СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

Работа с СУБД SQL

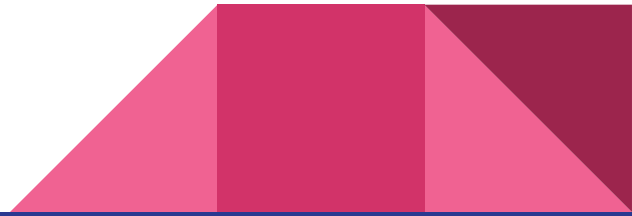
Обзор реляционных СУБД. Основы SQL.

Проверить, идет ли запись

Меня хорошо видно && слышно?

Знакомство

- настройка микрофона и аудио
- проверка работы чата
- напишите, пожалуйста, в чат кратко про свой опыт работы с SQL, Linux (1..10)



Обзор реляционных СУБД

Выбор СУБД

- максимальное число пользователей одновременно обращающихся к базе;
- характеристики клиентского ПО;
- аппаратные компоненты сервера;
- серверную операционную систему;
- уровень квалификации персонала
- масштабируемость
- гибкость модели данных
- и т.д.

Сравнение СУБД

Персональные:

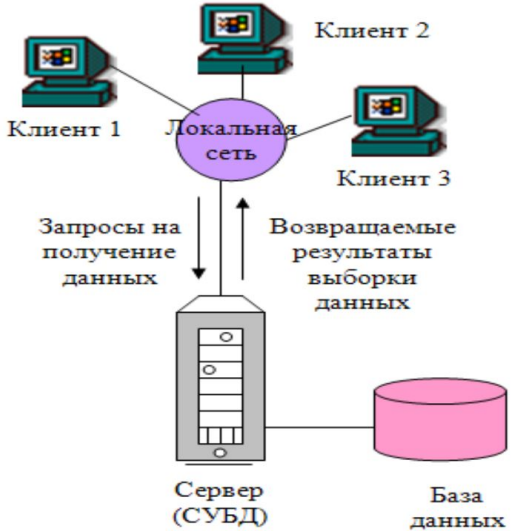
- dBASE,
- FoxPro,
- MS Access

Недостатки:

- большой объем сетевого трафика
- копия СУБД на каждой рабочей станции
- сложность обеспечения параллельной работы и целостности данных

Многопользовательские:

- Oracle,
- IBM DB2,
- Microsoft SQL Server
- **Postgres**

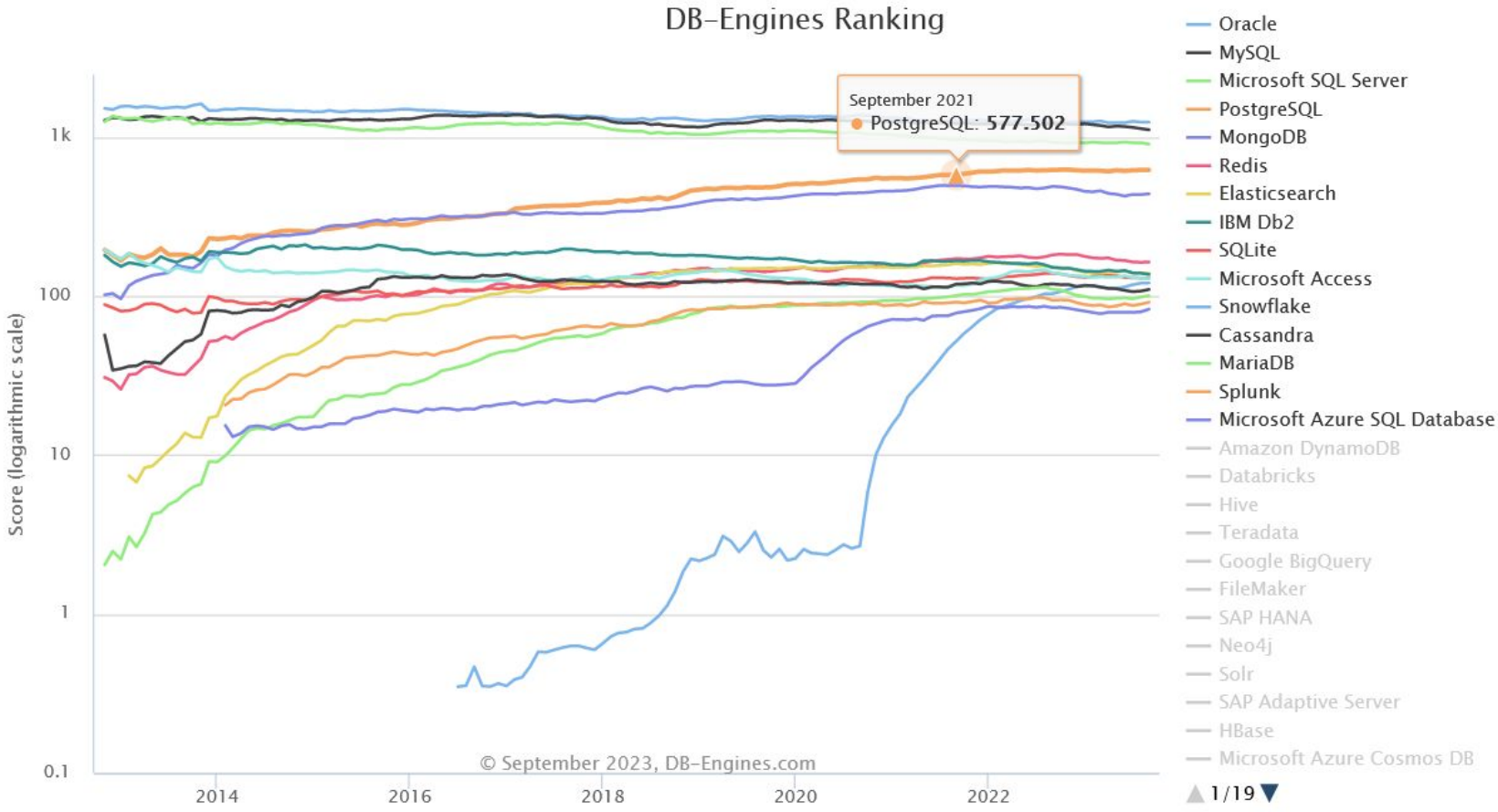


Многопользовательские СУБД

Преимущества:

