

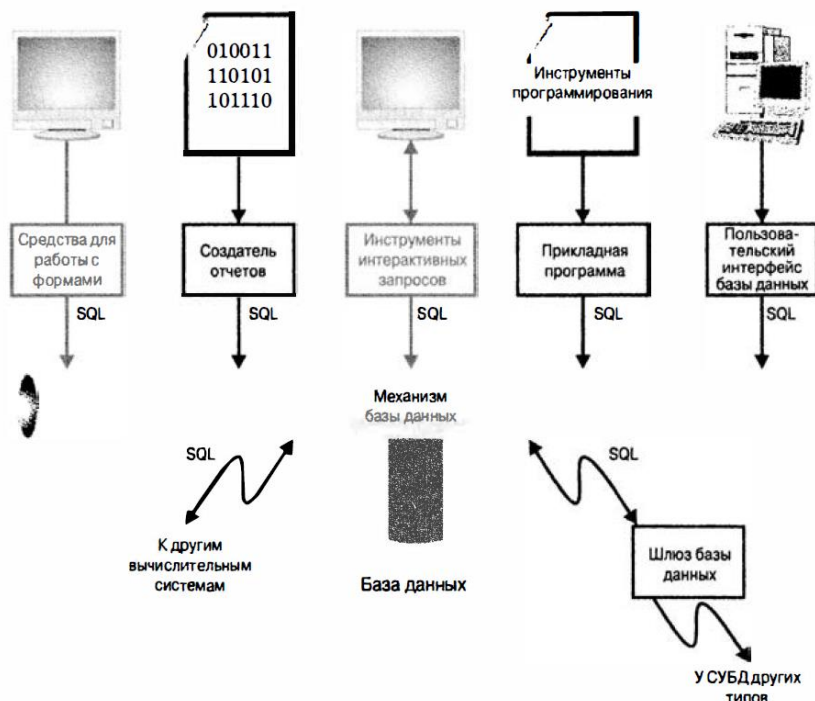
# SQL и базы данных



SQL является инструментом, предназначенным для организации, управления, выборки и обработки информации, содержащейся в реляционных базах данных.

- Определение данных. SQL позволяет пользователю определить структуру и организацию хранимых данных и взаимоотношения между элементами сохраненных данных.
- Выборка данных. SQL дает пользователю или приложению возможность извлекать из базы содержащиеся в ней данные и пользоваться ими.
- Обработка данных. SQL позволяет пользователю или приложению изменять базу данных, т.е. добавлять в нее новые данные, а также удалять или обновлять уже имеющиеся в ней данные.
- Управление доступом. С помощью SQL можно ограничить возможности пользователя по выборке, добавлению и изменению данных и защитить их от несанкционированного доступа.
- Совместное использование данных. SQL применяется для координации совместного использования данных пользователями, работающими одновременно, с тем чтобы изменения, вносимые одним пользователем, не приводили к непреднамеренному уничтожению изменений, вносимых примерно в то же время иным пользователем.
- Целостность данных. SQL позволяет обеспечить целостность базы данных, защищая ее от разрушения из-за несогласованных изменений или отказа системы.

Сердцем СУБД является механизм базы данных (database engine, часто называемый просто **движком**); он отвечает за структурирование данных, сохранение и получение их из базы данных. Он принимает SQL-запросы от других компонентов СУБД (таких, как генератор отчетов или модуль запросов), от пользовательских приложений и даже от других вычислительных систем. Как видно из рисунка, SQL выполняет много различных функций.



Вот некоторые основные свойства SQL, обеспечивающие такой небывалый его успех в течение последних десятилетий.

- Независимость от конкретных СУБД
- Межплатформенная переносимость
- Наличие стандартов
- Поддержка со стороны компании IBM
- Поддержка со стороны компании Microsoft
- Построение на реляционной модели
- Высокоуровневая структура, напоминающая естественный язык
- Возможность выполнения специальных интерактивных запросов
- Обеспечение программного доступа к базам данных
- Возможность различного представления данных
- Полноценность в качестве языка, предназначенного для работы с базами данных
- Возможность динамического определения данных
- Поддержка архитектуры клиент/ сервер
- Поддержка приложений уровня предприятия
- Расширяемость и поддержка объектно-ориентированных технологий
- Возможность доступа к данным в Интернете
- Интеграция с языком java (протокол JDBC)
- Поддержка открытого кода
- Промышленная инфраструктура

Мейнфрейм (также мэйнфрейм, от англ. mainframe) — большой универсальный высокопроизводительный отказоустойчивый сервер со значительными ресурсами ввода-вывода, большим объёмом оперативной и внешней памяти, предназначенный для использования в критически важных системах (англ.

**Реляционная база данных** — база данных, основанная на реляционной модели данных. Реляционной называется база данных, в которой все данные, доступные пользователю, организованы в виде таблиц, а все операции базы данных выполняются над этими таблицами. Столбцов не может быть 0, строк может быть 0. В правильно построенной реляционной базе данных в каждой таблице есть столбец (или комбинация столбцов), для которого значения во всех строках различны. Этот столбец (столбцы) называется первичным ключом (primary key). Таблица, в которой все строки отличаются друг от друга, в математических

терминах называется отношением (relation) .

Отношение "предок-потомок", существующее между офисами и работающими в них людьми, в реляционной модели не утеряно; просто оно реализовано в виде одинаковых значений данных, хранящихся в двух таблицах, а не в виде явного указателя. Таким способом реализуются все отношения, существующие между таблицами реляционной базы данных. Одним из главных преимуществ языка SQL является возможность извлекать связанные между собой данные, используя эти отношения.

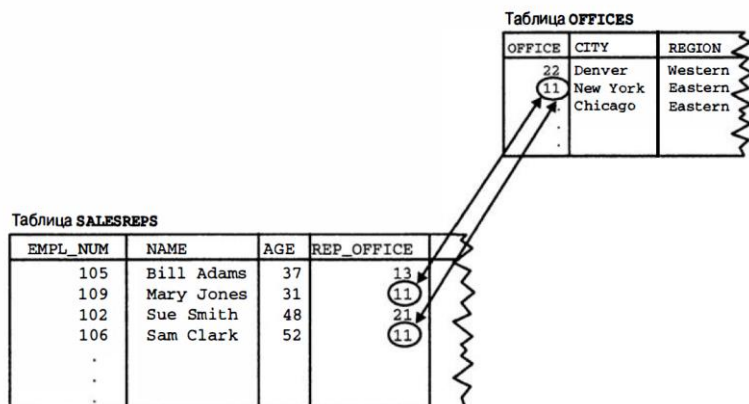


Рис. 4.9. Отношение "предок-потомок" в реляционной базе данных

На рис. 4.9 столбец REP\_OFFICE представляет собой внешний ключ для таблицы OFFICES. Хотя столбец REP\_OFFICE и находится в таблице SALESREPS, значения, содержащиеся в нем, представляют собой идентификаторы офисов. Эти значения соответствуют значениям в столбце OFFICE, который является первичным ключом таблицы OFFICES. Первичный и внешний ключи создают между таблицами, в которых они содержатся, такое же отношение "предок-потомок", как и в иерархической базе данных.

Фактически внешний ключ всегда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

двенадцать правил, которым должна соответствовать настоящая реляционная база данных

1. Правило представления информации. Вся информация в реляционной базе данных должна быть представлена исключительно на логическом уровне и только одним способом - в виде значений, содержащихся в таблицах.

2. Правило гарантированного доступа. Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляционной базе данных должен обеспечиваться путем использования комбинации имени таблицы, значения первичного ключа и имени столбца.

3. Систематическая трактовка значения NULL. В настоящей реляционной базе данных должна быть реализована полная поддержка значений NULL (которые отличаются от строки символов нулевой длины, строки пробельных символов, а также от нуля или любого другого числа), которые используются для представления отсутствующей и неприменимой информации систематическим образом независимо от типа этих данных.

4. Правило динамического каталога, основанного на реляционной модели. Описание базы данных на логическом уровне должно быть представлено в том же виде, что и обычные данные, чтобы пользователи, обладающие соответствующими

правами, могли работать с ним с помощью того же реляционного языка, который они применяют для работы с основными данными.

5. Правило исчерпывающего подязыка данных. Реляционная система может поддерживать несколько языков и режимов взаимодействия с пользователем. Однако должен существовать по крайней мере один язык, инструкции которого можно представить в виде строк символов в соответствии с некоторым точно определенным синтаксисом и который в полной мере поддерживает все следующие элементы:

- . определение данных;
- . определение представлений;
- . обработку данных (интерактивную и программную);
- . ограничения целостности данных;
- . авторизацию;
- . границы транзакций (начало, фиксацию и откат) .

6. Правило обновления представлений. Все представления, которые теоретически можно обновить, должны быть доступны для обновления системой.

