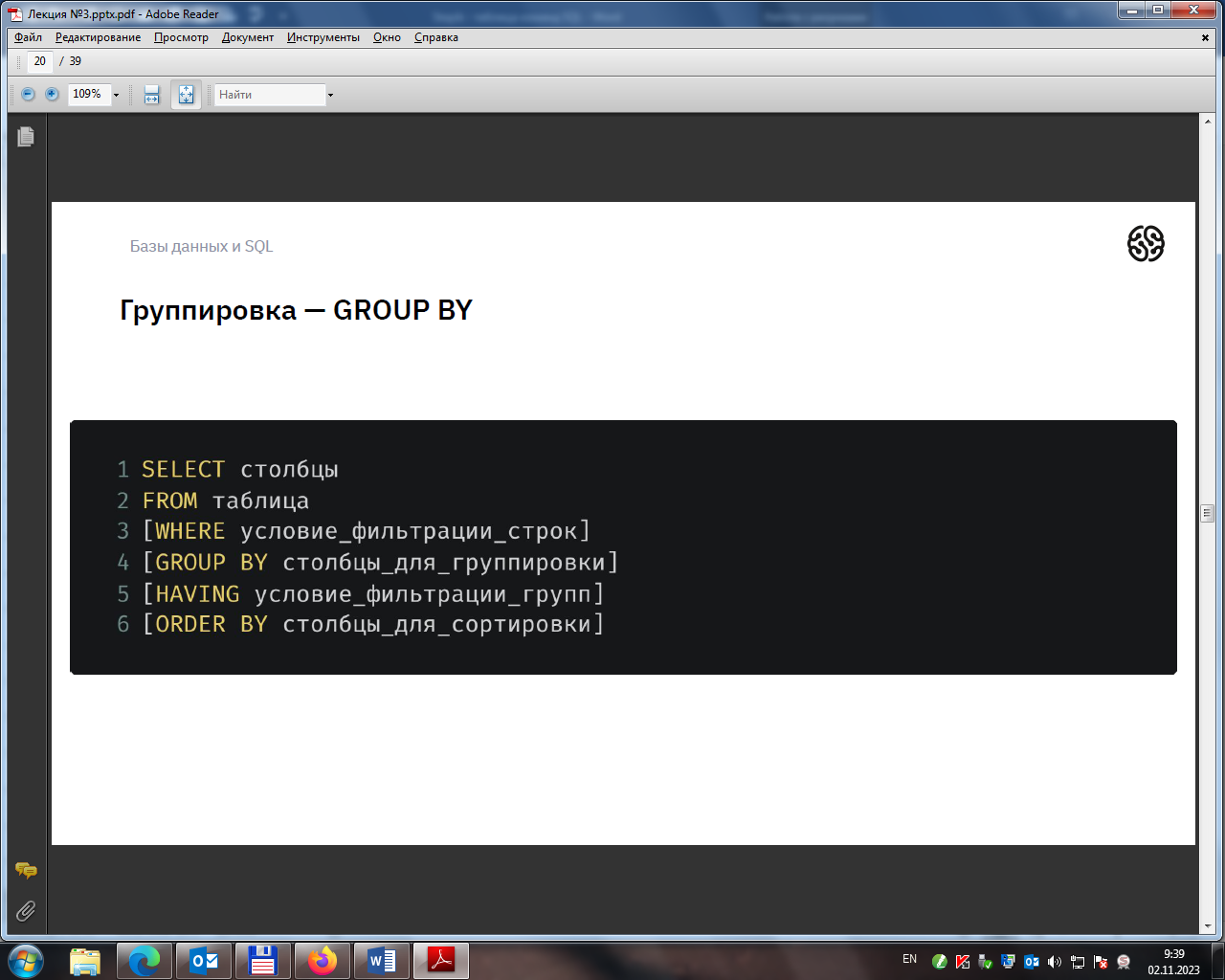
 **Порядок набора команд:**

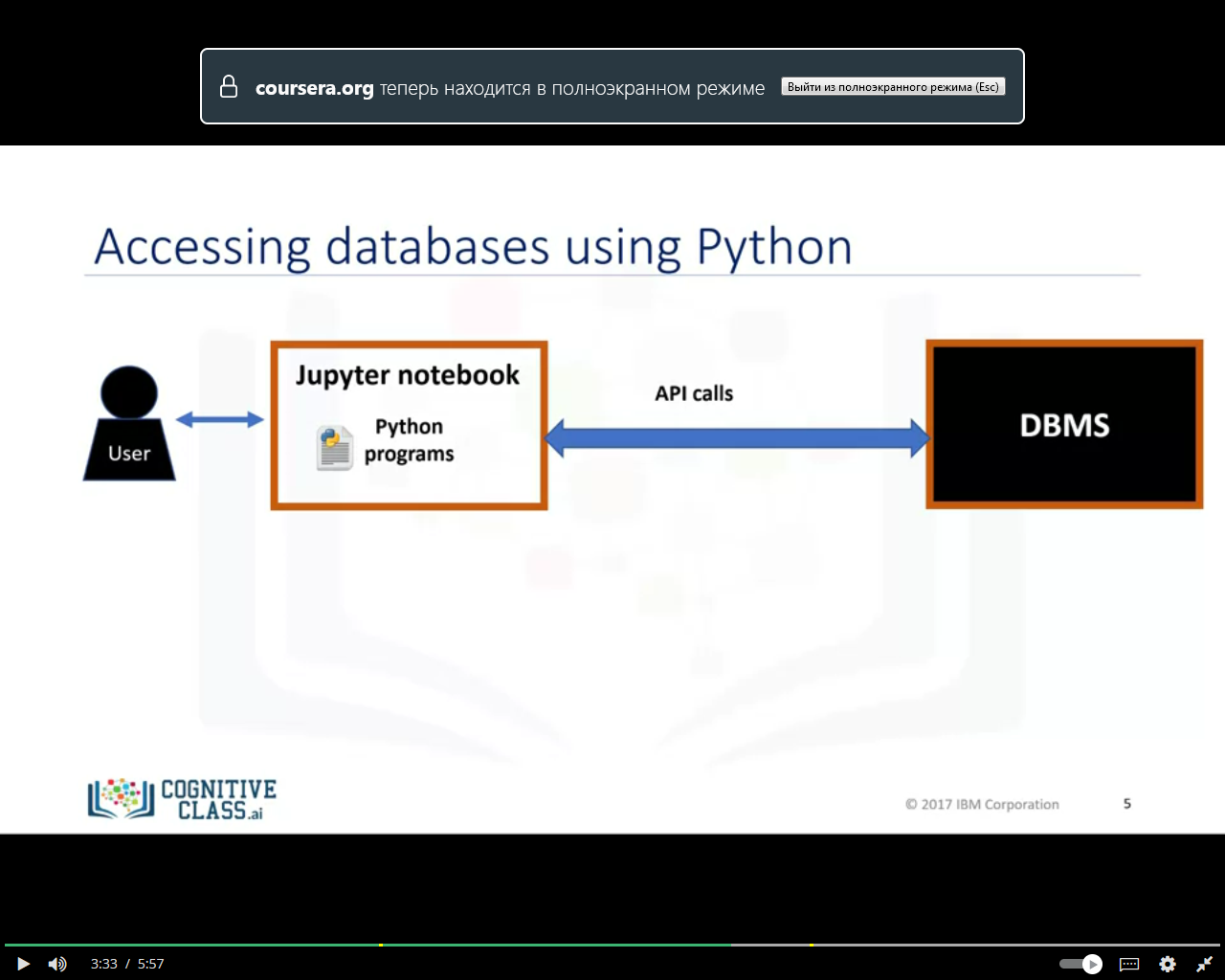
**Порядок выполнения  SQL запроса на выборку на СЕРВЕРЕ**:

1. FROM, включая JOINs
2. WHERE
3. GROUP BY
4. HAVING
5. Функции WINDOW
6. SELECT
7. DISTINCT
8. UNION
9. ORDER BY
10. LIMIT и OFFSET