- Широкий доступ к существующим БД
- Повышение производительности системы
- Снижение стоимости аппаратного обеспечения
- Повышение уровня непротиворечивости данных
- Отказоустойчивость

Рейтинг СУБД (2023)



Oracle

Лидирует в области крупномасштабных информационных систем

- Высочайшая надежность.
- Возможность разбиения крупных баз данных на разделы (large-database partition), что дает возможность эффективно управлять гигантскими терабайтными базами;
- Наличие универсальных средств защиты информации;
- Эффективные методы максимального повышения скорости обработки запросов;
- Индексация по битовому отображению;
- Свободные таблицы (в других СУБД все таблицы заполняются сразу при создании);
- Распараллеливание операций в запросе;

Oracle

Лидирует в области крупномасштабных информационных систем

- Наличие широкого спектра средств разработки, мониторинга и администрирования;
- Поддержка известных платформ: Windows, AIX, Compaq Tru64 UNIX, HP 9000 Series HP-UX, Linux Intel, Sun Solaris
- Ориентация на интернет технологии.
- Поддержка XML в хранимых процедурах, позволяющая разработчикам (традиционных) баз данных непосредственно использовать преимущества языка XML, применяя привычный механизм хранимых процедур
- Доступ по протоколу HTTP, поддерживающий отправку SQL-запросов к БД с применением URL-адресов

MS SQL Server

Идеально подходит для малых и средних организаций

- простота администрирования,
- возможность подключения к Web,
- быстродействие и функциональные возможности механизма сервера СУБД,
- наличие средств удаленного доступа,

