHERE.
- Этот набор операторов SQL-сервер компилирует в единый план выполнения. Этот план выполнения сохраняется в КЭШе процедур при первом выполнении хранимой процедуры. Затем этот план можно повторно использовать без рекомпиляции при каждом вызове процедуры.

Хранимые процедуры

- можно создать непосредственно на сервере, следовательно, когда для решения определенных задач требуется многократно выполнять одни и те же операторы, применение хранимой процедуры позволяет сократить сетевой трафик и обеспечить более высокую производительность.

Применение хранимых процедур

- упрощает доступ к БД, т.к. пользователи могут в этом случае и не знать архитектуры таблиц, им не потребуется прямой доступ к реальным таблицам, они будут просто запускать необходимые для решения своих задач соответствующие процедуры.

Хранимые процедуры

- могут содержать как входные, так и выходные параметры;
- могут содержать инструкции, которые управляют потоком кода, например IF и WHILE

Первоначальное выполнение хранимой процедуры:

1. Лексический анализ разбивает ХП на отдельные компоненты.
2. Разрешение ссылок – компоненты,ссылающиеся на объекты БД, сопоставляются с этими объектами.
3. В таблице syscomments сохраняется исходный текст ХП, а в sysobjects – ее название.
4. Создается предварительный план выполнения запроса (нормализованный план или дерево запроса) и сохраняется в sysprocedures.

5. Дерево запроса считывается и окончательно оптимизируется.

- При последующих вызовах ХП выполняется только шаг 5.
- План выполнения ХП после первого выполнения хранится в быстродействующем процедурном кэше.
- ХП могут принимать аргументы при запуске и возвращать значения.

Преимущества использования ХП

- Выполняются быстрее, чем последовательность отдельных операторов;
- Необходимые для выполнения операторы уже содержатся в БД;
- Они находятся в исполняемом формате;
- Поддерживается модульное программирование;

Преимущества использования ХП (продолжение):

- Могут вызывать другие ХП и функции;
- Могут быть вызваны из прикладных программ других типов;
- Уменьшается нагрузка на сеть, т.к. для запуска достаточно передать иногда только имя ХП.

Создание и запуск ХП.

- **CREATE PROC** имя-процедуры

[параметры]

AS

Запрос SQL

- Запуск

EXEC имя-процедуры

[параметры]

Пример 1 – процедура без параметров

Вывести, когда и какой предмет сдавал студент Иванов ВВ.

CREATE PROC my_proc1

AS

SELECT Предмет.Название,Ведомость.Дата

FROM Студенты **INNER JOIN** (Предметы **INNER JOIN** Ведомость **ON** Предметы.КодПредмета=Ведомость.КодПредмета) **ON** Студент.НомерЗачетки=Ведомость.НомерЗачетки

WHERE Студент.ФИО=‘Иванов ВВ’;

Запуск:

EXEC my_proc1

Пример 2 – процедура с входным параметром

Тоже, только для любого студента

```
CREATE PROC my_proc2
@F varchar(20)
AS
SELECT Предмет.Название,Ведомость.Дата
FROM  Студенты INNER JOIN (Предметы INNER JOIN
    Ведомость ON Предметы.КодПредмета=Ведомость.
    КодПредмета) ON Студент.НомерЗачетки=Ведомость.
    НомерЗачетки
WHERE Студент.ФИО= @F;
```

Запуск:

```
EXEC my_proc2 ‘Сидоров ВВ’   или
EXEC my_proc2 @F= ‘Сидоров ВВ’
```

Пример 3 – процедура с входными параметрами

Для некоторого студента снизить оплату на 5%.

```
CREATE PROC my_proc3
@N varchar(5), @P float
AS
UPDATE Оплата
SET Сумма= Сумма*(1- @P )
WHERE Оплата.НомерЗачетки=@N;
```

Запуск:

```
EXEC my_proc3 'ИС230', 0.05 или
EXEC my_proc3 @N= 'ИС230', @P= 0.05
```

Пример 4 – процедура с входными параметрами и значениями по умолчанию

```
CREATE PROC my_proc4
@N varchar(5)= 'ИС230',
@P float=0.05
AS
UPDATE Оплата
SET Сумма= Сумма*(1- @P )
WHERE Оплата.НомерЗачетки=@N;
```

