



Рейтинг

Конференции Олега Бунина (Онтико)

Конференции Олега Бунина

 olegbunin 2 февраля 2018 в 17:40

SQL ключи во всех подробностях

Автор оригинала: Joe Nelson

Блог компании Конференции Олега Бунина (Онтико), MySQL, PostgreSQL, SQL, Администрирование баз данных

[Перевод](#)

В Интернете полно догматических заповедей о том, как нужно выбирать и использовать ключи в реляционных базах данных. Иногда споры даже переходят в холивары: использовать естественные или искусственные ключи? Автоинкрементные целые или UUID?

Прочитав шестьдесят четыре статьи, пролистав разделы пяти книг и задав кучу вопросов в IRC и StackOverflow, я (автор оригинальной статьи **Joe «begriffs» Nelson**), как мне кажется, собрал куски паззла воедино и теперь смогу примирить противников. Многие споры относительно ключей возникают, на самом деле, из-за неправильного понимания чужой точки зрения.

Содержание

- Что же такое «ключи»?
- Любопытный случай первичных ключей
- Выбор естественных ключей
- Искусственные ключи
- Суррогатные ключи
- Автоинкрементные BIGINT
- UUID
- Итоги и рекомендации

Давайте разделим проблему на части, а в конце соберём её снова. Для начала зададим вопрос – что же такое «ключ»?

Что же такое «ключи»?

Забудем на минуту о *первичных* ключах, нас интересует более общая идея. Ключ — это колонка (column) или колонки, не имеющие в строках дублирующих значений. Кроме того, колонки должны быть неприводимо уникальными, то есть никакое подмножество колонок не обладает такой уникальностью.

Для примера рассмотрим таблицу для подсчёта карт в карточной игре:

```
CREATE TABLE cards_seen (  
    suit text,  
    face text  
);
```

Если мы отслеживаем одну колоду (то есть без повторяющихся карт), то сочетание рубашки и лица уникально и нам бы не хотелось вносить в таблицу одинаковые рубашку и лицо дважды, потому что это будет избыточно. Если карта есть в таблице, то мы видели её, в противном случае — не видели.

Мы можем и должны задать базе данных это ограничение, добавив следующее:

```
CREATE TABLE cards_seen (  
    suit text,  
    face text,  
  
    UNIQUE (suit, face)  
);
```

Сами по себе ни `suit` (рубашка), ни `face` (лицо) не являются уникальными, мы можем увидеть разные карты с одинаковыми рубашкой или лицом. Поскольку `(suit, face)` уникально, а отдельные колонки не уникальны, можно утверждать, что их сочетание неприводимо, а `(suit, face)` является ключом.

В более общей ситуации, когда нужно отслеживать несколько колод карт, можно добавить новое поле и записывать сколько раз мы видели карту:

```
CREATE TABLE cards_seen (  
    suit text,  
    face text,  
    seen int  
);
```

Хотя тройка `(suit, face, seen)` получается уникальной, она не является ключом, потому что подмножество `(suit, face)` тоже должно быть уникальным. Это необходимо, поскольку две строки с одинаковыми рубашкой и лицом, но разными значениями `seen` будут противоречащей информацией. Поэтому ключом является `(suit, face)`, и больше в этой таблице нет никаких ключей.

Ограничения уникальности

В PostgreSQL предпочтительным способом добавления ограничения уникальности является его прямое объявление, как в нашем примере. Использование индексов для соблюдения ограничения уникальности может понадобиться в отдельных случаях, но не стоит обращаться к ним напрямую. Нет необходимости в ручном создании индексов для колонок, уже объявленных уникальными; такие действия будут просто дублировать автоматическое создание индекса.

Также в таблице без проблем может быть несколько ключей, и мы должны объявить их *все*, чтобы соблюдать их уникальность в базе данных.

Вот два примера таблиц с несколькими ключами.

```
-- Три ключа  
CREATE TABLE tax_brackets (  
    min_income numeric(8,2),  
    max_income numeric(8,2),  
    tax_percent numeric(3,1),
```

```
    UNIQUE(min_income),
    UNIQUE(max_income),
    UNIQUE(tax_percent)
);

-- Два ключа
CREATE TABLE flight_roster (
    departure timestampz,
    gate text,
    pilot text

    UNIQUE(departure, gate),
    UNIQUE(departure, pilot)
);
```

Ради краткости в примерах отсутствуют любые другие ограничения, которые были бы на практике. Например, у карт не должно быть отрицательное число просмотров, и значение NULL недопустимо для большинства рассмотренных колонок (за исключением колонки `max_income` для налоговых групп, в которой NULL может обозначать бесконечность).

Любопытный случай первичных ключей

То, что в предыдущем разделе мы назвали просто «ключами», обычно называется «потенциальными ключами» (candidate keys). Термин «candidate» подразумевает, что все такие ключи конкурируют за почётную роль «первичного ключа» (primary key), а оставшиеся назначаются «альтернативными ключами» (alternate keys).

Потребовалось какое-то время, чтобы в реализациях SQL пропало несоответствие ключей и реляционной модели, самые ранние базы данных были заточены под низкоуровневую концепцию первичного ключа. Первичные ключи в таких базах требовались для идентификации физического расположения строки на носителях с последовательным доступом к данным. Вот как это объясняет Джо Селко:

Термин «ключ» означал ключ сортировки файла, который был нужен для выполнения любых операций обработки в последовательной файловой системе. Набор перфокарт считывался в одном и только в одном порядке; невозможно было «вернуться назад». Первые накопители на магнитных лентах имитировали такое же поведение и не позволяли выполнять двунаправленный доступ. Т.е., первоначальный Sybase SQL Server для чтения предыдущей строки требовал «перемотки» таблицы на начало.

В современном SQL не нужно ориентироваться на физическое представление информации, таблицы моделируют связи и внутренний порядок строк вообще не важен. Однако, и сейчас SQL-сервер по умолчанию создаёт кластерный индекс для первичных ключей и, по старой традиции, физически выстраивает порядок строк.

В большинстве баз данных первичные ключи сохранились как пережиток прошлого, и едва ли обеспечивают что-то, кроме отражения или определения физического расположения. Например, в таблице PostgreSQL объявление первичного ключа автоматически накладывает ограничение NOT NULL и определяет внешний ключ по умолчанию. К тому же первичные ключи являются предпочтительными столбцами для оператора JOIN.

Первичный ключ не отменяет возможности объявления и других ключей. В то же время, если ни один ключ не назначен первичным, то таблица все равно будет нормально работать. Молния, во всяком случае, в вас не ударит.

Нахождение естественных ключей



Рассмотренные выше ключи называются «естественными», потому что они являются свойствами моделируемого объекта

интересными сами по себе, даже если никто не стремится сделать из них ключ.

Первое, что стоит помнить при исследовании таблицы на предмет возможных естественных ключей — нужно стараться не перемудрить. Пользователь `sqlvogel` на `StackExchange` даёт следующий совет:

У некоторых людей возникают сложности с выбором «естественного» ключа из-за того, что они придумывают гипотетические ситуации, в которых определённый ключ может и не быть уникальным. Они не понимают самого смысла задачи. Смысл ключа в том, чтобы определить правило, по которому атрибуты в любой момент времени должны быть и всегда будут уникальными в конкретной таблице. Таблица содержит данные в конкретном и хорошо понимаемом контексте (в «предметной области» или в «области дискурса») и единственное значение имеет применение ограничения в этой конкретной области.

Практика показывает, что нужно вводить ограничение по ключу, когда колонка уникальна при имеющихся значениях и будет оставаться такой при вероятных сценариях. А при необходимости ограничение можно устранить (если это вас беспокоит, то ниже мы расскажем о стабильности ключа.)

Например, база данных членов хобби-клуба может иметь уникальность в двух колонках — `first_name`, `last_name`. При небольшом объёме данных дубликаты маловероятны, и до возникновения реального конфликта использовать такой ключ вполне разумно.

С ростом базы данных и увеличением объёма информации, выбор естественного ключа может стать сложнее. Хранимые нами данные являются упрощением внешней реальности, и не содержат в себе некоторые аспекты, которыми различаются объекты в мире, такие как их изменяющиеся со временем координаты. Если у объекта отсутствует какой-либо код, то как различить две банки с напитком или две коробки с овсянкой, кроме как по их расположению в пространстве или по небольшим различиям в весе или упаковке?

Именно поэтому органы стандартизации создают и наносят на продукцию различительные метки. На автомобилях штампуются `Vehicle Identification Number (VIN)`, в книгах печатается `ISBN`, на упаковке пищевых товаров есть `UPC`. Вы можете возразить, что эти числа не кажутся естественными. Так почему же я называю их естественными ключами?

Естественность или искусственность уникальных свойств в базе данных относительно к внешнему миру. Ключ, который при своём создании в органе стандартизации или государственном учреждении был искусственным, становится для нас естественным, потому что в целом мире он становится стандартом и/или печатается на объектах.

Существует множество отраслевых, общественных и международных стандартов для различных объектов, в том числе для валют, языков, финансовых инструментов, химических веществ и медицинских диагнозов. Вот некоторые из значений, которые часто используются в качестве естественных ключей:

- Коды стран по ISO 3166
- Коды языков по ISO 639
- Коды валют по ISO 4217
- Биржевые обозначения ISIN
- UPC/EAN, VIN, GTIN, ISBN
- имена логинов
- адреса электронной почты
- номера комнат
- mac-адрес в сети
- (широта, долгота) для точек на поверхности Земли

Рекомендуем объявлять ключи, когда это возможно и разумно, может быть, даже несколько ключей на таблицу. Но помните, что у всего вышеперечисленного могут быть исключения.

- Не у всех есть адрес электронной почты, хотя в некоторых условиях использования базы данных это может быть приемлемо. Кроме того, люди время от времени меняют свои электронные адреса. (Подробнее о стабильности ключей

позже.)

- Биржевые обозначения ISIN время от времени изменяются, например, символы GOOG и GOOGL не точно описывают реорганизацию компании из Google в Alphabet. Иногда может возникнуть путаница, как, например, с TWTR и TWTRQ, некоторые инвесторы ошибочно покупали последние во время IPO Twitter.
- Номера социального страхования используются только гражданами США, имеют ограничения конфиденциальности и повторно используются после смерти. Кроме того, после кражи документов люди могут получить новые номера. Наконец, один и тот же номер может идентифицировать и лицо, и идентификатор налога на прибыль.
- Почтовые индексы — плохой выбор для городов. У некоторых городов общий индекс, или наоборот в одном городе бывает несколько индексов.