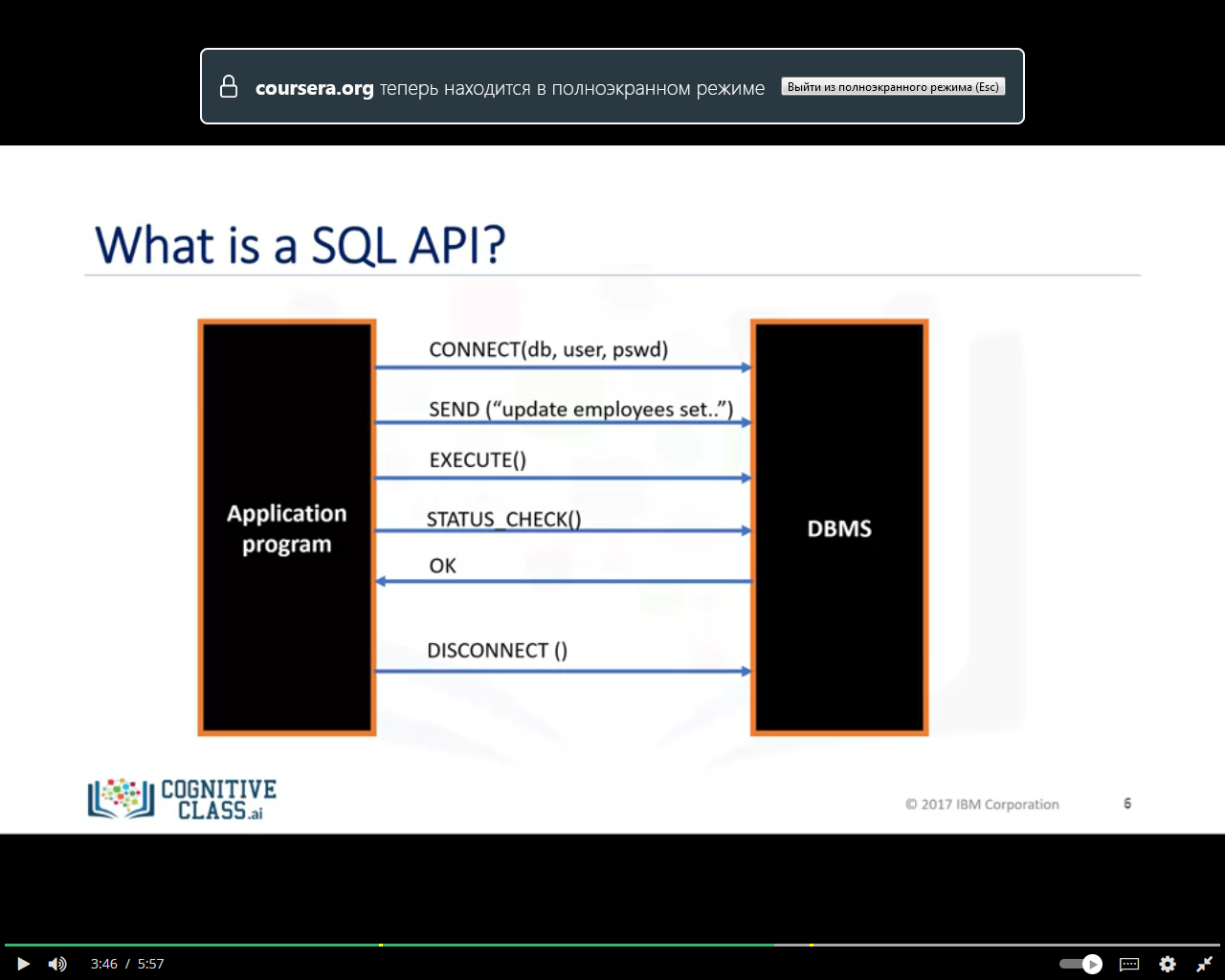
**Таблица команд по SQL**

|  |  |  |
| --- | --- | --- |
| Описание команды | Команда | Пример |
| **Тип данных** | INT - Целое число  DECIMAL - Вещественное число  DATE - Дата в формате ГГГГ-ММ-ДД  TIME — только время — часы, минуты, секунды — «hh:mm:ss»  DATETIME — соединяет дату и время YYYY-MM-DD HH:MI:SS  TIMESTAMP — хранит дату и время начиная с 1970 года  YEAR – год YYYY or YY  VARCHAR(N) - Строка не более 255 символов (N)  BOOL — 0 или 1 | -587  12.56  2022-10-14  Пример |
| Коллекционные типы | При объявлении списка допустимых значений ENUM и SET задаются списком строк. В случае ENUM поле может принимать лишь одно значение из списка. В случае SET — комбинацию заданных значений. | E:\Downloads\Untitled.png |
| **Работа с базой данных** |  |  |
| **Создание базы данных** | CREATE DATABASE имя базы данных; | CREATE DATABASE MySampleDB; |
| **Создание схемы** | CREATE SCHEMA имя схемы; | CREATE SCHEMA student; |
| **Просмотр созданных баз данных** | show databases; | show databases; |
| **Подключиться к базе** | **use имя\_БД;** | use MySampleDB; |
| **Для избежания ошибок (задвоения) БД, таблиц перед их созданием, удалением** | **DROP** объект **IF EXISTS** имя\_объекта; | **DROP DATABASE IF EXISTS vk;**  CREATE DATABASE vk;  USE vk;  DROP **TABLE** IF EXISTS users;  CREATE TABLE users(  ); |
| **Просмотр таблиц в БД** | SHOW TABLES; | SHOW TABLES; |
| **Просмотр столбцов таблицы (**Field,Type,Null,Key,Default, Extra**)** | DESCRIBE имя\_таблицы; | DESCRIBE pet; |
| **Создание и корректировка таблицы** |  |  |
| Создание таблицы  **CREATE** **TABLE** | CREATE TABLE имя таблицы(  название поля и его описание,  название поля и его описание,  …  ); | CREATE TABLE genre(  genre\_id INT PRIMARY KEY AUTO\_INCREMENT,  name\_genre VARCHAR(30),  price DECIMAL(8,2),  amount INT  ); |
| Создание таблицы с внешними ключами | FOREIGN KEY (связанное\_поле\_зависимой\_таблицы)  REFERENCES главная\_таблица (связанное\_поле\_главной\_таблицы) | CREATE TABLE book (  book\_id INT PRIMARY KEY AUTO\_INCREMENT,  title VARCHAR(50),  author\_id INT NOT NULL,  price DECIMAL(8,2),  amount INT,  **FOREIGN KEY (author\_id) REFERENCES author (author\_id)**  ); |
| Создание индексов для быстрого поиска **INDEX** | INDEX idx\_имя таблицы\_название индекса(имя\_ столбца\_1, имя\_столбца\_2) | INDEX idx\_users\_username(firstname, lastname) -- для быстрого поиска людей по ФИО |
| Действия при удалении записи главной таблицы | С помощью выраженияON DELETE можно установить действия, которые выполняются для записей подчиненной таблицы при удалении связанной строки из главной таблицы. При удалении можно установить следующие опции:   * CASCADE: автоматически удаляет строки из зависимой таблицы при удалении  связанных строк в главной таблице. * SET NULL: при удалении  связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение **NULL**. (В этом случае столбец внешнего ключа должен поддерживать установку **NULL**). * SET DEFAULT похоже на SET NULL за тем исключением, что значение  внешнего ключа устанавливается не в NULL, а в значение по умолчанию для данного столбца. * RESTRICT: отклоняет удаление строк в главной таблице при наличии связанных строк в зависимой таблице.   **Важно!** Если для столбца установлена опция SET NULL, то при его описании нельзя задать ограничение на пустое значение. | CREATE TABLE book (  book\_id INT PRIMARY KEY AUTO\_INCREMENT,  title VARCHAR(50),  author\_id INT NOT NULL,  price DECIMAL(8,2),  amount INT,  FOREIGN KEY (author\_id) REFERENCES author (author\_id) **ON DELETE CASCADE**  );  **id SERIAL, -- SERIAL = BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE** |
| Переименование таблицы **RENAME** | RENAME TABLE old\_name TO new\_name; | RENAME TABLE buyer TO customer; -- Изменить имя таблицы "buyer" на "customer" |
| Занесение новой записи в таблицу **INSERT** | INSERT INTO таблица(поле1, поле2)  VALUES (значение1, значение2); | INSERT INTO book (title, author, price, amount)  VALUES  ('Мастер и Маргарита', 'Булгаков', 670.99, NULL),  ('Идиот', 'Достоевский', 460.00, 10),  ('Братья Карамазовы', 'Достоевский', 799.01, 2); |
| Добавление записей из другой таблицы | С помощью запроса на добавление можно добавить записи **из другой таблицы**, отобранные с помощью запроса на выборку.  В этом случае вместо раздела VALUES записывается запрос на выборку, начинающийся с **SELECT**.  В нем можно использовать WHERE, GROUP BY, ORDER BY. | INSERT INTO book (title, author, price, amount)  SELECT title, author, price, amount  FROM supply  WHERE author NOT IN( 'Булгаков М.А.' , 'Достоевский Ф.М.'); |
| Добавление записей, вложенные запросы | В запросах на добавление можно использовать вложенные запросы. | INSERT INTO book (title, author, price, amount)  SELECT title, author, price, amount  FROM supply  WHERE title NOT IN (  SELECT title  FROM book  );  SELECT \* FROM book; |
| Запросы на обновление  **UPDATE** | UPDATE таблица SET поле = выражение  С помощью запросов на обновление можно изменять не все записи в таблице, а только часть из них. Для этого в запрос включается ключевое слово WHERE, после которого указывается условие отбора строк для изменения. | UPDATE book  SET price = 0.7 \* price  WHERE amount < 5; |
| Запросы на обновление нескольких столбцов | UPDATE таблица SET поле1 = выражение1, поле2 = выражение2 | UPDATE book  SET buy = IF(buy >= amount, amount, buy),  price = IF(buy = 0, 0.9 \* price, price); |
| Запросы на обновление нескольких таблиц | В запросах на обновление нескольких таблиц должны соблюдаться следующие требования:   * для столбцов, имеющих одинаковые имена, необходимо указывать имя таблицы, к которой они относятся, например, **book.price**; * все таблицы, используемые в запросе, нужно перечислить после ключевого слова UPDATE; * в запросе обязательно условие WHERE, в котором указывается условие, при котором обновляются данные. | UPDATE book, supply  SET book.amount = book.amount + supply.amount  WHERE book.title = supply.title AND book.author = supply.author; |
| Изменение таблицы **ALTER TABLE** | -- Добавить столбец "new\_column" в таблицу "Table\_name" ALTER TABLE Table\_name  ADD new\_column VARCHAR(50);  -- Удалить столбец "new\_column" из таблицы "Table\_name" ALTER TABLE Table\_name  DROP COLUMN new\_column; | ALTER TABLE Orders  ADD COLUMN status INT AFTER count\_order;  UPDATE Orders  SET status = RAND();  -- Возращает числа от 0 до 1 |
| Запросы на удаление  **DELETE** | Запросы корректировки данных позволяют удалить одну или несколько записей из таблицы.  DELETE FROM таблица;  Запрос на удаление позволяет удалить не все записи таблицы, а только те, которые удовлетворяют условию, указанному после ключевого слова WHERE:  DELETE FROM таблица  WHERE условие; | DELETE FROM supply;  DELETE FROM supply  WHERE title IN (  SELECT title  FROM book  ); |
| Пересоздание таблицы  **TRUNCATE** | TRUNCATE TABLE имя\_таблицы; | TRUNCATE TABLE messages; |
|  | TRUNCATE: обычно выполняется намного быстрее, чем DELETE, особенно при работе с большими таблицами. Операция выполняется как одна транзакция, не выполняя операции записи журнала. Поэтому TRUNCATE нельзя откатить (нельзя восстановить удаленные строки).  DELETE: позволяет использовать условия WHERE для точной фильтрации строк, которые нужно удалить. Записывает каждую удаленную строку в журнал транзакций, что позволяет проводить откат операции удаления, если это необходимо.  Таким образом, если нужно быстро удалить все строки из таблицы без возможности отката, можно использовать TRUNCATE. Если нужно удалить конкретные строки с использованием условий WHERE или возможности отката операции удаления, тогда используйте DELETE. | |
| Запросы на создание новой таблицы на основе данных из другой таблицы. | Для этого используется запрос SELECT, результирующая таблица которого и будет новой таблицей. При этом имена столбцов запроса становятся именами столбцов новой таблицы.  CREATE TABLE имя\_таблицы AS  SELECT ... | CREATE TABLE ordering AS  SELECT author, title, 5 AS amount  FROM book  WHERE amount < 4; |
| **Запросы на выборку SELECT** |  |  |
| Выбрать все записи таблицы | SELECT \* FROM имя таблицы; | SELECT \* FROM book; |
| Выборка отдельных столбцов | SELECT имя столбца, имя столбца... FROM имя таблицы; | SELECT title, amount FROM book; |
| Выборка столбцов и присвоение им новых имен | SELECT имя столбца **AS** псевдоним, имя столбца...  FROM имя таблицы; | SELECT title AS Название, amount  FROM book; |
| Выборка с созданием вычисляемого столбца | SELECT выражение AS псевдоним, имя столбца...  FROM имя таблицы; | SELECT title, author, price, amount,  **price \* amount AS total**  FROM book; |
| **Математические функции** |  |  |
| округляет до целого числа в большую сторону от х | CEILING(x) | CEILING(4.2)=5 CEILING(-5.8)=-5 |
| округляет значение **x** до **k** знаков после запятой, если **k** не указано – **x** округляется до целого | ROUND(x, k) | ROUND(4.361)=4 ROUND(5.86592,1)=5.9 |
| округляет до  целого числа в меньшую сторону от х | FLOOR(x) | FLOOR(4.2)=4 FLOOR(-5.8)=-6 |
| возведение **x** в степень **y** | POWER(x, y) | POWER(3,4)=81.0 |
| квадратный корень из **x** | SQRT(x) | SQRT(4)=2.0 SQRT(2)=1.41... |
| модуль числа **x** | ABS(x) | ABS(-1) = 1 ABS(1) = 1 |
| **Логические функции** |  |  |
| IF | IF(логическое\_выражение, выражение\_1 (истина), выражение\_2 (ложь)) | SELECT title, amount, price,  ROUND(IF(amount<4, price\*0.5, price\*0.7),2) AS sale  FROM book; |
| CASE - проверяет истинность набора условий и в  зависимости от результата проверки может возвращать тот или  иной результат. | CASE  WHEN условие\_1 THEN результат\_1  WHEN условие\_2 THEN результат\_2  .................................  WHEN условие\_N THEN условие\_N  [ELSE альтернативный\_результат]  END | SELECT ProductName, ProductCount,  CASE  WHEN ProductCount = 1  THEN 'Товар заканчивается'  WHEN ProductCount = 2  THEN 'Мало товара'  WHEN ProductCount = 3  THEN 'Есть в наличии'  ELSE 'Много товара'  END AS Category  FROM Products; |
| Выборка данных по условию | SELECT имя столбца, имя столбца... FROM имя таблицы  WHERE логическое выражение (**=**, **<>**, **>**, **<**, **>=**, **<=**); | SELECT title, price  FROM book  WHERE price < 600; |
| Логические операции | И «AND», ИЛИ «OR», НЕ «NOT»  Приоритеты операций:   1. круглые скобки 2. умножение  (\*),  деление (/) 3. сложение  (+), вычитание (-) 4. =, >, <, >=, <=, <> 5. NOT 6. AND 7. OR | SELECT title, author, price  FROM book  WHERE price > 600 AND author = 'Булгаков М.А.'; |
| Выборка данных из интервала, включая его границы **BETWEEN** | BETWEEN левая граница AND правая граница | SELECT title, amount  FROM book  WHERE amount BETWEEN 5 AND 14; |
| Выборка данных из значений из списка **IN** | IN (значение1, значение2… ) | SELECT title, price  FROM book  WHERE author IN ('Булгаков', 'Достоевский'); |
| Сортировка **ORDER BY**  ASC (по возрастанию, по умолчанию), DESC (по убыванию) | ORDER BY   * название столбца; * номер столбца; * псевдоним столбца.   В качестве критерия сортировки также можно использовать  сложное выражение на основе столбцов | SELECT title, author, amount AS Количество  FROM book  ORDER BY title; /\*ИЛИ \*/  ORDER BY **1, 3, Количество DESC**;  SELECT Price, Product Count  FROM Products  ORDER BY **Product Count \* Price**; |
| Сравнение с шаблоном **LIKE** | LIKE (не чувствителен к регистру)  % - любая строка, содержащая ноль или более символов,  \_ - любой одиночный символ | SELECT title  FROM book  WHERE title LIKE 'Б%'; /\*ИЛИ 'б%' \*/  SELECT title FROM book  WHERE title LIKE "\_% и \_%" /\*отбирает слово И внутри названия \*/  OR title LIKE "и \_%" /\*отбирает слово И в начале названия \*/  OR title LIKE "\_% и" /\*отбирает слово И в конце названия \*/  OR title LIKE "и" /\* отбирает название, состоящее из одного слова И \*/ |
|  | Товары, в названии которых есть **ЦИФРЫ**  **REGEXP '[0-9]'**  **REGEXP '[8]'** | SELECT \*  FROM home\_work.smartphones  WHERE product\_name **REGEXP '[0-9]'**  ; |
| Выбор уникальных элементов столбца **DISTINCT** | DISTINCT имя столбца  или  GROUP BY имя столбца | SELECT DISTINCT author  FROM book;  /\* ИЛИ \*/  SELECT author  FROM book  GROUP BY author; |
| CURRENT\_TIMESTAMP | CURRENT\_TIMESTAMP возвращает текущую дату и время | CREATE TABLE products (  created\_at DATETIME **DEFAULT CURRENT\_TIMESTAMP**,  updated\_at **DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP**); |
| В запросе CREATE TABLE можно использовать следующие операторы:  · DEFAULT  · CHECK | Оператор DEFAULT в запросе CREATE TABLE позволяет указать значение, которое сервер баз данных будет вставлять в столбец, если оно не будет указано явно.  Оператор CHECK позволяет задать условия, которые должны быть выполнены при добавлении данных в столбец запросами INSERT или UPDATE.  Если при добавлении или обновлении строки условие CHECK принимает значение FALSE (ЛОЖЬ), сервер базы данных возвращает ошибку. Если условие CHECK для строки принимает значение NULL, то сервер базы данных не возвращает ошибку. В некоторых случаях может быть необходимо использовать и ограничение CHECK, и ограничение NOT NULL. | |
| **Групповые функции** | SUM() – подсчет суммы значений столбца  COUNT() – подсчет кол-ва записей в столбце   * COUNT(\*) —  подсчитывает  все записи в группе, в том числе NULL; * COUNT(имя\_столбца) — возвращает количество записей столбца (только NOT NULL)в группе.   MIN()-вычисляют минимальное  MAX()- максимальное  AVG()- среднее значение эл-в столбца, относящихся к группе. | SELECT author, sum(amount), count(amount)  FROM book  GROUP BY author;  SELECT author, COUNT(amount), COUNT(\*)  FROM book  GROUP BY author;  SELECT author, MIN(price) AS Минимальная\_цена,  MAX(price) AS Максимальная\_цена, AVG(price) AS Средняя\_цена  FROM book  GROUP BY author; |
| Групповые функции c вычислением |  | SELECT author, SUM(price \* amount) AS Стоимость  FROM book  GROUP BY author;  SELECT author, ROUND(AVG(price),2) AS Средняя\_цена  FROM book  GROUP BY author; |
| Вычисления по таблице целиком | Результатом таких запросов является единственная строка с вычисленными по таблице значениями. | SELECT MIN(price), MAX(price), ROUND(AVG(DISTINCT price), 2)  FROM book; |
| Групповые функции с выборкой по условию **GROUP BY** | В запросах с групповыми функциями вместо WHERE используется ключевое слово HAVING , указываемое после GROUP BY. | SELECT author,  MIN(price) AS Минимальная\_цена,  FROM book  GROUP BY author  HAVING SUM(price \* amount) > 5000; |
| WHERE и HAVING могут использоваться в одном запросе | Вывести максимальную и минимальную цену книг каждого автора, кроме Есенина, количество экземпляров книг которого больше 10. | SELECT author,  MIN(price) AS Минимальная\_цена,  MAX(price) AS Максимальная\_цена  FROM book  WHERE author <> 'Есенин С.А.'  GROUP BY author  HAVING SUM(amount) > 10; |
| **Вложенные запросы** |  |  |
| Вложенный запрос, возвращающий **одно** значение | Вложенный запрос, возвращающий одно значение, может использоваться в условии отбора записей WHERE как обычное значение совместно с операциями =, <>, >=, <=, >, <. | SELECT title, author, price, amount  FROM book  WHERE price = (  SELECT MIN(price)  FROM book  ); |
| Использование вложенного запроса **в выражении** | Вложенный запрос, возвращающий одно значение, может использоваться в выражениях как обычный операнд, например, к нему можно что-то прибавить, вычесть и пр. | SELECT title, author, amount  FROM book  WHERE ABS(amount - (SELECT AVG(amount) FROM book)) >3; |
| Вложенный запрос, возвращающий **несколько** значений | Если вложенный запрос возвращает несколько значений одного столбца, его можно использовать в разделе WHERE совместно с оператором IN.  WHERE имя\_столбца IN (вложенный запрос, возвращающий один столбец) | SELECT title, author, amount, price  FROM book  WHERE author IN (  SELECT author  FROM book  GROUP BY author  HAVING SUM(amount) >= 12  ); |
| Вложенный запрос, **операторы ANY и ALL** | Вложенный запрос, возвращающий несколько значений одного столбца, можно использовать для отбора записей с помощью операторов ANY и ALL совместно с операциями отношения (=, <>, <=, >=, <, >). | SELECT title, author, amount, price  FROM book  WHERE amount < ALL (  SELECT AVG(amount)  FROM book  GROUP BY author  ); |
| Вложенный запрос **после** SELECT | Если вложенный запрос располагается после SELECT, то результат выполнения запроса выводится в отдельном столбце результирующей таблицы. При этом результатом запроса может быть только одно значение, тогда оно будет повторяться во всех строках. Также вложенный запрос может использоваться в выражениях. | SELECT title, author, amount,  (  SELECT AVG(amount)  FROM book  ) AS Среднее\_количество  FROM book  WHERE abs(amount - (SELECT AVG(amount) FROM book)) >3; |
| **Запросы на выборку** |  |  |
| Ограничение выборки и отбрасывание записей  **LIMIT**  **OFFSET FETCH** | LIMIT [кол-во\_пропущенных\_записей,] кол-во\_записей\_для\_вывода;  Существует и альтернативная форма записи:  LIMIT 2 OFFSET 2;  OFFSET FETCH  Это будет извлекать строки только от (m+1) до (m+1+p). | SELECT author,  FROM book  OFFSET 3  LIMIT 3;  SELECT ColumnNames  FROM TableName  ORDER BY ColumnNames  OFFSET m ROWS FETCH NEXT p ROWS ONLY; |