7. Правило высокоуровневого добавления, обновления и удаления. Операции вставки, обновления и удаления должны применяться к отношению в целом.

8. Правило физической независимости данных. Прикладные программы и утилиты для работы с данными на логическом уровне должны оставаться неизменными при любых изменениях способов хранения данных или методов доступа к ним.

9. Правило логической независимости данных. Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.

10. Правило независимости контроля целостности. Должна существовать возможность определять условия целостности, специфичные для конкретной реляционной базы данных, на подязыке этой базы данных и хранить их в каталоге, а не в прикладной программе.

11. Правило независимости распространения. Реляционная база данных должна быть переносима не только в пределах системы, но и по сети.

12. Правило согласования языковых уровней. Если в реляционной системе есть низкоуровневый язык (обрабатывающий одну запись за один раз), то он не должен иметь возможность обходить правила и условия целостности данных, выраженные на реляционном языке высокого уровня (обрабатывающем несколько записей за один раз). правило 12 предотвращает использование других средств работы с базой данных, помимо ее подязыка, поскольку это может нарушить ее целостность

SQL основан на реляционной модели данных, в которой данные организованы в виде коллекций таблиц.

- Каждая таблица имеет уникальное имя.

- В каждой таблице есть один или несколько именованных столбцов, расположенных в определенном порядке слева направо.

- В каждой таблице есть нуль или более строк, каждая из которых содержит одно значение данных в каждом столбце; строки в таблице не упорядочены.

Все значения данных в одном столбце имеют одинаковый тип данных и входят в набор допустимых значений, который называется доменом столбца.

Отношения между таблицами реализуются с помощью содержащихся в них данных. В реляционной модели данных для представления этих отношений используются первичные и внешние ключи.

- Первичным ключом может быть столбец или комбинация столбцов таблицы, значения которых уникальным образом идентифицируют каждую строку таблицы. У таблицы есть только один первичный ключ.

- Внешним ключом является столбец или группа столбцов таблицы, значения которых совпадают со значениями первичного ключа другой таблицы.

Таблица может содержать несколько внешних ключей, связывающих ее с одной или несколькими другими таблицами.

- Пара "первичный ключ-внешний ключ" создает отношение "предокпотомок" между таблицами, содержащими их.

Строки результатов запроса, как и строки таблицы базы данных, не имеют определенного порядка. Но, включив в инструкцию SELECT предложение ORDER BY, можно отсортировать результаты запроса.

**Репликация** — одна из техник масштабирования баз данных. Состоит эта техника в том, что данные с одного сервера базы данных постоянно копируются (реплицируются) на один или несколько других (называемые репликами). Для приложения появляется возможность использовать не один сервер для обработки всех запросов, а несколько. Таким образом появляется возможность распределить нагрузку с одного сервера на несколько.

Большинство СУБД позволяет различным пользователям создавать таблицы с одинаковыми именами (например, и пользователь Joe, и пользователь Sam могут создать таблицу BIRTHDAYS). СУБД обращается к необходимой таблице в зависимости от того, кто из пользователей запрашивает данные.

При наличии соответствующих прав можно обращаться к таблицам, владельцами которых являются другие пользователи, с помощью полного, или квалифицированного, имени таблицы. Оно состоит из имен владельца таблицы и собственно таблицы, разделенных точкой. Например, квалифицированное имя таблицы BIRTHDAYS, владельцем которой является пользователь SAM, имеет такой вид.  
SAM.BIRTHDAYS

Стандарт ANSI / ISO SQL еще больше обобщает понятие квалифицированного имени таблицы. Он разрешает создавать именованное множество таблиц, называемое **схемой**. Вы можете обращаться к таблице определенной схемы с использованием квалифицированного имени. Например, обращение к таблице BIRTHDAYS в схеме **EMPLOYEE\_INFO** имеет следующий вид.  
EMPLOYEE\_INFO.BIRTHDAYS

Полное имя столбца BIRTH\_DATE в таблице BIRTHDAYS, владельцем которой является пользователь SAM, имеет следующий вид.  
SAM.BIRTHDAYS.BIRTHDATE

**Таблица 5.4. Типы данных ANSI/ISO SQL**

Тип данных	Сокращение	Описание
CHARACTER (длина)	CHAR	Строки символов постоянной длины
CHARACTER VARYING (длина)	CHAR VARYING, VARCHAR	Строки символов переменной длины
CHARACTER LARGE OBJECT (длина)	CLOB	Большие строки символов переменной длины
NATIONAL CHARACTER (длина)	NATIONAL CHAR, NCHAR	Строки символов постоянной длины с наборами национальных символов
NATIONAL CHARACTER VARYING (длина)	NATIONAL CHAR VARYING, NCHAR	Строки символов переменной длины с наборами национальных символов
NATIONAL CHARACTER LARGE OBJECT (длина)	NCLOB	Большие строки символов переменной длины с наборами национальных символов
BIT (длина)	INT	Битовые строки постоянной длины
BIT VARYING (длина)		Битовые строки переменной длины
INTEGER		Целые числа
SMALLINT		Малые целые числа
NUMERIC (точность, масштаб)	DEC	Десятичные числа
DECIMAL (точность, масштаб)		Десятичные числа
FLOAT (точность)		Числа с плавающей точкой
REAL		Числа с плавающей точкой малой точности
DOUBLE PRECISION		Числа с плавающей точкой большой точности
DATE		Календарные даты
TIME (точность)		Время
TIME WITH TIME ZONE (точность)		Поясное время
TIMESTAMP (точность)		Дата и время
TIMESTAMP WITH TIME ZONE (точность)		Дата и поясное время
INTERVAL		Временные интервалы
XML (модификатор типа [вторичный модификатор типа])		Символьные данные в XML-формате

SQL поддерживает обработку отсутствующих, неизвестных или неприменимых данных с помощью концепции отсутствующего значения. **NULL** - это значение является индикатором, который показывает, что в конкретной строке определенный элемент данных отсутствует или что столбец вообще не подходит для этой строки.

#### Резюме

В настоящей главе были описаны основные элементы SQL.

- Язык SQL включает около тридцати инструкций, каждая из которых состоит из команды и одного или нескольких предложений. Каждая инструкция выполняет одно конкретное действие.
- В SQL-базах данных могут храниться значения различных типов, включая текстовые данные, целые и десятичные числа, числа с плавающей точкой и данные ряда других типов, введенных конкретным производителем.
- Инструкции SQL могут включать в себя выражения, в которых над именами столбцов, константами и встроенными функциями выполняются арифметические и другие операции.
- Разнообразие типов данных, констант и встроенных функций делает переносимость



инструкций SQL более сложной задачей, чем это может показаться на первый взгляд.

- Значения NULL используются для обработки отсутствующих или неприменимых элементов данных в SQL.

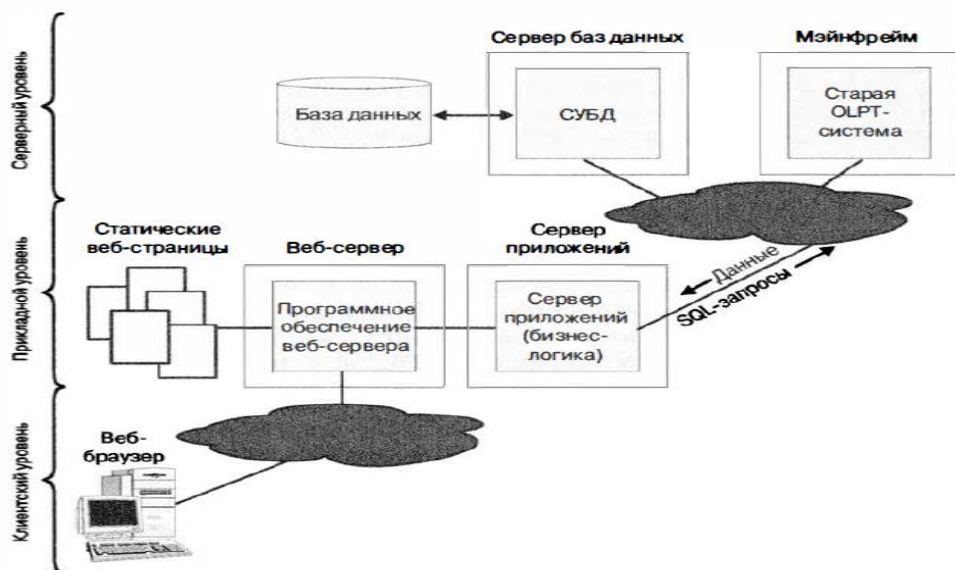


Рис. 3.5. Управление базами данных в трехуровневой интернет-архитектуре

XML (англ. eXtensible Markup Language) — расширяемый язык разметки

Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов

## Резюме

- Инструкция SELECT используется для формирования SQL-запроса. Каждая инструкция SELECT возвращает таблицу результатов запроса, содержащую один или более столбцов и нуль или более строк.
- Предложение FROM определяет таблицы, в которых содержатся выбираемые данные.
- Предложение SELECT определяет столбцы данных, которые необходимо включить в результаты запроса и которые могут представлять собой столбцы из базы данных или вычисляемые столбцы.
- Предложение WHERE отбирает строки, которые необходимо включить в результаты запроса, путем применения условия отбора к строкам базы данных.
- Условие отбора позволяет выбрать строки путем сравнения значений, проверки значений на принадлежность множеству или диапазону значений, проверки на соответствие шаблону и на равенство значению NULL.
- Простые условия отбора могут объединяться в более сложные условия с помощью операторов AND, OR и NOT.
- Предложение ORDER BY указывает, что результаты запроса должны быть отсортированы по возрастанию или убыванию на основе значений одного или нескольких столбцов.
- Для объединения результатов двух инструкций SELECT в одно множество можно использовать операцию UNION.