Искусственные ключи

lusab-babad
gutih-tugad
gutuk-bisog
mudof-sakat
haguz-biram

С учётом того, что ключ — это колонка, в каждой строке которой находятся уникальные значения, одним из способов его создания является жульничество — в каждую строку можно записать выдуманные уникальные значения. Это и есть искусственные ключи: придуманный код, используемый для ссылки на данные или объекты.

Очень важно то, что код генерируется из самой базы данных и неизвестен никому, кроме пользователей базы данных. Именно это отличает искусственные ключи от стандартизированных естественных ключей.

Преимущество естественных ключей заключается в защите от дублирования или противоречивости строк таблицы, искусственные же ключи полезны потому, что они позволяют людям или другим системам проще ссылаться на строку, а также повышают скорость операций поиска и объединения, так как не используют сравнения строковых (или многостолбцовых) ключей.

Суррогаты

Искусственные ключи используются в качестве привязки — вне зависимости от изменения правил и колонок, одну строку всегда можно идентифицировать одинаковым способом. Искусственный ключ, используемый для этой цели, называется «суррогатным ключом» и требует особого внимания. Суррогаты мы рассмотрим ниже.

Не являющиеся суррогатами искусственные ключи удобны для ссылок на строку *снаружи* базы данных. Искусственный ключ кратко идентифицирует данные или объект: он может быть указан как URL, прикреплен к счёту, продиктован по телефону, получен в банке или напечатан на номерном знаке. (Номерной знак автомобиля для нас является естественным ключом, но разработан государством как искусственный ключ.)

Искусственные ключи нужно выбирать, учитывая возможные способы их передачи, чтобы минимизировать опечатки и ошибки. Надо учесть, что ключ могут произносить, читать напечатанным, отправлять по SMS, читать написанным от руки, вводить с клавиатуры и встраивать в URL. Дополнительно, некоторые искусственные ключи, например, номера кредитных карт, содержат контрольную сумму, чтобы при возникновении определённых ошибок их можно было хотя бы распознать.

Примеры:

- Для номерных знаков США существуют правила об использовании неоднозначных признаков, например 0 и o.
- Больницы и аптеки должны быть особенно аккуратны, учитывая почерк врачей.
- Передаёте эсэмэской код подтверждения? Не выходите за пределы набора символов GSM 03.38.
- В отличие от Base64, кодирующего произвольные байтовые данные, Base32 использует ограниченный набор символов, который удобно использовать людям и обрабатывать на старых компьютерных системах.

- Proquints – это читаемые, записываемые и произносимые идентификаторы. Это произносимые (PRO-nouncable) пятёрки (QUINT-uplets) однозначно понимаемых согласных и гласных букв.

Учтите, что как только вы познакомите мир со своим искусственным ключом, люди странным образом начнут придавать ему особое внимание. Достаточно посмотреть на «блатные» номерные знаки или на систему создания произносимых идентификаторов, которая превратилась в печально известный автоматизированный генератор ругательств.

Даже, если ограничиться числовыми ключами, есть табу типа тринадцатого этажа. Несмотря на то, что proquints обладают большей плотностью информации на произносимый слог, числа тоже неплохи во многих случаях: в URL, пин-клавиатурах и написанных от руки записях, если получатель знает, что ключ состоит только из цифр.

Однако, обратите внимание, что не стоит использовать последовательный порядок в публично открытых числовых ключах, поскольку это позволяет рыться в ресурсах (/videos/1.mpeg, /videos/2.mpeg, и так далее), а также создаёт утечку информации о количестве данных. Наложите на последовательность чисел сеть Фейстеля и сохраните уникальность, скрыв при этом порядок чисел.

В wiki PostgreSQL есть пример функции псевдошифрования:

```
CREATE OR REPLACE FUNCTION pseudo_encrypt(VALUE int) returns int AS $$
DECLARE
l1 int;
l2 int;
r1 int;
r2 int;
i int:=0;
BEGIN
  l1:= (VALUE >> 16) & 65535;
  r1:= VALUE & 65535;
  WHILE i < 3 LOOP
    l2 := r1;
    r2 := l1 # (((1366 * r1 + 150889) % 714025) / 714025.0) * 32767)::int;
    l1 := l2;
    r1 := r2;
    i := i + 1;
  END LOOP;
  RETURN ((r1 << 16) + l1);
END;
$$ LANGUAGE plpgsql strict immutable;
```

Эта функция является обратной самой себе (т.е. $\text{pseudo_encrypt}(\text{pseudo_encrypt}(x)) = x$). Точное воспроизведение функции является своего рода безопасностью через неясность, и если кто-нибудь догадается, что вы использовали сеть Фейстеля из документации PostgreSQL, то ему будет легко получить исходную последовательность. Однако вместо $((1366 * r1 + 150889) \% 714025) / 714025.0$ можно использовать другую функцию с областью значений от 0 до 1, например, просто поэкспериментировать с числами в предыдущем выражении.

Вот, как использовать pseudo_encrypt:

```
CREATE SEQUENCE my_table_seq;

CREATE TABLE my_table (
  short_id int NOT NULL
    DEFAULT pseudo_encrypt(
      nextval('my_table_seq')::int
    ),
  -- другие колонки ...

  UNIQUE (short_id)
);
```

Такое решение сохраняет случайные значения в столбце short_id, если же важно поддерживать высокие скорости обработки данных, то можно хранить в таблице саму инкрементную последовательность и преобразовывать её при запросе отображения с помощью pseudo_encrypt. Как мы увидим позже, индексирование рандомизированных значений может привести к увеличению объёма записи.

В предыдущем примере для `short_id` использовались целые значения обычного размера, для `bigint` есть другие функции Фейстеля, например XTEA.

Ещё один способ запутать последовательность целых чисел заключается в преобразовании её в короткие строки. Попробуйте воспользоваться расширением `pg_hashids`:

```
CREATE EXTENSION pg_hashids;

CREATE SEQUENCE my_table_seq;

CREATE TABLE my_table (
  short_id text NOT NULL
  DEFAULT id_encode(
    nextval('my_table_seq'),
    ' long string as table-specific salt '
  ),
  -- другие колонки ...

  UNIQUE (short_id)
);

INSERT INTO my_table VALUES
  (DEFAULT), (DEFAULT), (DEFAULT);

SELECT * FROM my_table;
/*
| short_id |
|-----|
| R4      |
| ya      |
| LL      |
*/
```

Здесь снова будет быстрее хранить в таблице сами целые числа и преобразовывать их по запросу, но замерьте производительность и посмотрите, имеет ли это смысл на самом деле.

Теперь, чётко разграничив смысл искусственных и естественных ключей, мы видим, что споры «естественные против искусственных» являются ложной дихотомией. Искусственные и естественные ключи не исключают друг друга! В одной таблице могут быть и те, и другие. На самом деле, таблица с искусственным ключом должна обеспечивать и естественный ключ, за редким исключением, когда не существует естественного ключа (например, в таблице кодов купонов):

```
-- Редкий пример таблицы: нет потенциальных естественных ключей,
-- которые можно объявить вместе с искусственным ключом "code"

CREATE TABLE coupons (
  code text NOT NULL,
  amount numeric(5,2) NOT NULL,
  redeemed boolean NOT NULL DEFAULT false,

  UNIQUE (code)
);
```

Если у вас есть искусственный ключ и вы не объявляете естественные ключи, когда они существуют, то оставляете последние незащищёнными:

```
CREATE TABLE cars (
  car_id bigserial NOT NULL,
  vin varchar(17) NOT NULL,
  year int NOT NULL,

  UNIQUE (car_id)
```

```
-- нужно было добавить
-- UNIQUE (vin)
);

-- К сожалению, это успешно выполнится
INSERT INTO cars (vin, year) VALUES
('1FTJW36F2TEA03179', 1996),
('1FTJW36F2TEA03179', 1997);
```

Единственным аргументом против объявления дополнительных ключей является то, что каждый новый несёт за собой ещё один уникальный индекс и увеличивает затраты на запись в таблицу. Конечно, зависит от того, насколько вам важна корректность данных, но, скорее всего, ключи все же стоит объявлять.

Также стоит объявлять несколько искусственных ключей, если они есть. Например, у организации есть кандидаты на работу (Applicants) и сотрудники (Employees). Каждый сотрудник когда-то был кандидатом, и относится к кандидатам по своему собственному идентификатору, который также должен быть и ключом сотрудника. Ещё один пример, можно задать идентификатор сотрудника и имя логина как два ключа в Employees.

Суррогатные ключи



Как уже упоминалось, важный тип искусственного ключа называется «суррогатный ключ». Он не должен быть кратким и передаваемым, как другие искусственные ключи, а используется как внутренняя метка, всегда идентифицирующая строку. Он используется в SQL, но приложение не обращается к нему явным образом.

Если вам знакомы системные колонки (system columns) из PostgreSQL, то вы можете воспринимать суррогаты почти как параметр реализации базы данных (вроде ctid), который однако никогда не меняется. Значение суррогата выбирается один раз для каждой строки и потом никогда не изменяется.

Суррогатные ключи отлично подходят в качестве внешних ключей, при этом необходимо указать каскадные ограничения ON UPDATE RESTRICT, чтобы соответствовать неизменности суррогата.

С другой стороны, внешние ключи к публично передаваемым ключам должны быть помечены ON UPDATE CASCADE, чтобы обеспечить максимальную гибкость. (Каскадное обновление выполняется на том же уровне изоляции, что и окружающая его транзакция, поэтому не беспокойтесь о проблемах с параллельным доступом – база данных справится, если выбрать строгий уровень изоляции.)

Не делайте суррогатные ключи «естественными». Как только вы покажете значение суррогатного ключа конечным пользователям, или, что хуже, позволите им работать с этим значением (в частности через поиск), то фактически придадите ключу значимость. Потом показанный ключ из вашей базы данных может стать естественным ключом в чьей-то чужой БД.

Принуждение внешних систем к использованию других искусственных ключей, специально предназначенных для передачи, позволяет нам при необходимости изменять эти ключи в соответствии с меняющимися потребностями, в то же время поддерживая внутреннюю целостность ссылок с помощью суррогатов.

Автоинкрементные bigint

Чаще всего для суррогатных ключей используют автоинкрементную колонку «bigserial», также известную как IDENTITY. (На самом деле, PostgreSQL 10 теперь, как и Oracle, поддерживает конструкцию IDENTITY, см. CREATE TABLE.)

Однако, я считаю, что автоинкрементное целое плохой выбор для суррогатных ключей. Такое мнение непопулярно, поэтому позвольте мне объяснить.