| Вычитания двух дат | **DATEDIFF(дата\_1, дата\_2)**  результатом является количество дней между **дата\_1** и **дата\_2**.  Увеличьте разницу на 1, чтобы включить первый день | DATEDIFF('2020-04-01', '2020-03-28')=4  DATEDIFF('2020-05-09','2020-05-01')=8  DATEDIFF(date\_last, date\_first) + 1 |
| Выделить **номер** месяца из даты | **MONTH(дата)** | **MONTH('2020-04-12') = 4**  **MONTH(date\_first)** |
| Выделить **название** месяца из даты | **MONTHNAME(дата)** | **MONTHNAME('2020-04-12')='April'** |
| Выделить часть даты | День **DAY()**,  месяц **MONTH()**,  год **YEAR()** | DAY('2020-02-01') = 1  MONTH('2020-02-01') = 2  YEAR('2020-02-01') = 2020 |
| Генерации случайных чисел в интервале от 0 до 1 (не включительно)  **RAND()** | Функция RAND()  Если эту функцию умножить на 365, то она будет генерировать вещественные числа от 0 до 365 (не включительно). Таким образом, можно получить случайное число от 0 до 365. | FLOOR(RAND() \* 365) |
| Сложение даты с числом **INTERVAL** | DATE\_ADD(дата, INTERVAL число единица\_измерения),  Где  единица\_измерения (использовать прописные буквы) – это день (DAY), месяц(MONTH), неделя(WEEK) и пр.,  число – целое число,  дата – значение даты или даты и времени. | DATE\_ADD('2020-02-02', INTERVAL 45 DAY) возвращает 18 марта 2020 года  DATE\_ADD('2020-02-02', INTERVAL 6 MONTH) возвращает 2 августа 2020 года  SELECT '2018-10-01 0:00:00' - INTERVAL 1 DAY;  SELECT '2018-10-01 0:00:00' + INTERVAL 1 WEEK; SELECT '2018-10-01 0:00:00' + INTERVAL 1 YEAR; |
| Извлекаем первые символы **SUBSTRING** | извлекаем первые три цифры года рождения | SELECT id, name, SUBSTRING(birthday\_at, 1, 3) AS decade FROM users ORDER BY decade; |
| Использование временного имени таблицы | FROM fine AS f, traffic\_violation AS tv  После присвоения таблице алиаса, он используется во всех разделах запроса, в котором алиас задан. Например:  WHERE f.violation = tv.violation | UPDATE fine AS f, traffic\_violation AS tv  SET f.sum\_fine = tv.sum\_fine  WHERE f.sum\_fine IS NULL AND  f.violation = tv.violation; |
| Группировка данных по нескольким столбцам | В разделе GROUP BY можно указывать несколько столбцов, разделяя их запятыми. Тогда к одной группе будут относиться записи, у которых значения столбцов, входящих в группу, равны.  **Важно!** В разделе **GROUP BY** нужно перечислять все НЕАГРЕГИРОВАННЫЕ столбцы (к которым не применяются групповые функции) из **SELECT**. | SELECT name, number\_plate, violation, count(\*)  FROM fine  GROUP BY name, number\_plate, violation; |
| можно использовать вложенный запрос как отдельную таблицу, записанную после ключевого слова UPDATE, при этом вложенному запросу необходимо присвоить имя, например **query\_in** | UPDATE fine,  (  SELECT ...  ) query\_in  SET ...  WHERE | UPDATE fine,  (  SELECT name, number\_plate, violation  FROM fine  GROUP BY name, number\_plate, violation  HAVING COUNT(\*) >= 2  ORDER BY name, number\_plate, violation  ) query\_in  SET sum\_fine = sum\_fine \* 2  WHERE date\_payment IS NULL AND  fine.name=query\_in.name AND  fine.number\_plate=query\_in.number\_plate  AND fine.violation=query\_in.violation; |
| **E:\Downloads\Untitled(1).pngСоединение таблиц** | | |
| Соединение INNER JOIN | Оператор внутреннего соединения INNER JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.  SELECT  ...  FROM  таблица\_1 INNER JOIN таблица\_2  ON условие | SELECT title, name\_author  FROM  author INNER JOIN book  ON author.author\_id = book.author\_id; |
| Внешнее соединение LEFT и RIGHT OUTER JOIN | Оператор внешнего соединения LEFT OUTER JOIN (OUTER можно опустить) соединяет две таблицы. Порядок таблиц для оператора важен, поскольку оператор не является симметричным.  SELECT  ...  FROM  таблица\_1 LEFT JOIN таблица\_2  ON условие  ... | SELECT name\_author, title  FROM author LEFT JOIN book  ON author.author\_id = book.author\_id  ORDER BY name\_author; |
| Перекрестное**соединение CROSS JOIN** | Оператор перекрёстного соединения, или декартова произведения CROSS соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.  SELECT  ...  FROM  таблица\_1 CROSS JOIN таблица\_2  ...  или  SELECT  ...  FROM  таблица\_1, таблица\_2  ...  Результат запроса формируется так: каждая строка одной таблицы соединяется с каждой строкой другой таблицы, формируя в результате все возможные сочетания строк двух таблиц. | SELECT name\_author, name\_genre  FROM  author, genre; |
| Запросы на выборку из нескольких таблиц | Запрос на выборку может выбирать данные из двух и более таблиц базы данных. При этом таблицы должны быть логически связаны между собой. Для каждой пары таблиц, включаемых в запрос, необходимо указать свой оператор соединения. Наиболее распространенным является внутреннее соединение INNER JOIN  Пусть таблицы связаны между собой следующим образом:  тогда запрос на выборку для этих таблиц будет иметь вид ----->>>  Если же таблицы связаны так:  то запрос на выборку выглядит следующим образом --->>> | SELECT  ...  FROM  first  INNER JOIN second ON first.first\_id = second.first\_id  INNER JOIN third ON second.second\_id = third.second\_id  ...  SELECT  ...  FROM  first  INNER JOIN third ON first.first\_id = third.first\_id  INNER JOIN second ON third.second\_id = second.second\_id  ... |
| Операция соединение, использование USING() | При описании соединения таблиц с помощью **JOIN** в некоторых случаях вместо **ON** и следующего за ним условия можно использовать оператор **USING()**.  **USING** позволяет указать набор столбцов, которые есть в обеих объединяемых таблицах. Если база данных хорошо спроектирована, а каждый внешний ключ имеет такое же имя, как и соответствующий первичный ключ (например, **genre.genre\_id = book.genre\_id**), тогда можно использовать предложение **USING** для реализации операции **JOIN**.  При этом после **SELECT**,при использовании столбцов из **USING()**, необязательно указывать, из какой именно таблицы берется столбец. | Вариант с **ON**  SELECT title, name\_author, author.author\_id /\* явно указать таблицу - обязательно \*/  FROM  author INNER JOIN book  ON author.author\_id = book.author\_id;  Вариант с **USING**  SELECT title, name\_author, author\_id /\* имя таблицы, из которой берется author\_id, указывать не обязательно\*/  FROM  author INNER JOIN book  USING(author\_id); |
| Объединение **UNIONE:\Downloads\Untitled(2).png** | Если формат результирующих таблиц совпадает, возможно объединение результатов выполнения двух операторов **SELECT** в одну результирующую таблицу. Для этого используется оператор **UNION**. Важное условие — совпадение всех параметров результирующих запросов. Количество, порядок следования и тип столбцов должны совпадать.  В результирующий запрос попадают только не повторяющиеся результаты.  Если необходимо вывести все данные без повторения, используется оператор **UNION ALL** | SELECT name FROM catalogs  UNION  SELECT name FROM rubrics;  SELECT name FROM catalogs  UNION ALL  SELECT name FROM rubrics; |
| Оператор **EXISTS** | Оператор EXISTS проверяет, возвращает ли подзапрос  какое-либо значение. Как правило, этот оператор используется для индикации того, что как минимум одна строка в таблице удовлетворяет некоторому условию. Поскольку возвращения набора строк не происходит, то подзапросы с подобным оператором выполняются довольно быстро. | Например, найдем все товары из таблицы Products, на которые  есть заказы в таблице Orders:  SELECT \* FROM Products  WHERE EXISTS  (SELECT \* FROM Orders WHERE Orders.ProductId = Products.Id) |
| **Оконные функции** | Можно сравнить с агрегатными функциями, но при использовании оконной функции несколько строк не группируются в одну, а продолжают существовать отдельно. При этом результаты работы оконных функций просто добавляются к результирующей выборке как еще одно поле.  Этот функционал полезен для расчета скользящего среднего и нарастающих итогов.  SELECT Название\_функции (столбец для вычислений) OVER  (  PARTITION BY столбец для группировки  ORDER BY столбец для сортировки  ROWS или RANGE выражение для ограничения строк в пределах группы  ) | SELECT shopping\_day, department, count,  SUM(count) OVER(PARTITION BY shopping\_day ORDER BY department) AS 'Sum'  FROM shop; |
| Ранжирующие функции | Можно использовать для того, чтобы присвоить порядковый номер строки или составить рейтинг.  ROW\_NUMBER – функция возвращает номер строки и  используется для нумерации;  RANK — функция возвращает ранг каждой строки. В данном случае значения уже анализируются и, в случае нахождения одинаковых, возвращает одинаковый ранг с пропуском следующего значения;  DENSE\_RANK — функция возвращает ранг каждой строки. Но в отличие от функции RANK, она для одинаковых значений возвращает ранг, не пропуская следующий;  NTILE – это функция, которая позволяет определить к какой группе относится текущая строка. Количество групп задается в скобках. |  |
| Агрегирующие функции | Они используются для учета средней зарплаты, подсчета количества сотрудников, максимальную и минимальную ЗП и сумму по чеку. | SELECT shopping\_day, department, count,  SUM(count) OVER(PARTITION BY shopping\_day) AS 'Sum' ,  COUNT(count) OVER(PARTITION BY shopping\_day) AS 'Count' ,  AVG(count) OVER(PARTITION BY shopping\_day) AS 'Avg' ,  MAX(count) OVER(PARTITION BY shopping\_day) AS 'Max' ,  MIN(count) OVER(PARTITION BY shopping\_day) AS 'Min'  FROM shop; |
| Функции смещения | Функции, которые позволяют перемещаться и обращаться к разным строкам в окне, относительно текущей строки, а также обращаться к значениям в начале или в конце окна.  LAG или LEAD – функция LAG обращается к данным из  предыдущей строки окна, а LEAD к данным из следующей строки. Функцию можно использовать для того, чтобы сравнивать текущее значение строки с предыдущим или следующим. Имеет три параметра: столбец, значение которого необходимо вернуть, количество строк для смещения (по умолчанию 1), значение, которое необходимо вернуть если после смещения возвращается значение NULL;  FIRST\_VALUE или LAST\_VALUE — с помощью функции можно получить первое и последнее значение в окне. В качестве параметра принимает столбец, значение которого необходимо вернуть. | SELECT shopping\_day, department, count,  LAG(count) OVER(PARTITION BY shopping\_day ORDER BY shopping\_day) AS 'Lag' ,  LEAD(count) OVER(PARTITION BY shopping\_day ORDER BY shopping\_day) AS 'Lead' ,  FIRST\_VALUE(count) OVER(PARTITION BY shopping\_day ORDER BY shopping\_day) AS 'First\_Value' ,  LAST\_VALUE(count) OVER(PARTITION BY shopping\_day ORDER BY shopping\_day) AS 'Last\_Value'  FROM shop;  E:\Downloads\Untitled(3).png |
| Аналитические функции | Функции, которые возвращают информацию о распределении данных и используются для статистического анализа.  CUME\_DIST — вычисляет интегральное распределение  (относительное положение) значений в окне;  PERCENT\_RANK — вычисляет относительный ранг строки в окне;  PERCENTILE\_CONT — вычисляет процентиль на основе  постоянного распределения значения столбца. В качестве параметра принимает процентиль, который необходимо вычислить;  PERCENTILE\_DISC — вычисляет определенный процентиль для отсортированных значений в наборе данных. В качестве параметра принимает процентиль, который необходимо вычислить.  Важно! У функций PERCENTILE\_CONT и PERCENTILE\_DISC,  столбец, по которому будет происходить сортировка,  указывается с помощью ключевого слова WITHIN GROUP. |  |
| **Представление (VIEW)** | «Виртуальные таблицы». Представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему.  Это может поддерживать безопасность и целостность данных, а также пользователи могут выполнять свои задачи с соответствующими авторизованными столбцами. | SELECT \*  FROM Londonstaff; |
| **Удалить представление/виртуальную таблицу** | DROP VIEW |  |
| **Изменить представление/виртуальную таблицу** | ALTER VIEW |  |
| **Транзакции**  Принцип **ACID**  ● **Atomicy** — атомарность.  ● **Consistency** — согласованность. ● **Isolation** — изолированность.  ● **Durability** — сохраняемость.  Стандарт SQL определяет 4 уровня изоляции:  ● READ UNCOMMITTED  ● READ COMMITTED  ● REPEATABLE READ  ● SERIALIZABLE | Операции с денежными средствами — классический пример, показывающий, почему необходимы транзакции. Если при оплате покупки происходит перевод от клиента электронному магазину, то счет клиента должен уменьшиться на эту сумму, а счет электронного магазина — увеличиться на нее же. | START TRANSACTION;  -- Далее выполняем команды, входящие в транзакцию:  SELECT total FROM accounts WHERE user\_id = 4;  -- Убеждаемся, что на счету пользователя достаточно средств:  UPDATE accounts SET total = total - 2000 WHERE user\_id = 4;  -- Снимаем средства со счета пользователя:  UPDATE accounts SET total = total + 2000 WHERE user\_id IS NULL;  -- Чтобы изменения вступили в -- силу, мы должны выполнить команду COMMIT  COMMIT;  -- cкрипт выполнять полностью: начиная от первой и до самой последней строчки |
| Точка сохранения | Для работы с точками сохранения предназначены два оператора:  SAVEPOINT  ROLLBACK TO SAVEPOINT | START TRANSACTION;  SELECT total FROM accounts WHERE user\_id = 4;  SAVEPOINT accounts\_4;  UPDATE accounts SET total = total - 2000 WHERE user\_id = 4;  -- Допустим мы хотим отменить транзакцию и вернуться в точку сохранения. Можем воспользоваться оператором ROLLBACK TO SAVEPOINT:  ROLLBACK TO SAVEPOINT accounts\_4;  SELECT \* FROM accounts; |
| **Временная таблица** | Временная таблица автоматически удаляется по завершении соединения с сервером, а ее имя действительно только в течение данного соединения. Это означает, что два разных клиента могут использовать временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем. | CREATE TEMPORARY TABLE temp (id INT, name VARCHAR(255));  DESCRIBE temp; -- Показ всех столлбцов в таблице temp |
| **Переменные** | Часто результаты запроса необходимо использовать в последующих запросах. Для этого полученные данные следует сохранить во временных структурах. Эту задачу решают переменные SQL. | SELECT @total := COUNT(\*) FROM accounts;  SELECT @total; |
| **Хранимые процедуры и функции** | Позволяют сохранить последовательность SQL-операторов и вызывать их по имени функции или процедуры.  Разница между процедурой и функцией: функции возвращают значение и их можно встраивать в SQL-запросы, а хранимые процедуры вызываются явно. | |
| Процедуры | DELIMITER {custom delimiter}  CREATE PROCEDURE {proc\_name}([optional parameters])  BEGIN  // procedure body...  // procedure body...  END  {custom delimiter}   * delimiter - необходим для изменения разделителя SQL-инструкций с ; на // во время определения процедуры. Это позволяет разделитель ; использовать в теле процедуры для передачи на сервер. * proc\_name - уникальное имя хранимой процедуры, длиной не более 64 символа. Имена процедур не чувствительны к регистру, поэтому в одной схеме не может быть двух событий с именами procname и ProcName; * в процедуру можно передать параметры (optional\_parametrs)   **IN** - Параметр может ссылаться на процедуру. Значение параметра не может быть перезаписано процедурой.  **OUT** - Параметр не может ссылаться на процедуру, но значение параметра может быть перезаписано процедурой.  **IN OUT** - Параметр может ссылаться на процедуру, и значение параметра может быть перезаписано процедурой. | DELIMITER //  CREATE PROCEDURE my\_version ()  BEGIN  SELECT VERSION();  END //  -- CALL proc\_name;  CALL my\_version (); -- Вызов процедуры |
| Получить список хранимых процедур | SHOW PROCEDURE STATUS; -- Все процедуры |  |
| Удаление хранимых процедур и функций | DROP PROCEDURE  DROP FUNCTION | DROP PROCEDURE my\_version;  DROP PROCEDURE IF EXISTS my\_version; |
| Функции | DELIMITER {custom delimiter}  CREATE FUNCTION function\_name [ (parameter datatype [, parameter datatype]) ]  RETURNS return\_datatype  BEGIN  declaration\_section  executable\_section  END; | DROP FUNCTION IF EXISTS get\_version;  DELIMITER //  CREATE FUNCTION get\_version ()  RETURNS TEXT DETERMINISTIC  BEGIN  RETURN VERSION();  END//  SELECT get\_version(); -- вызов хранимой функции  Ключевое слово DETERMINISTIC сообщает, что при каждом вызове будет возвращаться одно и то же значение. Если значения, которые возвращает функция, каждый раз различны, то перед DETERMINISTIC следует добавить отрицание NOT. |
| Ветвление IF | Оператор IF позволяет реализовать ветвление программы по условию. IF принимает значение либо TRUE (истину), либо FALSE (ложь). В MySQL TRUE и FALSE — константы для целочисленных значений 1 и 0. Если логическое выражение истинно, IF выполняет SQL-выражения, которые размещаются в теле команды между ключевыми словами THEN и END IF. | DELIMITER //  DROP PROCEDURE IF EXISTS format\_now//  CREATE PROCEDURE format\_now (format CHAR(4))  BEGIN  IF(format = 'date') THEN  SELECT DATE\_FORMAT(NOW(), "%d.%m.%Y") AS format\_now;  ELSE  SELECT DATE\_FORMAT(NOW(), "%H:%i:%s") AS format\_now;  END IF;  END//  CALL format\_now('date')//  CALL format\_now('time')// |
| Циклы | Циклы являются важнейшей конструкцией, без которой хранимые процедуры и функции не имели бы достаточно функциональности. MySQL предоставляет три цикла: **while**, **repeat** и **loop**. Их можно использовать в теле хранимой процедуры или функции, т.е., между ключевыми словами BEGIN и END. | DELIMITER //  CREATE PROCEDURE while\_cycle ()  BEGIN  DECLARE i INT DEFAULT 3;  WHILE i > 0 DO  SELECT NOW();  SET i = i - 1;  END WHILE;  END//  CALL while\_cycle()// --2023-03-19 15:37:53 |