Запуск:

```
EXEC my_proc4 'ИС450', 0.1   или
EXEC my_proc4 @N='ИС450', @P = 0.1
EXEC my_proc4 @P = 0.1
EXEC my_proc4
```

Пример 5 – процедура с входными и выходными параметрами

Вывести общую сумму, уплаченную студентами за конкретный месяц

```
CREATE PROC my_proc5
@m int,
@S float output
AS
SELECT @S=SUM(Оплата.Сумма)
FROM Оплата
GROUP BY Month(Оплата.Дата)
HAVING Month(Оплата.Дата)= @m;
```

Запуск:

```
DECLARE @SS float
EXEC my_proc5 10, @SS output
SELECT @SS
```

Для чего в основном используют хранимые процедуры?

- Для упрощения сложных операций за счет инкапсуляции процессов в один блок, простой для выполнения.
- Для обеспечения непротиворечивости данных и вместе с тем без необходимости снова и снова воспроизводить одну и ту же последовательность шагов.
- Если все разработчики и приложения используют одни и те же хранимые процедуры, значит, один и тот же код будет использоваться всеми.
- Побочным эффектом этого является предотвращение ошибок. Чем больше шагов необходимо выполнить, тем выше вероятность появления ошибок. Предотвращение ошибок обеспечивает целостность данных.

Для чего в основном используют хранимые процедуры?

- Для упрощения управления изменениями. Если таблицы, имена столбцов, деловые правила (или что-то подобное) изменяются, обновлять приходиться только код хранимой процедуры и ничего больше.
- Побочным эффектом этого является повышение безопасности.
- Ограничение доступа к основным данным только через хранимые процедуры снижает вероятность повреждения данных (случайного или преднамеренного).
- Поскольку хранимые процедуры обычно сохраняются в компилированном виде, СУБД тратит меньше времени на обработку их команд. Это приводит к повышению производительности.

Для чего в основном используют хранимые процедуры?

- Существует элементы языка SQL и некоторые возможности, реализуемые только в хранимых процедурах. Хранимые процедуры, таким образом, можно использовать для написания более гибкого и мощного кода.

- **Использование хранимых процедур для всех повторяющихся действий в базе данных является хорошим стилем программирования.**

Триггеры

- это особые хранимые процедуры, автоматически выполняемые при обращении к БД с целью изменения данных.

- Содержат исполняемый код, но не могут быть вызваны из клиентского приложения или хранимой процедуры

Триггер

- всегда связан с конкретной таблицей и
- выполняется при совершении над ней операций **INSERT, UPDATE, DELETE** или их комбинаций, с которыми он должен быть связан;
- таблица может иметь любое количество любых триггеров.

Триггеры

- Все производимые **Т** модификации рассматриваются как одна транзакция (срабатывает как транзакция).
- Создает **Т** только владелец БД.
- При выполнении команд модификации данных сервер создает две таблицы:**inserted** и **deleted,** где хранятся списки соответствующих строк. Они связаны только с **Т** их создавшим, никакой другой Т не имеет к ним доступа.

Триггеры чаще всего используются для:

- Проверки корректности введенных данных и выполнении сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблиц;
- Для выполнения действий по отношению к другим таблицам, основанных на изменениях которые были сделаны в какой-то таблице;

Триггеры чаще всего используются для:

- Для выполнения дополнительной проверки и отмены введения данных (например, чтобы удостоверится, что разрешенная для клиента сумма кредита не превышена, в противном случае операция блокируется);
- Для подсчета значений вычисляемых полей или обновления меток даты/времени.

- В разных коммерческих СУБД рассматриваются разные триггеры. Так, в MS SQL Server триггеры определены только как постфильтры, то есть такие триггеры, которые выполняются после свершения события.
- В СУБД Oracle определены два типа триггеров: триггеры, которые могут быть запущены перед реализацией операции модификации, они называются BEFORE-триггерами, и триггеры, которые активизируются после выполнения соответствующей модификации, аналогично триггерам MS SQL Server, — они называются AFTER-тригерами.

