нетология

Реляционные базы данных: **Базы данных.**

Олег
Гежин

Олег Гежин

Инженер-программист, системный администратор УГМК

Модуль «Реляционные базы данных»

Цели модуля:

- Узнать принципы работы реляционных баз данных;
- Научиться писать SQL-запросы к базе данных;
- Научиться работать с индексами и оптимизировать выполнение запросов;
- Освоить репликацию и масштабирование баз данных;
- Научиться делать резервное копирование.

Структура модуля

1. **Базы данных.**
2. Работа с данными (DDL/DML).
3. SQL.
4. Индексы.
5. Репликация и масштабирование.
6. Резервное копирование.
7. Базы данных в облаке
   (на примере Яндекс.Облако)

План занятия

1. Реляционная модель
2. Ограничения
3. Нормализация
4. Денормализация
5. Типы данных
6. Итоги
7. Домашнее задание

Реляционная модель

Давайте вспомним предыдущие занятия

- Реляционная модель представляет собой фиксированную структуру математических понятий, которая описывает, как будут представлены данные;
- Базовой единицей данных в пределах реляционной модели является таблица;
- Таблица — это базовая единица данных. В реляционной алгебре она называется «отношение» (relation). Состоит из атрибутов (columns), которые определяют конкретные типы данных. Данные в таблице организованы в кортежи (rows), которые содержат множества значений столбцов.

Реляционная модель

Преимущества:

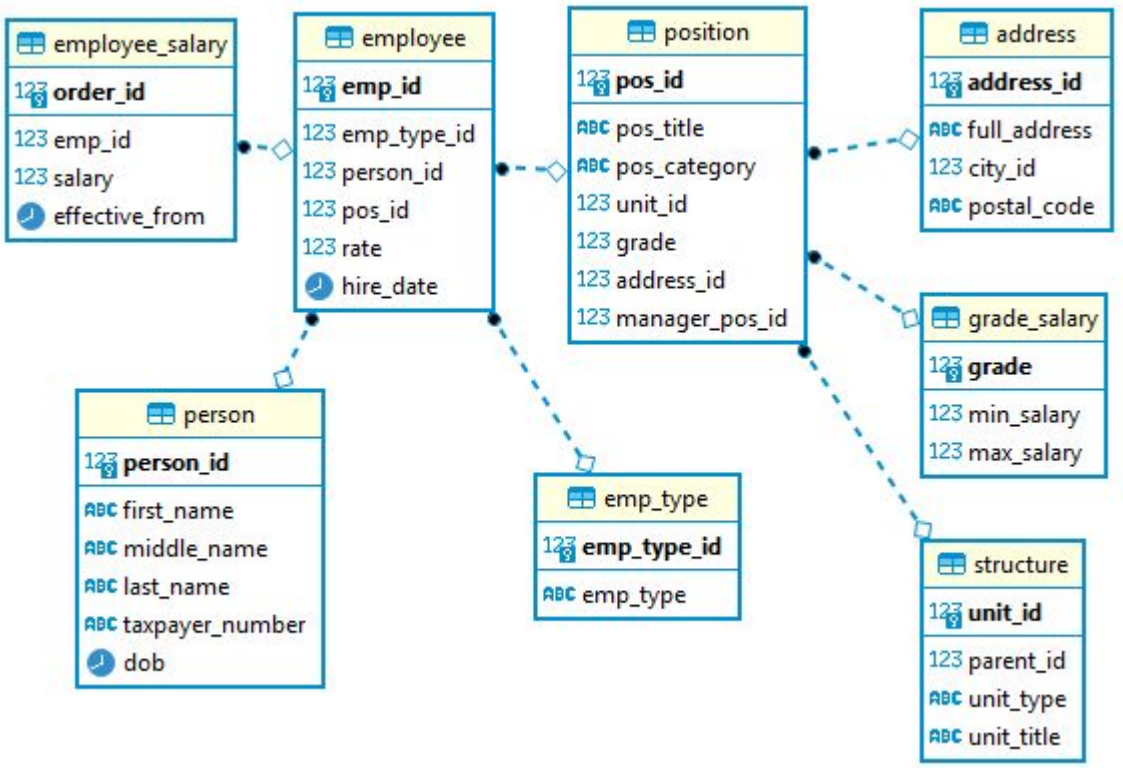
- Эффективное поддержание целостности данных;
- Блокировка и очередность доступа к данным;
- Атомарность данных (возможность использования сложных типов данных);
- Поддержка процедурных языков;
- Независимость физической и логической моделей.

Реляционная модель

Реляционная модель данных — созданная Эдгаром Коддом логическая модель данных, описывающая:

- структуры данных в виде наборов отношений;
- теоретико-множественные операции над данными: объединение, пересечение разность и декартово произведение;
- специальные реляционные операции: селекция, проекция, соединение и деление;
- специальные правила, обеспечивающие целостность данных.