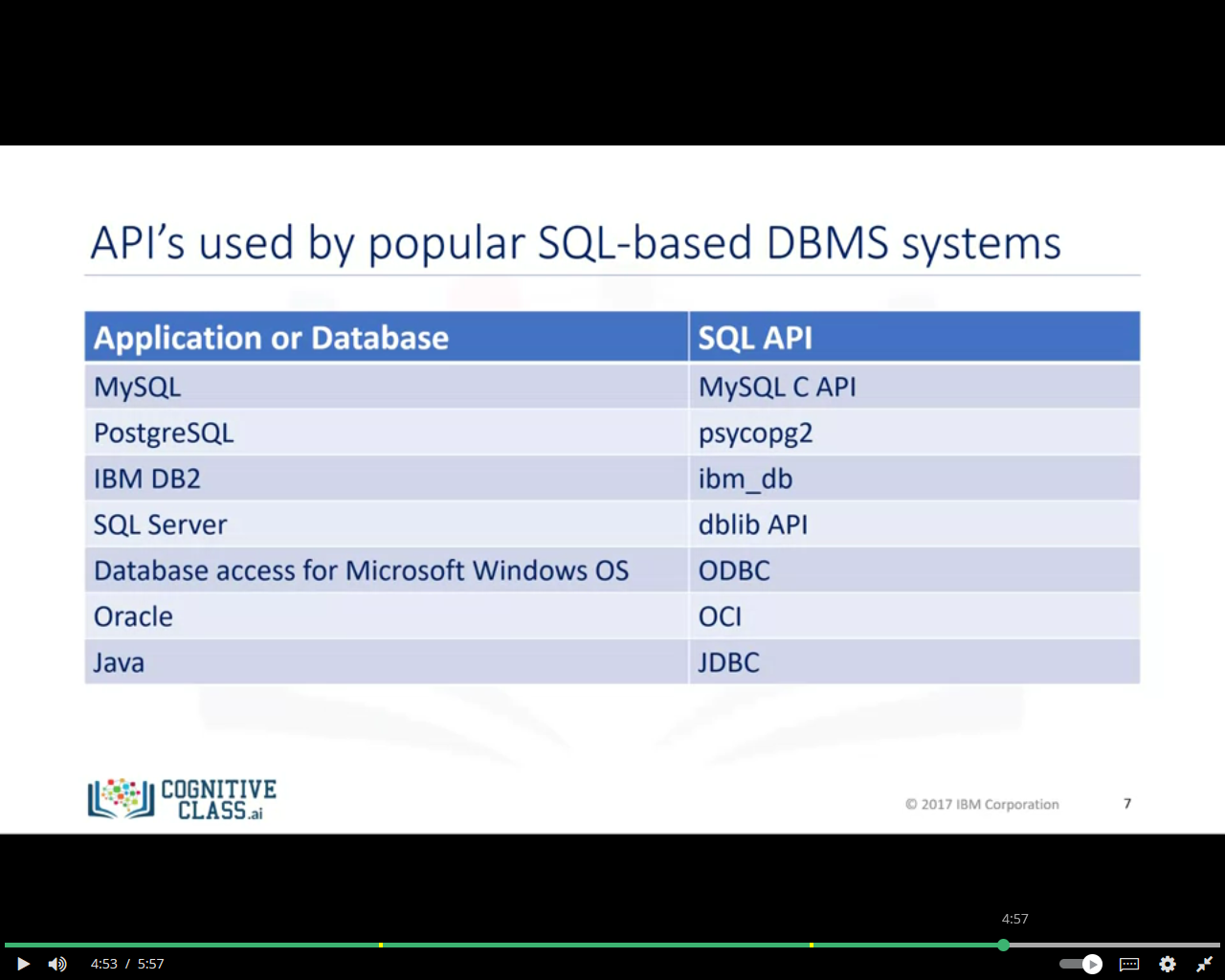
# Доступ к базам данных из программ на языке Python

