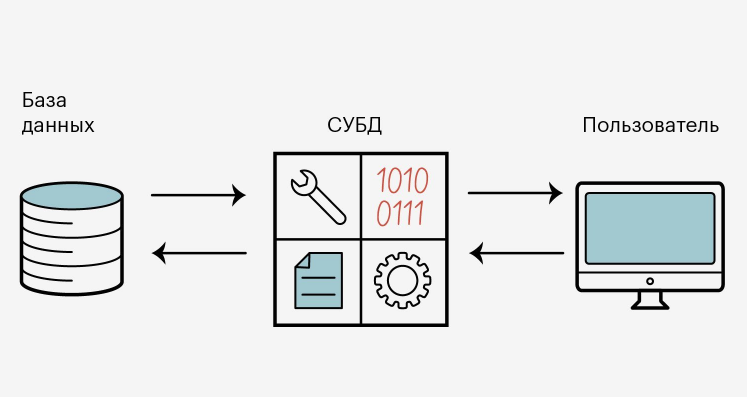
**Лекция 2: Особенности языковых средств управления и обеспечения безопасности данных в реляционных СУБД**

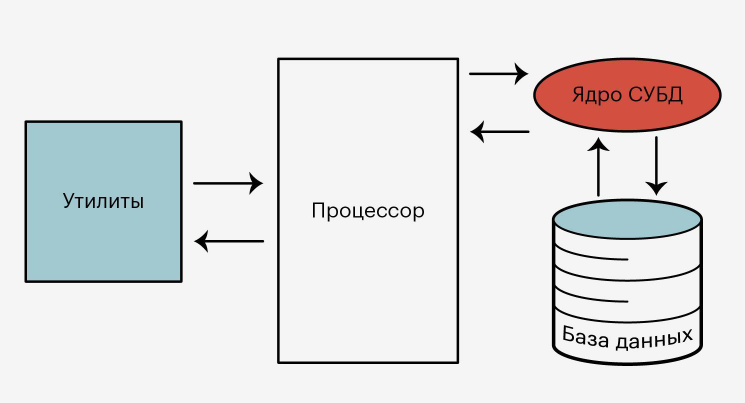
**Подготовили:** Сенокосов Владислав Владимирович, Мызников Сергей Анатольевич, Сергей СергеевЕвгеньевич

**База данных** — это набор упорядоченных и структурированных данных, которые хранятся в электронном виде на выделенных устройствах – серверах. Управление базой данных происходит с помощью СУБД. Обычно структура базы данных состоит из множества таблиц и установленными между ними связями по определенным правилам.

**Система управления базами данных (СУБД)** — это набор инструментов, которые позволяют удобно управлять базами данных: удалять, добавлять, фильтровать и находить элементы, менять их структуру и создавать резервные копии. То есть, можно сказать, что СУБД выступает в роли интерфейса между пользователем и самим сервером.



В качестве главных элементов СУБД выступают следующие: ядро, процессор, программные средства и базы данных. Поговорим о каждом из них подробнее.



**Ядро.** Отвечает за работу всей системы в целом. Через него проходят все процессы обработки данных и их хранения. Ядро следит за всеми изменениями баз данных и фиксирует их.

**Процессор, или компилятор.** Занимается обработкой запросов от пользователей. Главная его задача — преобразовать SQL-запрос в понятные для компьютера команды, а затем вернуть результаты.

**Программные средства, или утилиты.**Нужны, чтобы пользователь мог вводить запросы, а администраторы могли настраивать доступ и другие необходимые параметры.

**Базы данных.** Место, где хранятся данные в упорядоченном, а иногда и в зашифрованном виде. Базы могут различаться по структуре и типам представления данных.

### ****По расположению баз данных****

СУБД бывают локальные и распределённые.

**Локальные.** Это когда всё содержимое базы данных располагается на одном компьютере — обычно на сервере компании.

**Распределённые.** Это когда база данных частично находится на разных компьютерах — например, в облаке.

Современные СУБД могут быть одновременно локальными и распределёнными.

### ****По языку запросов****

СУБД поддерживают язык структурированных запросов и неструктурированных.