Реляционная модель



Ограничения

Первичные ключи

При создании таблицы могут быть использованы различные «ограничения» (CONSTRAINTS), которые содержат правила, указывающие, какие данные представлены в ней.

Одним из самых используемых ограничений является первичный ключ (PRIMARY KEY), который гарантирует, что каждая строка таблицы содержит уникальный идентификатор.

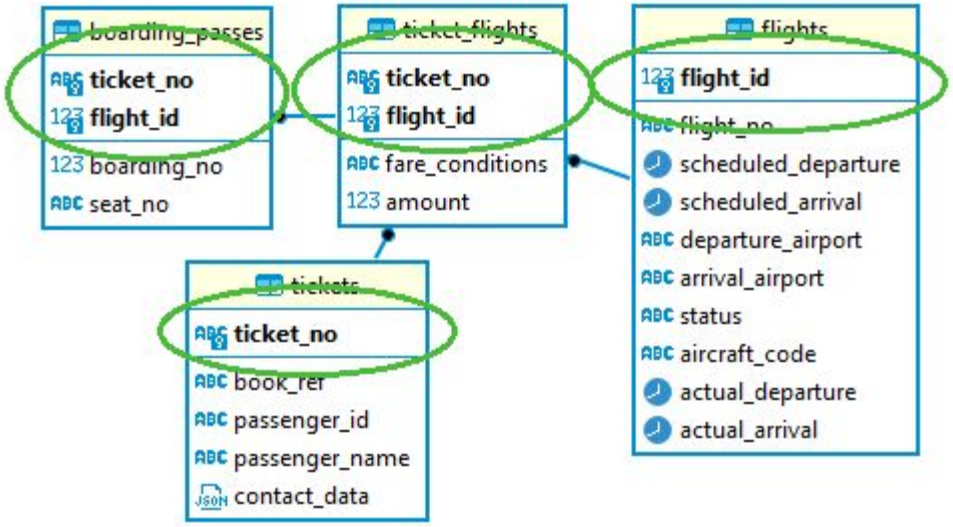
Правильным считается наличие первичного ключа во всех таблицах базы данных.

PRIMARY KEY = UNIQUE + NOT NULL + INDEX

Первичные ключи

Первичный ключ может состоять из одного или нескольких столбцов.

Первичные ключи, состоящие из нескольких столбцов называются «**составными**» (COMPOSITE).



Натуральные первичные ключи

Представляют собой данные, которые уже присутствуют в описываемой предметной области. Например, почтовые индексы могут быть использованы как естественные первичные ключи без дополнительной обработки.

Их использование, если оно возможно, считается более правильным, чем искусственных.

В справочнике стран натуральным первичным ключом может быть ISO код стран.

В справочнике граждан РФ натуральным составным первичным ключом может быть серия и номер паспорта.

Суррогатные первичные ключи

Представляют собой целочисленный идентификатор.

Применяется там, где нет возможности использовать натуральный первичный ключ. Позволяют решать те же практические задачи, что и естественные: улучшение производительности памяти и индексов при операциях обновления.

Внешние ключи

В то время как одна таблица имеет первичный ключ, другая таблица может иметь ограничение, описывающее, что её значения ссылаются на гарантированно существующие значения в первой таблице.

Это реализуется через создание в «дочерней» таблице столбца (может быть несколько столбцов), значениями которого являются значения первичного ключа из «родительской» таблицы.

Внешние ключи

Вместе наборы этих столбцов составляют внешний ключ (FOREIGN KEY), который является механизмом базы данных, гарантирующим, что значения в «дочерних» столбцах присутствуют как первичные ключи в «родительских».

Это ограничение контролирует все операции на этих таблицах:

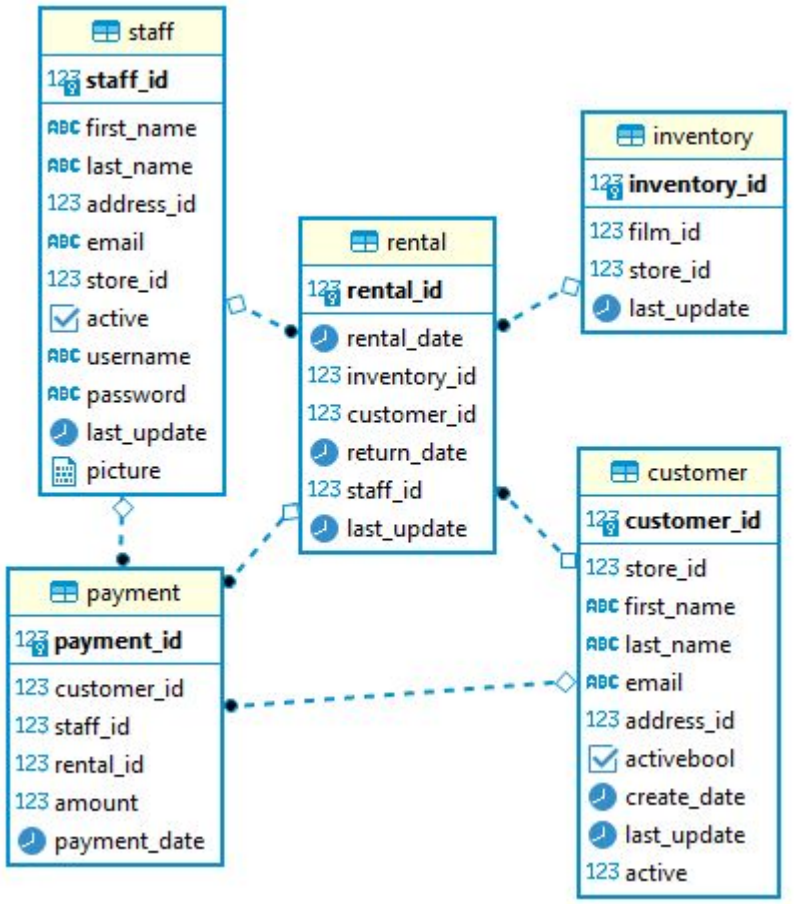
- добавление / изменение данных в «дочерней» таблице
- удаление / изменение данных в «родительской» таблице.