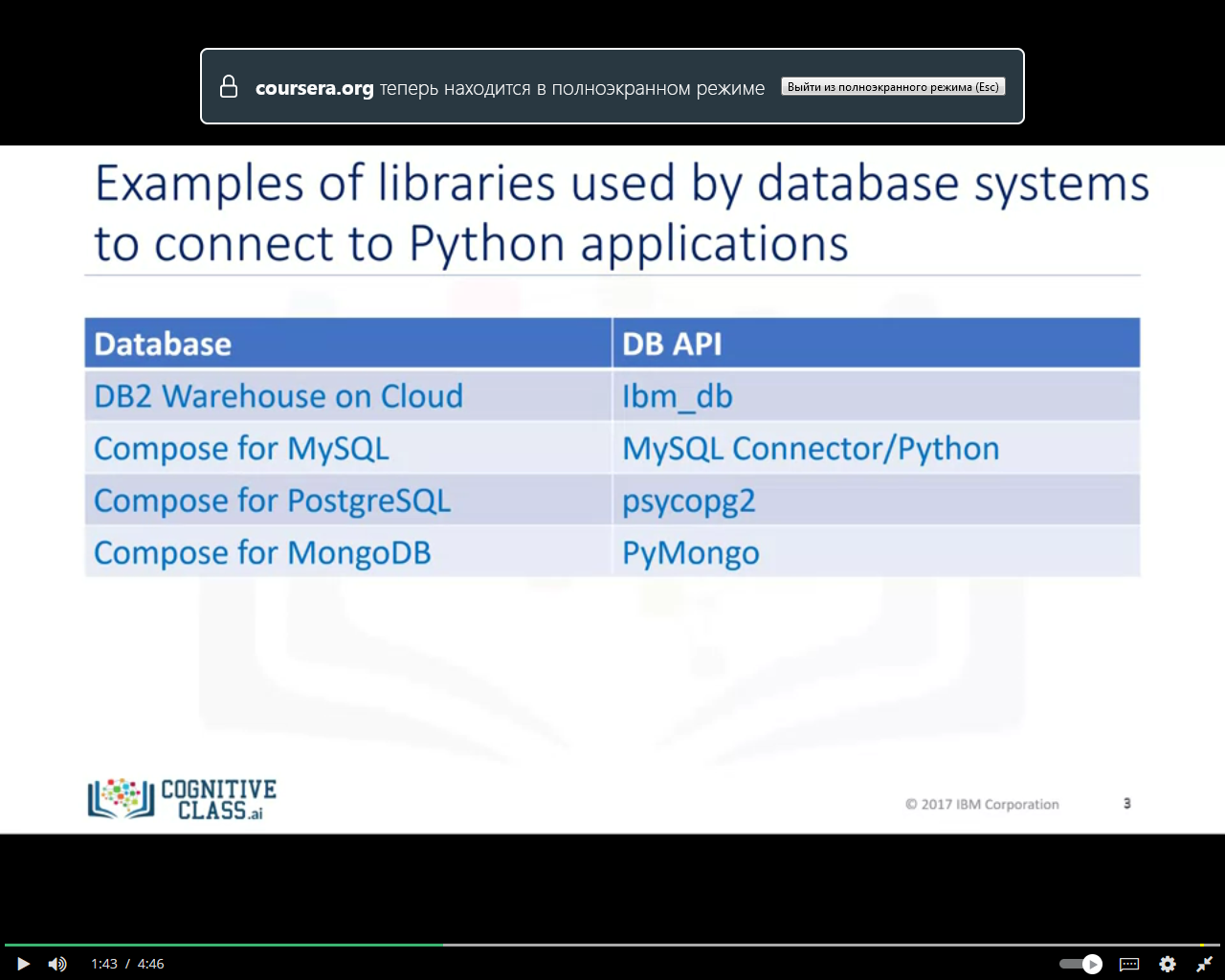
Написание кода на Python для доступа к БД упрощается благодаря существованию API баз данных на Python. Этот механизм называется DB-API. Существует подробная документация о его использовании на Python.