Недостатки последовательных ключей:

- Если все последовательности начинаются с 1 и постепенно увеличиваются, то у строк из разных таблиц будут одинаковые значения ключей. Такой вариант неидеален, предпочтительнее все же использовать непересекающиеся множества ключей в таблицах, чтобы, например, запросы не смогли бы случайно перепутать константы в JOIN и вернуть неожиданные результаты. (Как вариант для обеспечения отсутствия пересечений, можно составить каждую последовательность из чисел, кратных различным простым, но это будет довольно трудоёмко.)
- Вызов `nextval()` для генерации последовательности в современных распределённых SQL, приводит к тому, что вся система хуже масштабируется.
- Поглощение данных из базы данных, в которой тоже использовались последовательные ключи, приведет к конфликтам, потому что последовательные значения не будут уникальными в разных системах.
- С философской точки зрения последовательное увеличение чисел связано со старыми системами, в которых подразумевался порядок строк. Если же вы теперь хотите упорядочить строки, то делайте это явным образом, с помощью колонки меток времени или чего-то имеющего смысл в ваших данных. В противном случае нарушается первая нормальная форма.
- (Слабая причина, но) эти короткие идентификаторы так и тянет сообщить кому-нибудь.

UUID

Давайте рассмотрим другой вариант: использование больших целых чисел (128-битных), генерируемых в соответствии со случайным шаблоном. Алгоритмы генерации таких универсальных уникальных идентификаторов (universally unique identifier, UUID) имеют чрезвычайно малую вероятность выбора одного значения дважды, даже при одновременном выполнении на двух разных процессорах.

В таком случае, UUID кажутся естественным выбором для использования в качестве суррогатных ключей, не правда ли? Если вы хотите пометить строки уникальным образом, то ничто не сравнится с уникальной меткой!

Так почему же все не пользуются ими в PostgreSQL? На это есть несколько надуманных причин и одна логичная, которую можно обойти, и я представлю бенчмарки, чтобы проиллюстрировать свое мнение.

Для начала, расскажу о надуманных причинах. Некоторые люди думают, что UUID — это строки, потому что они записываются в традиционном шестнадцатеричном виде с дефисом: 5bd68e64-ff52-4f54-ace4-3cd9161c8b7f. Действительно, некоторые базы данных не имеют компактного (128-битного) типа `uuid`, но в PostgreSQL он есть и имеет размер двух `bigint`, т.е., по сравнению с объёмом прочей информации в базе данных, издержки незначительны.

Ещё UUID незаслуженно обвиняется в громоздкости, но кто будет их произносить, печатать или читать? Мы говорили, что это имеет смысл для показываемых искусственных ключей, но никто (по определению) не должен увидеть суррогатный UUID. Возможно, с UUID будет иметь дело разработчик, запускающий команды SQL в `psql` для отладки системы, но на этом всё. А разработчик может ссылаться на строки и с помощью более удобных ключей, если они заданы.

Реальная проблема с UUID в том, что сильно рандомизированные значения приводят к увеличению объёма записи (write amplification) из-за записей полных страниц в журнал с упреждающей записью (write-ahead log, WAL). Однако, на самом деле снижение производительности зависит от алгоритма генерации UUID.

Давайте измерим `write amplification`. По правде говоря, проблема в старых файловых системах. Когда PostgreSQL выполняет запись на диск, она изменяет «страницу» на диске. При отключении питания компьютера большинство файловых систем всё равно сообщит об успешной записи ещё до того, как данные безопасно сохранились на диске. Если PostgreSQL наивно воспримет такое действие завершённым, то при последующей загрузке системы база данных будет повреждена.

Раз PostgreSQL не может доверять большинству ОС/файловых систем/конфигураций дисков в вопросе обеспечения неразрывности, база данных сохраняет полное состояние изменённой дисковой страницы в журнал с упреждающей записью (`write-ahead log`), который можно будет использовать для восстановления после возможного сбоя. Индексирование сильно рандомизированных значений наподобие UUID обычно затрагивает кучу различных страниц диска и приводит к записи полного размера страницы (обычно 4 или 8 КБ) в WAL для каждой новой записи. Это так называемая полностраничная запись (`full-page write`, FPW).

Некоторые алгоритмы генерации UUID (такие, как «`snowflake`» от Twitter или `uuid_generate_v1()` в расширении `uuid-oss` для

PostgreSQL) создают на каждой машине монотонно увеличивающиеся значения. Такой подход консолидирует записи в меньшее количество страниц диска и снижает FPW.

Давайте измерим влияние FPW для различных алгоритмов генерации UUID, а также исследуем статистику WAL. Я использовал следующую конфигурацию для замера.

- Экземпляр EC2 с запущенным ami-aa2ea6d0
 - Ubuntu Server 16.04 LTS (HVM)
 - EBS General Purpose (SSD)
 - c3.xlarge
 - vCPU: 4
 - RAM GiB: 7.5
 - Disk GB: 2 x 40 (SSD)
- PostgreSQL, собранная из исходников
 - `ftp.postgresql.org/pub/source/v10.1/postgresql-10.1.tar.gz`
 - `./configure --with-uuid=openssl CFLAGS="-O3"`
- Конфигурация базы данных по умолчанию, со следующими исключениями:
 - `max_wal_size='10GB';`
 - `checkpoint_timeout='2h';`
 - `synchronous_commit='off';`

Схема:

```
CREATE EXTENSION "uuid-openssl";
CREATE EXTENSION pgcrypto;

CREATE TABLE u_v1 ( u uuid PRIMARY KEY );
CREATE TABLE u_crypto ( u uuid PRIMARY KEY );
```

Перед тем, как добавить UUID в каждую таблицу, находим текущую позицию write-ahead log.

```
SELECT pg_walfile_name(pg_current_wal_lsn());

/* Например,

   pg_walfile_name
-----
000000010000000000000001
*/
```

Я использовал такую позицию, чтобы получить статистику об использовании WAL после проведения бенчмарка. Так мы получим статистику событий, выполняемых последовательно после начальной позиции:

```
pg_waldump --stats 000000010000000000000001
```

Я провёл тесты трёх сценариев:

1. Добавление UUID, сгенерированных алгоритмом `gen_random_uuid()` (`pgcrypto`)

- Добавление из `uuid_generate_v1()` (предоставленного [uuid-oss] (<https://www.postgresql.org/docs/10/static/uuid-oss.html>))
- Снова добавление из `gen_random_uuid()`, но теперь с параметром `full_page_writes='off'` в конфигурации БД. Это покажет, насколько всё будет быстрее без увеличения FPW.

Для каждого из этих сценариев я начинал с пустой таблицы и вставлял 2^{20} UUID и повторял процедуру шестнадцать раз, измеряя каждый из них, чтобы отследить, как меняется производительность при большем количестве данных в таблице.

```
-- например, я выполнял это в psql 16 раз с параметром \timing

INSERT INTO u_crypto (
  SELECT gen_random_uuid()
  FROM generate_series(1, 1024*1024)
);
```

И вот результаты замеров скорости:

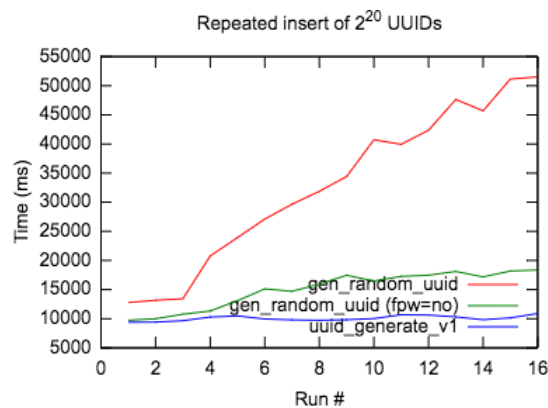


График скорости вставки UUID

Вот статистика WAL для каждого из способов:

`gen_random_uuid()`

Тип	N	(%)	Размер записи	(%)	Размер FPI	(%)
----	-	---	-----	---	-----	---
XLOG	260 (0.15)		13139 (0.09)		484420 (30.94)	
Heap2	765 (0.45)		265926 (1.77)		376832 (24.07)	
Heap	79423 (46.55)		6657121 (44.20)		299776 (19.14)	
Btree	89354 (52.37)		7959710 (52.85)		404832 (25.85)	

`uuid_generate_v1()`

Тип	N	(%)	Размер записи	(%)	Размер FPI	(%)
----	-	---	-----	---	-----	---
XLOG	0 (0.00)		0 (0.00)		0 (0.00)	
Heap2	0 (0.00)		0 (0.00)		0 (0.00)	
Heap	104326 (49.88)		7407146 (44.56)		0 (0.00)	
Btree	104816 (50.12)		9215394 (55.44)		0 (0.00)	

`gen_random_uuid()` with `fpw=off`

Тип	N	(%)	Размер записи	(%)	Размер FPI	(%)
----	-	---	-----	---	-----	---
XLOG	4 (0.00)		291 (0.00)		64 (0.84)	
Heap2	0 (0.00)		0 (0.00)		0 (0.00)	
Heap	107778 (49.88)		7654268 (46.08)		0 (0.00)	
Btree	108260 (50.11)		8956097 (53.91)		7556 (99.16)	

Результаты подтверждают, что `gen_random_uuid` создаёт существенную активность в WAL из-за полностраничных образов (full-page images, FPI), а другие способы этим не страдают. Конечно, в третьем методе я просто запретил базе данных делать это. Однако запрет FPW совсем не то, что стоило бы использовать в реальности, если только вы не полностью уверены в файловой системе и конфигурации дисков. В этой статье утверждается, что ZFS может быть безопасным для отключения FPW, но пользуйтесь им с осторожностью.

Явным победителем в моём бенчмарке оказался `uuid_generate_v1()` – он быстр и не замедляется при накоплении строк. Расширение `uuid-ossr` по умолчанию установлено в таких облачных базах данных, как RDS и Citus Cloud, и будет доступно без дополнительных усилий.

В документации есть предупреждение о `uuid_generate_v1`:

В нём используется MAC-адрес компьютера и метка времени. Учитывайте, что UUID такого типа раскрывают информацию о компьютере, который создал идентификатор, и время его создания, что может быть неприемлемым, когда требуется высокая безопасность.

Однако я не думаю, что настоящая проблема, потому что суррогатный ключ не передаётся. Если же это всё-таки важно для вас, в библиотеке есть `uuid_generate_v1mc()`, скрывающий mac-адрес компьютера.

Итоги и рекомендации

Теперь, когда мы познакомились с различными типами ключей и вариантами их использования, я хочу перечислить мои рекомендации по применению их в ваших базах данных.