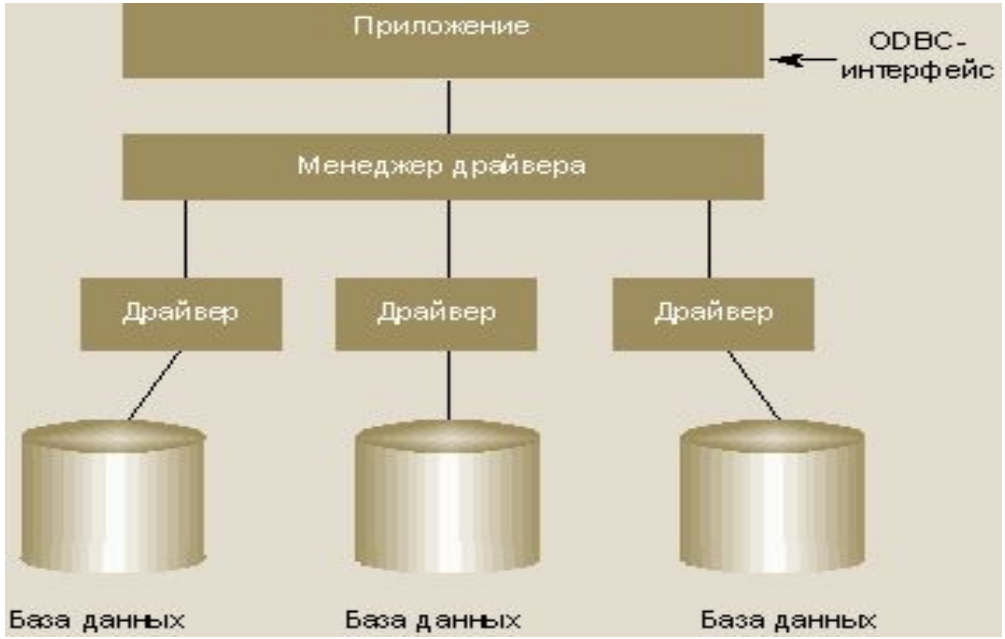
PostgreSQL

Иногда называют бесплатным аналогом Oracle Database

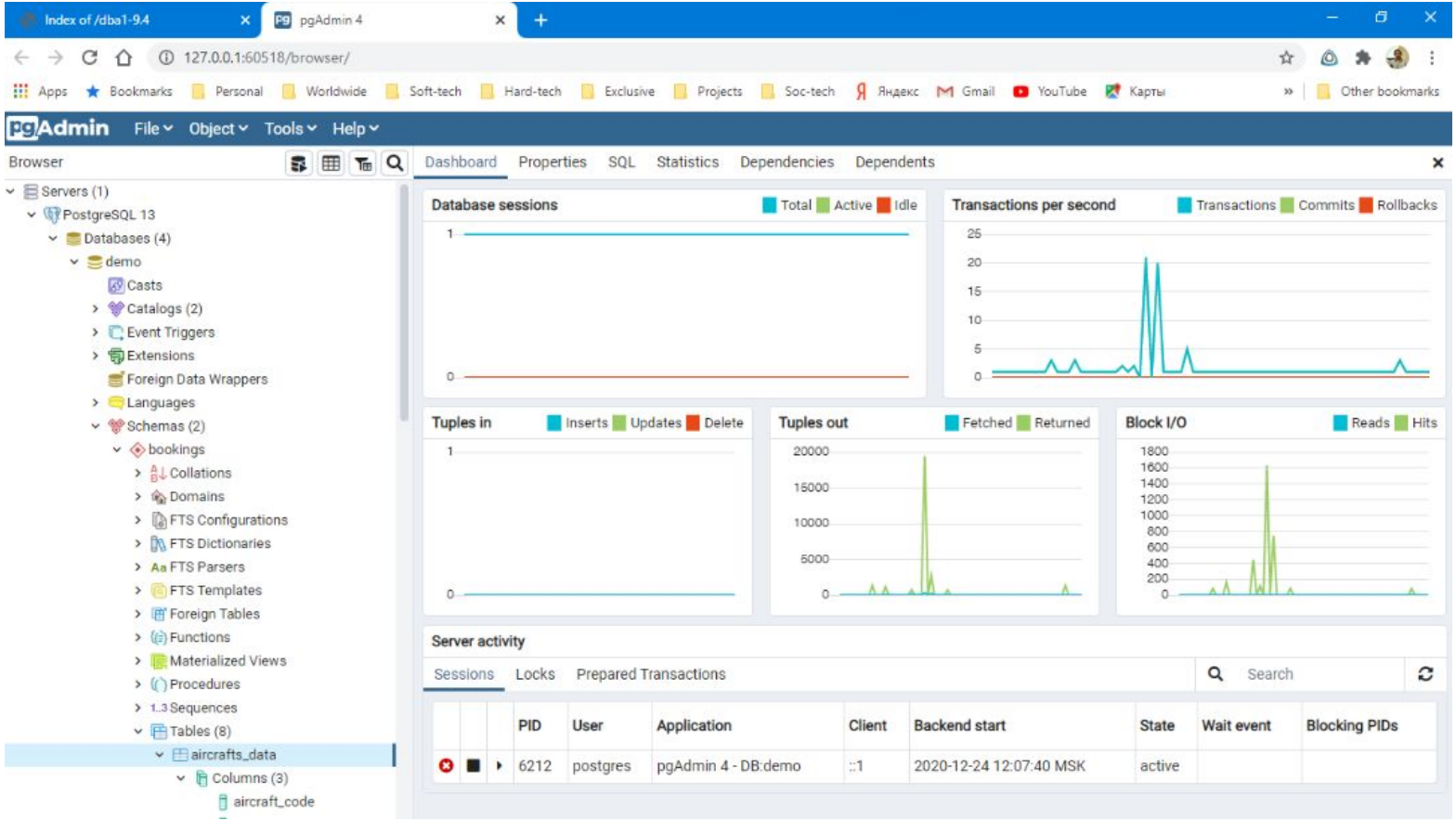
- Вместе со структурированными поддерживает также неструктурированные и перечисляемые типы данных.
- Инициирование нового соединения рассматривается как новый процесс.
- Близка к стандарту ANSI SQL.
- Транзакции (ACID).
- MVCC
- ODBC/JDBC/libpq
- Обобщенные табличные выражения (CTE).
- Полнотекстовый поиск.
- Логическая и 5 режимов синхронной репликации
- Оконные функции.

Технологии доступа к данным

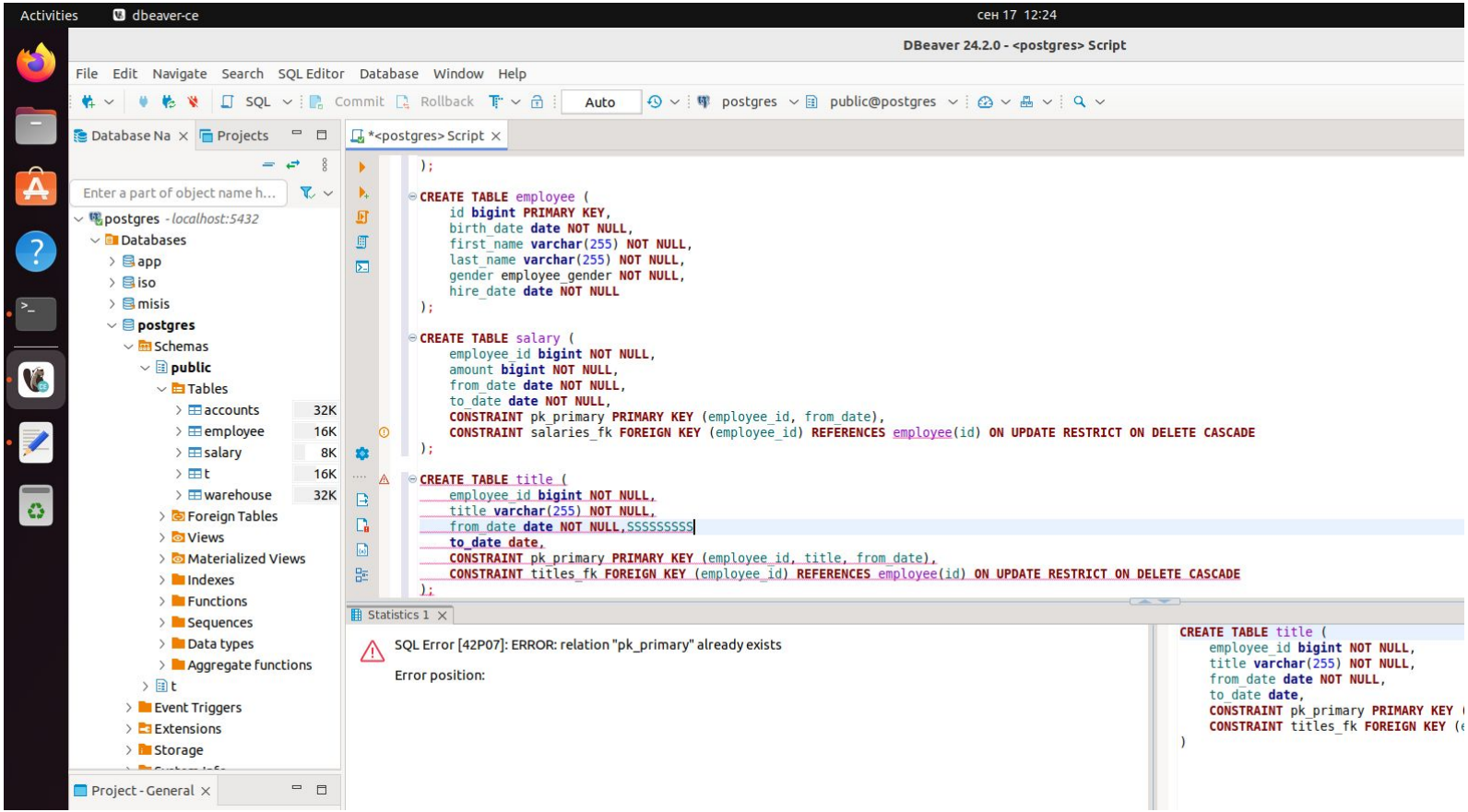
Архитектура **ODBC** (Open DataBase Connectivity) / JDBC



Система управления **pgAdmin**



Система управления DBeaver



psql

psql – терминальный клиент, который используется администраторами и разработчиками для интерактивной работы с PostgreSQL.