Вот так осуществляется доступ к БД на языке Python из среды Jupyter через редактор в веб-браузере. Есть механизм взаимодействия программы на Python с СУБД. Программа на Python подключается к БД с помощью вызовов API.

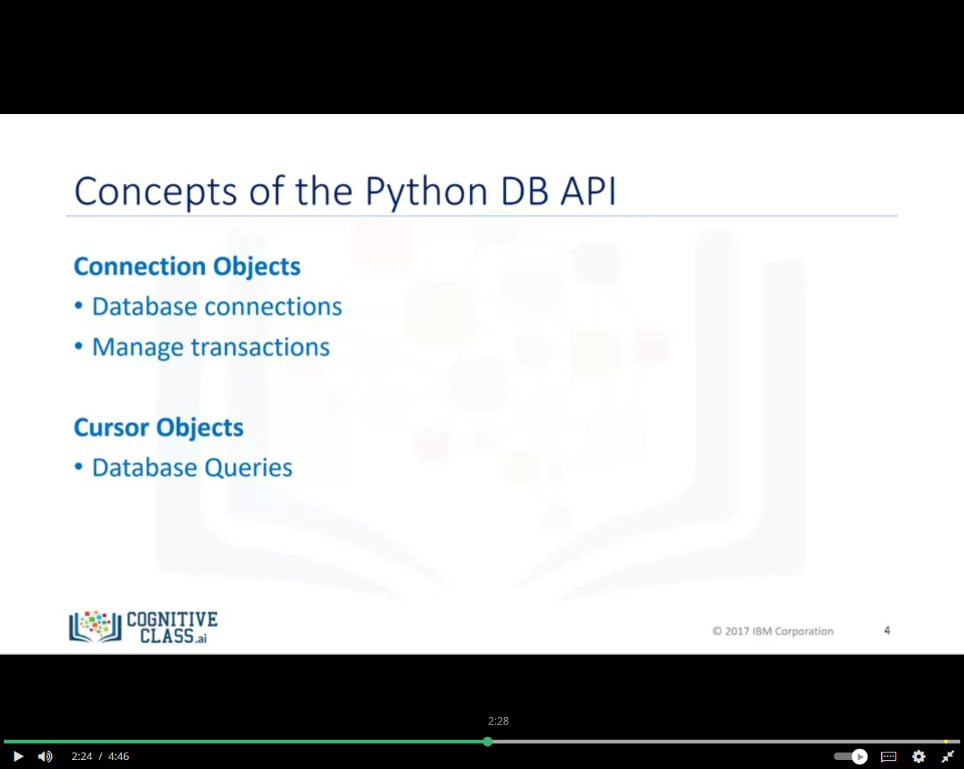


API — это программный интерфейс приложения, состоящий из набора функций, которые можно вызвать для получения доступа к различным типам серверов. API SQL состоит из библиотеки функций и является программным интерфейсом т. е. API для СУБД. Отправляя SQL-запросы к СУБД, приложение вызывает функции API. Затем оно вызывает другие функции для получения результатов запроса и информации о статусе от СУБД. Программа начинает доступ к базе данных с одного или нескольких вызовов API для подключения к СУБД. Чтобы отправить SQL-запрос к СУБД, программа составляет его как текстовую строку в буфере, а затем выполняет вызов API для передачи содержимого буфера в СУБД. Приложение вызывает функции API для проверки статуса запросов к СУБД и обработки ошибок. Программа завершает доступ к базе данных вызовом функции API для отключения от БД.

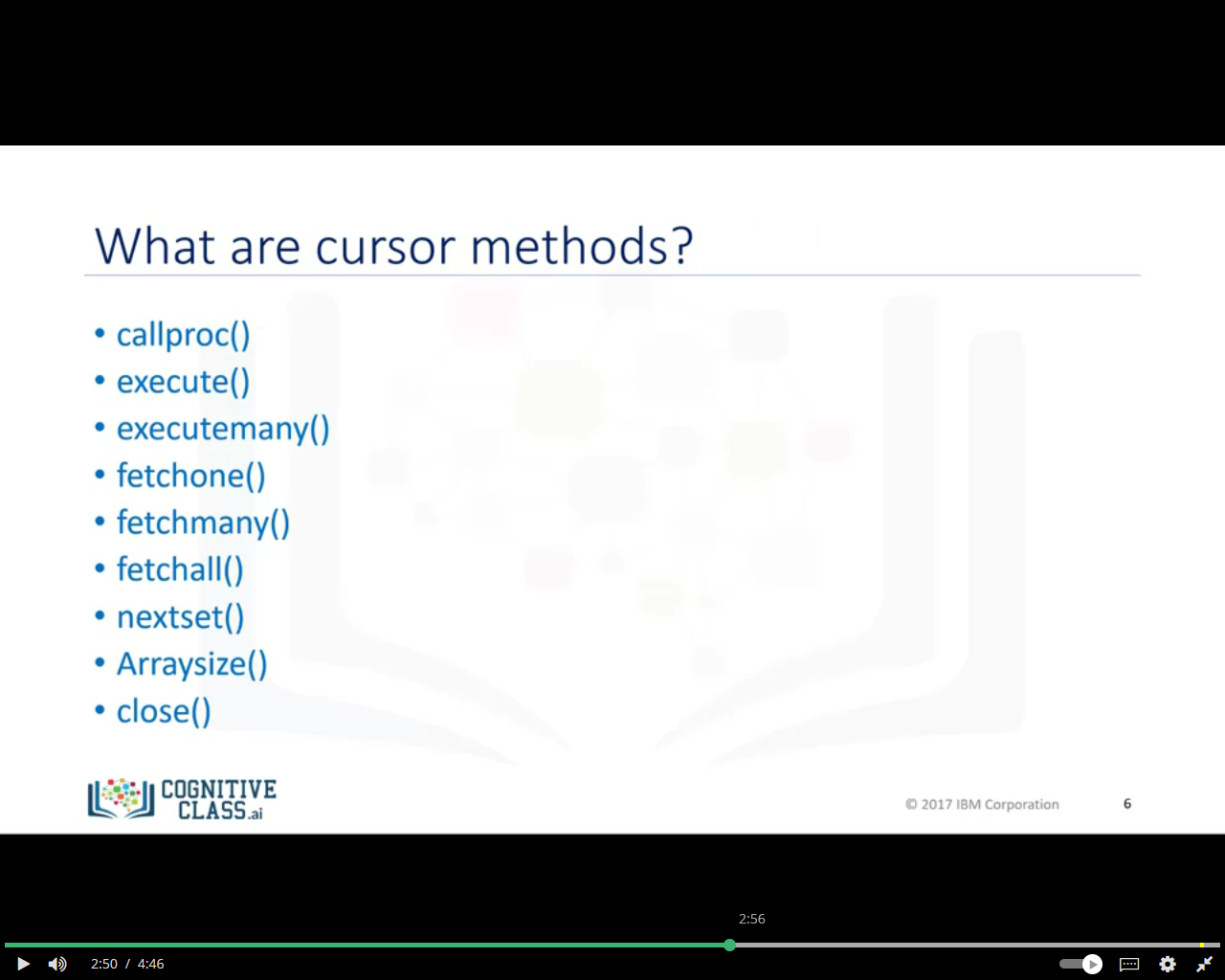
Теперь изучим основные концепции частных API, используемых популярными СУБД на базе SQL. У каждой системы баз данных есть своя библиотека. В таблице на слайде приведен список приложений и соответствующих API SQL. MySQL C API предоставляет низкоуровневый доступ к клиент-серверному протоколу MySQL и позволяет получать доступ к БД на языке С. API psycopg2 используется для подключения приложений на Python к БД PostgreSQL. API IBM\_DB используется для подключения приложения на Python к БД IBM DB2. API dblib применяется для подключения к БД SQL Server. ODBC дает доступ к БД операционной системе Microsoft Windows. OCI используют базы данных Oracle. И, наконец, JDBC предназначен для приложений на Java.