Для каждой таблицы:

1. Определите и объявите все естественные ключи.
2. Создайте суррогатный ключ `<table_name>_id` типа `uuid` со значением по умолчанию в `uuid_generate_v1()`. Можете даже пометить его как первичный ключ. Если добавить в этот идентификатор название таблицы, это упростит JOIN, т.е. получите `JOIN foo USING (bar_id)` вместо `JOIN foo ON (foo.bar_id = bar.id)`. Не передавайте этот ключ клиентам и вообще не выводите за пределы базы данных.
3. Для промежуточных таблиц, через которые происходит JOIN, объявляйте все колонки внешних ключей как единый составной первичный ключ.
4. При необходимости добавьте искусственный ключ, который можно использовать в URL или других указаниях ссылки на строку. Используйте сетку Фейстеля или `pg_hashids`, чтобы замаскировать автоинкрементные целые.
5. Указывайте каскадное ограничение `ON UPDATE RESTRICT`, используя суррогатные UUID в качестве внешних ключей, а для внешних искусственных ключей – `ON UPDATE CASCADE`. Выбирайте естественные ключи, исходя из собственной логики.

Такой подход обеспечивает стабильность внутренних ключей, в то же время допуская и даже защищая естественные ключи. К

Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп



Войти

Регистрация

Обсуждать подобные профессиональные вопросы мы предлагаем на наших конференциях. Если у вас за плечами большой опыт в ИТ-сфере, наболело, накопело и хочется высказаться, поделиться опытом или где-то попросить совета, то на майском фестивале конференций РИТ++ будут для этого все условия, 8 тематических направлений начиная от фронтенда и мобильной разработки, и заканчивая DevOps и управлением. Подать заявку на выступление можно [здесь](#).

Теги: [sql](#), [индексы](#)

Хабы: [Блог компании Конференции Олега Бунина \(Онтико\)](#), [MySQL](#), [PostgreSQL](#), [SQL](#), [Администрирование баз данных](#)

↑ +30 ↓ 336 109k 119 Поделиться



Конференции Олега Бунина (Онтико)

Конференции Олега Бунина



332,0

Карма

0,0

Рейтинг

Олег Бунин @olegbunin

Пользователь

Facebook Facebook Facebook

ПОХОЖИЕ ПУБЛИКАЦИИ

29 сентября 2020 в 17:37

Переезжаем на ClickHouse: 3 года спустя

↑ +35 9,6k 54 8

10 мая 2018 в 12:26

Архитектура платежной системы. Банальности, проверенные опытом

↑ +51 28,7k 177 27

8 сентября 2016 в 19:20

Основы индексирования и возможности EXPLAIN в MySQL

↑ +30 24,6k 229 3

Комментарии 119



VMichael 2 февраля 2018 в 21:27

↑ +3 ↓

Честно говоря у меня сложилось впечатление, что начинающим лучше статью не показывать. Запутывает и выводы не однозначные. Например:

1. Определите и объявите все естественные ключи.

Это зачем?

Могу посоветовать почитать книгу Криса Дейта «Введение в системы баз данных».

Объемно, конечно, но весьма познавательно.



menotal 2 февраля 2018 в 23:13

↑ +4 ↓

Описанные "Недостатки последовательных ключей" выглядят слабо.

1. как будто бы помогает бороться с ошибками в написании джойнов. На самом деле ошибочный запрос с гuidaми просто не будет возвращать ничего. Ошибочный запрос с сиквенсами будет возвращать лишнее. И первое не обязательно лучше.
2. Современные СУБД как раз придумали уже кучу всего, чтобы система с сиквенсами нормально масштабировалась
3. Надумано, так как импорт будет в любом случае в отдельную таблицу. Никто в здравом уме не будет вливать внешние данные в текущую структуру без ETL обработки, даже если там уже гуйд как ИД
4. предрассудок в чистом виде. Кроме того, например в Оракле, последовательность ИД как раз таки и не гарантируется. Как раз из-за п.2. Никто по ИД не сортирует.
5. личные тараканы автора



rjhdby 3 февраля 2018 в 12:09



↑ +1 ↓

Надумано, так как импорт будет в любом случае в отдельную таблицу. Никто в здравом уме не будет вливать внешние данные в текущую структуру без ETL обработки, даже если там уже гуйд как ИД

Есть у меня на поддержке приложение, хоть и legacy, но активно дорабатываемое под изменения законодательства. У этого приложения есть БД(Informix) с парой 200-гиговых таблиц с ключами типа SERIAL. Знали бы вы, сколько боли причиняет этот SERIAL при любом чихе в сторону реорганизации структуры данных...



arapasy 3 февраля 2018 в 12:58



↑ 0 ↓

Я могу представить что инкрементный ключ делает сложной или даже бесполезной кластеризации. Но не могу представить какие ещё проблемы он несёт. Поделитесь примерами если это не секрет



VMichael 3 февраля 2018 в 14:13



↑ +2 ↓

SERIAL приносит проблемы, если данные добавляются на разных серверах, а потом сливаются, например в репликации. Тогда приходится выделять диапазон номеров для каждого сервера и отслеживать эту систему. Проще GUID в таком случае. Если БД одна, то каких то особых проблем с ними нет. Бывает ставят условие, что бы был непрерывный счетчик, без «дыр», но это другая проблема.



rjhdby 4 февраля 2018 в 01:39



↑ 0 ↓

Тут скорее кумулятивная проблема. Главный пререквизит, что максимальное окно недоступности не более часа в неделю, невозможность кардинально переписать софт с этими данными работающий и завязка на текущее значение next. Любое разделение данных (а иначе альтер в окно не укладывается) приводит к шаманским пляскам вокруг счетчика



VMichael 4 февраля 2018 в 10:57



↑ 0 ↓

Сложно написали.

Главное, что мне не понятно, зачем вам изменять значение текущих ключей (т.е. полей с типом Serial) и какой альтер вы делаете.

Вашей проблемы в целом не понял.

Опишите подробнее, интересно.



VolCh 4 февраля 2018 в 12:59



↑ 0 ↓

Навскидку кейс: меняем структуру огромной таблицы. alter работает недопустимо долго. Типовое решение: создаём новую структуру как отдельную таблицу, приложение до окончания миграции пишет в обе таблицы, в фоне данные мигрируют, когда догоняют, приложение пишет в новую таблицу, а старая дропается. При такой схеме в идеальном случае минимум один раз нужно будет согласовывать текущие значения счётчиков. А уж если мы не можем по каким-то причинам просто использовать значения из одной таблицы для вставки в другую, используя в режиме двойной записи только одну физическую последовательность, то возникает множество проблем типа в одну таблицу запись прошла, а до другой не дошла. А последовательности часто нетранзакционны,



VMichael 4 февраля 2018 в 14:55



↑ 0 ↓

Да, примерно так и представил, но решил уточнить.

Возникает ситуация, когда пишут в две таблицы, как я и писал выше (там разные серверы, суть тоже).

В таком случае GUID удобнее конечно.

Как вариант, переключайте запись только в новую таблицу, затем перекидывайте данные из старой.

Впрочем, думаю, эти варианты вы продумывали уже, они очевидны.

Двойная запись одного и того же, это зло.

Выделить диапазон номеров для старой таблицы и диапазон номеров для новой таблицы, решение лучше.

Но вы лучше знаете ваш бизнес процесс.

Как вариант, откажитесь от встроенной последовательности и заведите свою, просто таблицу с одной строкой и одним полем, и процедурой, которая возвращает значение и обновляет в поле. Тогда можно навесить транзакционность. Но, как всегда, есть и но. Могут быть с производительностью проблемы, вопрос какой поток данные льется.



avyrupaev 2 февраля 2018 в 23:13



↑ +1 ↓

А как же производительность JOIN на 128-битных uuid, против 64-битных целых?



vdem 3 февраля 2018 в 12:30

↑ +1 ↓

UUID тоже как бы целые числа, только 128-битные, а не 64. Просто отображаются для пользователей как правило в HEX.

P.S. А производительность вряд ли особо страдает. Достаточно загрузить в регистр и проверить старшую/младшую 64-битную часть, если совпадает — проверить остальное.



DrPass 2 февраля 2018 в 23:28

↑ +1 ↓

Первое, что бросилось в глаза, автор несколько путает ключи и ограничения. Например, в `CREATE TABLE tax_brackets` и `CREATE TABLE flight_roster` ключей как таковых нет. То, что здесь указано — это ограничения, а не ключи. Ключ служит для идентификации записи, ограничение — для соблюдения бизнес-правил. Иногда они могут совпадать, иногда нет. Это как раз те примеры, когда для идентификации подходящего естественного ключа нет, нужно суррогатный вводить.

По поводу суррогатного ключа `uuid` vs автоинкремент vs что-то другое, могу сказать своё мнение: если с базой предполагается работа администратора/саппорта напрямую, делайте ключи читабельными. Если не предполагается, смело делайте `uuid`. Человеческое время поддержки намного ценнее, чем время разработки или несколько процентов плюса в бенчмарке.



mayorovp 3 февраля 2018 в 16:02

↑ 0 ↓

В теории реляционных БД есть такая сущность как «естественный ключ». На практике такой ключ должен стать либо первичным ключом, либо ограничением уникальности.



menotal 5 февраля 2018 в 09:58

↑ -1 ↓

Верно, только вот на практике случай, когда «естественный ключ» равен всему набору столбцов — весьма частый.



VolCh 4 февраля 2018 в 13:05

↑ 0 ↓

Часто делаю два суррогатных ключа в таблицах: автоинкремент/последовательность для РК и использования в качестве цели FK в пределах базы(приложения) и `uuid` для интеграций с внешними системами, особенно когда интеграция планируется в режимах мультимастер.



arapasy 3 февраля 2018 в 05:21

↑ 0 ↓

Есть несколько вопросов к автору. Где можно найти определения искусственных но не суррогатных ключей а так же внутренних ключей? Какое отношение имеет ограничение `UNIQUE` к ключам? (Напомню ограничение `UNIQUE` не нарушают повторяющиеся значения значения `NULL`) С моей точки зрения не совсем раскрыта тема что индекс, ключ, суррогатный ключ и ограничение — это вещи практически из разных галактик. Поясню что я имею в виду. Ключ относится к логической организации данных. Суррогатный ключ относится к уровню приложения. Наличие суррогатного ключа не делает базу нормализованной хотя формально первичный ключ вроде бы и есть. Индекс так это способ ускорить выборку. Кстати MySQL на каждый внешний ключ создает дополнительный индекс. А вот PostgreSQL — не создает. И многие удивляются что выборка тормозит. Нужно создавать явно увы. А ограничение это и есть ограничение отменит в случае чего добавление обновление удаление записи.