Запросы (входной поток) могут быть получены из файла или интерактивно из командной строки.

Строка подключения:

```
psql -d database -h host -p port -U username
```

Для подключения к базе данных нужно знать имя базы данных, имя сервера, номер порта сервера и имя пользователя, под которым следует подключиться.

```
⇒ \?
⇒ \h
⇒ \l
⇒ \c demo
⇒ \d
⇒ \q
```

- список управляющих команд psql
- список команд языка sql
- вывести список баз данных
- подключиться к базе данных с указанным именем
- вывести список таблиц в базе данных
- выйти из программы psql

psql

\dt – список всех таблиц в БД
\dn – список всех схем в БД
\df – список всех функций в БД
\du – список ролей БД

```
testdb=> CREATE TABLE my_table (
testdb(> first integer not null default 0,
testdb(> second text)
testdb-> ;
CREATE TABLE

testdb=> \d my_table
Таблица "my_table"
Атрибут | Тип | Модификаторы
---------+---------+--------------------
first | integer | NOT NULL DEFAULT 0
second | text |

testdb=> SELECT * FROM my_table;
```

Основы SQL. DDL.

Стандарт SQL

SQL (Structured Query Language, структурированный язык запросов) – стандарт языка для работы с данными в реляционных базах данных.

История:

- Прототип языка – сначала **QBE**, затем **SEQUEL** (Structured English Query Language) – был разработан в начале 70-х годов в IBM Research и реализован в СУБД System R.
- 1989 – первый ANSI/ISO стандарт языка SQL (вторая редакция, первая была в 1987 г.). Однако развитие технологий БД потребовали его доработки и расширения.
- 1992 – стандарт **SQL-92** или SQL 2. Практически все современные реляционные (и постреляционные) СУБД поддерживают этот стандарт полностью.
- 1999 – стандарт SQL 3. В стандарт введены структурированные типы данных и другие особенности, позволяющие сочетать реляционную и объектную модель данных.
- **SQL-2016** (введён JSON).
- **SQL-2023** (ISO/IEC 9075) - https://www.iso.org/standard/76584.html

Задача

Есть данные застройщика: квартиры, ЖК, сделки.
Нужно отобразить динамику прибыли за q1 2024-го года по ЖК по сравнению с 2023.

- Как это реализовать на Python/Java/C++ (без использования библиотек)?
- Как реализовать с помощью SQL?

SQL

SQL не является традиционным языком программирования он не содержит операторы, позволяющие осуществлять пошаговые действия, а ориентирован на работу со множествами
Другими словами, на SQL пишется, **ЧТО** должно получиться
в результате выполнения запроса, но не пишется, **КАК** это будет реализовано

Data Definition Language, DDL

- CREATE создаёт какую либо структуру базы данных либо саму базу
- ALTER изменяет свойства этой структуры
- DROP удаляет структуру

Data Manipulation Language, DML

- SELECT осуществляет выборку данных (с 2016 года **DQL - data query**)
- INSERT добавляет новые данные
- UPDATE изменяет существующие данные
- DELETE удаляет данные

Создание таблицы

Таблица БД может быть создана выполнением оператора CREATE TABLE.

Неполный синтаксис оператора:

```
CREATE TABLE [IF NOT EXISTS] table_name (
   column1 datatype(length) column_constraint,
   column2 datatype(length) column_constraint,
   ...
   table_constraints
);
```

Например:

```
CREATE TABLE test (a varchar(20));
```

Типы данных

Назначение	Тип	Размер	NOT NULL
Число	**int**(integer)/bigint	4/8	0 (-1)
Текст	**text**/varchar(256)/*char(5)*	4 байта длина+1-2 байта на символ	'' (прямые кавычки)
Деньги/вес/нецелое число	**numeric**/decimal	4 байта	0 (-1)
Логический	**boolean**	1 байт	false
Дата [`with time zone`]	**date**	4 байта	2099.12.31
Дата + время	**timestamp**	8 байт	2099.12.31 23:59:59

Удаление таблицы

Таблица может быть удалена оператором DROP TABLE

Например:

```
DROP TABLE test;
```

Значение по умолчанию. Ограничения.

Можно разрешить или запретить полю принимать неопределенные значения (null). По умолчанию – nullable.

Существует возможность указать значение, которое должно иметь поле, если ему не присвоено никакое значение. Такое значение называется **значением по умолчанию**.

Например:

```
CREATE TABLE NewTable (
     a varchar(10) default 'Привет',
     b varchar(10) not null default ‘test’ primary key
)
```

Ограничение Primary Key

- Первичный ключ может состоять из одного или более полей таблицы.
- Значение первичного ключа уникально в таблице.
- Попытка поместить в таблицу запись с дубликатным первичным ключом будет отвергнута.

Вычисляемые поля

Поле может быть вычисляемым.

- Вычисляемые поля не хранятся в таблице.
- Каждый раз, когда требуется его значение, происходит обращение к формуле.
- С помощью ключевого слова **STORED** задается физическое хранение данных в вычисляемом столбце. При наличии этого ключевого слова вычисление выполняется при добавлении или изменении строки, а результат физически хранится в таблице.
- Синтаксис определения вычисляемого поля:
  <имя поля> GENERATED AS <выражение> [STORED]

Пример:

```
CREATE TABLE employees (
  name TEXT,
  birthdate DATE,
  age_years INT GENERATED ALWAYS AS (EXTRACT(YEAR FROM (birthdate))) STORED
);
```

https://www.db-fiddle.com/f/g1wcJdaeKZ5Zu7aSnm3Q2y/0

Ограничения CHECK

На поле может быть наложено ограничение в виде логического выражения.