- Триггеры могут быть эффективно использованы для поддержки семантической целостности БД, однако приоритет их ниже, чем приоритет правил-ограничений (constraints), задаваемых на уровне описания таблиц и на уровне связей между таблицами. При написании триггеров всегда надо помнить об этом, при нарушении правил целостности по связям (DRI declarative Referential Integrity) триггер просто может никогда не сработать.

Предложения SQL для триггеров

- **CREATE TRIGGER** создает триггер
- **ALTER TRIGGER** изменяет триггер
- **DROP TRIGGER** удаляет триггер

Создание триггера

CREATE TRIGGER имя_триггера

ON имя таблицы или представления

FOR {INSERT, UPDATE, DELETE}
[WITH ENCRYPTION]

AS оператор_**SQL;**

Необязательный параметр **WITH ENCRYPTION** не позволяет увидеть код триггера в таблице syscomments

- По умолчанию триггер выполняется после изменения данных, но если указать параметр **INSEAD OF**, то создается триггер, выполняющийся вместо изменения данных.

Изменение триггера

- **ALTER TRIGGER** имеет тот же синтаксис, что и **CREATE TRIGGER**, и применяется для изменения текста имеющегося триггера, например, если мы не хотим, чтобы триггер реагировал на выполнение обновления, следует из его текста в предложении ALTER TRIGGER удалить UPDATE

Удаление триггера

- **DROP TRIGGER** имя триггера.

- Автоматически триггер удаляется при удалении таблицы, для которой он был создан.

Триггеры

По умолчанию триггер выполняется после изменения данных, но если указать параметр **INSTEAD OF**, то такой триггер будет выполняться вместо изменения данных.

В **T** можно использовать функцию **@@ROWCOUNT**,

которая возвращает количество строк обработанных последней командой.

Пример 1

Создать триггер, который будет при каждом добавлении или изменении данных таблицы «Преподаватели» возвращать сообщение о количестве измененных записей.

```
CREATE TRIGGER tr_ПРЕП
ON Преподаватели
FOR INSERT, UPDATE
AS raiserror ('Произведено изменений таблицы',0,1 @@rowcount)
RETURN
```

Запуск

INSERT (Табельный Номер, ФИО, Должность)

VALUES (’33п28’,’Иванов ВВ’,’доцент’)

Пример 2

Триггер срабатывает при удалении экземпляра некоторой книги, например, в случае утери. Он проверяет остался ли хотя бы один экземпляр этой книги, если нет, то книга удаляется из каталога.

```
CREATE TRIGGER DEL_EXEMP ON Экземпляры
FOR DELETE
AS
DECLARE @Ntek int (кол-во оставшихся экземпляров книги)
DECLARE @DEL_EX varchar(12) (шифр удаленного экземпляра)
BEGIN
SELECT @DEL_EX=ISBN FROM deleted
EXEC @Ntek = COUNT_EX @DEL_EX (вызов ХП, определяющей кол-во экземпляров книги с зад. шифром)
IF @Ntek =0 DELETE FROM Книги WHERE Книги.ISBN= @DEL_EX END
GO
```

Транзакция

- - это последовательность операций, производимых над **БД**, рассматриваемая СУБД как единое целое, и переводящая ее из одного непротиворечивого (согласованного) состояния в другое непротиворечивое (согласованное) состояние.
- Каждая **Т** начинается при целостном состоянии **БД** и сохраняет это состояние после своего завершения.

Транзакция

Инициализация **Т** м.б. вызвана как пользователем так и прикладной программой.

Т особенно важны для многопользовательских СУБД.

Проектирование транзакции заключается в определении:

- данных, используемых **Т**,
- функциональных характеристик **Т**,
- выходных данных, формируемых **Т**,
- степени важности и интенсивности использования.

Свойства транзакций

1. **Атомарность –** транзакция должна быть выполнена целиком или не выполнена вовсе.
2. **Согласованность –** гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое, т.е. транзакция не разрушает взаимной согласованности данных.

Свойства транзакций

3. **Изолированность –** означает, что конкурирующие за доступ к БД транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей иллюзия параллельности.
4. **Долговечность –** если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах.