Внешний ключ проверяет, чтобы данные корректно присутствовали в обеих таблицах. Иначе операции будут отменены.

Внешние ключи

Внешние ключи могут быть составными.

Как правило «родителем» во внешнем ключе является первичный ключ, но при необходимости «родителем» может быть любой столбец, который имеет ограничение уникальности (UNIQUE).



UNIQUE

Ограничение UNIQUE:

- гарантирует, что все значения в столбце будут уникальными.
- может быть наложено на один или несколько столбцов.

В каждой таблице может быть несколько уникальных ограничений.

Ограничение первичного ключа имеет ограничение UNIQUE по умолчанию.

NOT NULL

Изначально столбец в таблице может содержать любые значения, включая «нулевые» значения, то есть пустые значения.

Ограничение NOT NULL делает так, что в столбец нельзя записать «нулевые» значения.

Таким образом, поле всегда будет содержать значение и при попытке внести пустое значение при добавлении или изменении записи будет выбрасывать ошибку.

Ограничение первичного ключа имеет ограничение NOT NULL по умолчанию.

CHECK

Ограничение CHECK:

- используется для ограничения диапазона значений, который может быть записан в столбец.
- может быть указано для одного столбца и допускает только определенные значения для этого столбца.
- может быть указано для таблицы, таким образом оно ограничивает значения в определенных столбцах на основе значений в других столбцах строки.

Нормализация

Нормализация

Нормализация — это метод проектирования базы данных, который используется для разработки таблицы реляционной базы данных до более высокой нормальной формы.

При этом данный процесс является прогрессивным, и более высокий уровень нормализации базы данных не может быть достигнут, если не были выполнены предыдущие уровни.

Исходные данные

Представим условный онлайн магазин и предположим, что кто-то создал таблицу с данными о пользователе:

```
CREATE TABLE customer (
    customer_name varchar(50) NOT NULL,
    customer_login varchar(20) NOT NULL,
    customer_password varchar(10) NOT NULL,
    customer_address varchar(50) NOT NULL,
    delivery_address varchar(50) NOT NULL,
    purchases JSON, --{id_продукта: количество}
    amount decimal(10, 2),
    wish_list text[]
)
```

Исходные данные

В результате будет вот такая таблица:

	customer_name	customer_login	customer_password	customer_address	delivery_address	purchases	amount	wish_list
1	Федор Иванович	fivan123	1q2w3e	London, Near BigBen	Freedom str, 56	{"23":4, "17":2, "15":3}	1 800,35	{1,14,3}
2	Федор Иванович	fivan123	1q2w3e	London, Near BigBen	Freedom str, 56	{"14":3}	150,22	{1,3}
3	Петя Галушкин	numberone	556677	Москва, Научный проезд 23	Ленинский проспект 80	{"11":1, "12":1, "73":1}	350	{5}
4	Петя Галушкин	numberone	556677	Москва, Научный проезд 23	Расплетина 33	{"51":99}	5 000	{16,28}
5	Петя Галушкин	numberone	556677	Москва, Научный проезд 23	Широкая 12/3	{"16":3, "28":6}	1 800,35	{16,28}

Скорее всего, с этим будет не совсем удобно работать...

1 нормальная форма

Чтобы таблица удовлетворяла 1НФ, значения в каждом столбце должны быть **атомарными**. То есть, значения в домене каждого атрибута отношения не являются ни списками, ни множествами простых или сложных значений.

Столбцы *purchases* и *wish_list* хранят множества данных, а в столбце *customer_address* можно выделить город проживания.

Давайте проведем нормализацию.

1 нормальная форма

Создадим несколько новых таблиц.