- Если логическое значение принимает значение false, то БД отвергнет такую запись с выдачей соответствующего сообщения.
- Ограничение CHECK имеет имя. Имя даёт либо сервер, либо разработчик.
- Ограничение может касаться более чем одного столбца таблицы, тогда оно объявляется не на уровне поля, а на уровне таблицы

Пример:

```
CREATE TABLE employees (
  id SERIAL PRIMARY KEY,
  first_name VARCHAR (50) NOT NULL,
  last_name VARCHAR (50) NOT NULL,
  birth_date DATE NOT NULL,
  joined_date DATE NOT NULL,
  salary numeric CHECK(salary > 0)
);
```

https://www.db-fiddle.com/f/4F1sQCxNuSPX3Ed8B2LpC6/1

Ограничения UNIQUE

На поле может быть наложено ограничение UNIQUE.

- Это означает запрет на появления одного и того же значения поля более чем в одной записи.
- Попытка нарушить запрет приводит к тому, что запись будет отвергнута БД с выдачей сообщения.
- Обязательно NOT NULL !!! так как NULL != NULL

Пример:

```
CREATE TABLE  vv (
  a numeric,
  b numeric unique not null
);
```

https://www.db-fiddle.com/f/aR6194VH13WbDxx4XBW9Lh/2

Ограничения PRIMARY KEY и UNIQUE

Первичный ключ и уникальный ключ могут состоять из нескольких полей. В этом случае primary key или unique является ограничением на уровне таблицы.

Синтаксис:

< ограничение primary key или unique > ::=
[CONSTRAINT имя ограничения] { PRIMARY KEY | UNIQUE } { (column [,...n]) }

Оператор **ALTER TABLE**

Свойства уже существующей таблицы могут быть изменены оператором **ALTER TABLE**.

- добавить поле
  ALTER TABLE <имя таблицы> ADD {<имя поля> <свойства поля>}[,… n]
- удалить поле
  ALTER TABLE DROP COLUMN {<имя поля>[,… n]}
- изменить свойства поля
  ALTER TABLE ALTER COLUMN {<имя поля><новые свойства поля>
- добавить ограничение
  ALTER TABLE ADD CONSTRAINT {<определение ограничения>[,… n]}
- удалить ограничение
  ALTER TABLE DROP CONSTRAINT {<имя ограничения>[… n]}

В оператор ALTER TABLE может быть включено произвольное число модификаций.

Ограничения таблицы

[CONSTRAINT имя_ограничения
{ CHECK (выражение) [NO INHERIT] |
UNIQUE (имя_столбца [, ...]) параметры_индекса
PRIMARY KEY (имя_столбца [, ...]) параметры_индекса
EXCLUDE [USING индексный_метод] (элемент_исключения WITH оператор [, ...]) параметры_индекса [WHERE (предикат)] |
FOREIGN KEY (имя_столбца [, ...]) REFERENCES целевая_таблица [(целевой_столбец [, ...])

Ограничения столбца

```
[ CONSTRAINT имя_ограничения
{ NOT NULL | NULL |
        CHECK (выражение ) [ NO INHERIT ] |
        DEFAULT выражение_по_умолчанию
        UNIQUE параметры_индекса
        PRIMARY KEY параметры_индекса
        REFERENCES целевая_таблица [ (целевой_столбец ) ]
```

Основы SQL. DML.

Data Manipulation Language (DML)

- выборка данных (SELECT)
- вставка (INSERT)
- обновление (UPDATE)
- удаление (DELETE)

Оператор SELECT

Синтаксис:
1 SELECT [ALL | DISTINCT] <список вывода>
2 [INTO <имя новой таблицы>]
3 [FROM <список таблиц и условий соединения>]
4 [WHERE <условие отбора или соединения>]
5 [GROUP BY <список полей группировки>]
6 [HAVING <условия, накладываемые на группу>]
7 [ORDER BY <список полей для сортировки вывода>]
8 [OFFSET <смещение от начала>]
9 [LIMIT <ограничение на вывод>]
10 [UNION <запрос на выборку для объединения>]
…

порядок следования разделов нарушать нельзя!

```
SELECT ALL * FROM publishers;

SELECT p.country
FROM pubs.public.publishers p
LIMIT 5;

SELECT DISTINCT state
FROM authors;

SELECT 'Название книги: ',
title, pubdate
FROM titles

select 2+2 as "итого";

SELECT * INTO NewAuthors
FROM authors;
```

Раздел WHERE

- Раздел WHERE предназначен для наложения горизонтальных фильтров на данные, обрабатываемые запросом.
- Для этого указывается логическое условие, от результата вычисления которого зависит, будет ли строка включена в результат выборки или нет.

```
SELECT * FROM titles WHERE ytd_sales > 5000;

SELECT * FROM authors WHERE 1 = 1;

SELECT title FROM titles
WHERE extract('year' FROM pubdate) > 1991 AND extract('year' FROM pubdate) <= 1995;

SELECT * FROM titles
WHERE title LIKE '%Database%' OR "type" = 'popular_comp';

SELECT pub_name, 'не Америка!' AS "Особенность" FROM publishers WHERE state IS NULL;
```

Раздел WHERE

- Раздел WHERE не предназначен для создания связей между таблицами (по внешним ключам), хотя такая возможность есть.
- Данный способ является устаревшим и его категорически не рекомендуется использовать при профессиональной работе с СУБД.

```
SELECT title, pub_name FROM titles, publishers
WHERE titles.pub_id = publishers.pub_id AND country = 'USA';

SELECT * FROM authors
WHERE au_fname IN ('Sylvia', 'Anne', 'Livia');

SELECT * FROM titles
WHERE price NOT BETWEEN 10::money AND 20::money;

SELECT Count(*) FROM authors WHERE contract = FALSE;
```

Раздел FROM

- С помощью раздела FROM определяются источники данных, с которыми будет работать запрос.
- Можно создать одну или несколько связей между отношениями, явно указывая те поля, которые должны играть роль внешних ключей

```
SELECT title, pub_name FROM titles, publishers
WHERE titles.pub_id = publishers.pub_id AND country = 'USA';

SELECT * FROM authors
WHERE au_fname IN ('Sylvia', 'Anne', 'Livia');

SELECT * FROM titles
WHERE price NOT BETWEEN 10::money AND 20::money;

SELECT Count(*) FROM authors WHERE contract = FALSE;
```

Раздел **ORDER BY**

Раздел **ORDER BY** предназначен для упорядочения набора данных, возвращаемых после выполнения запроса.