ФИКСАЦИЯ ИЛИ ОТКАТ?

- Если все операторы выполнены успешно и не произошло никаких сбоев программного или аппаратного обеспечения, то ФИКСАЦИЯ, т.е. все результаты транзакции становятся постоянными.
- Если в процессе выполнения транзакции случилось нечто, что ее нормальное завершение становится невозможным, то БД возвращается в свое исходное состояние, т.е. ОТКАТ.

Модель транзакций

Стандартом ANSI/ISO SQL определена модель транзакций и функции операторов

COMMIT - фиксация

ROLLBACK - откат

Транзакция завершается одним из 4-х возможных вариантов:

1. Оператор **COMMIT** означает успешное завершение транзакции; его использование делает постоянными изменения, внесенные в БД в текущей транзакции.
2. Оператор **ROLLBACK** прерывает транзакцию, отменяя изменения, сделанные в БД в рамках этой транзакции; новая транзакция начинается непосредственно после использования ROLLBACK.

Транзакция завершается одним из 4-х возможных вариантов:

3. Успешное завершение программы, в которой была инициирована текущая транзакция, означает успешное ее завершение (как будто был использован оператор **COMMIT**).
4. Ошибочное завершение программы прерывает транзакцию (как будто был использован оператор **ROLLBACK**).

ПРИМЕР

В таблице Ведомость изменить значение поля Оценка на 0, если записано значение NULL.

BEGIN TRAN update_оценка

UPDATE Ведомость **SET** оценка=0 **WHERE** оценка IS NULL

COMMIT TRAN update_оценка

GO

Журнал транзакций

ЖТ – это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью, в которую поступают записи обо всех изменениях основной части БД.

ЖТ сохраняет промежуточные состояния БД необходимые для отката транзакции.

Предназначен для поддержки надежности хранения данных во внешней памяти.

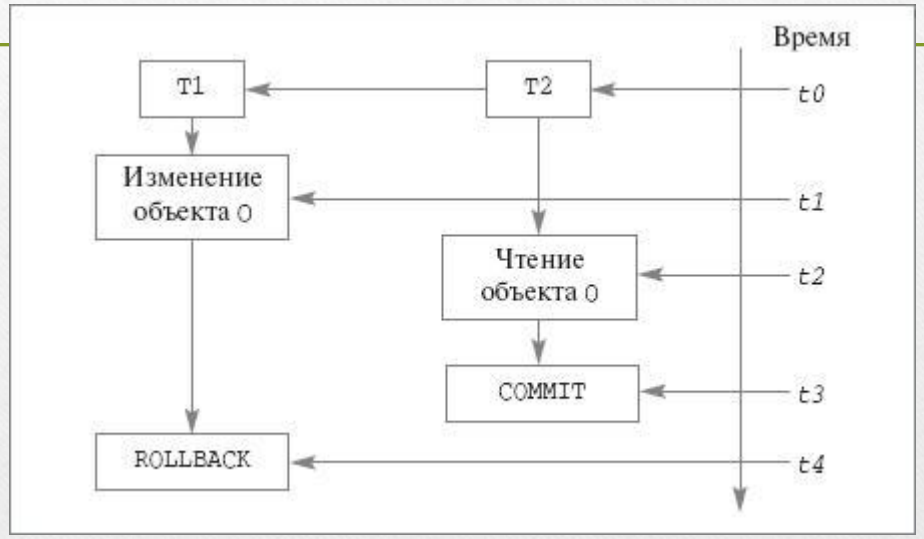
Журнал транзакций (продолжение)

Используется стратегия «упреждающей» записи (протокол WAL) – запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД.