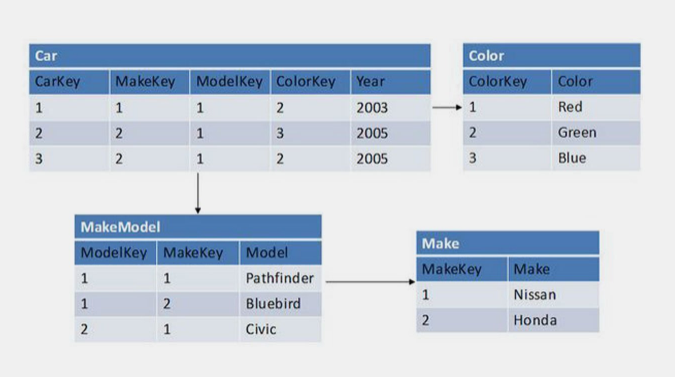
**SQL.** Это язык для создания структурированных запросов к базам данных. Относится к разделу декларативного программирования.

**NoSQL.** Это язык для запросов, который основан на другом языке программирования — например, Python или JavaScript. СУБД с NoSQL обычно используют при работе с большими данными.

### ****По структуре и организации данных****

### **Существуют различные типы баз данных, которые используются в различных сферах жизнедеятельности. Наиболее распространенные следующие типы:**

**Реляционные.** Данные в виде таблиц, которые связаны друг с другом через сквозные параметры. Такая архитектура обеспечивает построчное хранение данных и нужна для создания строгой структуры. Ещё одна их особенность: за одно обращение к базе пользователь сможет получить лишь небольшое число элементов.



Каждая строка имеет уникальный идентификатор, или ключ. Поэтому найти нужные данные и связать их между собой в такой базе данных легко.

Реляционные базы данных используют, когда объём данных не превышает нескольких терабайт. Это делает их подходящими практически любому проекту.

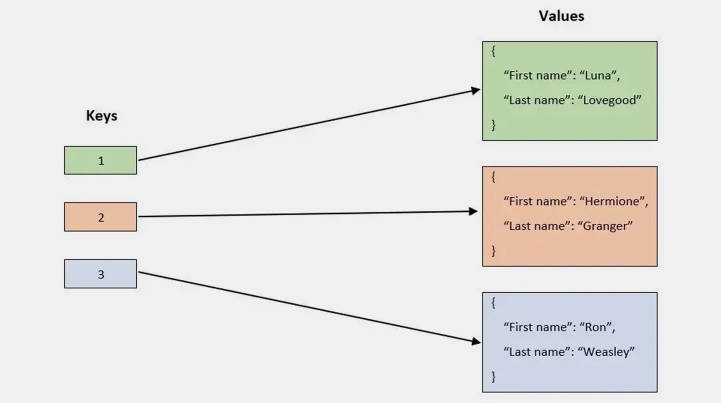
Самые популярные среди реляционных баз данных — PostgreSQL, Microsoft SQL Server, MySQL, Oracle.

Реляционные базы данных применяют, когда важны следующие характеристики:

* **Транзакционность.** Для выполнения операции нужно сделать несколько запросов к базе данных. Это может быть, например, перевод денег с одной карты на другую через приложение банка. Под транзакцией понимается минимальная неделимая операция, которая либо выполняется, либо нет.
* **Частые изменения данных.** Так как в реляционных базах данных строгая структура.
* **Поиск по индексам.** Каждый элемент такой базы данных — это строка, а искать элементы по строкам удобно.
* **Запросы небольшого количества записей за раз.** Реляционные базы данных работают как раз по этому принципу.
* **Объём данных не превышает нескольких терабайт.** Если будет больше, такая база данных будет работать медленнее.

**Ключ — значение.**Каждый элемент базы данных использует для хранения уникальный идентификатор, который состоит из ключа и значения. Этот способ хранения похож на тип данных «словарь» в языках программирования.

Базы данных типа «ключ — значение» имеют вид хеш-таблиц, в которых у каждой записи — только один индекс. При этом жёстких ограничений по структуре элементов нет.

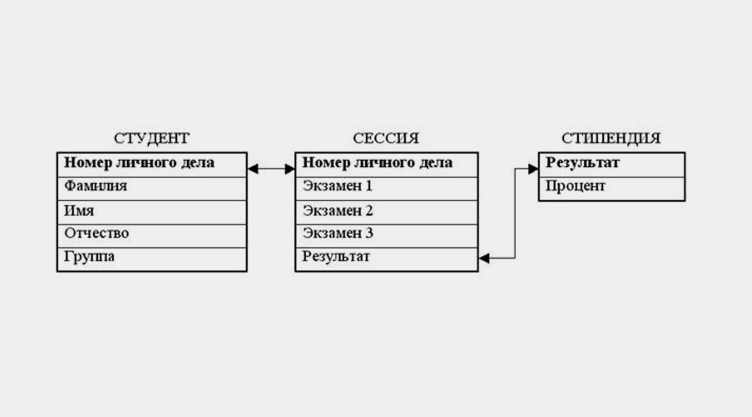


Такие базы данных используют по принципу логов, когда новые записи заносят в конец хранилища. Удаляют элементы с помощью добавления специальной записи.