В SQL-базе данных первичные и внешние ключи создают отношение "предок-потомок". Таблица, содержащая внешний ключ, является потомком, а таблица с первичным ключом - предком. Чтобы использовать в запросе отношение "предок-потомок", необходимо задать условие отбора, в котором первичный ключ сравнивается с внешним ключом.

*Вывести список всех служащих с городами и регионами, в которых они работают.*

```
SELECT NAME, CITY, REGION
FROM SALESREPS, OFFICES
WHERE REP_OFFICE = OFFICE;
```

NAME	CITY	REGION
-----	-----	-----
Mary Jones	New York	Eastern
Sam Clark	New York	Eastern
Bob Smith	Chicago	Eastern
Paul Cruz	Chicago	Eastern
Dan Roberts	Chicago	Eastern
Bill Adams	Atlanta	Eastern
Sue Smith	Los Angeles	Western
Larry Fitch	Los Angeles	Western
Nancy Angelli	Denver	Western

Таблица OFFICES

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	\$300,000.00	\$186,042.00
11	New York	Eastern	106	\$575,000.00	\$692,637.00
12	Chicago	Eastern	104	\$800,000.00	\$735,042.00
13	Atlanta	Eastern	NULL	\$350,000.00	\$367,911.00
21	Los Angeles	Western	108	\$725,000.00	\$835,915.00

Таблица SALESREPS

EMPL_NUM	NAME	AGE	REP_OFFICE	TITLE
105	Bill Adams	37	13	Sales Rep
109	Mary Jones	31	11	Sales Rep
102	Sue Smith	48	21	Sales Rep
106	Sam Clark	52	11	VP Sales
104	Bob Smith	33	12	Sales Mgr
101	Dan Roberts	45	12	Sales Rep
110	Tom Snyder	41	NULL	Sales Rep
108	Larry Fitch	62	21	Sales Mgr
103	Paul Cruz	29	12	Sales Rep
107	Nancy Angelli	49	22	Sales Rep

Результаты запроса

NAME	CITY	REGION

Таблица SALESREPS (потомок) содержит столбец REP\_OFFICE, который является внешним ключом для таблицы OFFICES (предок). Здесь отношение "предок-потомок" используется с целью поиска в таблице OFFICE для каждого служащего соответствующей строки, содержащей город и регион, и включения ее в результаты запроса.

## 1. Что такое DDL? Какие операции в него входят? Рассказать про них.

Какие существуют операторы SQL?

операторы определения данных (Data Definition Language, DDL):

- **CREATE** создает объект БД (базу, таблицу, представление, пользователя и т. д.),  
CREATE DATABASE Test
- **ALTER** изменяет объект (ALTER TABLE - изменить структуру таблицы)
- **DROP** удаляет объект;

Проконтролировать создание базы данных можно с помощью оператора

**SHOW DATABASES**,

**SHOW TABLES** кроме пользовательских таблиц отображает также и служебные таблицы.



**DESCRIBE** table\_name – показывает таблицу

**DELETE FROM** <table\_name> - Удаление всех данных из таблицы

Проверка на уже существующие базы данных:

CREATE DATABASE **IF NOT EXIST** имя\_базы\_данных;

Перед созданием таблицы необходимо выбрать базу данных, в которую таблица будет записана. Это делается с помощью оператора **USE**: USE имя\_базы\_данных

Для того, чтобы посмотреть описание созданной таблицы можно воспользоваться оператором DESCRIBE: **DESCRIBE** Users;

## 2. Что такое DML? Какие операции в него входят? Рассказать про них.

операторы манипуляции данными (Data Manipulation Language, DML):

- **SELECT** выбирает данные, удовлетворяющие заданным условиям,

SELECT [DISTINCT | ALL] поля\_таблиц

FROM список\_таблиц

[WHERE условия\_на\_ограничения\_строк]

[GROUP BY условия\_группировки]

[HAVING условия\_на\_ограничения\_строк\_после\_группировки по агрегатным функциям]

[ORDER BY порядок\_сортировки [ASC | DESC]]

[LIMIT ограничение\_количества\_записей]

SQL-псевдонимы: SELECT good\_type\_id AS id FROM GoodTypes;

Порядок выполнения инструкций (зависит от СУБД):

FROM

**WHERE**

GROUP BY

HAVING

**SELECT**

**DISTINCT**

ORDER BY

В предложении WHERE не доступны псевдонимы столбцов, определяемых в предложении SELECT, потому что, согласно списку, оно выполняется до SELECT

**WHERE** условия\_на\_ограничения\_строк [логический\_оператор

другое\_условия\_на\_ограничения\_строк]

- IS [NOT] NULL — позволяет узнать равно ли проверяемое значение NULL

- [NOT] BETWEEN min AND max

- [NOT] IN — позволяет узнать входит ли проверяемое значение столбца в список определённых значений

WHERE status IN ('father', 'mother');

- [NOT] LIKE шаблон [ESCAPE символ] — позволяет узнать соответствует ли строка определённому шаблону

Трафаретные символы:

Шаблон	Описание
never%	Сопоставляется любым строкам, начинающимся на «never».

Шаблон	Описание
<code>%ing</code>	Сопоставляется любым строкам, заканчивающимся на «ing».
<code>_ing</code>	Сопоставляется строкам, имеющим длину 4 символа, при этом 3 последних обязательно должны быть «ing». Например, слова «sing» и «wing».

**ESCAPE-символ** используется для экранирования трафаретных символов

WHERE progress LIKE '3!%' **ESCAPE '!' ;**

Если бы мы не экранировали трафаретный символ, то в выборку попало бы всё, что начинается на 3.

**Логические операторы:** NOT, OR (общее значение выражения истинно, если хотя бы одно из них истинно), AND, XOR (общее значение выражения истинно, если один и только один аргумент является истинным)

WHERE plane = 'Boeing' AND NOT town\_from = 'London';

- **INSERT** добавляет новые данные

Общая структура запроса с оператором INSERT

**INSERT INTO** имя\_таблицы [(поле\_таблицы, ...)]

**VALUES** (значение\_поля\_таблицы, ...)

| SELECT поле\_таблицы, ... FROM имя\_таблицы ...

```
INSERT INTO Goods (good_id, good_name, type)
```

```
VALUES (5, 'Table', 2);
```

```
INSERT INTO Goods VALUES (5, 'Table', 2);
```

```
INSERT INTO Goods
```

```
SELECT good_id, good_name, type FROM Goods where good_name = 2;
```

- **UPDATE** изменяет существующие данные

```
UPDATE FamilyMembers
```

```
SET member_name = "Andie Anthony"      что делаем
```

```
WHERE member_name = "Andie Quincey"    куда делаем
```

```
UPDATE Payments
```

```
SET unit_price = unit_price * 2
```

- **DELETE** удаляет данные;

```
DELETE Reservations, Rooms FROM
```

```
Reservations JOIN Rooms ON
```

```
Reservations.room_id = Rooms.id
```

```
WHERE Rooms.has_kitchen = false;
```

```
TRUNCATE TABLE имя_таблицы;
```

у оператора TRUNCATE есть ряд отличий:

- **Не срабатывают триггеры** (, в частности, триггер удаления
- Удаляет все строки в таблице, не записывая при этом удаление отдельных строк данных в журнал транзакций

- Сбрасывает счетчик идентификаторов до начального значения
- Чтобы использовать, необходимы права на изменение таблицы

Триггеры представляют специальный тип **хранимой процедуры**, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE. Допустим, в таблице Products хранятся данные о товарах. Но цена товара нередко содержит различные надбавки типа налога на добавленную стоимость, налога на добавленную коррупцию и так далее. Человек, добавляющий данные, может не знать все эти тонкости с налоговой базой, и он определяет чистую цену. С помощью триггера мы можем поправить цену товара на некоторую величину.

```
CREATE TRIGGER имя_триггера
ON {имя_таблицы | имя_представления}
{AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]
AS выражения_sql
```

### 3. Что такое TCL? Какие операции в него входят? Рассказать про них.

операторы управления транзакциями (Transaction Control Language, TCL):

- **COMMIT** Применяется для завершения транзакции и сохранения изменений в базе данных,

```
DELETE FROM developers
WHERE SPECIALTY = 'C++';
```

**COMMIT;**

- **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции,
- **SAVEPOINT** Создает точку к которой группа транзакций может откатиться, разбивает транзакцию на более мелкие.