Как известно, у каждой системы баз данных есть своя библиотека. В таблице приведен список баз данных и соответствующих интерфейсов DB-API для подключения к приложениям Python. Для подключения к БД DB2 Warehouse on Cloud используют библиотеку Ibm\_db и т.д.



Две основные концепции Python DB-API: объекты-подключения и объекты-запросы. Объекты-подключения отвечают за подключение к БД и управление транзакциями. Объекты-курсоры нужны для выполнения запросов. Сначала открываем объект-курсор, затем делаем запрос. Курсор БД выполняет ту же функцию, что и курсор в текстовом редакторе. Как будто вы прокручиваете результирующую выборку и переносите данные в программу. Курсоры используют для просмотра результатов в базе данных.



Вот методы для работы с объектами-подключениями.

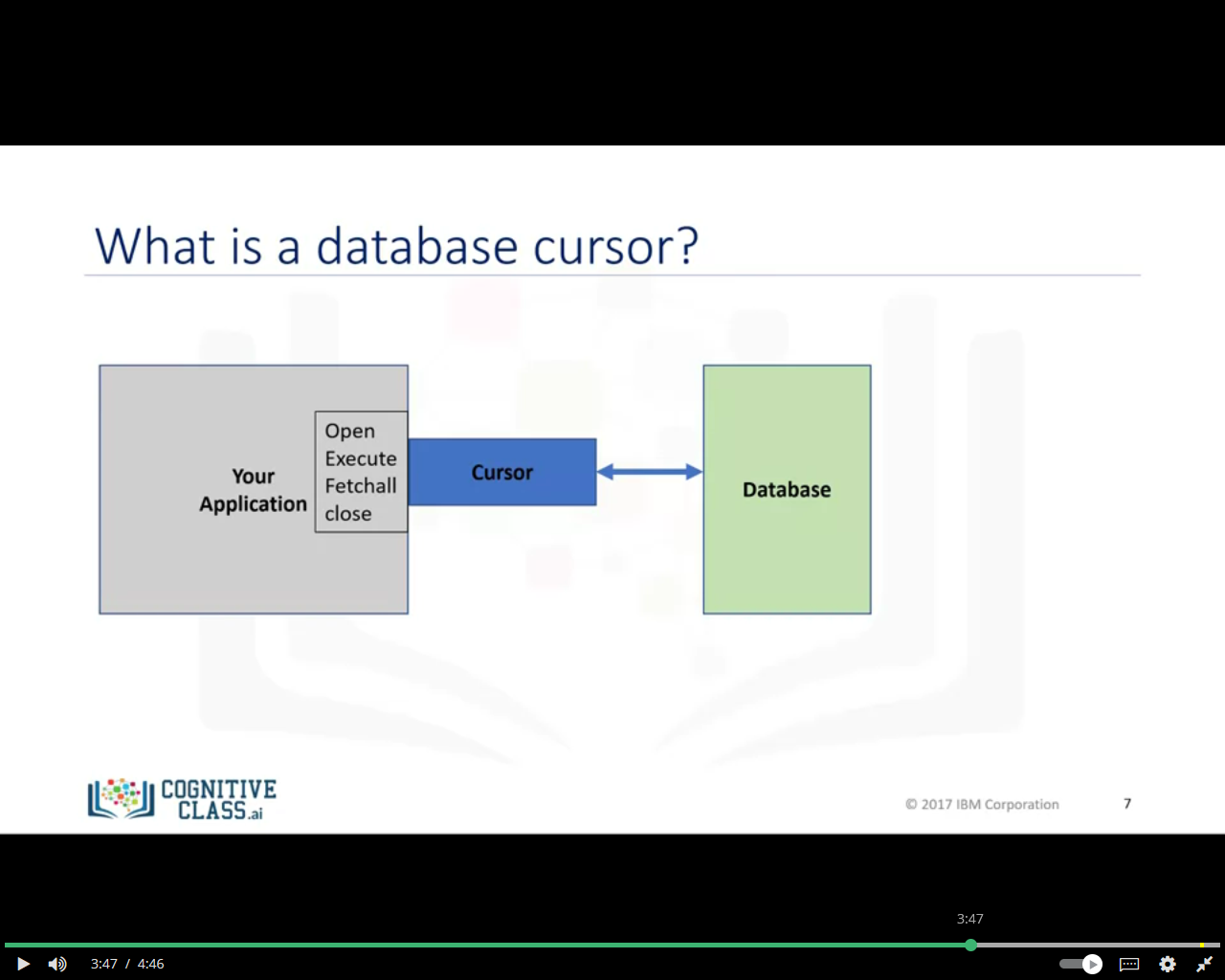
Метод cursor() возвращает новый курсор через подключение.

Метод commit() фиксирует отложенную транзакцию в базе данных.

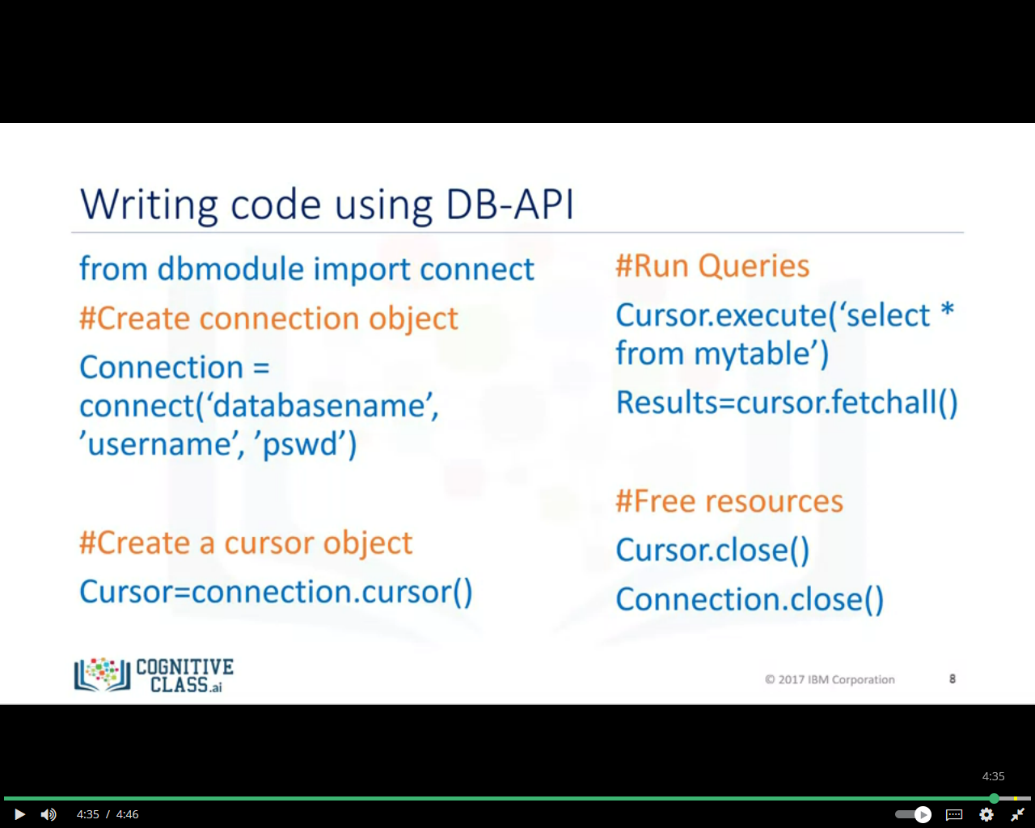
Метод rollback() откатывает базу данных к началу отложенной транзакции.

Метод close() закрывает соединение с базой данных.

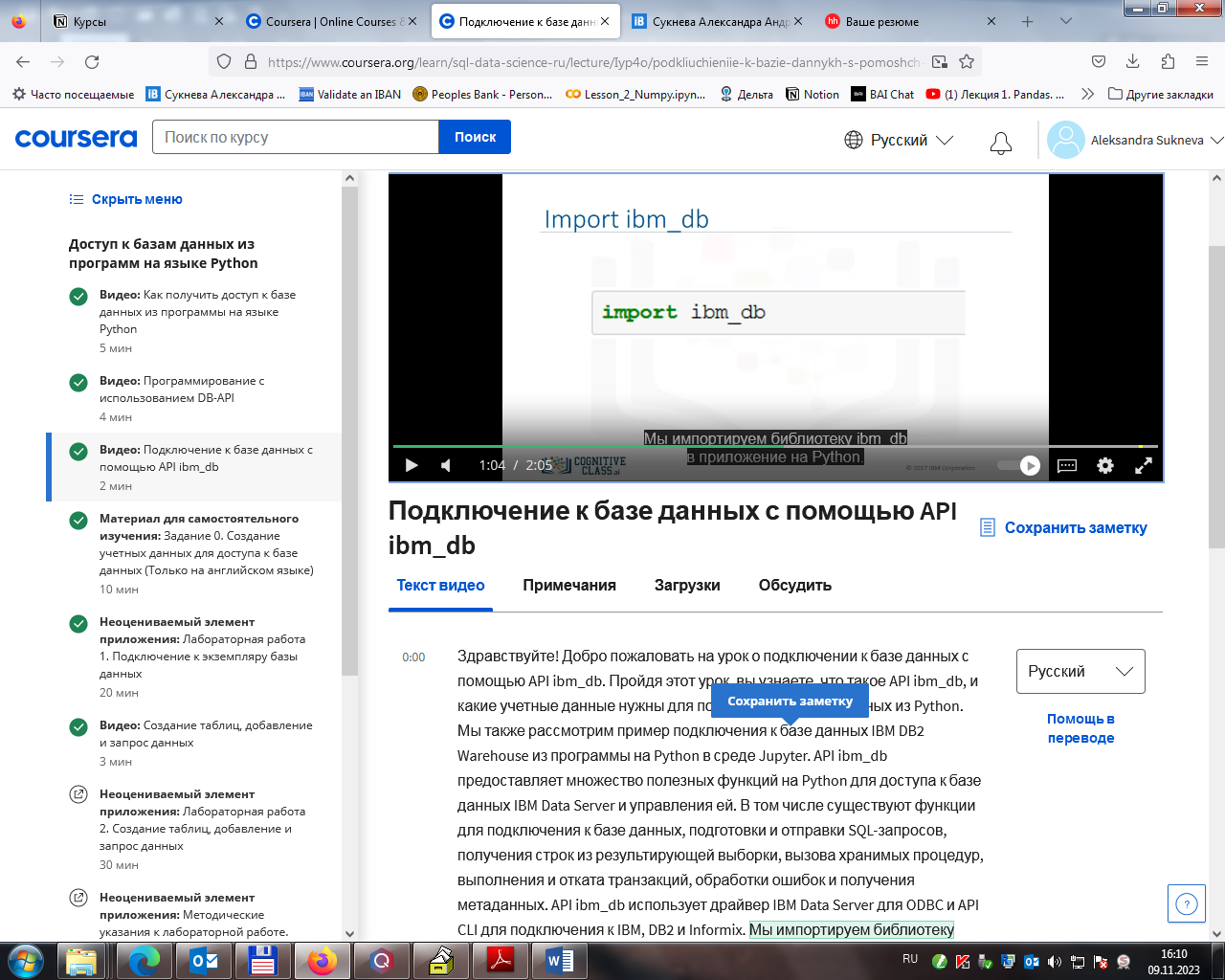
Эти объекты представляют курсор базы данных, используемый для управления результатом операции FETCH. Курсоры в рамках одного подключения не независимы, т. е. изменения, внесенные в БД одним курсором, сразу доступны другим курсорам. Курсоры, созданные в разных подключениях, могут быть или не быть независимыми в зависимости от реализации транзакции.