Ещё несколько способов использования баз данных типа «ключ — значение» — очереди, кэш и логирование.

Популярные базы данных типа «ключ — значение» — Redis, DynamoDB и Aerospike.

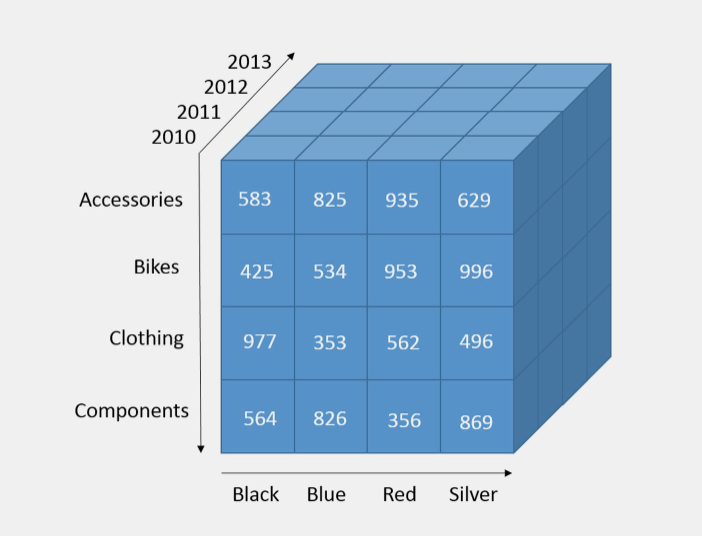
**Документные.** В отличие от реляционных баз данных, документно-ориентированные хранят информацию в виде «документов», а не таблиц и строк. Главное преимущество документных баз данных — возможность хранить информацию без строгого ограничения по структуре.



**Графовые.**В них элементы имеют взаимосвязи в виде графа, где у каждого его узла есть множество связей с другими узлами. Обычно эти базы данных используют в соцсетях или рекомендательных сервисах.

Каждый узел графовой базы данных содержит в себе данные, а чтобы составить взаимосвязи между узлами, используют рёбра. В них хранятся начальный и конечный узлы, направление и тип. Рёбра описывают взаимосвязи между двумя узлами — например, «родитель — потомок», — а также действия над ними.

**Колоночные (Хранилища).** В них данные хранятся последовательно в виде одной колонки, и предполагается, что на одном и том же месте в каждой колонке хранятся элементы, которые относятся к одной строке. Это позволяет эффективно сжимать данные и анализировать их — находить сумму, количество или среднее значение. Выборка данных производится по срезам, где сам срез производится по интересующей нас оси. На основе такой возможности можно осуществлять анализ данных и составлять удобные для логистики отчеты.



## **Популярные СУБД**

СУБД существует много, но часто программисты пользуются семью самыми популярными — PostgreSQL, Microsoft SQL Server (прости господи), MySQL, SQLite, MongoDB, Redis и Oracle.

**Вопрос 1: Определение управляющих структур.**

Под управляющими структурами в реляционной базе данных понимаются функции языка или архитектурные особенности базы данных, позволяющие пользователю осуществлять операции выборки, добавления, изменения, удаления, слияния, создания объекта и т.п. Под структурой понимается всякая конструкция языка SQL использующая при написании запроса.

Наиболее распространенные управляющие структуры, выступающие в роли функций языка:

1. **SELECT:** Используется для выбора данных из базы данных.

Пример: SELECT \* FROM table\_name;

1. **INSERT:** Используется для вставки новых записей в таблицу.

Пример: INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...);

1. **UPDATE:** Используется для обновления существующих записей в таблице.

Пример: UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;

1. **DELETE:** Используется для удаления записей из таблицы.

Пример: DELETE FROM table\_name WHERE condition;

1. **CREATE TABLE**: Используется для создания новой таблицы в базе данных.

Пример: CREATE TABLE table\_name (column1 datatype, column2 datatype, ...);

1. **ALTER TABLE**: Используется для изменения структуры существующей таблицы (добавление, удаление или изменение столбцов).

Пример: ALTER TABLE table\_name ADD column\_name datatype;

1. **DROP TABLE:** Используется для удаления существующей таблицы из базы данных. Пример: DROP TABLE table\_name;
2. **CREATE INDEX:** Используется для создания индекса в таблице для ускорения поиска данных.

Пример: CREATE INDEX index\_name ON table\_name (column1, column2, ...);

1. **DROP INDEX:** Используется для удаления индекса из таблицы.