- Города:

	city_id	city_name
1	1	London
2	2	Москва

- Список пожеланий:

customer_id	product_id
1	1
1	3
2	16
2	28

- Продажи:

purchase_id	customer_name	customer_login	customer_password	product_id	quantity
1	Федор Иванович	fivan123	1q2w3e	23	3
2	Федор Иванович	fivan123	1q2w3e	17	2
3	Федор Иванович	fivan123	1q2w3e	15	3
4	Федор Иванович	fivan123	1q2w3e	14	3
5	Петя Галушкин	numberone	556677	11	1
6	Петя Галушкин	numberone	556677	12	1
7	Петя Галушкин	numberone	556677	73	1
8	Петя Галушкин	numberone	556677	51	99
9	Петя Галушкин	numberone	556677	16	3
10	Петя Галушкин	numberone	556677	28	6

1 нормальная форма

И таблица *customer* будет выглядеть следующим образом:

customer_name	customer_login	customer_password	city_id	customer_address	delivery_address
Федор Иванович	fivan123	1q2w3e	1	Near BigBen	Freedom str, 56
Федор Иванович	fivan123	1q2w3e	1	Near BigBen	Freedom str, 56
Петя Галушкин	numberone	556677	2	Научный проезд 23	Ленинский проспект 80
Петя Галушкин	numberone	556677	2	Научный проезд 23	Расплетина 33
Петя Галушкин	numberone	556677	2	Научный проезд 23	Широкая 12/3

Теперь мы можем сказать, что наша таблица удовлетворяет требованиям 1НФ. Так же как и созданные дополнительные три таблицы.

2 нормальная форма

Таблица обязана соответствовать первой нормальной форме.

Все столбцы, которые не являются частью ключа, зависят от этого ключа. Чтобы соответствовать 2НФ и удалить дубликаты, каждый неключевой атрибут должен зависеть от всего ключа, а не только от его части.

Можно обратить внимание, как нехорошо выглядит таблица с продажами, надо вносить изменения в структуру продаж и менять первичный ключ.

2 нормальная форма

В таблице *customer* создадим идентификатор, который сделаем первичным ключом.

customer_id	customer_name	customer_login	customer_password	city_id	customer_address	delivery_address
1	Федор Иванович	fivan123	1q2w3e	1	Near BigBen	Freedom str, 56
2	Федор Иванович	fivan123	1q2w3e	1	Near BigBen	Freedom str, 56
3	Петя Галушкин	numberone	556677	2	Научный проезд 23	Ленинский проспект 80
4	Петя Галушкин	numberone	556677	2	Научный проезд 23	Расплетина 33
5	Петя Галушкин	numberone	556677	2	Научный проезд 23	Широкая 12/3

Таким образом можно привести таблицу с продажами к нормальному виду:

purchase_id	customer_id	product_id	quantity
1	1	23	3
2	1	17	2
3	1	15	3
4	1	14	3
5	2	11	1
6	2	12	1
7	2	73	1
8	2	51	99
9	2	16	3
10	2	28	6

3 нормальная форма

Таблица обязана соответствовать второй нормальной форме.

Значения, входящие в запись и не являющиеся частью ключа этой записи, не принадлежат таблице.

В таблице *customer* осталась информация по адресу доставки, которая относится к продажам.

Создадим таблицу, в которую будем записывать адреса для доставки и уберем информация из *customer*.

3 нормальная форма

Таблица с пользователями:

customer_id	customer_name	customer_login	customer_password	city_id	customer_address
1	Федор Иванович	fivan123	1q2w3e	1	Near BigBen
2	Петя Галушкин	numberone	556677	2	Научный проезд 23

Таблица по доставке. Обратите внимание, что добавили информацию по городу:

delivery_id	customer_id	city_id	address
1	1	1	Freedom str, 56
2	1	1	Freedom str, 56
3	2	2	Ленинский проспект 80
4	2	2	Расплетина 33
5	2	2	Широкая 12/3

3 нормальная форма

И теперь необходимо привязать идентификатор доставки к продажам:

purchase_id	customer_id	product_id	quantity	delivery_id
1	1	23	3	1
2	1	17	2	1
3	1	15	3	1
4	1	14	3	2
5	2	11	1	3
6	2	12	1	3
7	2	73	1	3
8	2	51	99	4
9	2	16	3	5
10	2	28	6	5

Нормальная Форма Бойса-Кодда

Реляционная схема считается в нормальной форме Бойса-Кодда (НФБК), если для каждой из ее зависимостей A → B выполняется одно из следующих условий:

- A → B является тривиальной функциональной зависимостью (то есть B является подмножеством A);
- A — первичный ключ для схемы реляционной схемы.

То есть, если таблица находится в 3НФ и все ее столбцы являются частью составного первичного ключа, то эта таблица находится в НФБК.

НФБК — это расширенная 3НФ.

Нормальная Форма Бойса-Кодда

Как правило 3НФ является желаемым результатом и дальнейшая нормализация может приводить к ненужному результату, из-за которого усложняется выборка данных.