VolCh 4 февраля 2018 в 13:14

↑ 0 ↓

Суррогатный ключ — это подмножество ключей.

Уточните, пожалуйста, что вы имеете в виду под «логической организацией данных» — это бизнес-модель, о которой, как правило, знает пользователь, или, наоборот, уровень хранилища о котором и приложению знать не особенно надо?



arapasy 3 февраля 2018 в 05:59

↑ 0 ↓

Потребовалось какое-то время, чтобы в реализациях SQL пропало несоответствие ключей и реляционной модели, самые ранние базы данных были заточены под низкоуровневую концепцию первичного ключа. Первичные ключи в таких базах требовались для идентификации физического расположения строки на носителях с последовательным доступом к данным. Вот как это объясняет Джо Селко:

Термин «ключ» означал ключ сортировки файла, который был нужен для выполнения любых операций обработки в последовательной файловой системе. Набор перфокарт считывался в одном и только в одном порядке; невозможно было

«вернуться назад». Первые накопители на магнитных лентах имитировали такое же поведение и не позволяли выполнять двунаправленный доступ. Т.е., первоначальный Sybase SQL Server для чтения предыдущей строки требовал «перемотки» таблицы на начало.

Хотелось бы посмотреть на оригинал или переиздание цитаты. Я ее не нашел в переводе популярной книги Джо Селко (который катит был противникам суррогатных ключей судя по его рекомендациям в этой книге).

Есть два момента. Действительно с 1963 года начали использовать индексный метод доступа к данным ISAM который предвосхитил современные базы данных. Фишкой этого метода было как раз то что в индексе содержались физические адреса данных а доступ к индексу можно было организовать по человекопонятным цифрам (например код товара, ISBN и т.п.) Это я о первом абзаце (слова автора)

Что касается цитаты Джо Селко (я конечно надеюсь узнать о первоисточнике цитаты) Хранение индексных наборов было дорогостоящей операцией а до изобретения НЖМД еще и медленной т.к. для поиска лента перематывалась в разные стороны. Поэтому реляционную модель реализовывали при помощи плоских файлов и их сортировки по ключу. Это позволяло делать расчеты. Приведу пример. Есть два плоских файла остатки товара и прайс. Оба файла сортируются по коду товара и сливаются в один файл. После этого полученный файл сортируется например по материально ответственному лицу и получаются итоги в денежном выражении «в разрезе» как это было принято говорить материально ответственных лиц.



michael_v89 3 февраля 2018 в 09:08



0

Если у объекта отсутствует какой-либо код, то как различить две банки с напитком или две коробки с овсянкой, кроме как по их расположению в пространстве или по небольшим различиям в весе или упаковке?

В физическом мире происходит примерно так. Есть совокупность элементов материи (материальные точки), связанных между собой некими взаимодействиями (векторами). То есть материя организуется в независимые кластеры. При действии внешнего взаимодействия на некоторые точки оно распространяется по векторам в пределах этого кластера. В результате у всех точек меняется поведение. Пример: взять коробку с полки из ряда других.

Кластеры как объекты мы выделяем при наблюдении поведения. У них нет естественных ключей. Можно выделить нечто вроде отношения "равно / не равно", которое показывает, распространится ли взаимодействие между двумя точками или нет, то есть принадлежат ли они одному кластеру или разным. А это основное свойство первичного ключа.

А поскольку кластеры это счетное множество, то можно их пронумеровать. Именно поэтому автоинкрементные ключи удобны в использовании.



апарасу 3 февраля 2018 в 12:39



0

Если говорить о теории реляционных баз данных то двух не различимых строк не может быть суррогат здесь не спасёт. Говоря о конкретном примере например продукт питания то давайте проследим его путь от производства до потребителя. Производитель присваивает изделию номер партии на которую составляет документы о его качестве. Внутри партии коробки с овсянкой неразличимы и выступают в своём количестве. Далее партия или её часть отгружается промежуточному складу или напрямую магазину. так появляется номер накладной. После этого в магазине вы покупаете товар по чеку. И все ещё по прежнему вы можете посмотреть номер партии производителя. Хотя например для вас как для клиента теряется информация о номере накладной.



michael_v89 3 февраля 2018 в 13:02



0

Суррогат как раз спасет, так как именно он и будет различаться.



michael_v89 3 февраля 2018 в 13:05



0

Мне кстати не нравится термин «суррогат». Именно искусственная нумерация первична. Именно она позволяет со 100% вероятностью различить 2 объекта.



апарасу 3 февраля 2018 в 13:14



0

Как простите суррогат который в базе данных сделает различимыми два объекта которые на полке?



michael_v89 3 февраля 2018 в 13:17



0

База это модель. Он моделирует их различие.

Вы не сможете различить 2 одинаковых объекта на полке ни по каким внешним признакам самих объектов.

**арпасу** 3 февраля 2018 в 13:25

0



Да именно. Модель. То есть если объекты в отношении различаются только суррогатным полем то или их различие несущественно и в данном случае должно быть не две строчки коробка, ещё коробка а одна строка — две коробки. Или же нужно выявить их различие. Как будет удаляться коробка которая из?

**michael_v89** 3 февраля 2018 в 13:33

0



Вы не оттуда начинаете. Начинать надо не с базы и отношений, а с моделируемых объектов. Объекты в реальности различаются поведением. 2 разных объекта на полке это уже свершившийся факт, который надо смоделировать.

**арпасу** 3 февраля 2018 в 13:44

0



Тут я с вами согласен. И вот если мы принимаем реляционную модель то нужно её создать по реляционными правилам нормализованной. Тогда получим профит который делает проще операции `crud` — не более того. Тут ни о каком суррогате речь ещё не идёт. Далее мы имеем `orm` или просто библиотеки которые привыкли работать с суррогатов и мы его добавляем как технический приём. К логике модели суррогат не имеет никакого отношения

**michael_v89** 3 февраля 2018 в 14:00

0



Нет. Реляционная модель это не абстрактная магия «надо делать так», а она и появилась для моделирования этих фактов. Если она не позволяет смоделировать 2 разных объекта с одинаковыми характеристиками, значит она применяется неправильно.

Мы некоторым образом различаем 2 объекта на полке, можем сказать, что вот это один объект, а вот это не тот а другой объект. Числовой ключ — это именно модель нашего восприятия объектов. Вернее, модель это отношение равенства чисел. Можно использовать любые другие ключи с таким свойством, набор разных строк например. У самих объектов таких характеристик нет.

**арпасу** 3 февраля 2018 в 14:31

0



Согласен. В правильно смоделированном отношении (таблице) два кортежа (две строки) всегда различаются (без применения суррогата). Весь вопрос в том чтобы правильно выделить объект с точки зрения решаемой задачи.

Что касается «магии нормализации». То я не утверждал что магия существует. Нормализация сильно упрощает операции вставки-изменения-удаления и выборки данных. И не более того. И сама нормализация не является абстракцией а всегда связана с решением конкретной задачи.

**michael_v89** 3 февраля 2018 в 15:46

0



Но у вас нет возможности сделать нормализацию на основе свойств самих объектов. Потому что в данном примере они все одинаковые.

Про магию я имел в виду вывод «мы используем реляционную модель, значит нам надо искать естественные ключи и не использовать искусственные, потому что она так требует по каким-то там магическим причинам». Это не так, потому что все наоборот. Нам нужно смоделировать 2 объекта и способ их различия, для этого можно использовать нумерацию, в реляционной модели нумерация будет являться первичным ключом.

**арпасу** 3 февраля 2018 в 16:22

0



Свойство объекта если перейти на другой уровень абстракции — не только его внутреннее свство а еще может быть и такое свойство как принадлежность к некоторому виду объектов. Которое будет характеризоваться количеством таких объектов. Если мы реализуем реляционную модель то мы не должны искать ключи в объекте в котором их не найдешь. Мы должны правильно выделить объект из окружающего мира. Например для серийного изделия это будет партия.

**michael_v89** 3 февраля 2018 в 17:01

0



Количество — это не свойство объекта, это свойство его контейнера. А контейнер это другой объект.

Ок, вот у нас 2 коробки. В базе есть строка, у которой свойство quantity = 2. Какой у нее первичный ключ?

2 одинаковые коробки и 2 человека с одинаковым именем и внешностью по принципам моделирования ничем не отличаются. Как вы таких людей будете в систему заносить?



VMichael 3 февраля 2018 в 14:22

↑ +1 ↓

Если вам нужно различить для каких то целей два абсолютно одинаковых объекта на полке, то вам придется присваивать еще одно свойство, по которому вы их различите.

Этим свойством как раз и может быть суррогатный ключ. Сформировав его в БД, затем вы, например, печатаете наклейки со штрих кодом и клеите их на ваши объекты. И вот они становятся различимы.

Если у вас такая задача.

Обычно такой задачи нет, и существует количество одинаковых объектов.

Но количество, это уже не свойство данного объекта, а атрибут других сущностей, например табличной части приходной накладной.



арарасы 3 февраля 2018 в 13:11

↑ 0 ↓

Суррогат не касается логики данных. Таблица с суррогатным ключом по прежнему останется таблицей без первичного ключа. Суррогатный ключ это всего лишь технический приём который сильно облегчает работу приложения т.к. избавляет от составных ключей и делает ненужным каскадное обновление при изменении естественного ключа.



michael_v89 3 февраля 2018 в 13:14

↑ 0 ↓

У реальных объектов нет первичного ключа. Всегда может быть 2 объекта с одинаковыми характеристиками.



арарасы 3 февраля 2018 в 13:28

↑ 0 ↓

Два объекта с одинаковыми характеристиками это объекты которые описываются характеристикой р количеством! Если это две книги например одинаковые то при поступлении в библиотеку они для различения нумеруются и тогда они в базе данных две строки. А в магазине это номер накладно с количеством, номер чека с кошечесовом



michael_v89 3 февраля 2018 в 13:39

↑ 0 ↓

То есть, по вашему, 2 близнеца с одинаковым именем могут получить только один паспорт на двоих?

Или наоборот, были 2 разных коробки, различались допустим цветом, в модели у них были разные первичные ключи. Потом краска осыпалась. По-вашему, они должны схлопнуться в один объект с количеством? А с первичными и внешними ключами что делать?

Контейнер с количеством это сущность более высокого уровня, чем объект в этом контейнере.



