***Structured Query Language, SQL*** (**язык структурированных запросов**) – это непроцедурный язык программирования, который работает практически с любой базой данных. Он является **декларативным** и описывает что, а не как нужно сделать. Он не чувствителен к реестру, но лучше соблюдать стиль.

**Реляционная БД** – это БД, в которой данные представлены в виде связанных таблиц. Столбцы называются **полями, колонками** или **признаками**, строки – **записями**, объекты пересечения полей и записей – **ячейками**.

***Database management system*, СУБД** – это система управления базами данных.



Для корректного написания SQL-запросов можно использовать [*SQLFormater*](https://sqlformat.org/) или следующие правила:

1. Писать операторы в верхнем регистре, чтобы визуально отделить их от ключевых слов.
2. Переносить новое ключевое слово на другую строку, чтобы легче понять какие столбцы будут выгружены.

Чтобы закомментировать строку в SQL используются два дефиса ***--***. Чтобы закомментировать несколько строк используются символы ***/\**** в начале и ***\*/*** в конце.

Типы данных в *PostgreSQL*:

* ***integer*** хранит целые числа от -2147483649 до +2147483647. Его обозначают либо ***int***, либо ***int4*** (занимает 4 байта).
* ***real*** обозначает вещественные числа (дробные). Его обозначают ***float4***.
* ***character*** содержит текст фиксированной длины. Псевдоним – ***char(n)***, где *n* – максимальное количество символов в строке (недостающие символы будут заполнены пробелами). По умолчанию = 1.
* ***character varying*** содержит тексты нефиксированной длины. Псевдоним – ***varchar(n)***, где *n* управляет лимитом строки (можно не указывать).
* ***text*** содержит строку любой длины. Нестандартный тип в *SQL*, используется в некоторых СУБД.
* ***date*** содержит только дату. Обычно используется в международном формате *ISO* *‘YYYY-MM-DD’*.
* ***timestamp*** содержит дату и время. Подразделяется на *timestamp without time zone ‘2004-10-19 10:23:54’* (используется по умолчанию) и *timestamp with time zone ‘2004-10-19 10:23:54+02’*.
* ***boolean*** обозначает логический тип. Содержит два значения *TRUE* ('true', 't', 'yes', 'y', 'on', '1') и *FALSE* (‘false', 'f', 'no', 'n', 'off', '0').

Операторы обращения к БД:

* ***SELECT*** – описывает что выгружать из базы.
* ***FROM*** – описывает откуда именно выгружать данные.
* ***WHERE*** – оператор сравнения, используется как фильтр при выборке (***>, <, >=, <=, =, !=, <>***). Ставят после *FROM*. При сравнении с символьным типом, датой и временем набор символов нужно взять в ***‘***одинарные кавычки***’***. Работает только с изначальной таблицей, но не с результатом агрегирующих функций.
* ***HAVING*** – используется, чтобы получить срез данных группировки (результат агрегирующих функций). Используется после *GROUP BY* (если есть). Например, *HAVING SUM(total) > 41*.
* ***AND, OR, NOT*** – логические операторы, используются для комбинирования условий выборки через оператор *WHERE*. Приоритет операторов в порядке уменьшения – *NOT, AND, OR*. В остальных случаях используют ***(***скобки***)***, действия в которых имеет наивысший приоритет.
* ***IS NULL, IS NOT NULL*** – используются для фильтрации или проверки наличия данных. В *SQL* используется специальное значение ***NULL*** для обозначения отсутствия данных. С *NULL* нельзя сравнивать какое-либо значение.
* ***GROUP BY*** – оператор группировки данных по полю. Ставят после *WHERE* (если отсутствует – после *FROM*). Порядок группировки по нескольким полям не влияет на выдачу. Запрос *SELECT city FROM table GROUP BY city* выгрузит все уникальные значения в поле (аналогично *DISTINCT*).
* ***ORDER BY*** – оператор сортировки. После него можно указать только *LIMIT*. По умолчанию сортирует данные от меньшего к большем. Чтобы изменить порядок сортировки вручную, после названия поля указывают ключевое слово ***DESC*** (по убыванию) или ***ASC*** (по возрастанию). Строки будут отсортированы в лексикографическом порядке. Порядок сортировки по нескольким полям влияет на выдачу.
* ***IN, NOT IN*** – позволяет определить, совпадает ли значение объекта со значением в списке. Например, *WHERE billing\_country IN ('USA', 'United Kingdom', 'Germany')*.
* ***LIMIT*** – указывает количество первых записей, которые нужно выгрузить. Ставят в конце запроса.
* ***OFFSET*** – указывает сколько элементов пропустить. Используется для пагинации.