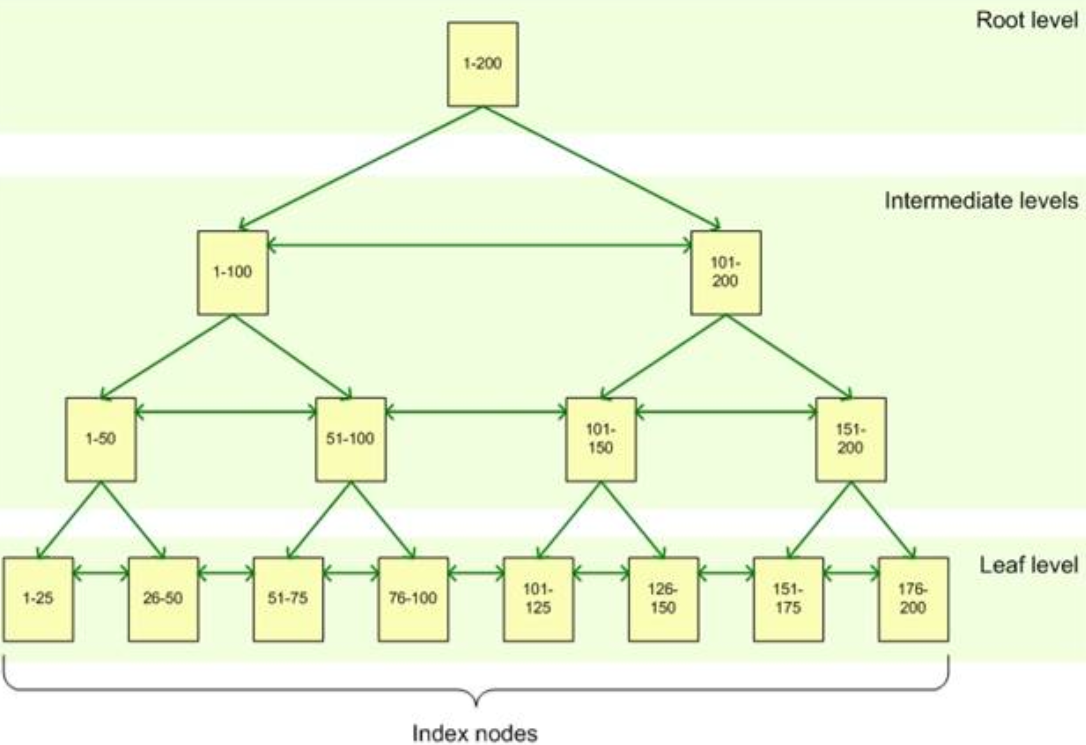
Пример: DROP INDEX index\_name;

1. **WHERE:** Используется для фильтрации результатов в командах SELECT, UPDATE и DELETE.

Пример: SELECT \* FROM table\_name WHERE condition;

Если рассматривать управляющие структуры, которые принадлежат к архитектурным особенностям той или иной базы данных, то выделяют следующие:

1. **Таблицы:** Они представляют собой основные структурные элементы базы данных, где данные хранятся в виде строк и столбцов. Каждая таблица имеет определенное количество столбцов, которые обычно представляют собой атрибуты, и строки, которые представляют собой отдельные записи.
2. **Индексы:** Индексы используются для ускорения поиска и извлечения данных из таблицы. Они представляют собой отдельные структуры данных, которые содержат упорядоченные значения столбца и указатели на соответствующие строки в таблице. Выделяют два основных типов индексов – кластерный и некластерный. Ввиду того, что индекс сам по себе строится как сбалансированное дерево, где происходит поиск по ключам, то в случае если листовые узлы дерева будут хранить данные, то такой индекс называется кластерным, а в случае если они содержат указатели на реальные данные – некластерным.



1. **Первичные ключи:** Первичный ключ - это уникальный идентификатор каждой записи в таблице. Он гарантирует уникальность идентификации записей в таблице и обеспечивает целостность данных.
2. **Внешние ключи:** Внешний ключ - это атрибут или набор атрибутов, которые связывают одну таблицу с другой. Они обеспечивают ссылочную целостность данных, определяя связи между различными таблицами в базе данных.
3. **Представления:** Представления представляют собой виртуальные таблицы, которые состоят из данных, извлеченных из одной или нескольких реальных таблиц. Они предоставляют удобный способ просмотра данных из базы данных без необходимости изменения базовой структуры данных.
4. **Хранимые процедуры и функции:** представляет собой предварительно скомпилированную и сохраненную в базе данных подпрограмму, которая может выполняться многократно при вызове из различных программ или запросов.