**SET TRANSACTION** Применяется для установки параметров доступа к данным в текущей транзакции

указать, что транзакция предназначена только для чтения, то мы должны использовать следующий запрос:

```
SET TRANSACTION READ ONLY;
```

**Команды управление транзакциями** используются только для **DML команд**: INSERT, UPDATE, DELETE

**транкейт не используется когда в таблице есть внешние ключи** надо использовать делит

### 4. Что такое DCL? Какие операции в него входят? Рассказать про них.

операторы определения доступа к данным (Data **Control** Language, DCL):

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

-- Предоставление права чтения таблицы students пользователю alex.

```
GRANT SELECT ON students TO alex;
```

-- Запрет права выборки из таблицы orders пользователя alex.

```
DENY SELECT ON orders TO alex;
```

-- Отменить запрет.

```
REVOKE SELECT ON total FROM piter;
```

### 5. Нюансы работы с NULL в SQL. Как проверить поле на NULL?

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

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

команды: **IS NULL, IS NOT NULL**

## 6. Виды Join'ов?

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

Особенностями операции соединения являются следующее:

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

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

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

**SELECT** поля\_таблиц

**FROM** таблица\_1

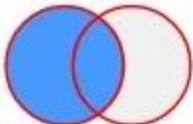
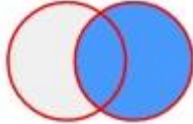
**[INNER] | [[LEFT | RIGHT | FULL][OUTER]] JOIN** таблица\_2

**ON** условие\_соединения

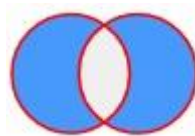
**[INNER] | [[LEFT | RIGHT | FULL][OUTER]] JOIN** таблица\_n

**ON** условие\_соединения]

<https://habr.com/ru/post/450528/>

LEFT [OUTER]	 <table><tr><td></td><td>1</td><td>1</td><td>2</td><td>3</td><td>5</td><td></td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td><td></td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td><td></td></tr><tr><td>6</td><td>6, 1</td><td>6, 1</td><td>6, 2</td><td>6, 3</td><td>6, 5</td><td>6, null</td></tr><tr><td>5</td><td>5, 1</td><td>5, 1</td><td>5, 2</td><td>5, 3</td><td>5, 5</td><td></td></tr></table>		1	1	2	3	5		1	1, 1	1, 1	1, 2	1, 3	1, 5		1	1, 1	1, 1	1, 2	1, 3	1, 5		6	6, 1	6, 1	6, 2	6, 3	6, 5	6, null	5	5, 1	5, 1	5, 2	5, 3	5, 5		<p>SELECT поля_таблиц FROM левая_таблица <b>LEFT JOIN</b> правая_таблица ON правая_таблица.ключ = левая_таблица.ключ</p> <p>Получение всех данных из левой таблицы, соединённых с соответствующими данными из правой</p> <p>INNER JOIN, но дополнительно мы добавляем null для строк из первой таблицы, для которой ничего не нашлось во второй</p>
	1	1	2	3	5																																
1	1, 1	1, 1	1, 2	1, 3	1, 5																																
1	1, 1	1, 1	1, 2	1, 3	1, 5																																
6	6, 1	6, 1	6, 2	6, 3	6, 5	6, null																															
5	5, 1	5, 1	5, 2	5, 3	5, 5																																
RIGHT [OUTER]		<p>SELECT поля_таблиц FROM левая_таблица <b>RIGHT JOIN</b> правая_таблица ON правая_таблица.ключ = левая_таблица.ключ</p>																																			

	<table><tr><td></td><td></td><td>1</td><td>1</td><td>2</td><td>3</td><td>5</td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td><td></td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td><td></td></tr><tr><td>6</td><td>6, 1</td><td>6, 1</td><td>6, 2</td><td>6, 3</td><td>6, 5</td><td></td></tr><tr><td>5</td><td>5, 1</td><td>5, 1</td><td>5, 2</td><td>5, 3</td><td>5, 5</td><td></td></tr><tr><td></td><td></td><td></td><td>null, 2</td><td>null, 3</td><td></td><td></td></tr></table>			1	1	2	3	5	1	1, 1	1, 1	1, 2	1, 3	1, 5		1	1, 1	1, 1	1, 2	1, 3	1, 5		6	6, 1	6, 1	6, 2	6, 3	6, 5		5	5, 1	5, 1	5, 2	5, 3	5, 5					null, 2	null, 3			<p>Получение всех данных из правой таблицы, соединённых с соответствующими данными из левой</p> <p>это INNER JOIN + null для строк из второй таблицы, для которой ничего не нашлось в первой</p>
		1	1	2	3	5																																						
1	1, 1	1, 1	1, 2	1, 3	1, 5																																							
1	1, 1	1, 1	1, 2	1, 3	1, 5																																							
6	6, 1	6, 1	6, 2	6, 3	6, 5																																							
5	5, 1	5, 1	5, 2	5, 3	5, 5																																							
			null, 2	null, 3																																								
<p>LEFT [OUTER] правая_таблица.ключ IS NULL</p>		<p>SELECT поля_таблиц FROM левая_таблица LEFT JOIN правая_таблица ON правая_таблица.ключ = левая_таблица.ключ WHERE правая_таблица.ключ IS NULL</p>																																										
<p>RIGHT [OUTER] левая_таблица.ключ IS NULL</p>		<p>SELECT поля_таблиц FROM левая_таблица RIGHT JOIN правая_таблица ON правая_таблица.ключ = левая_таблица.ключ WHERE левая_таблица.ключ IS NULL</p> <p>Получение данных, относящихся только к правой таблице</p>																																										
<p>INNER</p>	<table><tr><td></td><td></td><td>1</td><td>1</td><td>2</td><td>3</td><td>5</td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td><td></td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td><td></td></tr><tr><td>6</td><td>6, 1</td><td>6, 1</td><td>6, 2</td><td>6, 3</td><td>6, 5</td><td></td></tr><tr><td>5</td><td>5, 1</td><td>5, 1</td><td>5, 2</td><td>5, 3</td><td>5, 5</td><td></td></tr></table>			1	1	2	3	5	1	1, 1	1, 1	1, 2	1, 3	1, 5		1	1, 1	1, 1	1, 2	1, 3	1, 5		6	6, 1	6, 1	6, 2	6, 3	6, 5		5	5, 1	5, 1	5, 2	5, 3	5, 5		<p>SELECT поля_таблиц FROM левая_таблица INNER JOIN правая_таблица ON правая_таблица.ключ = левая_таблица.ключ</p> <p>тот же самый CROSS JOIN, у которого оставлены только те элементы, которые удовлетворяют условию, записанному в конструкции "ON".</p>							
		1	1	2	3	5																																						
1	1, 1	1, 1	1, 2	1, 3	1, 5																																							
1	1, 1	1, 1	1, 2	1, 3	1, 5																																							
6	6, 1	6, 1	6, 2	6, 3	6, 5																																							
5	5, 1	5, 1	5, 2	5, 3	5, 5																																							
<p>FULL OUTER</p>		<p>SELECT поля_таблиц FROM левая_таблица FULL OUTER JOIN правая_таблица ON правая_таблица.ключ = левая_таблица.ключ</p>																																										