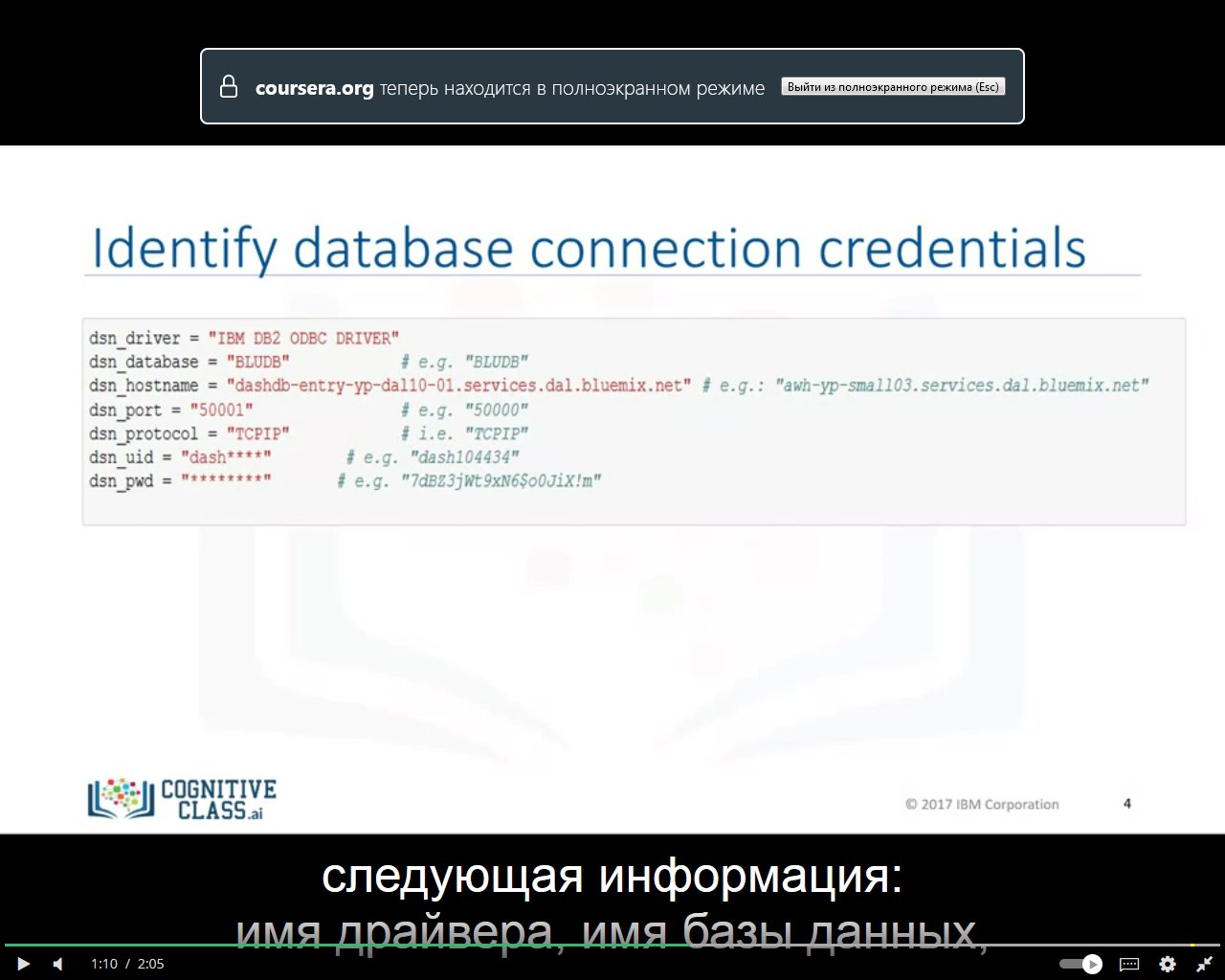
Курсор базы данных — это управляющая структура, которая позволяет просматривать записи в базе данных. Он похож на имя файла или файловый дескриптор в языке программирования. Подобно тому, как программа открывает файл и получает доступ к его содержимому, она открывает курсор и получает доступ к результату выполнения запроса. Также программа закрывает файл, завершая доступ, и закрывает курсор, завершая доступ к результатам запроса. Еще одно сходство в том, что файловый дескриптор запоминает текущую позицию программы в открытом файле, а курсор — позицию программы в результатах запроса.

Рассмотрим программу на Python, использующую DB-API для запроса к базе данных. Сначала импортируем модуль базы данных, используя API подключения из этого модуля. Чтобы открыть соединение с базой данных, используем функцию connect и передаем в нее параметры: название базы данных, имя пользователя и пароль. Функция connect возвращает объект-подключение. На базе объекта-подключения создаем объект-курсор. Курсор используем для выполнения запросов и получения результатов. После выполнения запросов через курсор также получаем результаты запроса. Наконец, когда система закончила выполнять запросы, нужно освободить ресурсы, закрыв соединение. Помните, что нужно обязательно закрывать соединения, чтобы лишние подключения не расходовали ресурсы.

API ibm\_db предоставляет множество полезных функций на Python для доступа к базе данных IBM Data Server и управления ей. В том числе существуют функции для подключения к базе данных, подготовки и отправки SQL-запросов, получения строк из результирующей выборки, вызова хранимых процедур, выполнения и отката транзакций, обработки ошибок и получения метаданных. API ibm\_db использует драйвер IBM Data Server для ODBC и API CLI для подключения к IBM, DB2 и Informix.



Мы импортируем библиотеку ibm\_db в приложение на Python.



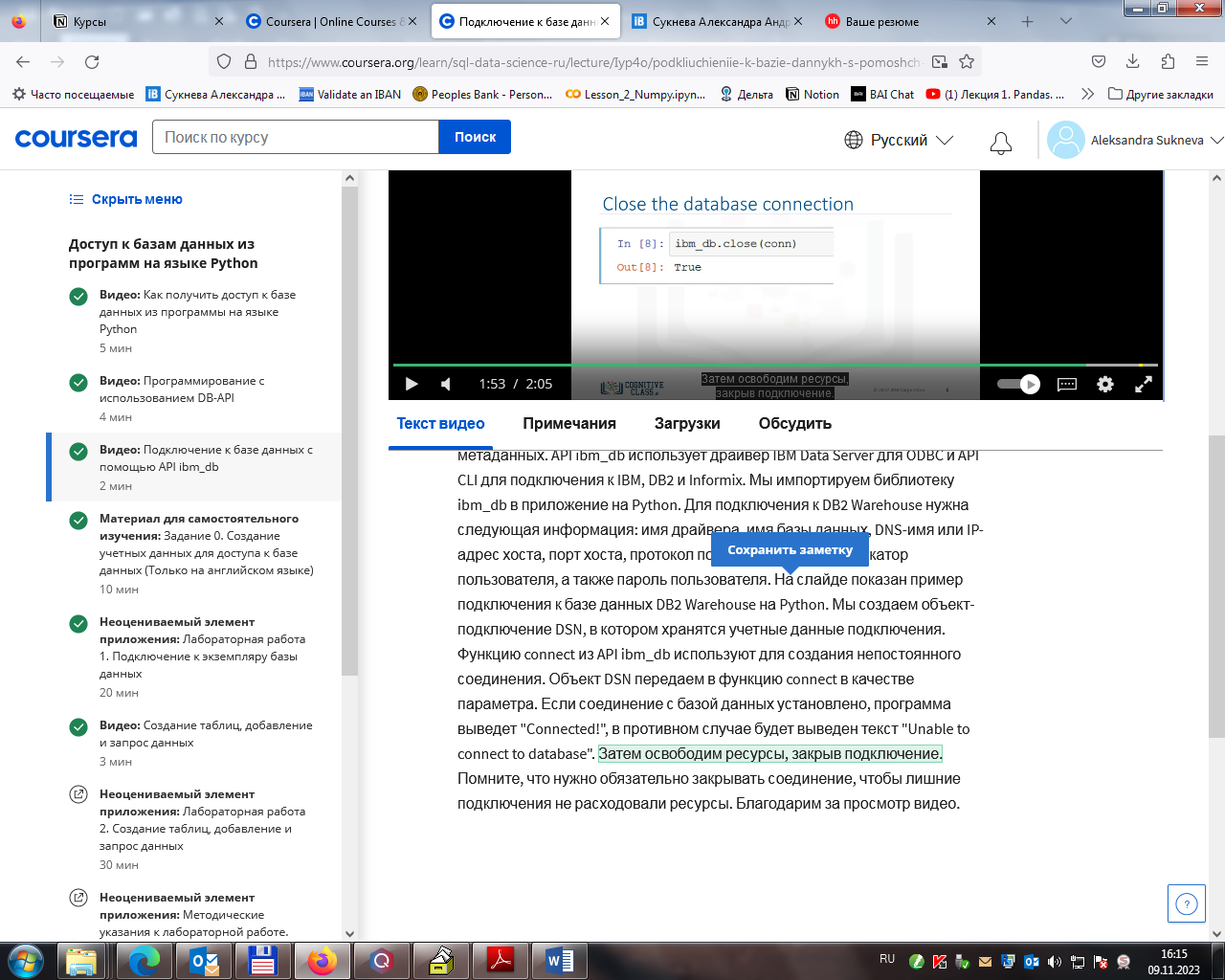
Для подключения к DB2 Warehouse нужна следующая информация: имя драйвера, имя базы данных, DNS-имя или IP-адрес хоста, порт хоста, протокол подключения, идентификатор пользователя, а также пароль пользователя.



На слайде показан пример подключения к базе данных DB2 Warehouse на Python.

Мы создаем объект-подключение DSN, в котором хранятся учетные данные подключения. Функцию connect из API ibm\_db используют для создания непостоянного соединения. Объект DSN передаем в функцию connect в качестве параметра. Если соединение с базой данных установлено, программа выведет "Connected!", в противном случае будет выведен текст "Unable to connect to database".

Затем освободим ресурсы, закрыв подключение. Помните, что нужно обязательно закрывать соединение, чтобы лишние подключения не расходовали ресурсы.



[Щелкните здесь,](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DB0201EN-SkillsNetwork/labs/Module%205/LAB-0v6_Create_Database_Credentials.md.html)чтобы открыть описание «Создание учетных данных для доступа к базе данных»