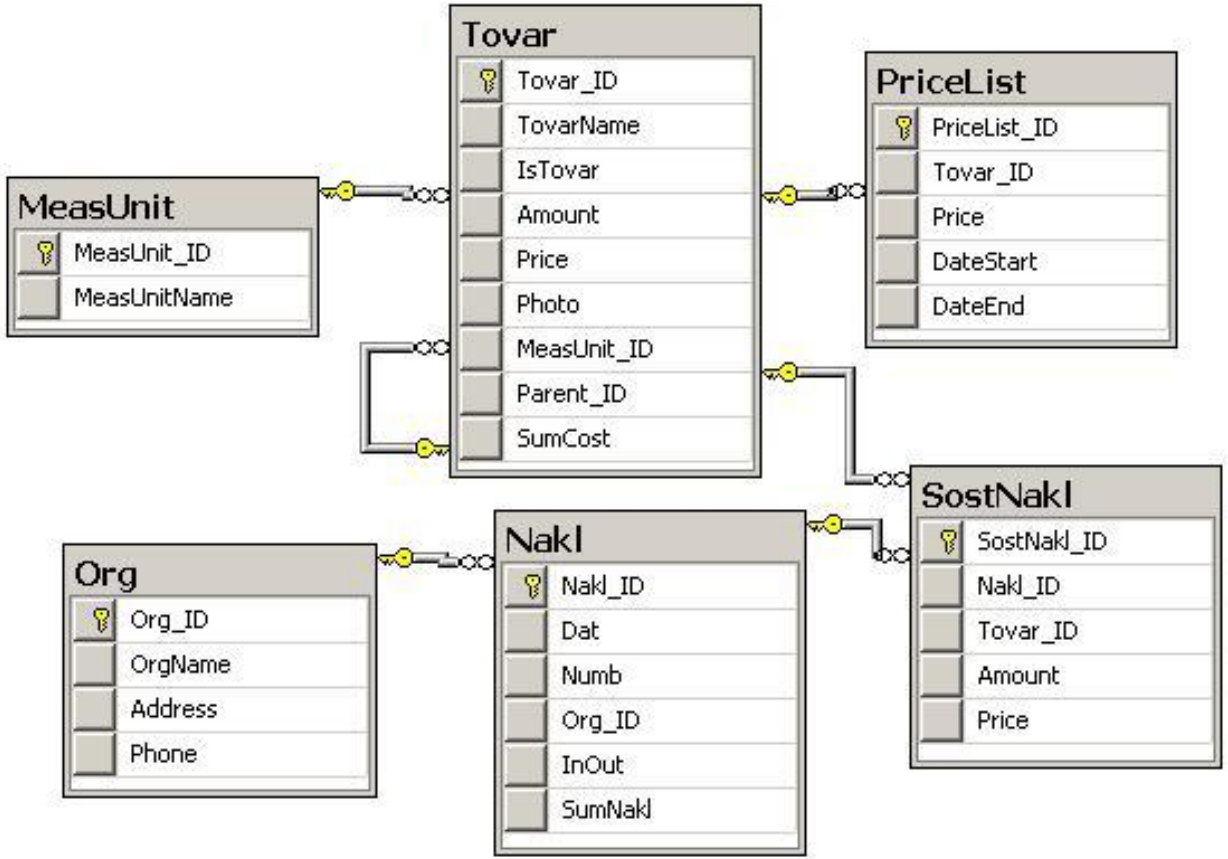
Синтаксис:

```
[ ORDER BY { выражение [ ASC | DESC ] } [ ,...n] ]
```

Например:
…ORDER BY A.t, B.x DESC
что означает, что выборка упорядочивается по A.t в порядке возрастания (ASC), а в пределах подмножеств записей с одинаковыми значениями A.t – по B.t в порядке убывания (DESC).

В список выражений упорядочения могут входить псевдонимы полей

Пример


Tovar
Tovar_ID
TovarName
IsTovar
Amount
Price
Photo
MeasUnit_ID
Parent_ID
SumCost
PriceList
PriceList_ID
Tovar_ID
Price
DateStart
DateEnd
MeasUnit
MeasUnit_ID
MeasUnitName
SostNakl
SostNakl_ID
Nakl_ID
Tovar_ID
Amount
Price
Nakl
Nakl_ID
Dat
Numb
Org_ID
InOut
SumNakl
Org
Org_ID
OrgName
Address
Phone

Предикат LIKE

<выражение> [NOT] LIKE <шаблон> [ESCAPE <символ>]

Возвращает истину, если выражение удовлетворяет шаблону. В шаблон могут входить обычные символы, которые обозначают сами себя, а также символы – заменители:

- % - любая строка, например, x LIKE '%eee%' означает поиск буквосочетания 'eee' в переменной x.

```
select * from Student where StudName like 'e%e%'
```

Предикат BETWEEN

x between 1 and 12

Результат будет иметь значение *true*, если x>=1 и x<=12

Пример 2

```
create table test(i int);
insert into test values (1),(13);
select i
from test
where i between 1 and 12;
```

https://www.db-fiddle.com/f/nAjPsVbP44iBwCqfkJ7H9a/1

Предикат IN

принадлежность множеству значений. Имеет 2 формы. Во первых, может быть указано фиксированное множество значений, например:

…WHERE x in(2,8,11)

Во втором случае множество задается оператором *SELECT*:

…WHERE x in (SELECT rrr FROM TT)

Предикат ALL

<скалярное выражение><операция сравнения> ALL(<подзапрос>)

пример:

x <= ALL(select y from MyTable)

Результат – истина, если *x* больше или равно всех значений *y*, возвращаемых оператором *SELECT*.