арарасы 3 февраля 2018 в 13:49

↑ 0 ↓

Пример не корректен. Все рассматривается с точки зрения решаемой задачи. Для какого то класса задач неразличимы люди даже не состоящие в родственных связях



арарасы 3 февраля 2018 в 13:51

↑ 0 ↓

Мы обсуждаемых не объект и сущность а кокрентнг строгую теорию реляционных баз данных.



michael_v89 3 февраля 2018 в 14:04

↑ 0 ↓

Мы обсуждаем общее средство моделирования, используем общие термины типа «сущность» и «первичный ключ». Если теория не позволяет смоделировать какие-то ситуации, значит она подмножество более общей теории.



арарасу

3 февраля 2018 в 15:13



Почему Вы считаете что мы обсуждаем не теорию реляционных баз данных? Статья посвящена именно теории реляционных баз данных и все мои комментарии также касались теории реляционных баз данных. В теории реляционных баз данных нет понятия сущность, объект, модель. Там все строго кортеж, отношение.



michael_v89

3 февраля 2018 в 15:35



Мы обсуждаем моделирование с помощью этих кортежей и отношений. Основным понятием является первичный ключ. Кортеж это аналог понятия сущность.



арарасу

3 февраля 2018 в 15:51



Какую модель данных применять при моделировании это уже другой вопрос. Если задача не укладывается в реляционную модель то она там не применяется. Если применяется то будет профит если придерживаться нормализации данных. Если нормализация нарушается то если это не ошибка проектирования — денормализацию проводят осознанно для решения других задач. (например классический вариант проход расход и баланс баланс является зависимым от прихода и расхода. Тем не менее для выборки когда сумма прихода и расхода вычисляется по терабайтам порой хранят или окончательный баланс или промежуточный как это было в ранних версиях 1с не знаю как сейчас)



michael_v89

3 февраля 2018 в 16:01



Да, только я о том, что ваше понятие о нормализации не совсем верное. Искусственный ключ не нарушает нормализацию.

Моя мысль в том, что у объектов не бывает собственных естественных ключей. Номер договора, номер счета, название из заданного списка — это все заранее придуманные искусственные ключи. Нет никакой проблемы в том, чтобы использовать свой искусственный ключ.

Даже если он у сущности уже придуман, необязательно использовать именно его, достаточно задать соответствие 1:1 с другим ключом. Ограничения сугубо технические, и сводятся к вопросам вида «можем ли мы позволить перестройку unique индекса на вставку, чтобы ускорить чтение по более удобному ключу».



арарасу

3 февраля 2018 в 16:06



Я не говорил что искусственный ключ нарушает нормализацию. Я говорил что суррогатный ключ не делает базу нормализованной. Это не просто перестановка слов.



michael_v89

3 февраля 2018 в 17:04



Что означает "не делает нормализованной" и почему естественный "делает"?



арарасу

3 февраля 2018 в 20:37



Я допустил неточность формулировки. Надо было сказать вместо не делает нормализованной — не делает реляционной. Различимость кортежей это первое требование которое называется целостность сущностей. В отношении где каждый кортеж различим существует по крайней мере один ключ.



michael_v89

3 февраля 2018 в 21:36



Ну так искусственный ключ делает то же самое, причем многие из них уже существуют в предметной области.



DrPass

3 февраля 2018 в 15:37



Там не всё так просто. Есть **теория реляционной модели данных** и есть реляционные базы данных, основанные на этой модели. Это не одно и то же. Реляционная модель данных оперирует кортежами, отношениями и т.д., и это математическая теория. Реляционная БД оперирует таблицами, записями, и это

прикладная инженерная область. И при этом таблица РБД на первый взгляд напоминает отношение в реляционной модели, но на самом деле отношением не является, равно как и строка таблицы не является кортежем. Поэтому надо эти вещи различать. Если мы обсуждаем вопросы по проектированию хранилища для каких-то реальных данных, то надо оперировать понятиями реляционных БД, а не абстрактной реляционной модели.



arapasy 3 февраля 2018 в 16:04



↑ 0 ↓

Реляционную модель данных применяли еще до создания серверов реляционных баз данных и реализовывали на плоских файлах с их сортировкой по ключам. Реляционную модель данных применяют и сейчас в проектах на движках NOSQL баз если например есть преимущества от использования движка базы по его кластеризации и шардированию но данные по природе реляционные. Движки реляционных баз данных современные все без исключения реализуют удобную работу с реляционной моделью данных. Но можно на них естественно реализовать нереляционную модель. Например все в одной таблице типа key/value.



DrPass 3 февраля 2018 в 16:58



↑ 0 ↓

Вы не поняли. Реляционная модель данных не полностью соответствует тому, что реализовано в реляционных БД, независимо от их архитектуры. Например, у вас может быть в базе данных два человека с одинаковым ФИО и без других атрибутов. Это нормально и допустимо (насколько такая архитектура правильна, вопрос за рамками этого обсуждения). А в реляционной модели данных такое в принципе невозможно. Поэтому если речь идет не о математике, а о БД, следует оперировать объектами, сущностями и т.д., а не отношениями/кортежами. Это разные вещи.



arapasy 3 февраля 2018 в 17:04



↑ 0 ↓

Я все понял. С моей точки зрения не может быть неправильной реляционной модели. Есть реляционная модель и не реляционная модель. В базе данных нет объектов сущностей. Там таблицы и строки в которые можно записать все что угодно. Объект, сущность — это уже совсем другой уровень абстракции.



DrPass 4 февраля 2018 в 00:40



↑ 0 ↓

С моей точки зрения не может быть неправильной реляционной модели.

Это вещь, не зависящая от вашей точки зрения :) Есть же определения. Отношение в реляционной модели — это подмножество декартового произведения атрибутов входящих в него кортежей. Соответственно, набор атрибутов, который не выражается как подмножество декартового произведения, отношением не является. Поскольку табличные данные реляционной БД этому в общем случае не соответствуют, то это как раз тот парадокс, когда это на самом деле неправильная реляционная модель. Т.е. она построена в общем на реляционных правилах, но не соответствует им целиком.



arapasy 3 февраля 2018 в 13:59



↑ 0 ↓

Реляционные базы данных это как алгебра

Мы же не пишем в алгебре так

*** + ** = *****



VolCh 4 февраля 2018 в 10:47



↑ +1 ↓

Не может в общем случае. Если у вас у двух объектов, представляющих сущности предметной области, нет различимых характеристик, то или вы недостаточно углубились в предметную область, например, не учитываете факт того, что объекты имеют разные пространственной временные координаты, либо то, что вы выделили в объект является не объектом, а классом объектов в предметной области, а поэкземплярный учёт не нужен в ней.



michael_v89 4 февраля 2018 в 12:02



↑ -1 ↓

Пространственные/временные координаты не являются ключом. Едет грузовик, на полках стоят коробки, подпрыгивают на кочках. Координаты меняются, даже относительно грузовика, и обратившись по запомненным ранее координатам вы можете получить совсем другую коробку. Допустим, на одной полке едет 3 полных коробки и

3 пустых. Какие им можно задать первичные ключи? А еще бывают смешанные жидкости, поля или силы. И точность измерений.


Реальные объекты могут иметь одинаковые характеристики, потому что состоят из одинаковых частиц материи, которые связаны одинаковыми силами и взаимодействуют одинаковыми квантами энергии. Вам тот же вопрос. 2 близнеца с одинаковым именем. По какому первичному ключу их различать?

 **VolCh** 4 февраля 2018 в 13:32     0 

Ключом является то, что позволяет однозначно идентифицировать объект. Есть у нас 6 коробок в грузовике — для каждой из них можно э задать как идентифицирующий признак, например, параметры какой-то функции от времени, в первом приближении начальные координаты в грузовике. Или хранить историю изменения координат в виде набора пар время-координаты.

 **michael_v89** 4 февраля 2018 в 13:46     0 

Это не ответ) Какие именно свойства будут первичным ключом и какие будут 6 значений?

 **VolCh** 4 февраля 2018 в 13:50     0 


Начальные координаты в грузовике.

 **michael_v89** 4 февраля 2018 в 13:57     0 

То есть такие: [(0, 0, 0), (0, 0, 1), (0, 0, 2), (0, 0, 3), ...]
Здравствуй, инкремент.


Ок, я обращаюсь по координатам (0,0,0), ожидаю получить первую полную коробку, но оказывается из-за тряски они поменялись местами и теперь там пустая. Вывод — местоположение это не первичный ключ.

Зато если задать соответствие числа и коробки, все проблемы исчезают. Мы можем отличить 2 коробки независимо от любых характеристик.

 **VolCh** 4 февраля 2018 в 13:48     0 

С близнецами и просто полными тезками, пока не решён вопрос биометрической идентификации, поступать нужно путём введения бизнес-правила, запрещающего давать/брать имена, приводящие к коллизиям в рамках набора идентифицирующих признаков (составного ключа). Это если мы говорим о разработке системы первичного учёта физических лиц а-ля системы ЗАГСа. Кто первый встал — того и тапки. Остальные системы должны уже ссылаться на первичную.

Кстати, насколько я знаю, в СССР и странах, унаследовавших его систему ЗАГС, не дадут зарегистрировать детей с одинаковыми ФИО, датой и местом (населенным пунктом) рождения, поскольку именно эти данные являются идентифицирующими на государственном уровне. С другой стороны, регистрируя близнецов государство (по крайней мере до недавнего времени) доверяло родителям и самим близнецам вопрос их идентификации, не будучи способным проверить их утверждения кто из них кем является.

 **michael_v89** 4 февраля 2018 в 14:01     0 

Но вы ведь идентифицируете 2 близнецов как 2 разных объекта, наблюдая их одновременно. Значит для различения необязательно использование биометрической идентификации.

Нам ведь не нужны конкретные характеристики. Нужен только способ отличить 2 сущности и обратиться к одной из них. Независимые числовые ключи это самый простой и естественный способ.

 **VolCh** 4 февраля 2018 в 14:12     0 

Эта идентификация сугубо локальная, основанная обычно на текущих пространственных координатах или каких-то иных временных характеристиках типа одежды или причёски. Я знаю, что передо мной Таня и Галя, но кто из них кто я могу решить только доверившись авторитетным в данной ситуации источникам. Которые могут и обмануть, и сами находиться в заблуждении.

**kuftachev** 3 февраля 2018 в 14:30

↑ +3 ↓

Статью нужно было сократить раз в 5 минимум и не вводить вилами по воде. (Это конечно к автору, а не к переводчику).

**evocatus** 4 февраля 2018 в 01:39

↑ -1 ↓

Если коротко, то суррогатных ключей следует, по возможности, избегать.