Давайте разделим таблицу по продажам на две:

purchase_id	product_id	quantity
1	23	3
2	17	2
3	15	3
4	14	3
5	11	1
6	12	1
7	73	1
8	51	99
9	16	3
10	28	6

purchase_id	delivery_id
1	1
2	1
3	1
4	2
5	3
6	3
7	3
8	4
9	5
10	5

4 нормальная форма

4 нормальная форма применяется для устранения многозначных зависимостей — таких зависимостей, где столбец с первичным ключом имеет связь один-ко-многим со столбцом, который не является ключом. Эта нормальная форма устраняет некорректные отношения многие-ко-многим.

4 нормальная форма

Предположим, что из-за большого количества заказов пришлось открыть несколько складов в разных городах и мы создали таблицу, которая хранит количество товара на каждом складе:

store_id	product_id	city_id	amount
1	2	1	100
1	3	1	200
1	15	1	300
2	2	2	1 230
2	17	2	80
2	89	2	171

Так как в этой таблице составной первичный ключ (store_id, product_id, city_id), она находится в НФБК.

4 нормальная форма

Для того чтобы удовлетворить требования 4 нормальной формы, разделим таблицу по складам:

store_id	city_id
1	1
2	2

store_id	product_id	amount
1	2	100
1	3	200
1	15	300
2	2	1 230
2	17	80
2	89	171

5 нормальная форма

5 нормальная форма разделяет таблицы на более малые таблицы для устранения избыточности данных. Разбиение идет до тех пор, пока нельзя будет воссоздать оригинальную таблицу путем объединения малых таблиц.

5 нормальная форма

Таблица с продажами в НФБК выглядела следующим образом:

store_id	product_id	city_id	amount
1	2	1	100
1	3	1	200
1	15	1	300
2	2	2	1 230
2	17	2	80
2	89	2	171

Мы ее разбили на две таблицы и пусть каждый склад работает на несколько городов:

store_id	city_id
1	1
2	2
1	3
2	4

store_id	product_id	amount
1	2	100
1	3	200
1	15	300
2	2	1 230
2	17	80
2	89	171

5 нормальная форма

Если мы выполним join двух малых таблиц, то получим следующий результат:

store_id	city_id	product_id	amount
1	1	2	100
1	3	2	100
1	1	3	200
1	3	3	200
1	1	15	300
1	3	15	300
2	2	2	1 230
2	4	2	1 230
2	2	17	80
2	4	17	80
2	2	89	171
2	4	89	171

Где темно-синим выделен ложный результат.

5 нормальная форма

Давайте разобьем исходную таблицу на три:

store_id	city_id
1	1
2	2
1	3
2	4

store_id	product_id	amount
1	2	100
1	3	200
1	15	300
2	2	1 230
2	17	80
2	89	171

city_id	product_id
1	2
1	3
1	15
2	2
2	17
2	89

Необходимо помнить, что при извлечении информации (например, о городе и товарах) необходимо в запросе соединить все три отношения. Любая комбинация соединения двух отношений из трех неминуемо приведет к извлечению ложной информации.