Предикат ANY (SOME)

Синтаксис:

<скалярное выражение><операция сравнения> {ANY|SOME}(<подзапрос>)

пример:

x <= ANY(select y from MyTable)

Результат – истина, если *x* меньше или равно чем хотя бы одного значения *y*, возвращаемого оператором *SELECT*.

Квантор существования (EXISTS)

Синтаксис:

[NOT] EXISTS(<подзапрос>)

пример:

… WHERE EXISTS (SELECT * FROM TT)

Результат – истина, если оператор *SELECT* возвращает непустое множество записей.

Квантор общности – не существует такого объекта, для которого не выполнено условие.

SELECT INTO

Оператор *SELECT* может помещать результат выборки в новую таблицу.

Например:

```
SELECT T1.x,T2.y into NewTable
FROM T1,T2
WHERE T1.z=T2.z
```

Если таблица NewTable уже существует, то операция будет отвергнута.

Пример:

```
select g.GruppName,s.StudName into ttt
from student s, Gruppa g
where g.Grupp_ID=s.grupp_ID
order by GruppName,StudName
```

Ограничение объема выборки

В операторе SELECT может присутствовать фраза, ограничивающая объем выборки:

SELECT [ALL | DISTINCT] …

Умолчанием является *ALL*; в этом случае оператор *SELECT* возвращает все записи, удовлетворяющие условию во фразе *WHERE*.

DISTINCT означает, что все возвращаемые записи должны быть различны. Дубликаты будут исключены из результирующего множества.

Примеры операторов SELECT

1) Товары и история цен до сегодняшнего дня.

```
SELECT Tovar.Tovar_ID, Tovar.TovarName,
  DateStart, PriceList.Price
FROM Tovar, PriceList
where Tovar.Tovar_ID*=PriceList.Tovar_ID
  and IsTovar=1
  and DateStart<=getdate()
  and (DateEnd is null or DateEnd>=getdate())
ORDER BY TovarName
```

Примечание: используется левое внешнее соединение таблиц *Tovar* и *PriceList*, с тем, чтобы в результат вошли все товары, в том числе не имеющие цены.

Примеры операторов SELECT

2) Список организаций, закупавших «Апельсин».

```
SELECT distinct Org.*
FROM Org, Nakl, SostNakl, Tovar
WHERE Nakl.Org_ID=Org.Org_ID
  and Nakl.Nakl_ID=SostNakl.Nakl_ID
  and SostNakl.Tovar_ID=Tovar.Tovar_ID
  and Nakl.InOut='-'
  and Tovar.TovarName='Апельсин'
ORDER BY OrgName
```

Примеры операторов SELECT

2) Другое решение с квантором существования:

```
SELECT Org.*
FROM Org
WHERE exists
  (SELECT SostNakl.*
   FROM Nakl, SostNakl, Tovar
   WHERE Nakl.Org_ID=Org.Org_ID
     and Nakl.Nakl_ID=SostNakl.Nakl_ID
     and SostNakl.Tovar_ID=Tovar.Tovar_ID
     and Tovar.TovarName='Апельсин'
     and Nakl.InOut='-'
  )
ORDER BY OrgName
```

СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

Вставка, обновление данных

Команда INSERT

Команда INSERT используется для ввода данных: добавляет одну или несколько строк в одну таблицу.

Синтаксис (упрощённый):

```
INSERT INTO <имя_таблицы> [
(<имя_столбца>,...) ] {
[VALUES (<значение>,..) ]
|[ <SELECT-запрос> ]
|[ DEFAULT VALUES ]
}
```

Все столбцы, не представленные в явном или неявном списке столбцов, получат значения по умолчанию, если для них заданы эти значения, либо NULL в противном случае.

Команда INSERT

```
INSERT INTO exams VALUES
(301667, 1, 2, 2, '20.06.2020'::date);

INSERT INTO subjects(subj_id, subj_name, description)
VALUES
(1, 'Базы данных', 'основы работы с PostgreSQL'),
(2, 'Физика','Часть 3. Оптика'),
(3, 'Английский язык','');

INSERT INTO authors DEFAULT VALUES;

INSERT INTO films
SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';
```

Команда UPDATE

Команда UPDATE используется для изменения данных: модифицирует одну или несколько строк в таблице.

Синтаксис (упрощённый):

```
UPDATE <имя_таблицы>
SET <имя_столбца> = <значение>, ...
[FROM {<имя таблицы>
|<SELECT-запрос>},…]
[WHERE <условие>]
```

- Если в UPDATE будет пропущен раздел WHERE, то заданные в разделе SET изменения будут сделаны в каждой строке таблицы.
- В PostgreSQL надёжнее ссылаться на другие таблицы в подзапросах, хотя такие запросы часто работают медленнее, чем FROM-соединение.

Команда UPDATE

```
UPDATE students SET fio = 'Нетсуорти Эстер'
WHERE fio = 'Шоу Эстер';

UPDATE subjects SET
     subj_name = 'Иностранный язык',
     description = 'группы английского и французского языков'
WHERE subj_id = 3;

UPDATE exams e SET grade = 5
FROM students s, subjects j
WHERE e.stud_id = s.stud_id AND e.subj_id = j.subj_id
     AND s.fio = 'Стравинский Фёдор Михайлович'
     AND j.subj_name = 'Базы данных'
     AND e.exam_date = '20.06.2020'::date;
```

Команда DELETE

Команда DELETE используется для удаления данных: удаляет одну или несколько строк из таблицы.

Синтаксис (упрощённый):

```
DELETE FROM <имя_таблицы>
[ WHERE <условие> ]}
```

- Без WHERE будут удалены все строки таблицы. В разделе WHERE также можно использовать вложенные подзапросы (т.е. используя данные других таблиц).
- Быстрый вариант удаления всех данных в одной или нескольких таблицах:
  TRUNCATE table1
  Оператор TRUNCATE TABLE (**DDL**) работает много быстрее, чем DELETE FROM TabName, особенно при больших размерах таблицы.