Вот пост разработчика PostgreSQL на эту тему:

www.databasesoup.com/2015/03/primary-keyvil-reprised.html

**DrPass** 4 февраля 2018 в 05:13

↑ +1 ↓

А вы не слушайте этого разработчика, все разработчики PostgreSQL поголовно совсем не обязательно должны разбираться и в проектировании БД. Он неправ. Не надо избегать суррогатных ключей. Наоборот, если вы стоите перед выбором, натуральный ключ или суррогатный, у вас должны быть проверки:

1. Натуральный ключ составной — значит, нафиг его.
2. Данные, которые будут в натуральном ключе, имеют хоть малейшую вероятность изменения — значит, нафиг его.
3. Натуральный ключ громоздкий, и негативно влияет на производительность — значит, нафиг его.
4. Суррогатный ключ можно превратить в используемый в бизнес-процессах идентификатор, опять же таки, нафиг натуральный. Собственно, у вас в реальной жизни будет крайне мало кейсов, где вам будет лучше использовать натуральный ключ.

**evocatus** 4 февраля 2018 в 13:10

↑ 0 ↓

Я почти со всем согласен, но с п.1 не совсем:

не вижу ни одной разумной причины создавать суррогатный ключ для таблицы many-to-many, там самым правильным вариантом мне видится использовать составной естественный ключ, являющийся комбинацией foreign key, которые хранит эта таблица.

Есть, конечно, Django и др. недоразвитые фреймворки, которые заставляют использовать численный id с автоинкрементом в каждой таблице, но у них, как правило, помимо этого столько всяких других недостатков, что я не понимаю как можно их использовать в 2018 году.

**DrPass** 4 февраля 2018 в 14:46

↑ 0 ↓

Согласен, many-to-many — тот самый случай, когда не нужно вводить какой-то идентификатор.

**evocatus** 4 февраля 2018 в 13:17

↑ 0 ↓

Ещё одно соображение: у меня часто бывает, что использование естественных ключей (даже если это VARCHAR(30) или составной ключ) позволяет вообще избежать JOIN'ов, потому что нужная информация уже в ключе.

**DrPass** 4 февраля 2018 в 16:04

↑ 0 ↓

Здесь есть нюанс. Вполне вероятно, что если вы закидываете в ключевое текстовое поле какую-то информацию, присутствующую в дочерних таблицах, которую потом приходится извлекать из ключа, то вам с точки зрения производительности было бы эффективнее просто денормализовать эти данные.

**evocatus** 4 февраля 2018 в 17:29

↑ +1 ↓

Приведу пример: мероприятия проходят в разных помещениях, в таблице locations с помещениями я сделал название помещения (varchar(30)) первичным ключом. Мало того, что оно уникально, так это наверное, самая первая и важная информация, которую мы хотим знать о нём.

И теперь на странице, где находится таблица мероприятий я сразу имею название помещения без всяких JOIN'ов, потому что оно и есть foreign key.

Следует ли это как-то денормализовать?

**arapasy** 4 февраля 2018 в 19:35

↑ 0 ↓

Та небольшие проблемы начнутся если у Вас этот ключ будет участвовать например в расписании мероприятия и будет пару миллионов записей. Когда нужно будет изменить наименование помещения то все будет каскадно довольно долго обновляться в транзакции (а если к базе параллельно сотни тысяч обращений в это время — вообще все очень тормозит).

Поэтому использовать суррогат здесь свмый нормальный вариант. Ну или графориентированные базы данных у которых вы просто задаёте связь и никаких суррогатов или JOIN-ов не нужно.



evocatus 4 февраля 2018 в 20:31



↑ 0 ↓

Даже если изменение названия будет, это по определению редкое событие.

К тому же, если речь идёт о малом бизнесе, где количество событий на помещение измеряется единицами в месяц, а большинство бизнесов не переживают рубеж в 2 года, либо количество данных настолько небольшое, что проблемы с производительностью находятся вообще на другой планете, чем большинство предприятий, которые используют базы данных.



DrPass 5 февраля 2018 в 00:30



↑ 0 ↓

В вашем случае, пожалуй, не стоит. Но это частный кейс, хорошо работающий только на небольших объемах данных.



michael_v89 5 февраля 2018 в 10:24



↑ 0 ↓

Если у вас мало данных/нагрузки и это не представляет проблемы, то зачем JOIN-ов бояться? С ORM это могут быть не JOIN, а отдельные запросы по id, которые проще и результаты можно кешировать. А если нужна дополнительная информация (этаж, вместимость), то все равно придется джойнить или дублировать все в названии.



evocatus 4 февраля 2018 в 13:22



↑ +1 ↓

Насчёт п.2: хоть малейшую вероятность изменения имеет буквально всё. Меняющиеся постоянно требования и условия — тяжёлая реальность и никакой магии, которая бы нас от этого спасла, нет. Во всяком случае это не суррогатные ключи.

Если мы работаем с базами вручную — пишем схемы и запросы сами, то любое изменение характера данных будет очень больно бить, именно поэтому создают ORM.

А если используется ORM, то менять схему и запросы гораздо легче, потому что они генерируются автоматически. В таком случае буквально все аргументы против естественных ключей (кроме производительности — а её надо сначала измерять прежде чем что-то говорить) отпадают. И если использовать нормальный ORM типа SQLAlchemy, то он позволяет и естественные ключи и композитные — всё, что угодно.



michael_v89 4 февраля 2018 в 13:49



↑ 0 ↓

Так не бывает естественных ключей. Все ключи специально придуманные искусственные.



VolCh 4 февраля 2018 в 14:01



↑ 0 ↓

Вроде разница объяснена в посте — ключи, пришедшие извне системы, для системы являются естественными. Если она генерирует их сама — искусственными.



michael_v89 4 февраля 2018 в 14:06



↑ 0 ↓

Ну так это не значит, что своих ключей надо избегать. Если у сущности уже есть удобный искусственный ключ, можно использовать его, если нет, надо придумать самим. Не надо искать уникальный набор атрибутов, это невозможно. На то она и сущность, а не значение. Число 2 одно, а переменных со значением 2 может быть много.



VolCh 4 февраля 2018 в 14:20



↑ 0 ↓

На то она и сущность, что у неё должен быть как набор идентифицирующих её значений, так и набор значений, характеризующих её состояние. Если мы не можем определить идентифицирующие признаки, то, скорее всего, это просто сложное значение, value object например. И отдельный ключ для него создавать надо только в рамках каких-то технических оптимизаций типа нормализации. Или в рамках обхода технических ограничений, не позволяющих, например, эффективно работать с массивами как с одним столбцом.



michael_v89 4 февраля 2018 в 15:18



↑ +2 ↓

Нет. В том то и дело. У сущности есть характеристики состояния, и всё. Больше у нее ничего нет. У любых 2 сущностей одного типа может быть одинаковое состояние. Именно поэтому и навешиваются искусственные ключи, такие как номер счета или инвентарный номер.



VolCh 4 февраля 2018 в 20:15



↑ -1 ↓

У них есть индивидуальность. Сущности с идентичным состоянием — это всё равно разные сущности. Тут есть небольшая нестыковка с традиционным ООП при применении его для моделирования сущностей — и состояние сущности, и её индивидуальность обычно выражаются через состояние объекта, через его свойства. Но, грубо говоря, свойства, которые задаются в конструкторе, не меняются во время жизни и не влияют, обычно, на поведение и в совокупности реализуют индивидуальность, не являются состоянием сущности, хотя являются состоянием объекта.



michael_v89 4 февраля 2018 в 21:21



↑ 0 ↓

Они разные, я о том и говорю. В оперативной памяти они различаются адресом — сторонней независимой характеристикой. В более абстрактном хранилище нужен более абстрактный адрес, для чего хорошо подходит числовой id.



VolCh 4 февраля 2018 в 21:29



↑ 0 ↓

В оперативной памяти объекты, а не сущности. Два объекта с идентичным состоянием представляют одну сущность в одном состоянии. Два объекта с идентичными идентифицирующими свойствами, но разными другими, представляют собой одну сущность в разных состояниях и наоборот, разные сущности в одном состоянии.



michael_v89 4 февраля 2018 в 21:41



↑ 0 ↓

Не-не, я о той ситуации, если бы не надо было в базу сохранять, если бы оперативная память не сбрасывала данные при выключении. Мы бы просто создавали объекты, соответствующие коробкам, и устанавливали одинаковое состояние, соответствующее свойствам этих коробок. Без всяких первичных ключей. Все действия и связи были бы по ссылке на объект. Вот id это то же самое, только уровнем выше.



VolCh 4 февраля 2018 в 23:01



↑ 0 ↓

Ну, в целом, да. Если ещё эта оперативка шарится на весь кластер серверов и всех клиентов и биндинги на все языки. По uuid :)



VolCh 4 февраля 2018 в 14:06



↑ 0 ↓

Пример реально естественного ключа — количество протонов и нейтронов в ядре химического элемента. Формула молекулы вещества. Эти значения объективны, существуют независимо от человека.



michael_v89 4 февраля 2018 в 14:08



↑ 0 ↓

Неа) Это ключ для типа вещества. А молекул с конкретно таким количеством может быть много. Кроме того, он сам по себе числовой и инкрементный.



VolCh 4 февраля 2018 в 14:22



↑ 0 ↓

Не понял, что значит «тип вещества».



michael_v89 4 февраля 2018 в 15:36



↑ 0 ↓

Нет реального объекта «водород» с одним протоном и электроном. Есть много реальных атомов с одним протоном и электроном, проявляющих одинаковые свойства. 2 атома вы не можете различить по этому ключу. А еще атомы могут ионизироваться.

Собираемые типы да, могут основываться на естественных свойствах. Их значения и являются критерием

объединения. Такие типы существуют каждый в единственном экземпляре и образуют перечисление. Для химических элементов значение свойства соответствует номеру в перечислении.



DrPass 4 февраля 2018 в 15:58



↑ +1 ↓

Я думаю, что речь шла о справочнике химических элементов, а не о справочнике атомов Вселенной :)



michael_v89 4 февраля 2018 в 16:51



↑ 0 ↓

Ну, я говорил в контексте того, как различить 2 одинаковых объекта.



evocatus 4 февраля 2018 в 14:23

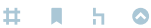


↑ 0 ↓

Про изотопы вы, наверное, слышали?



DrPass 4 февраля 2018 в 15:25



↑ 0 ↓

Изотопы же тоже количеством нейтронов отличаются. Количество протонов определяет элемент, количество нейтронов — изотоп.