5 нормальная форма

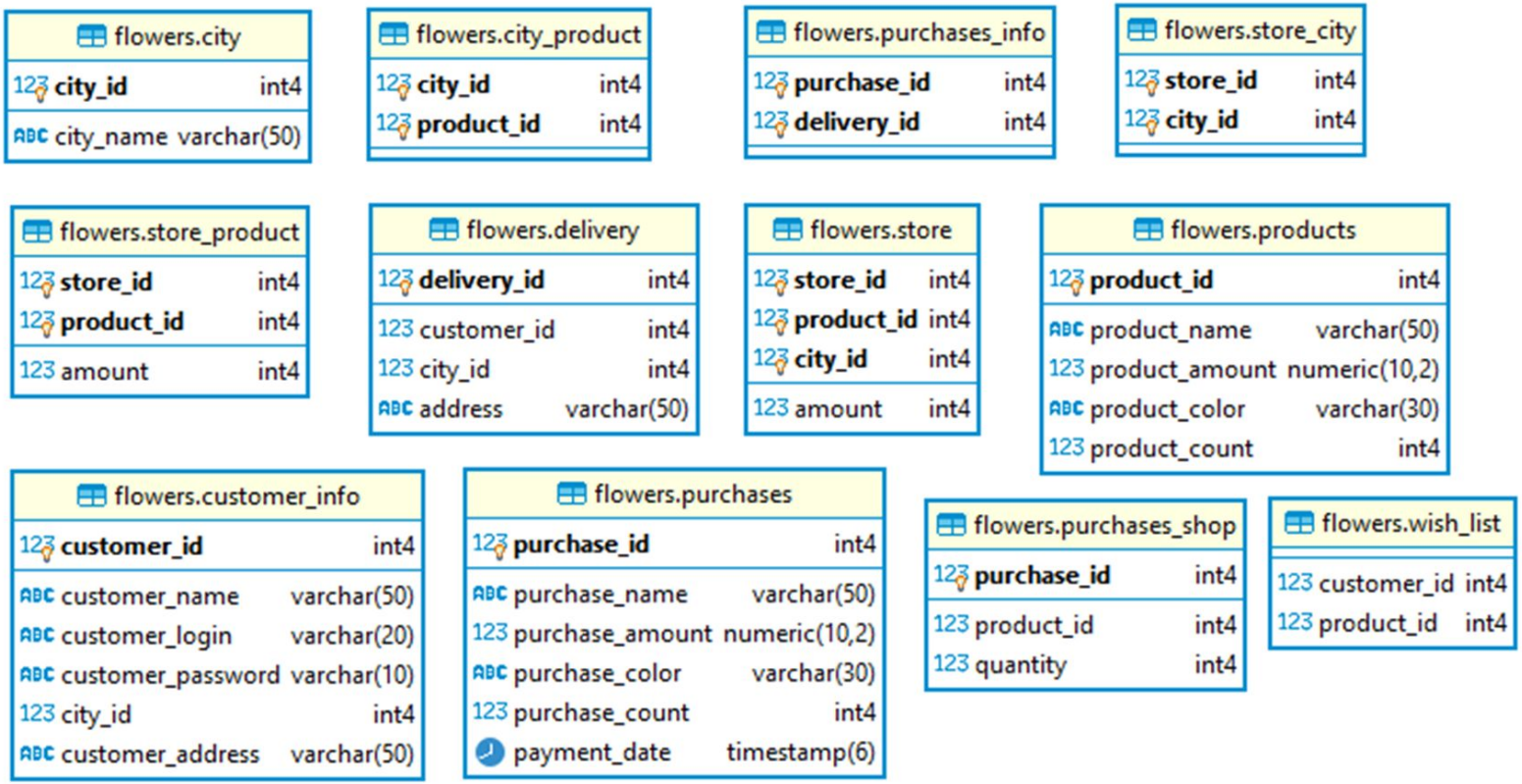
Проверим на практике:

```
SELECT sc.store_id,
       sc.city_id,
       sp.product_id,
       sp.amount
FROM store_city sc
JOIN store_product sp ON
sp.store_id = sc.store_id
JOIN city_product cp ON
cp.city_id = sc.city_id
 AND cp.product_id =
 sp.product_id
```

store_id	city_id	product_id	amount
1	1	2	100
1	1	3	200
1	1	15	300
2	2	2	1 230
2	2	17	80
2	2	89	171

Соответственно наши таблицы удовлетворяют пятой нормальной форме.

Результат


flowers.city
city_id int4
city_name varchar(50)
flowers.city_product
city_id int4
product_id int4
flowers.purchases_info
purchase_id int4
delivery_id int4
flowers.store_city
store_id int4
city_id int4
flowers.store_product
store_id int4
product_id int4
amount int4
flowers.delivery
delivery_id int4
customer_id int4
city_id int4
address varchar(50)
flowers.store
store_id int4
product_id int4
city_id int4
amount int4
flowers.products
product_id int4
product_name varchar(50)
product_amount numeric(10,2)
product_color varchar(30)
product_count int4
flowers.customer_info
customer_id int4
customer_name varchar(50)
customer_login varchar(20)
customer_password varchar(10)
city_id int4
customer_address varchar(50)
flowers.purchases
purchase_id int4
purchase_name varchar(50)
purchase_amount numeric(10,2)
purchase_color varchar(30)
purchase_count int4
payment_date timestamp(6)
flowers.purchases_shop
purchase_id int4
product_id int4
quantity int4
flowers.wish_list
customer_id int4
product_id int4

Нормализация

При проектировании базы можно избегать какие-либо формы нормализации. Если изначально есть понимание разделения сущностей и связей по своим местам, то можно пропустить начальные формы нормализации.

Также при исправлении нарушений одной нормальной формы можно заранее учесть нарушения более высокой формы.

Рассматривать ETNF (основного домена), DKNF (ключа домена), 6 нормальную форму не будем, так как они носят больше научный характер и на практике не применяются.

Функциональные зависимости

Функциональные зависимости — это основа нормализации баз данных.

Под функциональной зависимостью подразумевается зависимость значения одного атрибута от другого.

Если даны два атрибута А и Б некоторого отношения, то говорят, что Б функционально зависит от А, если в любой момент времени каждому значению А соответствует ровно одно значение Б.

Аксиомы Армстронга

Используются для вывода всех функциональных зависимостей в реляционной базе данных.

- Рефлексивность,
- Пополнение,
- Транзитивность,
- Самодетерминированность,
- Декомпозиция,
- Объединение,
- Композиция,
- Накопление.

Факультативно можно ознакомиться по ссылке.

Денормализация

Денормализация

Денормализация — это процесс ухода от правил нормализации там, где это необходимо.

Для процесса денормализации не существует стандартного алгоритма. Процесс денормализации индивидуален и требует четкого понимания, для чего он необходим в связи с появлением избыточности.

Денормализация

К денормализации прибегают для сокращения времени обработки запросов и уменьшения затрат ресурсов. В нормализованных базах часто приходится соединять большое количество таблиц или добавлять агрегацию.