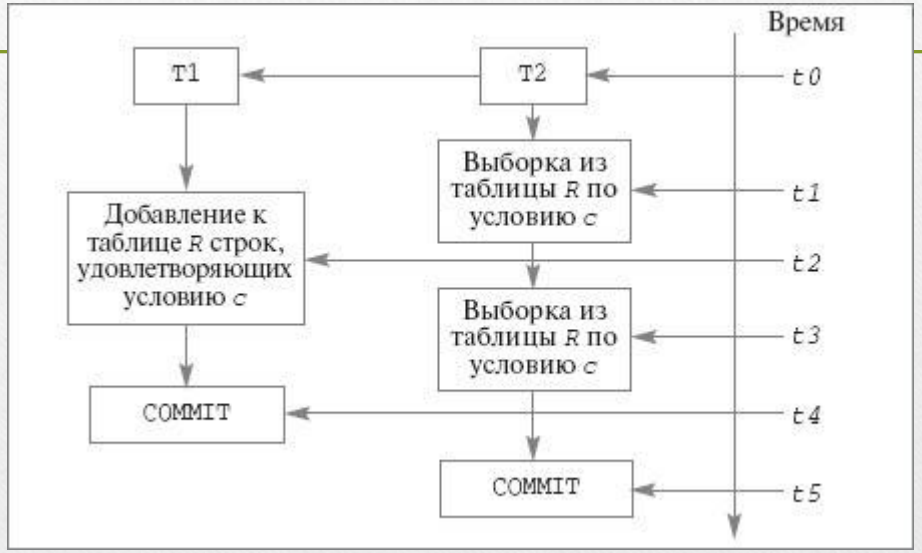
Проблемы параллельного выполнения транзакций.

- Пропавшие изменения,
- Проблемы промежуточных данных,
- Проблемы несогласованных данных,
- Проблемы строк-фантомов.

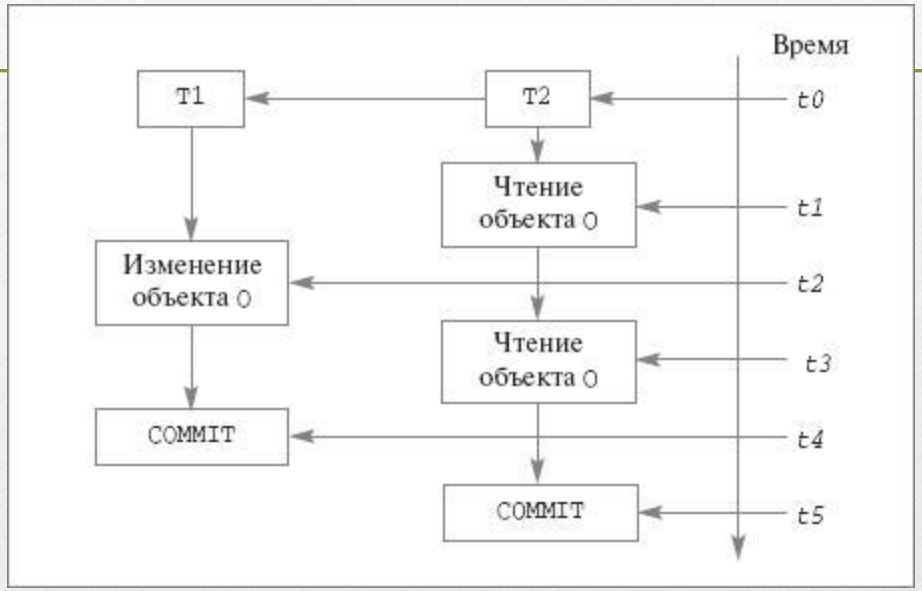
Феномен «грязное чтение»



Феномен фантомов



Феномен неповторяемого чтения



Как бороться с проблемами? Сериализация транзакций.

Выработать процедуру согласованного выполнения **T**:

- В ходе выполнения **T** пользователь видит только согласованные данные,
- При параллельном выполнении 2-х **T** СУБД гарантирует принцип независимого выполнения **T**

Сериализация транзакций.

Самый популярный механизм реализации сериализации **T** – механизм блокировок.

Блокировки

Чтобы запретить нескольким пользователям одновременно изменять данные в базе и считывать «грязные» данные используется ***блокировка.***

Блокирование обеспечивает логическую целостность транзакций и данных.

Уровни гранулярности - блокируемые ресурсы включают запись данных, страницу данных, экстент, таблицу или всю БД. Самый низкий уровень гранулярности – блокировка записи.

Режимы блокировок:

- Разделяемый
- Обновления
- Монопольный
- Намерения
- Схемы
- Массового обновления

Разделяемая блокировка

Применяется только для чтения и позволяет параллельным транзакциям одновременно считывать данные из одного и того же источника.