Помимо операций выборки данных, *SQL* позволяет манипулировать данными.

1. Поддерживаются математические операции ***+, -, \*, /*** с числами, но не поддерживает операции конкатенации строк.
2. Поддерживаются агрегирующие функции.

* ***SUM(поле)*** – возвращает сумму значений в поле.
* ***AVG(поле)*** – находит среднее арифметическое для значений в поле.
* ***MIN(поле)*** – возвращает минимальное значение в поле.
* ***MAX(поле)*** – возвращает максимальное значение в поле.
* ***COUNT(поле)*** – выводит количество записей в поле без учета пропусков. Можно использовать для всей таблицы (пропуски будут учтены).
* ***ROUND(поле)*** – округляет поле до ближайшего целого числа.

**Первичный ключ** (***primary key, PK***) – это уникальный признак записи. Он обеспечивает уникальность записей и может быть представлен набором любых символов.

**Внешний ключ** (***foreign key, FK***) – это поле, которое отсылает к первичному ключу другой таблицы.

В реляционных БД существует три основных вида связи таблиц:

1. **Один к одному (*one-to-one*)** – одна запись первой таблицы связана только с одной записью другой таблицы. Используется редко, т.к. такие таблицы можно объединить в одну.

Например, запись о сотруднике из таблицы отдела разработки связана с таблицей всего персонала связью *one-to-one*.

1. **Один ко многим (*one-to-many*)** – одной записи первой таблицы соответствует несколько записей другой таблицы. Самый популярный вид связи.

Например, запись о клиенте из таблицы с клиентами связана с несколькими заказами из таблицы заказов связью *one-to-many*.

1. **Многие ко многим (*many-to-many*)** – одной записи первой таблицы соответствует несколько записей другой таблицы и наоборот, одной записи другой таблицы соответствует несколько записей первой таблицы.

Например, отношение таблиц с вкладчиками и банками. Один человек может открыть несколько вкладов в разных банках, а один банк может открывать вклады разным людям.

Реализовать такую связь бывает сложно и неудобно, поэтому используются **соединительные таблицы**. В нашем примере это будет таблица с *id* вкладчиков и *id* банков реализующая связь *one-to-many*.

На *ER*-диаграмме типы связей отображены таким образом:



Пример связывания таблиц:



Ключевое слово ***DISTINCT*** возвращает уникальные значения. Обычно используется после *SELECT*. Если перечислить несколько полей после *DISTINCT*, запрос выведет все уникальные комбинации значений в этих полях. Может сочетаться с агрегирующими функциями.

Функция ***CAST*** приводит значение к заданному типу данных. Например, *CAST(invoice\_date AS DATE)* конвертирует значение поля *invoice\_date* к типу *date*.

Функция ***EXTRACT*** позволяет извлечь значение из данных. Например, можно извлечь год с помощью такой записи *EXTRACT(YEAR FROM CAST(invoice\_date AS DATE))*.

Предикат ***LIKE*** ищет нужную строку в текстовых значениях поля. Символ ***%*** соответствует любому количеству символов. Символ ***\_*** соответствует одному символу. Например, отфильтруем записи, содержащие слово *Epic* – *WHERE descrition LIKE ‘%Epic%’*.

**Псевдоним (*alias*)** – временно название, которое присваивают в запросе таблице или полю. При этом в исходной БД названия не меняются. К псевдонимам нельзя обращаться в *WHERE* или *HAVING* (во многих и в *GROUP BY*), потому что сначала данные выбираются по условию, а только потом полям назначают псевдонимы.

Задать псевдоним можно двумя способами – через пробел или ***AS***:

*SELECT EXTRACT(YEAR FROM CAST(invoice\_date AS DATE)) AS year\_of\_purchase*

*SELECT EXTRACT(YEAR FROM CAST(invoice\_date AS DATE)) year\_of\_purchase*

При использовании псевдонимов таблиц можно указывать имена полей через точку после псевдонима. При работе с несколькими таблицами в одном запросе полезно указывать вместе с полем и таблицу-источник.