Таким образом денормализацию можно выполнить сократив количество таблиц или добавив новые столбцы в существующую таблицу. При этом учитывая избыточность данных необходимо следить за целостностью данных.

Процесс денормализации зависит от СУБД в которой происходит работа.

Денормализация

К примеру, в PostgreSQL есть возможность создания материализованных представлений (MATERIALIZED VIEW), то есть можно создать МП, внутри которого будет выполнена логика по соединению данных из нескольких таблиц, произведена агрегация и другие действия, а результат этих действий будет физически храниться на жестком диске. Когда нужно будет актуализировать данные, достаточно обновить данные командой:

```
REFRESH MATERIALIZED VIEW имя_МП
```

Далее при обращении к МП данные будут читаться с диска, а не выполняться вся логика запроса, что сокращает время работы с данными во множество раз.

Денормализация

К примеру, в MySQL нет поддержки МП, здесь можно создать денормализованную таблицу, и с помощью триггерных функций формировать данные в денормализованной таблице.

При изменении данных в нормализованных таблицах или при добавлении новых данных в эти таблицы должны отрабатывать триггеры, которые будут вызывать **единую хранимую процедуру**, которая будет производить необходимые вычисления, соединения данных и полученный результат записывать в денормализованную таблицу.

Важно помнить, что при появлении избыточности или дублировании атрибутов необходимо контролировать целостность при внесении и модификации данных.

Типы данных

Типы данных

Выделяют следующие типы данных:

- Числовые,
- Строковые,
- Дата и время,
- Сложные, бинарные, геометрические…

При этом в зависимости от СУБД названия и поддержка разных типов данных может отличаться.

Рассмотрим типы данных на примере MySQL.

Числовые

TINYINT — очень малое целое число. Диапазон со знаком от -128 до 127. Диапазон без знака от 0 до 255.

SMALLINT — малое целое число. Диапазон со знаком от -32768 до 32767. Диапазон без знака от 0 до 65535.

MEDIUMINT — целое число среднего размера. Диапазон со знаком от -8388608 до 8388607. Диапазон без знака от 0 до 16777215.

INTEGER — целое число нормального размера. Диапазон со знаком от -2147483648 до 2147483647. Диапазон без знака от 0 до 4294967295.

BIGINT — большое целое число. Диапазон со знаком от -9223372036854775808 до 9223372036854775807. Диапазон без знака от 0 до 18446744073709551615.

Числовые

FLOAT — малое число с плавающей точкой обычной точности. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38, 0, и от 1,175494351E-38 до 3,402823466E+38.

DOUBLE — число с плавающей точкой удвоенной точности нормального размера. Допустимые значения: от -1,7976931348623157E+308 до -2,2250738585072014E-308, 0, и от 2,2250738585072014E-308 до 1,7976931348623157E+308.

DECIMAL/NUMERIC — число с плавающей точкой. Ведет себя подобно столбцу CHAR, содержащему цифровое значение. Число хранится в виде строки и при этом для каждого десятичного знака используется один символ. Используется для хранения финансовых значений.

Числовые

Отличие работы FLOAT от NUMERIC:

```
577 SELECT x,
578    round(x::numeric) AS num_round,
579    round(x::double precision) AS dbl_round
580 FROM generate_series(-3.5, 3.5, 1) as x;
```

Log

Результат 1

SELECT x, round(x::numeric) AS num_round, round(x::double pre

Таблица

Текст

	x	num_round	dbl_round
1	-3,5	-4	-4
2	-2,5	-3	-2
3	-1,5	-2	-2
4	-0,5	-1	-0
5	0,5	1	0
6	1,5	2	2
7	2,5	3	2
8	3,5	4	4

Строковые

CHAR — строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов.

VARCHAR — строка переменной длины. Примечание: концевые пробелы удаляются при сохранении значения (в этом заключается отличие от спецификации ANSI SQL). Диапазон аргумента M составляет от 0 до 255 символов.

TINYTEXT — столбец типа BLOB или TEXT с максимальной длиной 255 (2^8 — 1) символов.

TEXT — столбец типа BLOB или TEXT с максимальной длиной 65535 (2^16 — 1) символов.

MEDIUMTEXT — столбец типа BLOB или TEXT с максимальной длиной 16777215 (2^24 — 1) символов.

Строковые

LONGTEXT — столбец типа BLOB или TEXT с максимальной длиной 4294967295 (2^32 — 1).

ENUM('значение1','значение2',...) — перечисляемый тип данных. Объект строки может иметь только одно значение, выбранное из заданного списка величин 'значение1', 'значение2', ..., NULL или специальная величина ошибки "". Список ENUM может содержать максимум 65535 различных величин.

SET('значение1','значение2',...) — набор. Объект строки может иметь ноль или более значений, каждое из которых должно быть выбрано из заданного списка величин 'значение1', 'значение2', ... Список SET может содержать максимум 64 элемента.

Дата и время

DATE — дата. Поддерживается интервал от '1000-01-01' до '9999-12-31'.