	<table><tr><th>Id</th><th>Name</th></tr><tr><td>1</td><td>Москва</td></tr><tr><td>2</td><td>Санкт-Петербург</td></tr><tr><td>3</td><td>Казань</td></tr></table> <table><tr><th colspan="2">Person (Люди)</th></tr><tr><th>Name</th><th>CityId</th></tr><tr><td>Андрей</td><td>1</td></tr><tr><td>Леонид</td><td>2</td></tr><tr><td>Сергей</td><td>1</td></tr><tr><td>Григорий</td><td>4</td></tr></table> <table><tr><th>Person.Name</th><th>Person.CityId</th><th>City.Id</th><th>City.Name</th></tr><tr><td>Андрей</td><td>1</td><td>1</td><td>Москва</td></tr><tr><td>Сергей</td><td>1</td><td>1</td><td>Москва</td></tr><tr><td>Леонид</td><td>2</td><td>2</td><td>Санкт-Петербург</td></tr><tr><td>NULL</td><td>NULL</td><td>3</td><td>Казань</td></tr><tr><td>Григорий</td><td>4</td><td>NULL</td><td>NULL</td></tr></table>	Id	Name	1	Москва	2	Санкт-Петербург	3	Казань	Person (Люди)		Name	CityId	Андрей	1	Леонид	2	Сергей	1	Григорий	4	Person.Name	Person.CityId	City.Id	City.Name	Андрей	1	1	Москва	Сергей	1	1	Москва	Леонид	2	2	Санкт-Петербург	NULL	NULL	3	Казань	Григорий	4	NULL	NULL	
Id	Name																																													
1	Москва																																													
2	Санкт-Петербург																																													
3	Казань																																													
Person (Люди)																																														
Name	CityId																																													
Андрей	1																																													
Леонид	2																																													
Сергей	1																																													
Григорий	4																																													
Person.Name	Person.CityId	City.Id	City.Name																																											
Андрей	1	1	Москва																																											
Сергей	1	1	Москва																																											
Леонид	2	2	Санкт-Петербург																																											
NULL	NULL	3	Казань																																											
Григорий	4	NULL	NULL																																											
<div>FULL OUTER</div> <div>Без совместных данных</div>	<div></div>	<div>SELECT поля_таблиц</div> <div>FROM левая_таблица</div> <div>FULL OUTER JOIN</div> <div>правая_таблица</div> <div>ON правая_таблица.ключ =</div> <div>левая_таблица.ключ</div> <div>WHERE левая_таблица.ключ IS</div> <div>NULL</div> <div>OR правая_таблица.ключ IS</div> <div>NULL</div>																																												
<div>CROSS JOIN</div> <div>декартово</div> <div>произведение</div>	<table><tr><td></td><td>1</td><td>1</td><td>2</td><td>3</td><td>5</td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td></tr><tr><td>1</td><td>1, 1</td><td>1, 1</td><td>1, 2</td><td>1, 3</td><td>1, 5</td></tr><tr><td>6</td><td>6, 1</td><td>6, 1</td><td>6, 2</td><td>6, 3</td><td>6, 5</td></tr><tr><td>5</td><td>5, 1</td><td>5, 1</td><td>5, 2</td><td>5, 3</td><td>5, 5</td></tr></table>		1	1	2	3	5	1	1, 1	1, 1	1, 2	1, 3	1, 5	1	1, 1	1, 1	1, 2	1, 3	1, 5	6	6, 1	6, 1	6, 2	6, 3	6, 5	5	5, 1	5, 1	5, 2	5, 3	5, 5	<div>При выборе каждая строка одной</div> <div>таблицы объединяется с каждой</div> <div>строкой второй таблицы, давая</div> <div>тем самым все возможные</div> <div>сочетания строк двух таблиц.</div> <div>Порядок таблиц для оператора не</div> <div>важен, поскольку оператор</div> <div>является симметричным.</div>														
	1	1	2	3	5																																									
1	1, 1	1, 1	1, 2	1, 3	1, 5																																									
1	1, 1	1, 1	1, 2	1, 3	1, 5																																									
6	6, 1	6, 1	6, 2	6, 3	6, 5																																									
5	5, 1	5, 1	5, 2	5, 3	5, 5																																									

## 7. Что лучше использовать join или подзапросы? Почему?

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

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

## 8. Что делает UNION?

Объединяет запросы в одну таблицу

```
SELECT поля_таблиц FROM список_таблиц ...
UNION [ALL]
SELECT поля_таблиц FROM список_таблиц ... ;
```

UNION по умолчанию убирает повторения в результирующей таблице. **Для отображения с повторением есть необязательный параметр ALL.**

Не путайте операции объединения запросов с операциями объединения таблиц. Для этого служит оператор JOIN.



Не путайте операции объединения запросов с подзапросами. Подзапросы выполняются для связанных таблиц.

Объединение таблиц оператором UNION выполняется для таблиц **никак не связанных, но со схожей структурой.**

```
SELECT DISTINCT Goods.good_name AS name FROM Goods
UNION
SELECT DISTINCT FamilyMembers.member_name AS name FROM FamilyMembers;
```

Скидывает все значения в один столбец (сначала первая таблица, потом вторая)

Для того, чтобы UNION корректно сработал нужно: чтобы результирующие таблицы каждого из SQL запросов имели одинаковое число столбцов, с одним и тем же типом данных и в той же самой последовательности.

### 9. Чем WHERE отличается от HAVING (ответа про то что используются в разных частях запроса - недостаточно)?

Отличие HAVING от WHERE:

- **WHERE** — **сначала выбираются записи по условию**, а затем могут быть сгруппированы, отсортированы и т.д. Это ограничивающее выражение. Оно выполняется **до того, как будет получен результат операции.**

- **HAVING** — **сначала группируются записи, а затем выбираются по условию**, при этом, в отличие от WHERE, в нём **можно использовать значения агрегатных функций**

- Выражения **WHERE** используются вместе с операциями **SELECT, UPDATE, DELETE**, в то время как **HAVING** только с **SELECT** и предложением **GROUP BY**. То есть, использовать WHERE в запросах с агрегатными функциями нельзя, для этого и был введен HAVING.

### 10. Что такое ORDER BY?

```
SELECT поля_таблиц FROM список_таблиц
ORDER BY столбец_1 [ASC | DESC][, столбец_n [ASC | DESC]];
```

Правило сортировки применяется только к тому столбцу, за которым оно следует.

### 11. Что такое GROUP BY?

Иногда требуется узнать информацию не о самих объектах, а об определенных группах, которые они образуют. Для этого используется оператор GROUP BY и агрегатные функции.

```
SELECT family_member, SUM(unit_price * amount) FROM Payments
GROUP BY family_member;
```

При использовании GROUP BY все значения NULL считаются равными

Агрегатные функции применяются для значений, не равных NULL. Исключением является функция COUNT()

SUM(поле\_таблицы) Возвращает сумму значений

AVG(поле\_таблицы) Возвращает среднее значение

COUNT(поле\_таблицы) Возвращает количество записей

MIN(поле\_таблицы) Возвращает минимальное значение

MAX(поле\_таблицы) Возвращает максимальное значение

### 12. Что такое DISTINCT?

Оператор SQL DISTINCT используется для указания на то, что следует работать только с уникальными значениями столбца.

Может использоваться с агрегатными функциями

```
SELECT COUNT(DISTINCT Singer)
```

```
AS CountOfSingers
```

```
FROM Artists
```

### 13. Что такое LIMIT?

```
SELECT поля_выборки
```

```
FROM список_таблиц
```

```
LIMIT [ количество_пропущенных_записей, ] количество_записей_для_вывода;
```

когда необходимо сделать отступ от начала таблицы, предназначена конструкция **OFFSET** **FETCH**

Для того, чтобы вывести строки с 3 по 5, нужно использовать такой запрос:

```
SELECT * FROM Company LIMIT 2, 3;
```

### 14. Что такое EXISTS?

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

```
CREATE DATABASE IF NOT EXISTS имя_базы_данных;
```

```
DROP DATABASE IF EXISTS имя_базы_данных;
```

Обычно предикат EXISTS используется в зависимых (коррелирующих) подзапросах. Этот вид подзапроса имеет внешнюю ссылку, связанную со значением в основном запросе.

Результат подзапроса может зависеть от этого значения и должен оцениваться отдельно для каждой строки запроса, в котором содержится данный подзапрос. Поэтому предикат EXISTS может иметь разные значения для разных строк основного запроса

Найти тех производителей портативных компьютеров, которые также производят принтеры:

```
1. SELECT DISTINCT maker
2. FROM Product AS lap_product
3. WHERE type = 'laptop' AND
4. NOT EXISTS (SELECT maker
5. FROM Product
6. WHERE type = 'printer' AND
7. maker = lap_product.maker
8. );
```

### 15. Расскажите про операторы IN, BETWEEN, LIKE.

**[NOT] IN** — позволяет узнать входит ли проверяемое значение столбца в список определённых значений

<pre>SELECT * FROM Salespeople WHERE city IN ( 'Barcelona', 'London' );</pre>	<pre>SELECT * FROM Salespeople WHERE city = 'Barcelona' OR city = 'London';</pre>
---	---

**[NOT] BETWEEN min AND max** — позволяет узнать расположено ли проверяемое значение столбца в интервале между **min** и **max**. **BETWEEN**

может работать с символьными полями в терминах эквивалентов ASCII. Это означает что вы можете использовать BETWEEN чтобы выбирать ряд значений из упорядоченных по алфавиту значений.

```
SELECT *
  FROM Salespeople
 WHERE ( comm BETWEEN .10, AND .12 )
       AND NOT comm IN ( .10, .12 );

SELECT *
  FROM Customers
 WHERE cname BETWEEN 'A' AND 'G';
```

**[NOT] LIKE шаблон [ESCAPE символ]** — позволяет узнать соответствует ли строка (только CHAR или VARCHAR) определённому шаблону. В качестве условия используются символы трафаретные символы (wildkards)

**Трафаретные символы:**

- символ подчеркивания (\_), который можно применять вместо любого единичного символа в проверяемом значении
- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении.