*FROM invoice AS i*

*MIN(i.total)*

Операторы JOIN объединяют таблицы “по горизонтали”, а операторы UNION – “по вертикали”.



Оператор ***INNER JOIN*** (внутреннее объединение), или кратко ***JOIN***, предполагает объединение по “внутренней области”, общей для двух таблиц. Например, объединим таблицы по актеру.

Оператор ***LEFT OUTER JOIN*** (левое внешнее объединение), или кратко ***LEFT JOIN***, предполагает, что в результат слияния войдут все записи из левой таблицы. Записи из правой таблицы сохранятся только в том случае, если значения в нужном поле совпадают со значениями в левой таблице. Все из левой плюс общее из правой.

Оператор ***RIGHT OUTER JOIN*** (правое внешнее объединение), или кратко ***RIGHT JOIN***, предполагает, что в результат слияния войдут все записи из правой таблицы. Записи из левой таблицы сохранятся только в том случае, если значения в нужном поле совпадают со значениями в правой таблице. Все из правой плюс общее из левой.

Оператор ***FULL OUTER JOIN*** (полное внешнее объединение), или кратко ***FULL JOIN***, объединяет все данные из левой и правой таблиц. Если не нашлось совпадений, на месте значения будет *NULL*.



Объединять таблицы можно не только по полям, которые являются внешними ключами, но и по другим полям, если типы этих полей совпадают.

Оператор ***UNION*** предполагает склейку двух таблиц с одинаковым количеством и типом полей. Абсолютные дубликаты в итоговую таблицу не входят. Используется между *SELECT*.

Оператор ***UNION ALL*** предполагает склейку двух таблиц с одинаковым количеством и типом полей. Абсолютные дубликаты в итоговую таблицу входят. Используется между *SELECT*.

Таблицу можно присоединять саму к себе. Например, для выстраивания иерархии сотрудников отдела.



**Подзапрос**, или **вложенный запрос** – это обычный запрос, который используется после *FROM* или *WHERE*, т.е. полученную таблицу с данными можно использовать вместо условия. Преимущества – сначала идет отбор данных, а потом объединение таблиц, что позволяет экономить время.



**Нормализация** – это метод проектирования БД, который позволяет привести БД к минимальной избыточности. Другими словами – это процесс удаления избыточных данных. Она нужна для устранения аномалий, повышения производительности и повышения удобства управления данными (где-то до *3NF*).

Степень нормализации градируется по следующим формам (в порядке увеличения нормальности):

1. **Ненормализованная форма или нулевая нормальная форма (*UNF*)**:

* БД приведена к табличному виду так, чтобы он отвечал базовым принципам реляционной теории (строки и столбцы не должны быть пронумерованы, их порядок не должен иметь значения).



1. **Первая нормальная форма (*1NF*)**:

* В таблице не должно быть дублирующих строк.
* В каждой ячейке таблицы хранится атомарное (одно не составное) значение.
* В столбце хранятся данные одного типа.
* Отсутствуют массивы и списки в любом виде.



1. **Вторая нормальная форма (*2NF*)**:

* Таблица должна иметь ключ, по которому можно идентифицировать каждую строку.
* Все не ключевые столбцы таблицы должны зависеть от полного ключа (если он составной). Если какой-то столбец зависит только от части составного ключа, то БД не находится в *2NF*.



Пример с составным ключом (потребует декомпозиции и соединительной таблицы):



1. **Третья нормальная форма (*3NF*)**:

* В таблицах должна отсутствовать **транзитивная зависимость** (когда не ключевые столбцы зависят от значений других не ключевых столбцов).



1. **Нормальная форма Бойса-Кодда (*BCNF*)**:

* Ключевые столбцы составного ключа не должны зависеть от не ключевых столбцов.



1. **Четвертая нормальная форма (*4NF*)**:

* В таблице не должно быть многозначных зависимостей (когда два столбца никак не связаны друг с другом, но оба зависят от третьего столбца).



1. **Пятая нормальная форма (*5NF*)**:

* В таблице каждая нетривиальная зависимость соединения должна определяться потенциальным ключом этой таблицы.

1. **Доменно-ключевая нормальная форма (*DKNF*)**:

* Каждое наложенное ограничение на таблицу должно являться логическим следствием ограничений доменов и ограничений ключей, которые накладываются на данную таблицу.