**Вопрос 2: Авторизация доступа к отношениям и их полям.**

Для обеспечения защиты информация, хранящейся в базе данных используются методы разграничения прав доступа, а также механизмы аутентификации и авторизации.

Авторизация доступа к отношениям и их полям является важной частью обеспечения безопасности данных в реляционных базах данных. Этот процесс предоставляет контроль над тем, кто и как может получить доступ к определенным таблицам (отношениям) и их полям в базе данных.

В реляционных СУБД авторизация обычно осуществляется с помощью управления ролями и привилегиями.

**Роли** — это группы пользователей с общими правами доступа, а привилегии определяют уровень доступа к конкретным объектам базы данных.

Основная идея такого подхода состоит в том, что по отношению к любому отношению БД и любому столбцу отношения вводится предопределенный набор привилегий. С каждой транзакцией неявно связывается идентификатор пользователя, от имени которого она выполняется (способы связи и идентификации пользователей не фиксируются в языке и определяются в реализации).

Технически передача привилегий осуществляется при выполнении оператора SQL GRANT. Существует также привилегия изъятия всех или части привилегий у пользователя, которому они ранее были переданы. Эта привилегия также может передаваться. Технически изъятие привилегий происходит при выполнении оператора SQL REVOKE.

Пример SQL запроса для предоставления привилегии SELECT на определенные поля таблицы:

GRANT SELECT ON table\_name(column1, column2) TO user\_name;

Этот запрос предоставляет пользователю user\_name право на выполнение операции SELECT только на указанные поля column1 и column2 таблицы table\_name.

**Вопрос 3: Точки сохранения и откаты транзакции.**

Для сохранения целостности информации и функционирования БД используются механизмы точек сохранения и откаты транзакций.

**Точки сохранения (далее сокращенно ТС)** – это внутренний механизм СУБД, привязывающий любые изменения в БД к конкретному моменту времени в рамках транзакции, и позволяющий при необходимости отменить все изменения, выполненные после установки данной ТС (так называемый откат до ТС).

**Транзакция** – это минимальная неделимая (атомарная) операция, осуществляемая в базе данных, она может быть выполнена либо полностью, либо не выполнена вовсе. Атомарность подразумевает, что операция не состоит не из каких других подзадач.  
  
По умолчанию сервер использует глобальную ТС для осуществления отката транзакции. Данная ТС устанавливается автоматически при старте транзакции и является первой в ее контексте. Когда инициируется откат транзакции, то все изменения, выполненные в ее контексте, откатываются с помощью глобальной ТС транзакции, после чего данная транзакция подтверждается в TIP (Transaction Inventory Page). Но если количество изменений, выполненных в контексте транзакции, становится слишком велико (порядка 104 – 106 записей), то хранение списков отката получается дорогостоящим.

Помимо использования ТС для отката транзакции, сервер также использует их для обработки исключений. Каждый оператор обрамляется ТС-фреймом, позволяющим откатить именно этот оператор, не затрагивая предыдущие. Это гарантирует, что оператор либо выполнился успешно, либо все его изменения автоматически отменены и инициирована соответствующая ошибка. Для обработки исключений каждый BEGIN…END блок также обрамляется фреймом, позволяющим отменить все изменения, выполненные данным блоком.

**Как это устроено?**

ТС представляет собой структуру данных, размещенную в динамической памяти сервера (в пуле транзакции) и имеющую уникальный числовой идентификатор. К каждой ТС привязан список действий, совершенных в ее контексте (так называемый журнал отмены). В пределах транзакции ТС образуют стек и, следовательно, их откат всегда возможен только последовательно. Фрагменты журнала отмены распределены между ТС, которые инкрементно хранят историю всех изменений, выполненных в контексте транзакции.  
  
ТС, активная на момент изменения какой-либо записи, называется текущей. Информация об изменении записи помещается в журнал отмены текущей ТС. В случае инициации отката до ТС журнал отмены раскручивается в обратную сторону, реконструируя запись к виду, в каком она существовала на момент установки данной ТС. После реконструкции всех измененных записей ТС обычно удаляется из контекста транзакции.

Существует несколько событий, которые приводят к созданию сервером системных (то есть неуправляемых пользователем) ТС:

1. Выполнение любого клиентского SQL-запроса. Как уже было сказано выше, это делается для обеспечения атомарности данного запроса, то есть при возниковении любого исключения во время выполнения запроса, изменения, внесенные им в БД, всегда будут отменены. По окончании выполнения запроса ТС автоматически удаляется.
2. Выполнение BEGIN…END блока в случае, если этот блок содержит обработчик ошибок (WHEN-блок), либо если любой из вышестоящих блоков содержит обработчик ошибок. В этом случае каждый оператор BEGIN устанавливает точку сохранения и соответствующий ему оператор END удаляет ее.