**ESCAPE-символ** используется для экранирования трафаретных символов. Например, вы хотите получить идентификаторы задач, прогресс которых равен 3%:

```
SELECT job_id FROM Jobs
WHERE progress LIKE '3!%' ESCAPE '!';
```

## 16. Что делает оператор MERGE? Какие у него есть ограничения?

**MERGE** позволяет осуществить слияние данных одной таблицы с данными другой таблицы.

При слиянии таблиц проверяется условие, и если оно истинно, то выполняется UPDATE, а если нет - INSERT. При этом изменять поля таблицы в секции UPDATE, по которым идет связывание двух таблиц, нельзя.

Является командой DML.

```
MERGE INTO table_name USING table_reference ON (condition)
  WHEN MATCHED THEN
    UPDATE SET column1 = value1 [, column2 = value2 ...]
  WHEN NOT MATCHED THEN
    INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...]);
```

```
MERGE INTO person p
  USING ( SELECT tabn, name, age FROM person1) p1
  ON (p.tabn = p1.tabn)
  WHEN MATCHED THEN UPDATE SET p.age = p1.age
  WHEN NOT MATCHED THEN INSERT (p.tabn, p.name, p.age)
  VALUES (p1.tabn, p1.name, p1.age)
```

## 17. Какие агрегатные функции вы знаете?

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

SQL предоставляет несколько агрегатных функций:

- COUNT - производит подсчет записей, удовлетворяющих условию запроса;
- SUM - вычисляет арифметическую сумму всех значений колонки;
- AVG - вычисляет среднее арифметическое всех значений;
- MAX - определяет наибольшее из всех выбранных значений;
- MIN - определяет наименьшее из всех выбранных значений
- SUBSTRING(выражение, начальная позиция, длина) позволяет извлечь из выражения его часть заданной длины, начиная от заданной начальной позиции
- STRING\_AGG - конкатенирует строки
- STRING\_SPLIT выполняет операцию, обратную STRING\_AGG. Она принимает на входе символьную строку и разбивает её на подстроки по заданному вторым параметром разделителю
- LOWER(строковое выражение) и UPPER(строковое выражение) преобразуют все символы аргумента, соответственно, к нижнему и верхнему регистру

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

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

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

## 18. Что такое ограничения (constraints)? Какие вы знаете

Когда вы создаете таблицу (или, когда вы ее изменяете), вы можете помещать ограничение на значения которые могут быть введены:

```
CREATE TABLE < table name >
(
  < column name > < column constraint >,
  < column name > < data type > < column constraint > ...
  < table constraint > ( < column name >
  [, < column name > ])... );
```

- PRIMARY KEY - набор полей (1 или более), значения которых образуют уникальную комбинацию и используются для однозначной идентификации записи в таблице. Для таблицы может быть создано только одно такое ограничение. Данное ограничение используется для обеспечения целостности сущности, которая описана таблицей.

Первичные ключи не могут позволить значений NULL

- CHECK позволяет установить свое условие, которому должно удовлетворять значение вводимое в таблицу, прежде чем оно будет принято.
- UNIQUE обеспечивает отсутствие дубликатов в столбце или наборе столбцов.
- FOREIGN KEY защищает от действий, которые могут нарушить связи между таблицами. FOREIGN KEY в одной таблице указывает на PRIMARY KEY в другой. Поэтому данное ограничение нацелено на то, чтобы не было записей FOREIGN KEY, которым не отвечают записи PRIMARY KEY.

[CONSTRAINT имя\_ограничения]

FOREIGN KEY (столбец1, столбец2, ... столбецN)

REFERENCES главная\_таблица (столбец\_главной\_таблицы1, столбец\_главной\_таблицы2, ... столбец\_главной\_таблицыN)

[ON DELETE действие]

[ON UPDATE действие]

## - NOT NULL

Какие отличия между ограничениями PRIMARY и UNIQUE?

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

Кластерный индекс — это древовидная структура данных, при которой значения индекса хранятся вместе с данными, им соответствующими. И индексы, и данные при такой организации упорядочены. При добавлении новой строки в таблицу, она дописывается не в конец файла\*, не в конец плоского списка, а в нужную ветку древовидной структуры, соответствующую ей по сортировке.

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

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

## 19. Что такое суррогатные ключи?

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

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

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

**Потенциальный ключ** - простой или составной ключ, который уникально идентифицирует каждую запись набора данных. При этом потенциальный ключ должен обладать критерием избыточности: при удалении любого из полей набор полей перестает уникально идентифицировать запись. Термин «candidate» подразумевает, что все такие ключи конкурируют за почётную роль «первичного ключа» (primary key), а оставшиеся назначаются «альтернативными ключами» (alternate keys).

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

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

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

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

Что такое «внешний ключ» (foreign key)?

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

## 20. Что такое индексы? Какие они бывают?

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

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

### Преимущества

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

### Недостатки

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

### Индексы предпочтительней для:

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

### Использование индексов нецелесообразно для:

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

### Типы индексов:

По порядку сортировки: упорядоченные — индексы, в которых элементы упорядочены;

По источнику данных: индексы по представлению (view); индексы по выражениям.

### По воздействию на источник данных вы можете помещать

- кластерный индекс - при определении в наборе данных физическое расположение данных перестраивается в соответствии со структурой индекса. Логическая структура набора данных в этом случае представляет собой скорее словарь, чем индекс. Данные в словаре физически упорядочены, например по алфавиту. Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными индексами. Увеличение производительности особенно заметно при работе с последовательными данными.



- некластерный индекс — наиболее типичные представители семейства индексов. В отличие от кластерных, они не перестраивают физическую структуру набора данных, а лишь организуют ссылки на соответствующие записи. Для идентификации нужной записи в наборе данных некластерный индекс организует специальные указатели, включающие в себя: информацию об идентификационном номере файла, в котором хранится запись; идентификационный номер страницы соответствующих данных; номер искомой записи на соответствующей странице; содержимое столбца.

По структуре: В\*-деревья; В+-деревья; В-деревья; Хэши.

По количественному составу

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

По характеристике содержимого

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

- обратный индекс (reverse index) — B-tree индекс, но с реверсированным ключом, используемый в основном для монотонно возрастающих значений (например, автоинкрементный идентификатор) в OLTP системах с целью снятия конкуренции за последний листовой блок индекса, т.к. благодаря переворачиванию значения две соседние записи индекса попадают в разные блоки индекса. Он не может использоваться для диапазонного поиска.
- функциональный индекс, индекс по вычисляемому полю (function-based index) — индекс, ключи которого хранят результат пользовательских функций. Функциональные индексы часто строятся для полей, значения которых проходят предварительную обработку перед сравнением в команде SQL. Например, при сравнении строковых данных без учета регистра символов часто используется функция UPPER. Кроме того, функциональный индекс может помочь реализовать любой другой отсутствующий тип индексов данной СУБД.
- первичный индекс — уникальный индекс по полю первичного ключа.
- вторичный индекс — индекс по другим полям (кроме поля первичного ключа).
- XML-индекс — вырезанное материализованное представление больших двоичных XML-объектов (BLOB) в столбце с типом данных xml.

По механизму обновления

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

По покрытию индексируемого содержимого

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

Индексы в кластерных системах

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

В чем отличие между кластерными и некластерными индексами?

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

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

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

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

Когда полное сканирование набора данных выгоднее доступа по индексу?

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

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

Как создать индекс?

Индекс можно создать либо с помощью выражения **CREATE INDEX**:

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

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

## 21. Чем TRUNCATE отличается от DELETE?

Delete в целом не предназначена для полной очистки таблицы - генерит редо, поддерживает индексы, то есть работает медленно.

Truncate наоборот предназначен именно для быстрой очистки ВСЕЙ таблицы или партии - освобождает занятые экстенды.

- Операция TRUNCATE не записывает в журнал событий удаление отдельных строк. Вследствие чего не может активировать триггеры.
- После операции TRUNCATE для некоторых СУБД (например, Oracle) следует неявная операция COMMIT. Поэтому удаленные в таблице записи нельзя восстановить операцией ROLLBACK. Но существуют и СУБД, в которых операция TRUNCATE может участвовать в транзакциях, например, PostgreSQL и Microsoft SQL Server.
- Операция DELETE блокирует каждую строку, а TRUNCATE — всю таблицу.
- Операция TRUNCATE не возвращает какого-то осмысленного значения (обычно возвращает 0) в отличие от DELETE, которая возвращает число удаленных строк.
- Операция TRUNCATE в некоторых СУБД (например, MySQL или Microsoft SQL Server), сбрасывает значение счетчиков (для полей с AUTOINCREMENT / IDENTITY). В PostgreSQL для сброса счётчиков необходимо указывать модификатор RESTART IDENTITY.
- Операция TRUNCATE в некоторых СУБД (например, MySQL, PostgreSQL или Microsoft SQL Server) запрещена для таблиц, содержащих внешние ключи других таблиц. В PostgreSQL существует, однако, модификатор CASCADE, который разрешает TRUNCATE в этой ситуации — данные из зависимых таблиц удаляются в той же транзакции.