DATETIME — комбинация даты и времени. Поддерживается интервал от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'.
TIMESTAMP — Временная метка. Интервал от '1970-01-01 00:00:00' до некоторого значения времени в 2037 году.

TIME — время. Интервал от '-838:59:59' до '838:59:59'.

YEAR — год в двухзначном или четырехзначном форматах (по умолчанию формат четырехзначный). Допустимы следующие значения: с 1901 по 2155, 0000 для четырехзначного формата года и 1970-2069 при использовании двухзначного формата (70-69).

Сложные, бинарные, геометрические

Как было сказано ранее, разные СУБД поддерживают разные типы данных.

Например, в MySQL нет поддержки массивов, но есть возможность работы с JSON. В PostgreSQL есть поддержка массивов:

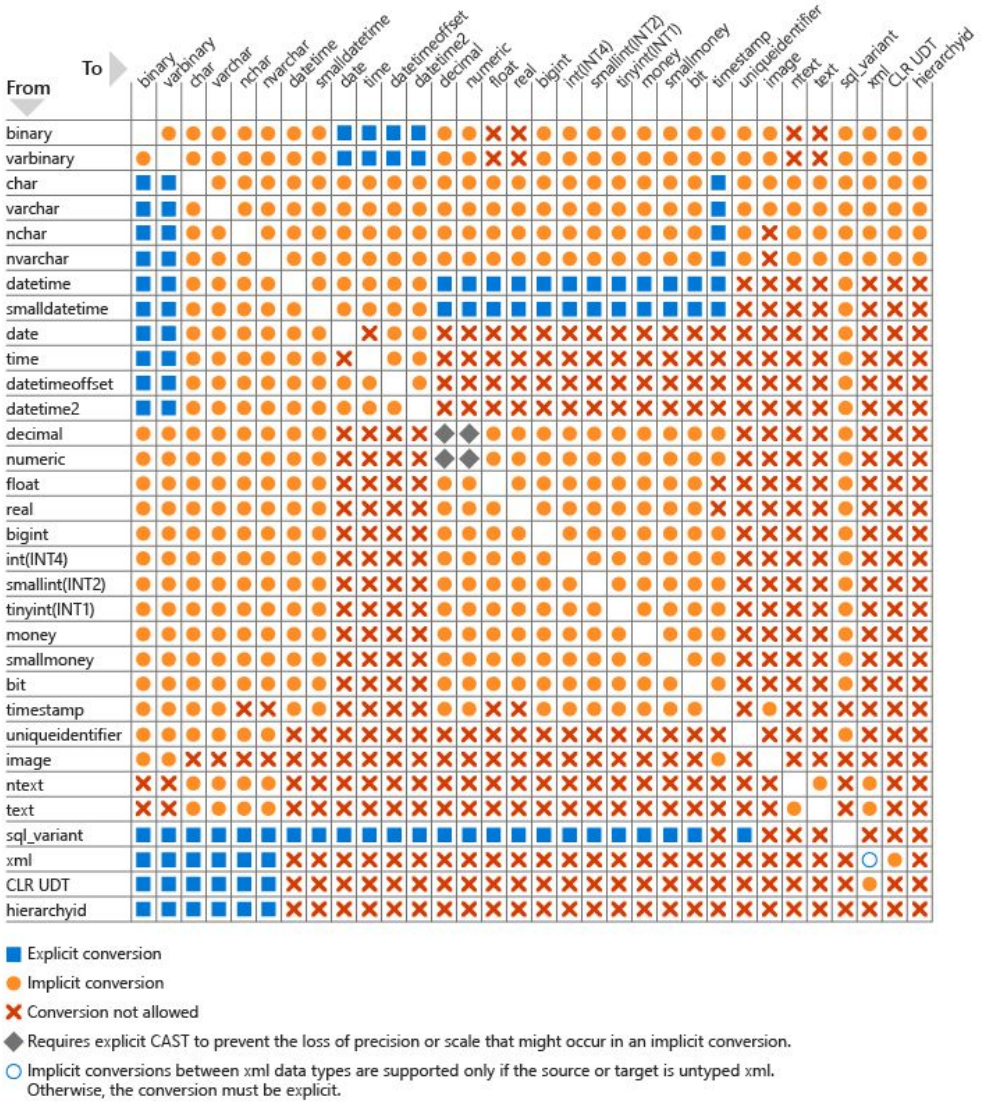
int[] — массив, где элементами могут быть только числа,

text[] — массив, где элементами могут быть только строки.

Также в некоторых СУБД есть возможность подключать различные расширения, что позволяет работать со сложными типами данных, такими как геометрия, география и т.д.

Типы данных

Таблица преобразования типов данных на примере MS SQL:



Итоги

Итоги

Сегодня на лекции мы:

- вспомнили реляционную модель данных;
- узнали, какие существуют ограничения;
- разобрались с нормализацией и денормализацией;
- познакомились с типами данных.

Домашнее задание

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Олег Гежин