**Ограничение домена** – это ограничение, предписывающее использование для определенного столбца значений только из некоторого заданного домена (набора значений).

**Ограничение ключа** – это ограничение, утверждающее, что некоторый столбец или их комбинация представляет собой потенциальный ключ.

1. **Шестая нормальная форма (*6NF*)**:

* Таблица должна удовлетворять всем нетривиальным зависимостям соединения.

Приведение БД к какой-то конкретной нормальной форме требует, чтобы эта база уже находилась в предыдущей нормальной форме. БД считается нормализованной, если она находится как минимум в третьей нормальной форме.

**CRUD** (Create/Read/Update/Delete) – стандартная классификация функций по манипуляции данными. **CRUD приложение** – это приложение, реализующее весь этот функционал. Рассмотрим общепринятую конвенцию именования запросов URL на примере сущности *Post*:

|  |  |  |
| --- | --- | --- |
| **HTTP метод** | **URL** | **Действие** |
| GET | /posts | Получаем все записи (READ) |
| POST | /posts | Создаем новую запись (CREATE) |
| GET | /posts/new | HTML форма создания записи |
| GET | /posts/:id/edit | HTML форма редактирования записи |
| GET | /posts/:id | Получаем одну запись (READ) |
| PATCH | /posts/:id | Обновляем запись (UPDATE) |
| DELETE | /posts/:id | Удаляем запись (DELETE) |

Создать новую пустую таблицу можно с помощью оператора ***CREATE TABLE***. Владельцем таблицы будет пользователь, выполнивший команду.

Свойство ***GENERATED BY DEFAULT AS IDENTITY*** означает, что столбец является столбцом идентификации с автоинкрементом.

Задавать ограничения ***CONSTRANT*** (например, *NOT NULL*, *PRIMARY KEY*) можно индивидуально на каждый столбец либо на всю таблицу сразу. Определяя ограничения на уровне таблиц, можно задать ему любое имя.

Оператор ***CHECK*** задает условия проверки.

*CREATE TABLE films ( id INTEGER GENERATED BY DEFAULT AS IDENTITY,*

*title varchar(40) NOT NULL,*

*produced integer NOT NULL,*

*date\_prod date,*

*kind varchar(10),*

*len\_min integer,*

*CONSTRAINT films\_pk PRIMARY KEY (id),*

*CONSTRAINT constr\_example CHECK (len\_min > 100 AND title <> '') );*

Свойство ***DEFAULT*** позволяет задавать значения по умолчанию для столбцов.

*name varchar(40) DEFAULT 'unknown'*

Свойство ***REFERENCES*** позволяет указать, что столбец является вторичным ключом для какой-то другой таблицы. После него указывается имя таблицы и связанный столбец. Все таблицы, на которые мы ссылаемся, к этому моменту должны быть созданы.

При удалении полей из связанных столбцов можно использовать ограничения:

* ***RESTRICT*** – запрещает удаление записи, на которую есть ссылка
* ***CASCADE*** – удаляет все связанные записи
* ***NO ACTION*** – выдаст ошибку. Используется по умолчанию.
* ***SET NULL*** и ***SET DEFAULT*** – присваивает значение NULL или значение по умолчанию

*CREATE TABLE favorite\_films (*

*id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,*

*user\_id INTEGER REFERENCES users (id) ON DELETE RESTRICT,*

*film\_id INTEGER REFERENCES films (id) ON DELETE CASCADE );*

Удалить таблицу и все связанные с ней объекты можно с помощью оператора ***DROP TABLE***. Если на удаляемую таблицу ссылается другая таблица, то удаление не сработает, т.к. по умолчанию действует режим *RESTRICT*. Если явно использовать режим *CASCADE*, то таблица, которая ссылается на данные из текущей таблицы, будет полностью удалена.

*DROP TABLE films, users CASCADE;*

Оператор ***IF EXISTS*** позволяет проверить наличие таблицы перед операцией над ней.

*DROP TABLE IF EXISTS films;*

Изменить существующую таблицу можно с помощью оператора ***ALTER TABLE***. С его помощью можно добавлять и удалять столбцы и ограничения, а также переименовывать столбцы и таблицы. Новые столбцы заполняются *NULL* или заданными значениями по умолчанию.