- В [SQLite](#) операция как таковая отсутствует, но есть оптимизация операции DELETE, которая «значительно ускоряет её работу, если отсутствует аргумент WHERE».

## 22. Что такое хранимые процедуры? Для чего они нужны?

Это муветон, нарушается принцип ,..., надо лезть в базу чтобы исправить её, а без неё вся логика в коде

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

## 23. Что такое представления (VIEW)? Для чего они нужны?

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

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

```
CREATE VIEW Londonstaff
AS SELECT *
FROM Salespeople
WHERE city = 'London';
```

## 24. Что такое временные таблицы? Для чего они нужны?

**Временная таблица** - это объект базы данных, который хранится и управляется системой **базы данных на временной основе**. Они могут быть локальными (только я могу работать) или глобальными (все). Используется для сохранения результатов вызова хранимой процедуры, уменьшение числа строк при соединениях, агрегирование данных из различных источников или как замена курсоров и параметризованных представлений. Срок жизни временной таблицы – сеанс с БД

## 25. Что такое транзакции? Расскажите про принципы **ACID**.

**Транзакция** - это воздействие на базу данных, переводящее её из одного целостного состояния в другое и **выражаемое в изменении данных**, хранящихся в базе данных. это N ( $N \geq 1$ ) запросов к БД, которые выполняются успешно все вместе или не выполняются вовсе.

Назовите основные свойства транзакции.

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

- **Согласованность (consistency)**. Транзакция, достигающая своего нормального завершения (EOT — end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты.
- **Изолированность (isolation)**. Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на ее результат.
- **Долговечность (durability)**. Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбой в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

## 26. Расскажите про уровни изолированности транзакций.

Выбирая используемый уровень изолированности транзакций, разработчик информационной системы в определённой мере обеспечивает выбор между скоростью работы и обеспечением гарантированной согласованности получаемых из системы данных. При параллельном выполнении транзакций возможны следующие проблемы:

1. **потерянное обновление (англ. lost update)** — при одновременном изменении одного блока данных разными транзакциями теряются все изменения, кроме последнего;
2. **«грязное» чтение (англ. dirty read)** — чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится);
3. **неповторяющееся чтение (англ. non-repeatable read)** — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;
4. **фантомное чтение (англ. phantom reads)** — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет строки или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк.

Уровни изолированности транзакций:

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

- Чтение неподтверждённых данных (грязное чтение) (**read uncommitted, dirty read**) — чтение незафиксированных изменений как своей транзакции, так и параллельных транзакций. Нет гарантии, что данные, изменённые другими транзакциями, не будут в любой момент изменены в результате их отката, поэтому такое чтение является потенциальным источником ошибок. Невозможны потерянные изменения, возможны неповторяемое чтение и фантомы.
- Чтение подтверждённых данных (**read committed**) — чтение всех изменений своей транзакции и зафиксированных изменений параллельных транзакций. Потерянные изменения и грязное чтение не допускается, возможны неповторяемое чтение (когда мы видим обновлённые и удалённые строки (UPDATE, DELETE)) и фантомы (когда мы видим добавленные записи (INSERT)).
- Повторяемость чтения (**repeatable read, snapshot**) — чтение всех изменений своей транзакции, любые изменения, внесённые параллельными транзакциями после начала своей, недоступны. Потерянные изменения, грязное и неповторяемое чтение невозможны, возможны фантомы.

- Упорядочиваемость (**serializable**) — результат параллельного выполнения сериализуемой транзакции с другими транзакциями должен быть логически эквивалентен результату их какого-либо последовательного выполнения. Проблемы синхронизации не возникают.

Уровни	Не должно быть	Будут проблемы
<b>read uncommitted, dirty read</b> видят не закомиченные результаты TP + 2 проблемы	lost update	dirty read + non-repeatable read+ phantom reads
<b>read committed</b> видят закомиченные результаты + 2 проблемы	lost update + dirty read	non-repeatable read+ phantom reads
<b>repeatable read, snapshot</b> видят результаты Update и Delete + 1 проблема	lost update + dirty read + non-repeatable read	phantom reads
<b>Serializable</b>	lost update + dirty read + non-repeatable read+ phantom reads	Нет проблем с синхронизацией

Какие проблемы могут возникать при параллельном доступе с использованием транзакций? При параллельном выполнении транзакций возможны следующие проблемы:

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

## 27. Что такое нормализация и денормализация? Расскажите про 3 нормальные формы?

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

Нормализация – это и есть здравый смысл в проектировании БД.

Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей (строк таблицы).

**Назначение процесса нормализации** заключается в следующем:

- **исключение некоторых типов избыточности** (чтобы в каждом отношении хранились только первичные факты (то есть факты, не выводимые из других хранимых фактов);



- устранение некоторых аномалий обновления;
- разработка проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;
- упрощение процедуры применения необходимых ограничений целостности.

#### нормальные формы:

- Первая нормальная форма (1NF) - Отношение находится в 1NF, если значения всех его атрибутов атомарны (неделимы).
- Вторая нормальная форма (2NF) - Отношение находится в 2NF, если оно находится в 1NF, и при этом все неключевые атрибуты зависят только от ключа целиком, а не от какой-то его части.
- Третья нормальная форма (3NF) - Отношение находится в 3NF, если оно находится в 2NF и все неключевые атрибуты не зависят друг от друга.
- Четвёртая нормальная форма (4NF) - Отношение находится в 4NF, если оно находится в 3NF и если в нем не содержатся независимые группы атрибутов, между которыми существует отношение «многие-ко-многим».
- Пятая нормальная форма (5NF) - Отношение находится в 5NF, когда каждая нетривиальная зависимость соединения в ней определяется потенциальным ключом (ключами) этого отношения.
- Шестая нормальная форма (6NF) - Отношение находится в 6NF, когда она удовлетворяет всем нетривиальным зависимостям соединения, т.е. когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6NF, также находится и в 5NF. Введена как обобщение пятой нормальной формы для хронологической базы данных.
- Нормальная форма Бойса-Кодда, усиленная 3 нормальной формой (BCNF) - Отношение находится в BCNF, когда каждая её нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ.
- Доменно-ключевая нормальная форма (DKNF) - Отношение находится в DKNF, когда каждое наложенное на нее ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данное отношение.

#### Первая нормальная форма

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения (может быть выражено одним, как правило, действительным числом). Не должно быть повторений строк в таблице.

Например, есть таблица «Автомобили»:

Фирма	Модели
BMW	M5, X5M, M1
Nissan	GT-R

Нарушение нормализации 1НФ происходит в моделях BMW, т.к. в одной ячейке содержится список из 3 элементов: M5, X5M, M1, т.е. он не является атомарным.

Преобразуем таблицу к 1НФ (в одной ячейке – одно значение):

Фирма	Модели
BMW	M5
BMW	X5M
BMW	M1
Nissan	GT-R

**Вторая нормальная форма** (прослеживается 2 зависимости из которых можно выделить 2-ю таблицу)

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Например, дана таблица:

<u>Модель</u>	<u>Фирма</u>	Цена	Скидка
M5	BMW	5500000	5%
X5M	BMW	6000000	5%
M1	BMW	2500000	5%
GT-R	Nissan	5000000	10%

Таблица находится в первой нормальной форме, но не во второй. Цена машины зависит от модели и фирмы. Скидка зависит от фирмы, то есть зависимость от первичного ключа неполная. Исправляется это путем декомпозиции на два отношения, в которых не ключевые атрибуты зависят от ПК.

<u>Модель</u>	<u>Фирма</u>	Цена
M5	BMW	5500000
X5M	BMW	6000000
M1	BMW	2500000
GT-R	Nissan	5000000

<u>Фирма</u>	Скидка
--------------	--------

BMW	5%
Nissan	10%

### Третья нормальная форма

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Рассмотрим таблицу:

<u>Модель</u>	Магазин	Телефон
BMW	Риал-авто	87-33-98
Audi	Риал-авто	87-33-98
Nissan	Некст-Авто	94-54-12

Таблица находится во 2НФ, но не в 3НФ.

В отношении атрибут «Модель» является первичным ключом. Личных телефонов у автомобилей нет, и телефон зависит исключительно от магазина.