Команда DELETE

```sql
DELETE FROM titleauthor;

DELETE FROM publishers p
WHERE p.pub_name = 'Microsoft Press';

DELETE FROM students s
WHERE s.stud_id IN (
     SELECT e.stud_id FROM exams e
     WHERE e.grade = 2
     GROUP BY e.stud_id
     HAVING count(*) > 3
);

DELETE FROM tasks
WHERE status = 'DONE' RETURNING *;
```

Команда INSERT

Пример: поместить в таблицу *Tovar1* список товаров, ранее покупавшихся организацией с идентификатором Org_ID=14. Предполагается, что таблица *Tovar1* уже существует.

```
INSERT INTO Tovar1(TovarName, IsTovar, Parent_ID)
SELECT TovarName, IsTovar, Parent_ID FROM Tovar
WHERE Tovar_ID in
   (SELECT Tovar_ID
    FROM SostNakl, Nakl
    WHERE SostNakl_ID=Nakl.Nakl_ID
     and Nakl.Org_ID=14)
```

Команда UPDATE

Пример 1:

```
UPDATE Nakl SET
  Dat='20121231',
  Numb=1234
WHERE Nakl_ID=12
```

Пример 2.

Поступила накладная на приход товара на склад. Значение Nakl_ID=234. Необходимо изменить значение количества товара на складе в связи с этим новым поступлением.

```
UPDATE Tovar
SET Amount=Tovar.Amount+SostNakl.Amount
FROM Tovar, SostNakl
WHERE Tovar.Tovar_ID=SostNakl.Tovar_ID
  AND Nakl_ID=234
```

СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

АИС

VIEW

Представления

- **Представление** для пользователей базы данных выглядит как таблица, однако на самом деле его содержимое формируется запросом.
- Физически данные, виртуально принадлежащие представлению, находятся в таблицах, к которым обращается этот запрос.
- Имя представления должно отличаться от имён других представлений, таблиц, последовательностей, индексов или сторонних таблиц в той схеме данных, где оно создаётся.
- При анализе запроса нет абсолютно никакой разницы между таблицами и представлениями.
- Столбец представления будет изменяемым, если это простая ссылка на изменяемый столбец нижележащего базового отношения; в противном случае этот столбец будет доступен только для чтения.
- Представления, выбирающие данные не из одной таблицы, недоступны для добавления или изменения данных

Представления

```
DROP VIEW IF EXISTS publications;

CREATE VIEW publications AS
    SELECT
        a.au_lname AS "Фамилия",
        a.au_fname AS "Имя",
        t.title AS "Название книги"
    FROM authors a
    JOIN titleauthor ta ON a.au_id = ta.au_id
    JOIN titles t ON ta.title_id = t.title_id;

SELECT * FROM publications ORDER BY Фамилия, Имя;

ALTER VIEW publications RENAME TO summary1;
```

Пример оператора CREATE VIEW

Пример 1. В результате выполнения следующего оператора будет добавлена одна запись в таблицу Tovar.

```
INSERT INTO FirstView(TovarName,IsTovar,Amount)
VALUES('Яблоки',1,32)
```

Пример 2. К названию товара для уровней классификации добавить символ '?'.

```
UPDATE FirstView SET TovarName=TovarName+'?' WHERE IsTovar=0
```

СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

Задание

Задания

Создать базу данных школы (id, name, date_entered) + 3 записи

Создать базу данных предприятия (product_name,company,price) + 3 записи

Задание со *:

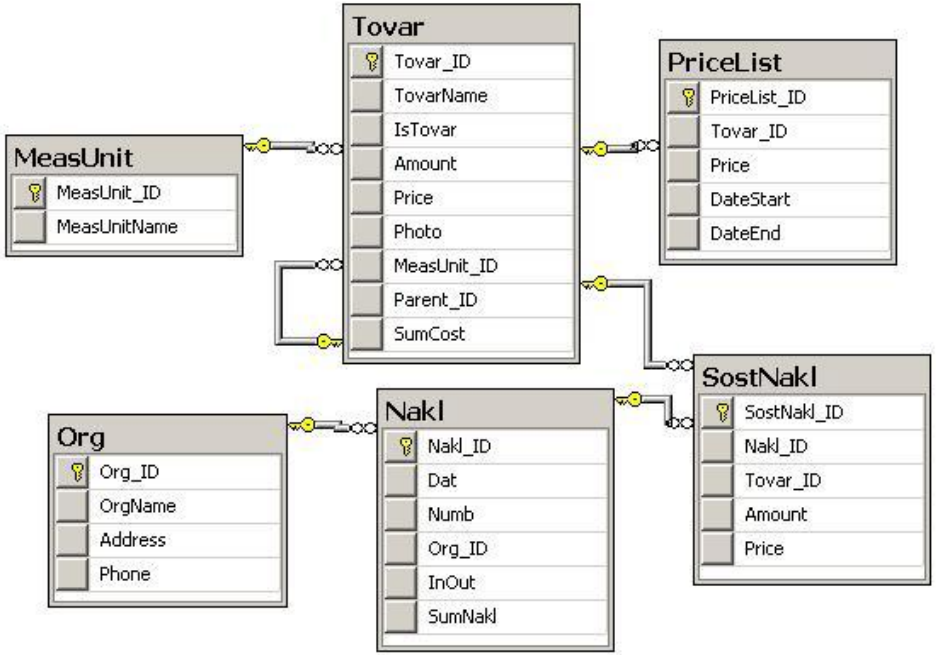
создать 2 таблицы склад и товары с первичными ключами, заполнить их и связать по внешнему ключу

https://www.postgresql.org/docs/current/ddl-constraints.html

Задание с **:
реализуем схему с 41 слайда

Задания

Теперь вместе реализуем схему с 41 слайда



СПАСИБО ЗА ВНИМАНИЕ!

#аис
#учисьваис