Блокировка обновления

Применяется, когда допустимо обновление ресурса. Только одна транзакция единовременно может наложить ее. Если транзакция действительно производит изменение, то эта блокировка преобразуется в *монопольную*, иначе – в *разделяемую*.

Блокировка намерения
Монопольная блокировка

Применяется для модификации данных. При этом никакая другая транзакция не может читать или изменять этот ресурс.

Блокировка намерения

применяется для иерархического упорядочения блокировок.

Блокировка схемы

применяется при выполнении операции, затрагивающей схему таблицы, например, при добавлении поля в таблицу или компиляции запросов.

Блокировка массового обновления

Применяется при копировании больших объемов данных в таблицу с указанием TABLOCK.

Индексы

- это набор ссылок, упорядоченных по определенному столбцу таблицы,
- это наборы уникальных значений для некоторой таблицы с соответствующими ссылками на данные,
- это средство логической сортировки данных для повышения скорости поиска и выполнения в последующем сортировки.

Индекс также ускоряет поиск, как предметный указатель в книге.

Файлы

- ***Индексированный файл*** – это основной файл, содержащий данные отношения, для которого создан индексный файл.
- ***Индексный файл*** – это файл, в котором каждая запись состоит из двух значений: данных поля по которому индексировали и указателя, который связывает с соответствующим кортежем индексированного файла

Индекс

- представляет собой физический объект базы данных, имеющий структуру b-дерева и используемый оптимизатором для ускорения доступа к данным по сравнению с линейным просмотром таблиц. Дополнительное преимущество индексов заключается в том, что они используются для обеспечения уникальности ключа, — это означает, что конкретные значения ключа в столбце не должны повторяться. В SQL Server существуют индексы двух типов: кластерные и некластерные.

Кластерный индекс

- Создание кластерного индекса приводит к физической сортировке данных.
- У каждой таблицы может быть лишь один кластерный индекс.
- Один элемент индекса соответствует странице данных. Страница представляет собой физический блок для хранения данных объемом 8 Кбайт. После того как сервер определит страницу (посредством перебора индекса или просмотра таблицы), он находит на странице нужную запись.
- Кластерный индекс позволяет выйти на первую запись результата и организовать последующий перебор таблицы (вместо повторного перемещения по индексу) в поисках остальных записей без возврата на уровень индекса, кластерные индексы отлично подходят для выборки интервалов данных. К этой категории относятся внешние ключи (объединения), списки имен и любые другие совокупности данных с последовательными ключами.

Некластерные индексы

- позволяют индексировать данные на основании значения ключа в каждой записи.
- SQL Server позволяет создать до 249 некластерных индексов.
- Один элемент индекса приходится на одну запись. Когда запись будет идентифицирована, к ней происходит непосредственное обращение. Каждой записи в таблице назначается идентификатор. Поскольку в некластерном индексе каждый указатель соответствует одной записи, а также потому, что их указатели имеют больший размер, чем у кластерных индексов (поскольку содержат идентификатор записи, а не просто идентификатор страницы), некластерные индексы обычно занимают намного больше места.

Полнотекстовый индекс

- хранящаяся в базе проиндексированных текстов совокупность идентификаторов, определяющих учитываемую совокупность индексов для всех слов текстов проиндексированных информационных ресурсов. Полнотекстовый индекс предназначен для обеспечения поиска по тексту проиндексированного документа и может включать в себя морфологический индекс, синтаксический индекс, семантический индекс.

Индексы (создание)

CREATE INDEX имя_индекса

ON имя_таблицы (столбцы для индексации)

Индексы. Что следует учитывать:

- Индексы повышают производительность операций выборки, но ухудшают производительность операций-действий;
- Для хранения данных индекса требуется много места на диске;
- Не все данные подходят для индексации. Данные, неявляющиеся по своей сути уникальными (ГРУППА а таблице СТУДЕНТ), не дадут большого эффекта;

Индексы. Что следует учитывать:

- Выбирайте для индексации данные, которые часто используются для фильтрации и сортировки данных;
- В качестве индекса можно использовать несколько столбцов.