*ALTER TABLE products ADD COLUMN description text;*

*ALTER TABLE products DROP COLUMN description;*

При добавлении ограничения указывается его имя, через которое потом можно к нему обратиться, а также выполняется автоматическая проверка уже внесенных данных. Примеры редактирования ограничений:

*// установить ограничение на значение имени (не должно совпадать с 'что-то странное')*

*ALTER TABLE products ADD CHECK (name <> 'что-то странное');*

*// установить/удалить ограничение на уникальность имени*

*ALTER TABLE products ADD CONSTRAINT uq\_products\_name UNIQUE (name);*

*ALTER TABLE products DROP CONSTRAINT uq\_products\_name UNIQUE (name);*

*// поле some\_id должно быть внешним ключом к таблице another\_table*

*ALTER TABLE example ADD CONSTRAINT fk\_example\_to\_another\_table FOREIGN KEY (some\_id) REFERENCES another\_table;*

*// установить/удалить ограничение на категорию NOT NULL*

*ALTER TABLE products ALTER COLUMN category SET NOT NULL;*

*ALTER TABLE products ALTER COLUMN category DROP NOT NULL;*

*// установить/удалить столбцу новое значение по умолчанию*

*ALTER TABLE products ALTER COLUMN price SET DEFAULT 100;*

*ALTER TABLE products ALTER COLUMN price DROP DEFAULT;*

*// преобразовать столбец в другой тип данных*

*ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);*

*// переименовать столбец и таблицу*

*ALTER TABLE products RENAME COLUMN name TO title;*

*ALTER TABLE products RENAME TO items;*

**Для добавления новой записи** в таблицу используется оператор ***INSERT***. Далее указываются имя таблицы и столбцов (в любом порядке), а сами значения перечисляются после свойства ***VALUES***. Если список столбцов опущен, то он генерируется автоматически (не рекомендуется).

*INSERT INTO clients (name, phone, email) VALUES ('Кое-кто', '80932334444', 'koe\_kto@mail.ru'),*

*('Еще-кто-то', '80932334445', 'kto@mail.ru');*

Скрыть потенциальную ошибку при добавлении можно с помощью свойства ***ON CONFLICT***. Например:

*INSERT INTO products (id, name, price, category)*

*VALUES (1, 'Шампунь 2', 1200, 'косметика')*

*ON CONFLICT DO NOTHING;*

Вместо ***DO NOTHING*** можно написать ***DO UPDATE*** – она гарантирует добавление или обновление данных. Для обращения к значениям, изначально предлагаемым для добавления, используется специальная таблица ***excluded***.

*INSERT INTO clients (id, name, phone, email)*

*VALUES (9, 'Анна Васильевна Орешкина', '805565423422', 'anna@mail.com')*

*ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name, phone = EXCLUDED.phone, email = EXCLUDED.email;*

Наполнить таблицу выборкой данных на основании другой таблицы можно с помощью ***подзапросов*** внутри *INSERT*. При этом свойство ***VALUES*** не используется. Например:

*INSERT INTO old\_orders (id, client\_id, date, status, address)*

*SELECT id, client\_id, date, status, address*

*FROM orders*

*WHERE date < '2020-01-01';*

**Для обновления значений полей записей** в таблице используется оператор ***UPDATE***. После команды ***SET*** указываются поля с новыми значениями. Также можно использовать фильтры для массовой установки значений. Этот оператор также поддерживает подзапросы.

*UPDATE clients*

*SET phone = 85553332211*

*WHERE name = 'Иванов Иван Иванович';*

С помощью свойства ***FROM***можно объединить несколько таблиц, но целевая строка не должна соединяться с более чем одной строкой из других таблиц.

*UPDATE positions*

*SET amount = amount + 1*

*FROM orders*

*WHERE positions\_.order\_id = orders.id*

*AND orders.client\_id = ( SELECT id*

*FROM clients*

*WHERE name = 'Бук Василий Петрович' );*

**Для удаления записей** из таблицы используется оператор ***DELETE***. Этот оператор также поддерживает подзапросы. Использовать оператор для параллельного удаления из нескольких таблиц запрещено.

*DELETE FROM clients WHERE id = 111;*

Для сброса номера последовательности счетчика используется команда:

*ALTER SEQUENCE customer\_id\_seq RESTART WITH 1;*

**Дамп БД** – это текстовые файлы в формате *SQL* и запросы внутри них. С его помощью можно восстановить БД с нуля.