VolCh 4 февраля 2018 в 21:25



↑ 0 ↓

поэтому написал про составной ключ протоны+нейтроны



DrPass 4 февраля 2018 в 15:56



↑ +1 ↓

Насчёт п.2: хоть малейшую вероятность изменения имеет буквально всё

Абсолютно верно. Вот поэтому суррогатный ключ чаще всего предпочтительней. Ключ «Код клиента КЛН-123456» плюс констрейнт «Номер паспорта — уникальный» всяко лучше, чем ключ «Номер паспорта AA 654321»



evocatus 4 февраля 2018 в 17:32



↑ 0 ↓

Я думаю практически все согласны с тем, что для таблицы людей нужно вводить суррогатные ключи. А вот дальше... например, я делаю возможность некоторым людям выступать не только в роли записей в таблице, но и логиниться на сайте и что-то делать. Создал таблицу users, которая имеет поле person_id, ссылающееся на суррогатный первичный ключ таблицы людей. Но этот же ключ используется как первичный и для этой таблицы — зачем создавать ещё один?



VolCh 4 февраля 2018 в 23:03



↑ 0 ↓

А потом у вас появятся коллективные пользователи, а также «боты».



DrPass 5 февраля 2018 в 00:25



↑ 0 ↓

Вот табличка users — это как раз тот редкий случай, когда натуральный ключ вполне уместен в качестве первичного. Я бы там сделал первичным ключом логин. Очень удобно в различных полях, связанных с настройками или журналированием действий пользователей, видеть их логины, а не person_id. Заодно и решается проблема, какой person_id завести для пользователя, который является собой бездушного робота, выполняющего какие-то регламентные задачи или синхронизацию с чем-то в вашей базе.



VolCh 4 февраля 2018 в 14:24



↑ 0 ↓

1. А как ссылочную целостность контролировать и обеспечивать?



DrPass 4 февраля 2018 в 15:27



↑ 0 ↓

Не понял сути вопроса, если честно. Так же, как и обычно, какая разница, составной ключ или нет?



VolCh 5 февраля 2018 в 11:22



↑ -1 ↓

Есть, например, составной естественный ключ химического элемента из количества протонов и нейтронов. При попытках указания элемента где-то снаружи таблицы можно сразу определить есть такая запись или нет, имея гарантию, что сочетание протонов и нейтронов уникально. Если первичным ключом делаете какой-нибудь автоинкремент или uuid, то для обеспечения ссылочной целостности вам нужно будет как-то контролировать и обеспечивать, что записи в других таблицах вида "2 протона 3 нейтрона" ссылаются на существующую запись в таблице элементов.



mayorovp 5 февраля 2018 в 11:24



↑ 0 ↓

А нахрена в других таблицах писать «2 протона 3 нейтрона» если для этой цели первичный ключ есть?



VolCh 5 февраля 2018 в 11:53



↑ 0 ↓

Потому что это естественный первичный ключ, а не введенный с какими-то целями суррогатный.



mayorovp 5 февраля 2018 в 11:56



↑ 0 ↓

Нет, после ввода суррогатного ключа естественный ключ более не является первичным. В частности, это означает, что для ссылок на эту таблицу следует использовать именно суррогатный ключ.



VolCh 5 февраля 2018 в 12:58



↑ 0 ↓

Усложнение архитектуры получается и нарушение инкапсуляции. Клиенту мало знаний об естественном идентификаторе какой-то бизнес сущности, ему ещё нужно получить её суррогатный идентификатор, в бизнес-процессах никого не интересующий.



mayorovp 5 февраля 2018 в 13:03



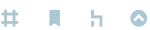
↑ 0 ↓

Клиенту знать суррогатный идентификатор не обязательно, его узнаванием вполне может заняться слой доступа к данным или любой другой.

А инкапсуляция — это не про РСУБД, у РСУБД другие ценности.



VolCh 5 февраля 2018 в 18:44

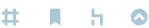


↑ 0 ↓

Смысл делать сугубо внутренний суррогатный идентификатор, если основной операцией будет извлечение или изменение данных по естественному внешнему ключу?



mayorovp 5 февраля 2018 в 19:10



↑ 0 ↓

Обычно так получается проще.



DrPass 5 февраля 2018 в 14:17



↑ 0 ↓

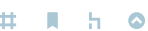
что записи в других таблицах вида «2 протона 3 нейтрона» ссылаются на существующую запись в таблице элементов.

В общем случае таких записей там быть не должно, это уже денормализация данных без оснований. Если же основания есть, то это будет составной внешний ключ, а в основной таблице там будет констрейнт на уникальность.

В конкретно вашем примере лучшим первичным ключом напрашивается как раз наименование элемента по таблице Менделеева, в нотации, как это принято, с атомной массой для изотопов.



VolCh 5 февраля 2018 в 18:46



↑ 0 ↓

На сколько я помню школу, то номер ячейки в таблице Менделеева однозначно определяет количество протонов в ядре, а количество нейтронов плавающее. Откуда тут денормализация?

**DrPass** 5 февраля 2018 в 22:26 # 0

↑ -1 ↓

На сколько я помню школу, то номер ячейки в таблице Менделеева однозначно определяет количество протонов в ядре, а количество нейтронов плавающее

Нотация этот вопрос уже давно решила, H — водород, а ^2H — дейтерий. Для первичного ключа, который содержит всю необходимую информацию, этого достаточно.

Откуда тут денормализация?

По факту, вы **группу свойств** описываемого объекта из предметной области предлагаете использовать для его идентификации, и дублировать в разных наборах данных. Это чистой воды денормализация.

**VolCh** 6 февраля 2018 в 14:47 # 0

↑ 0 ↓

Чем это отличается от 1-0 и 1-1? По сути лишь разные формы нотаций одного и того же естественного ключа. При том, что "моя" является более простым преобразованием, не требует таблиц преобразования.

Это не я предложил, а Менделеев, насколько я понимаю, идентифицировав даже не обнаруженные на то время элементы.

**DrPass** 6 февраля 2018 в 23:26 # 0

↑ 0 ↓

Чем это отличается от 1-0 и 1-1

Примерно тем же, чем IVANOFF отличается от {D12BEB59-6259-4FA1-A733-ADCD523D72DC}. Первое блюдо готово к употреблению, второе лишь полуфабрикат. Для использования нужно лезть в таблицу, искать и выводить «человеческое описание».

**VolCh** 7 февраля 2018 в 11:12 # 0

↑ 0 ↓

Зависит от задач. Я так же могу сказать, что чтобы узнать количество протонов и нейтронов по нотации типа 2H , мне нужно лезть в таблицу. По сути такие нотации и есть суррогатные ключи поверх естественных, в буквальном смысле слова — данных природой. Причём изначальная нотация, чисто буквенная, оказалась недостаточной для практических задач, когда выяснилось, что количество нейтронов в ядре, масса ядра, не выводятся однозначно из количества протонов.

**potapuff** 5 февраля 2018 в 10:38 # 0

↑ 0 ↓

Логика статьи понятна. Помогите разобраться с терминологией. Всю жизнь думал суррогатный ключ — искусственно созданный ключ для упрощения доступа (например serial или uuid вместо составного естественного ключа).

И тут, внезапно, наткнулся на объяснение Тома Кайта, почему в Оракле нет on update:

«Primary keys are supposed to be immutable, never changing, constant. It is an excessively bad practice to have to update them ever. If there is a 0.00001% chance you will have to update a primary key — then **it is not a primary key, its a surrogate key** and you need to find the true primary key (even if you have to make it up via a sequence)» (asktom.oracle.com/pls/asktom/f?

p=100:11:0:::P11_QUESTION_ID:5773459616034)

Здесь, просто «surrogate key» не «суррогатный ключ», что-то другое?

**VolCh** 5 февраля 2018 в 11:10 # -1

↑ -1 ↓

Есть два, как минимум, контекста, когда речь заходит о первичных, альтернативных, суррогатных и прочих ключах: контекст модели данных приложения и контекст модели хранения. В обоих контекстах понятия разных ключей схожи, но в первом ключи идентифицируют сущности, а во втором — записи.

С точки зрения базы, идентификатор записи не должен изменяться никогда, а если бизнес-процессы допускают изменение идентификатора сущности (например опечатка при вводе ИНН или возможность смены логина пользователем), то этот идентификатор не может служить идентификатором записи, для базы он искусственно созданный для упрощения кому-то (не базе) ключ.

С точки зрения же приложения, бизнес-модели при наличии естественного идентификатора, который в некоторых случаях может меняться, суррогатным будет ключ который в базе никогда меняться не будет, о связи которого с первичным ключом приложения

приложению нужно заботиться отдельно.

**potapuff**

5 февраля 2018 в 11:27



0

Перефразирую вопрос.

У Тома фраза звучит как: суррогатный ключ — не настоящий ключ, а ерунда на постном масле. В этом же предложении есть про первичный ключ на основе последовательности, как один из вариантов определения «настоящего» первичного ключа. А в терминах хабровской статьи первичный ключ на основе последовательности — это суррогатный ключ.

Возможно, просто, наложились термин «суррогатный ключ» и какой-то альтернативный перевод слова «surrogate».

**michael_v89**

5 февраля 2018 в 12:37



0

Там немного в другом смысле употребляется. Общепринятое мнение таково, что суррогатный ключ — это ключ, не являющийся свойством сущности в предметной области и выдуманный при проектировании БД, и что это плохо. Но оно неправильное. При этом путаются 2 ситуации — когда уже в самой предметной области у сущности есть суррогатный ключ, и когда его нет. В первом случае он почему-то считается обычным свойством реального объекта. Поэтому во втором случае возникают проблемы, когда пытаются применить тот же подход и найти уникальное сочетание свойств. On update одна из них. Если в предметной области нет ключа, его надо придумать самим, а не выискивать заведомо ненастоящий ключ. Например пронумеровать сущности. Формулировка в цитате выглядит кривовато, но в целом верно.

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки

Неделя

Месяц

Почему Джефф Безос – самый опасный политикан на планете

↑ +59

👁 58,2k

📖 150

💬 479

Дуров рассказал, как будет монетизироваться Telegram

↑ +25

👁 20,8k

📖 7

💬 45

Добраться до Марса: новые семь минут ужаса всего через семь дней

↑ +70

👁 21,6k

📖 35

💬 75

Украденные данные CD Projekt Red выставили на аукционе

↑ +21

👁 37,1k

📖 13

💬 60

Расскажем о soft skills всё, что можно

Подборка

Войти / аккаунт	Разделы	Информация	Услуги
Регистрация	Новости	Для авторов	Тарифы
	Хабы	Для компаний	Контент
	Компании	Документы	Семинары
	Пользователи	Соглашение	Мегапроекты
	Песочница	Конфиденциальность	Мерч