## **Примеры**

Установление точки сохранения и затем отмена действия всех команд, выполненных после установленной точки:

BEGIN;

INSERT INTO table1 VALUES (1);

SAVEPOINT my\_savepoint;

INSERT INTO table1 VALUES (2);

ROLLBACK TO SAVEPOINT my\_savepoint;

INSERT INTO table1 VALUES (3);

COMMIT;

Показанная транзакция вставит в таблицу значения 1 и 3, но не 2.

Этот пример показывает, как установить и затем уничтожить точку сохранения:

BEGIN;

INSERT INTO table1 VALUES (3);

SAVEPOINT my\_savepoint;

INSERT INTO table1 VALUES (4);

RELEASE SAVEPOINT my\_savepoint;

COMMIT;

Данная транзакция вставит значения 3 и 4.

**Вопрос 4:** Встроенный SQL. Динамический SQL. Язык SQL в коммерческих реализациях.

**Встроенный SQL**

Язык SQL, как мы уже видели, предназначен для организации доступа к базам данных. При этом предполагается, что доступ к БД может быть осуществлен в двух режимах: в интерактивном режиме и в режиме выполнения прикладных программ (приложений).

Эта двойственность SQL создает ряд преимуществ:

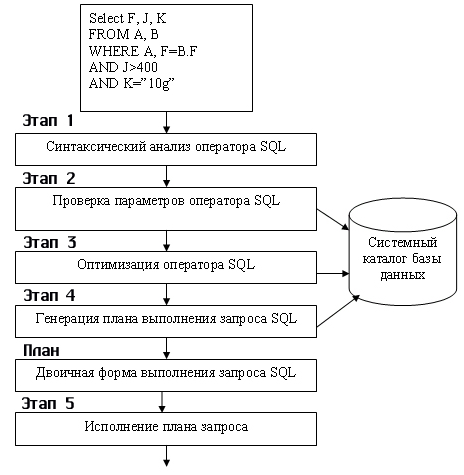
* Все возможности интерактивного языка запросов доступны и в прикладном программировании.
* Можно в интерактивном режиме отладить основные алгоритмы обработки информации, которые в дальнейшем могут быть готовыми вставлены в работающие приложения.

SQL действительно является языком по работе с базами данных, но в явном виде он не является языком программирования. В нем отсутствуют традиционные операторы, организующие циклы, позволяющие объявить и использовать внутренние переменные, организовать анализ некоторых условий и возможность изменения хода программы в зависимости от выполненного условия. В общем случае можно назвать SQL подъязыком, который служит исключительно для управления базами данных. Для создания приложений, настоящих программ необходимо использовать другие, базовые языки программирования, в которые операторы языка SQL будут встраиваться.

Базовыми языками программирования могут быть языки С++, COBOL, PL/1, Pascal, PHP, Java. Существуют два способа применения SQL в прикладных программах:

* *Встроенный SQL.*При таком подходе операторы SQL встраиваются непосредственно в исходный текст программы на базовом языке. При компиляции программы со встроенными операторами SQL используется специальный препроцессор SQL, который преобразует исходный текст в исполняемую программу.
* *Интерфейс программирования приложений (API application program interface).*При использовании данного метода прикладная программа взаимодействует с СУБД путем применения специальных функций. Вызывая эти функции, программа передает СУБД операторы SQL и получает обратно результаты запросов. В этом случае не требуется специализированный препроцессор.

Процесс выполнения операторов SQL может быть условно разделен на 5 этапов

****

1. На первом этапе выполняется синтаксический анализ оператора SQL. На этом этапе проверяется корректность записи SQL-оператора в соответствии с правилами синтаксиса.
2. На этом этапе проверяется корректность параметров оператора SQL: имен отношений, имен полей данных, привилегий пользователя по работе с указанными объектами. Здесь обнаруживаются семантические ошибки.
3. На этом этапе проводится оптимизация запроса. СУБД проводит разделение целостного запроса на ряд минимальных операций иоптимизирует последовательность их выполнения с точки зрения стоимости выполнения запроса. На этом этапе строится несколько планов выполнения запроса и выбирается из них один — оптимальный для данного состояния БД.
4. На четвертом этапе СУБД генерирует двоичную версию оптимального плана запроса, подготовленного на этапе 3. Двоичный план выполнения запроса в СУБД фактически является эквивалентом объектного кода программы.
5. И наконец, только на пятом этапе СУБД реализует (выполняет) разработанный план, тем самым выполняя оператор SQL.