Таким образом, в отношении существуют следующие функциональные зависимости:

Модель → Магазин, Магазин → Телефон, Модель → Телефон.

Зависимость Модель → Телефон является транзитивной, следовательно, отношение не находится в 3НФ.

В результате разделения исходного отношения получаются два отношения, находящиеся в 3НФ:

<u>Магазин</u>	Телефон
Риал-авто	87-33-98
Некст-Авто	94-54-12

<u>Модель</u>	Магазин
BMW	Риал-авто
Audi	Риал-авто
Nissan	Некст-Авто

Нормальная форма Бойса-Кодда (НФБК) (частная форма третьей нормальной формы)

Определение ЗНФ не совсем подходит для следующих отношений:

- 1) отношение имеет два или более потенциальных ключа;
- 2) два и более потенциальных ключа являются составными;
- 3) они пересекаются, т.е. имеют хотя бы один общий атрибут.

Для отношений, имеющих один потенциальный ключ (первичный), НФБК является ЗНФ. Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта. Предположим, рассматривается отношение, представляющее данные о бронировании стоянки на день:

Номер стоянки	Время начала	Время окончания	Тариф
1	09:30	10:30	Бережливый
1	11:00	12:00	Бережливый
1	14:00	15:30	Стандарт
2	10:00	12:00	Премиум-В
2	12:00	14:00	Премиум-В
2	15:00	18:00	Премиум-А

Тариф имеет уникальное название и зависит от выбранной стоянки и наличия льгот, в частности:

- «Бережливый»: стоянка 1 для льготников
- «Стандарт»: стоянка 1 для не льготников
- «Премиум-А»: стоянка 2 для льготников
- «Премиум-В»: стоянка 2 для не льготников.

Таким образом, возможны следующие составные первичные ключи: {Номер стоянки, Время начала}, {Номер стоянки, Время окончания}, {Тариф, Время начала}, {Тариф, Время окончания}.

Отношение находится в ЗНФ. Требования второй нормальной формы выполняются, так как все атрибуты входят в какой-то из потенциальных ключей, а неключевых атрибутов в отношении нет. Также нет и транзитивных зависимостей, что соответствует требованиям третьей нормальной формы. Тем не менее, существует функциональная зависимость Тариф  $\rightarrow$  Номер стоянки, в которой левая часть (детерминант) не является потенциальным ключом отношения, то есть отношение не находится в нормальной форме Бойса — Кодда.

Недостатком данной структуры является то, что, например, по ошибке можно приписать тариф «Бережливый» к бронированию второй стоянки, хотя он может относиться только к первой стоянке.

Можно улучшить структуру с помощью декомпозиции отношения на два и добавления атрибута **Имеет льготы**, получив отношения, удовлетворяющие НФБК (подчёркнуты атрибуты, входящие в первичный ключ.):

### Тарифы

<u>Тариф</u>	Номер стоянки	Имеет льготы
Бережливый	1	Да
Стандарт	1	Нет
Премиум-А	2	Да
Премиум-В	2	Нет

### Бронирование

<u>Тариф</u>	<u>Время начала</u>	Время окончания
Бережливый	09:30	10:30
Бережливый	11:00	12:00
Стандарт	14:00	15:30
Премиум-В	10:00	12:00
Премиум-В	12:00	14:00
Премиум-А	15:00	18:00

### Четвертая нормальная форма

Отношение находится в 4НФ, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.

В отношении R (A, B, C) существует **многозначная зависимость** R.A  $\twoheadrightarrow$  R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Предположим, что рестораны производят разные виды пиццы, а службы доставки ресторанов работают только в определенных районах города. Составной первичный ключ соответствующей переменной отношения включает три атрибута: {Ресторан, Вид пиццы, Район доставки}.

Такая переменная отношения не соответствует 4НФ, так как существует следующая многозначная зависимость:

{Ресторан}  $\twoheadrightarrow$  {Вид пиццы}

{Ресторан}  $\twoheadrightarrow$  {Район доставки}

То есть, например, при добавлении нового вида пиццы придется внести по одному новому кортежу для каждого района доставки. Возможна логическая аномалия, при которой определенному виду пиццы будут соответствовать лишь некоторые районы доставки из обслуживаемых рестораном районов.

Для предотвращения аномалии нужно декомпозировать отношение, разместив независимые факты в разных отношениях. В данном примере следует выполнить декомпозицию на {Ресторан, Вид пиццы} и {Ресторан, Район доставки}.

Однако, если к исходной переменной отношения добавить атрибут, функционально зависящий от потенциального ключа, например цену с учётом стоимости доставки ({Ресторан, Вид пиццы, Район доставки} → Цена), то полученное отношение будет находиться в 4НФ и его уже нельзя подвергнуть декомпозиции без потерь.

### Пятая нормальная форма

Отношения находятся в 5НФ, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

Если «Атрибут\_1» зависит от «Атрибута\_2», а «Атрибут\_2» в свою очередь зависит от «Атрибута\_3», а «Атрибут\_3» зависит от «Атрибута\_1», то все три атрибута обязательно входят в один кортеж.

Это очень жесткое требование, которое можно выполнить лишь при дополнительных условиях. На практике трудно найти пример реализации этого требования в чистом виде.

Например, некоторая таблица содержит три атрибута «Поставщик», «Товар» и «Покупатель». Покупатель\_1 приобретает несколько Товаров у Поставщика\_1. Покупатель\_1 приобрел новый Товар у Поставщика\_2. Тогда в силу изложенного выше требования Поставщик\_1 обязан поставлять Покупателю\_1 тот же самый новый Товар, а Поставщик\_2 должен поставлять Покупателю\_1, кроме нового Товара, всю номенклатуру Товаров Поставщика\_1. Этого на практике не бывает. Покупатель свободен в своем выборе товаров. Поэтому для устранения отмеченного затруднения все три атрибута разносят по разным отношениям (таблицам). После выделения трех новых отношений (Поставщик, Товар и Покупатель) необходимо помнить, что при извлечении информации (например, о покупателях и товарах) необходимо в запросе соединить все три отношения. Любая комбинация соединения двух отношений из трех неминуемо приведет к извлечению неверной (некорректной) информации. Некоторые СУБД снабжены специальными механизмами, устраняющими извлечение недостоверной информации. Тем не менее, следует придерживаться общей рекомендации: структуру базы данных строить таким образом, чтобы избежать применения 4НФ и 5НФ.

Пятая нормальная форма ориентирована на работу с зависимыми соединениями. Указанные зависимые соединения между тремя атрибутами встречаются очень редко. Зависимые соединения между четырьмя, пятью и более атрибутами указать практически невозможно.

### Доменно-ключевая нормальная форма



Переменная отношения находится в ДКНФ тогда и только тогда, когда каждое наложенное на неё ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данную переменную отношения.

Ограничение домена – ограничение, предписывающее использовать для определённого атрибута значения только из некоторого заданного домена. Ограничение по своей сути является заданием перечня (или логического эквивалента перечня) допустимых значений типа и объявлением о том, что указанный атрибут имеет данный тип.

Ограничение ключа – ограничение, утверждающее, что некоторый атрибут или комбинация атрибутов является потенциальным ключом.

Любая переменная отношения, находящаяся в ДКНФ, обязательно находится в 5НФ. Однако не любую переменную отношения можно привести к ДКНФ.

### Шестая нормальная форма

Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ.

Идея «декомпозиции до конца» выдвигалась до начала исследований в области хронологических данных, но не нашла поддержки. Однако для хронологических баз данных максимально возможная декомпозиция позволяет бороться с избыточностью и упрощает поддержание целостности базы данных.

Для хронологических баз данных определены U\_операторы, которые распаковывают отношения по указанным атрибутам, выполняют соответствующую операцию и упаковывают полученный результат. В данном примере соединение проекций отношения должно производиться при помощи оператора U\_JOIN.

### Работники

<u>Таб.№</u>	<u>Время</u>	Должность	Домашний адрес
6575	01-01-2000:10-02-2003	слесарь	ул.Ленина,10
6575	11-02-2003:15-06-2006	слесарь	ул.Советская,22
6575	16-06-2006:05-03-2009	бригадир	ул.Советская,22

Переменная отношения «Работники» не находится в 6НФ и может быть подвергнута декомпозиции на переменные отношения «Должности работников» и «Домашние адреса работников».

### Должности работников

<u>Таб.№</u>	<u>Время</u>	Должность
6575	01-01-2000:10-02-2003	слесарь
6575	16-06-2006:05-03-2009	бригадир

#### Домашние адреса работников

<u>Таб.№</u>	<u>Время</u>	Домашний адрес
6575	01-01-2000:10-02-2003	ул.Ленина,10
6575	11-02-2003:15-06-2006	ул.Советская,22

#### 28. Что такое TIMESTAMP?

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

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