Следует отметить, что перечисленные этапы отличаются по числу обращений к БД и по процессорному времени, требуемому для их выполнения. Синтаксический анализ проводится очень быстро, он не требует обращения к системным каталогам БД. Семантический анализ уже требует работы с базой метаданных, то есть с системными каталогами БД, поэтому при выполнении этого этапа происходит обращение к системному каталогу и серьезная работа с ним. Этап, связанный с оптимизаций плана запроса, требует работы не только с системным каталогом, но и со статистической информацией о БД, которая характеризует текущее состояние всех отношений, используемых в запросе, их физическое расположение на страницах и сегментах внешней памяти. В силу указанных причин этап оптимизации наиболее трудоемкий и длительный в процессе выполнения запроса. Однако если не проводить этап оптимизации, то стоимость (время) выполнения неоптимизированного запроса может в несколько раз превысить стоимость оптимизированного запроса. Время, потраченное на оптимизацию запроса, с лихвой компенсирует затраты на выполнение неоптимизированного запроса.

**Особенности встроенного SQL**

При объединении операторов SQL с базовым языком программирования должны соблюдаться следующие принципы:

* Операторы SQL включаются непосредственно в текст программы на исходном языке программирования. Исходная программа поступает на вход препроцессора SQL, который компилирует операторы SQL.
* Встроенные операторы SQL могут ссылаться на переменные базового языка программирования.
* Встроенные операторы SQL получают результаты SQL-запросов с помощью переменных базового языка программирования.
* Для присвоения неопределенных значений (NULL) атрибутам отношений БД используются специальные функции.
* Для обеспечения построчной обработки результатов запросов во встроенный SQL добавляются несколько новых операторов, которые отсутствуют в интерактивном SQL.

## **Что такое динамический SQL?**

**Динамический**[**SQL**](https://coderlessons.com/tutorials/bazy-dannykh/osnovy-sql/osnovy-sql) — это методология программирования для генерации и выполнения операторов во время выполнения. Он в основном используется для написания универсальных и гибких программ, в которых операторы SQL будут создаваться и выполняться во время выполнения в соответствии с требованиями.

## Способы написания динамического SQL

PL / SQL предоставляет два способа написания динамического SQL

1. NDS — собственный динамический SQL
2. DBMS\_SQL

## **NDS (собственный динамический SQL)** — немедленное выполнение

Собственный динамический SQL — это более простой способ написания динамического SQL. Он использует команду «EXECUTE IMMEDIATE» для создания и выполнения SQL во время выполнения. Но чтобы использовать этот способ, тип данных и номер переменной, которая будет использоваться во время выполнения, должны быть известны заранее. Это также дает лучшую производительность и меньшую сложность по сравнению с DBMS\_SQL.

## **DBMS\_SQL** для динамического SQL

PL / SQL предоставляет пакет DBMS\_SQL, который позволяет вам работать с динамическим SQL. Процесс создания и выполнения динамического SQL содержит следующий процесс.

* **OPEN CURSOR** : динамический SQL будет выполняться так же, как курсор. Таким образом, чтобы выполнить инструкцию SQL, мы должны открыть курсор.
* **PARSE SQL** : следующий шаг — анализ динамического SQL. Этот процесс будет просто проверять синтаксис и держать запрос готовым к выполнению.
* Переменные **BIND Значения** : Следующим шагом является назначение значений для переменных связывания, если таковые имеются.
* **ОПРЕДЕЛИТЬ КОЛОННУ** : Следующий шаг — определить столбец, используя его относительные позиции в операторе выбора.
* **ВЫПОЛНИТЬ** : Следующий шаг — выполнить проанализированный запрос.
* **FETCH VALUES** : Следующий шаг — выборка выполненных значений.
* **ЗАКРЫТЬ КУРСОР** : После того, как результаты получены, курсор должен быть закрыт.

**Язык SQL в коммерческих реализациях. Стандартизация SQL.**

Язык SQL (Structured Query Language) широко используется в коммерческих реализациях систем управления базами данных (СУБД) для выполнения запросов к данным, управления данными и выполнения других операций с базой данных. Некоторые из наиболее популярных коммерческих СУБД, которые используют SQL, включают в себя:

**Oracle Database:** Oracle Database является одной из наиболее распространенных коммерческих СУБД и широко используется в корпоративных средах для хранения и управления данными с использованием языка SQL.

**Microsoft SQL Server:** Microsoft SQL Server предоставляет мощные инструменты для управления данными и поддерживает язык SQL для выполнения запросов и операций с данными. Он широко используется в среде Windows.

**IBM Db2:** IBM Db2 представляет собой семейство продуктов СУБД, которые поддерживают SQL и используются в различных сферах, включая корпоративные приложения, банковское дело и аналитику данных.

**SAP HANA:** SAP HANA - это платформа для обработки и анализа данных, которая также использует SQL для выполнения запросов к данным и выполнения операций с базой данных.

Стандартизация SQL происходит через процесс, управляемый международной организацией ANSI (American National Standards Institute) и международной организацией ISO (International Organization for Standardization). Основные версии стандарта SQL включают:

**SQL-86:** Оригинальный стандарт SQL, утвержденный ANSI в 1986 году.

**SQL-92:** Обновленный стандарт, утвержденный ANSI и ISO в 1992 году, который включил множество дополнений и улучшений.

**SQL:1999 (SQL-99):** Стандарт, опубликованный в 1999 году, включает расширения, такие как поддержка объектов, рекурсивные запросы и аналитические функции.

**SQL:2003 (SQL-2003):** Стандарт, опубликованный в 2003 году, включает еще больше расширений и улучшений, включая поддержку XML.

**SQL:2008 (SQL-2008):** Версия стандарта, опубликованная в 2008 году, включает дополнительные функции и расширения.

**SQL:2011 (SQL-2011):** Стандарт, опубликованный в 2011 году, который включает новые функции и корректировки.

**SQL:2016 (SQL-2016):** Версия стандарта, опубликованная в 2016 году, содержит еще больше новых функций и расширений.

**SQL:2019 (SQL-2019):** Стандарт, опубликованный в 2019 году, который включает новые возможности, такие как поддержка JSON и расширения для аналитики данных.

Хотя стандарт SQL определяет базовый язык запросов, различные коммерческие СУБД могут расширять язык SQL, включая свои собственные функции и возможности, чтобы удовлетворить свои конкретные требования и особенности.

**Вопрос 5:** Стандартизация SQL. Оптимизация производительности и характеристик доступа к базам данных.

Оптимизация производительности и характеристик доступа к базам данных - это ключевые аспекты при разработке и поддержке приложений, которые работают с данными. Это включает в себя оптимизацию запросов, индексирование, настройку конфигурации СУБД, кэширование и другие методы для обеспечения эффективного доступа к данным. Вот несколько методов оптимизации производительности и характеристик доступа к базам данных:

Анализ и оптимизация запросов: Один из наиболее важных аспектов оптимизации производительности баз данных - это анализ и оптимизация запросов SQL. Это включает в себя выбор правильных индексов, использование соответствующих операторов и функций SQL, избегание избыточных запросов и уменьшение количества операций с данными.

Индексирование: Создание правильных индексов на столбцах, используемых для поиска, сортировки и объединения данных, может значительно улучшить производительность запросов. Однако следует избегать создания избыточных индексов, так как это может повлиять на производительность вставки, обновления и удаления данных.

Настройка конфигурации СУБД: Настройка параметров и конфигурационных файлов СУБД может существенно повлиять на производительность. Это включает в себя настройку параметров памяти, размеров буферов, количества одновременных соединений и других параметров, чтобы оптимизировать использование ресурсов и улучшить производительность.

Кэширование: Использование кэширования для хранения результатов запросов или промежуточных данных может существенно ускорить доступ к данным и снизить нагрузку на базу данных. Кэширование может быть реализовано на уровне приложения или в самой базе данных, используя кэш запросов или кэш планов выполнения.

Фрагментация таблиц: Оптимизация фрагментации таблиц может повысить производительность запросов. Это включает в себя периодическое выполнение операций оптимизации и перестроения индексов, чтобы минимизировать фрагментацию данных на диске.

Горизонтальное и вертикальное разделение данных: Разделение данных на разные таблицы или разные части таблицы может помочь улучшить производительность запросов. Горизонтальное разделение (partitioning) данных может ускорить операции поиска и агрегирования данных, а вертикальное разделение (vertical partitioning) - уменьшить количество данных, которые необходимо обрабатывать в каждом запросе.

Оптимизация сетевого взаимодействия: Уменьшение объема передаваемых данных между приложением и базой данных, например, путем использования кэширования на уровне приложения или сжатия данных, также может улучшить производительность и характеристики доступа к базе данных.

Каждая из этих методик может быть эффективной при оптимизации производительности и характеристик доступа к базам данных, и часто комбинируются в рамках комплексного подхода к оптимизации баз данных.