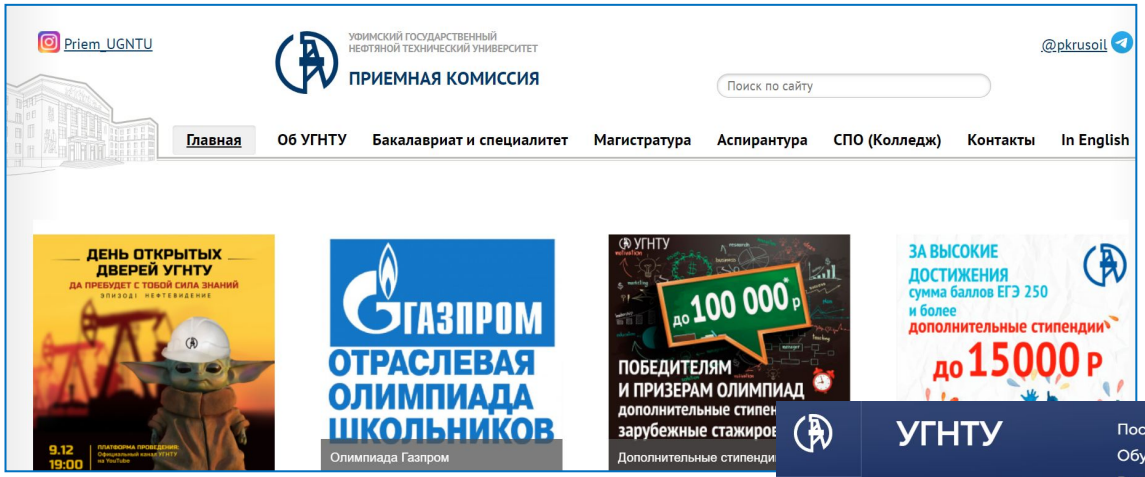
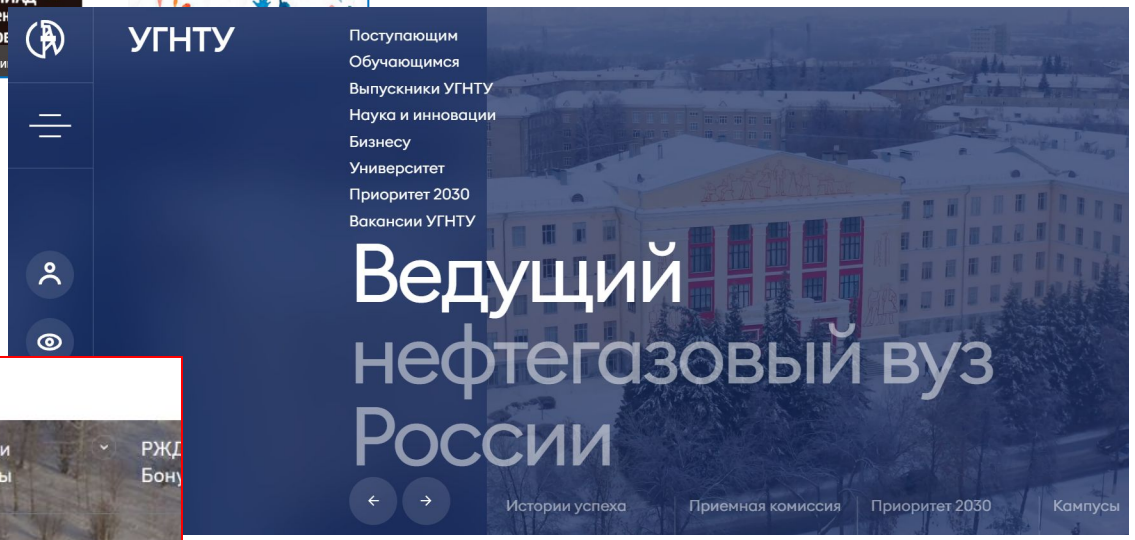
Базы данных

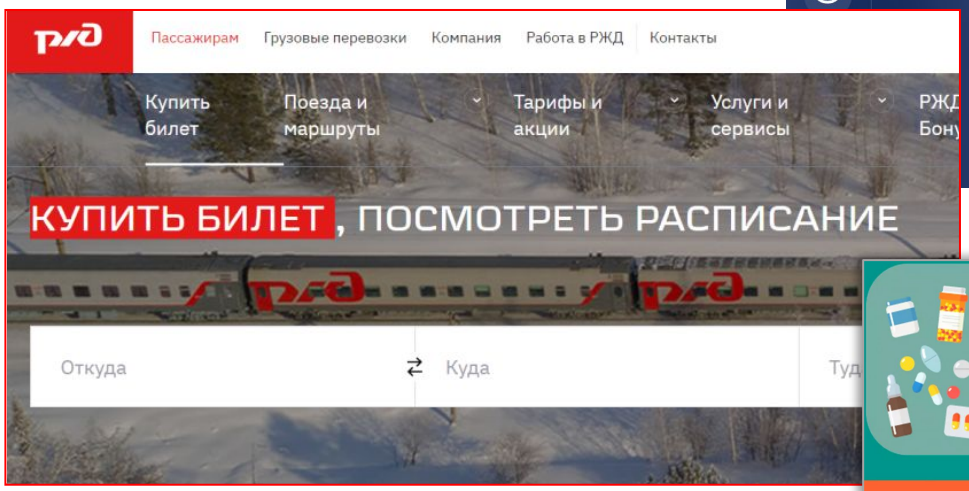
Содержание

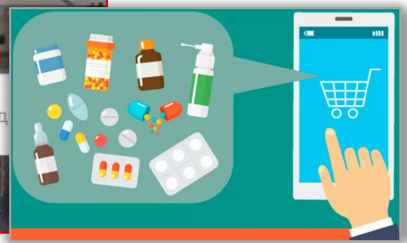
Зачем нужно изучать базы данных?

1) Практически в каждом приложение реализована БД

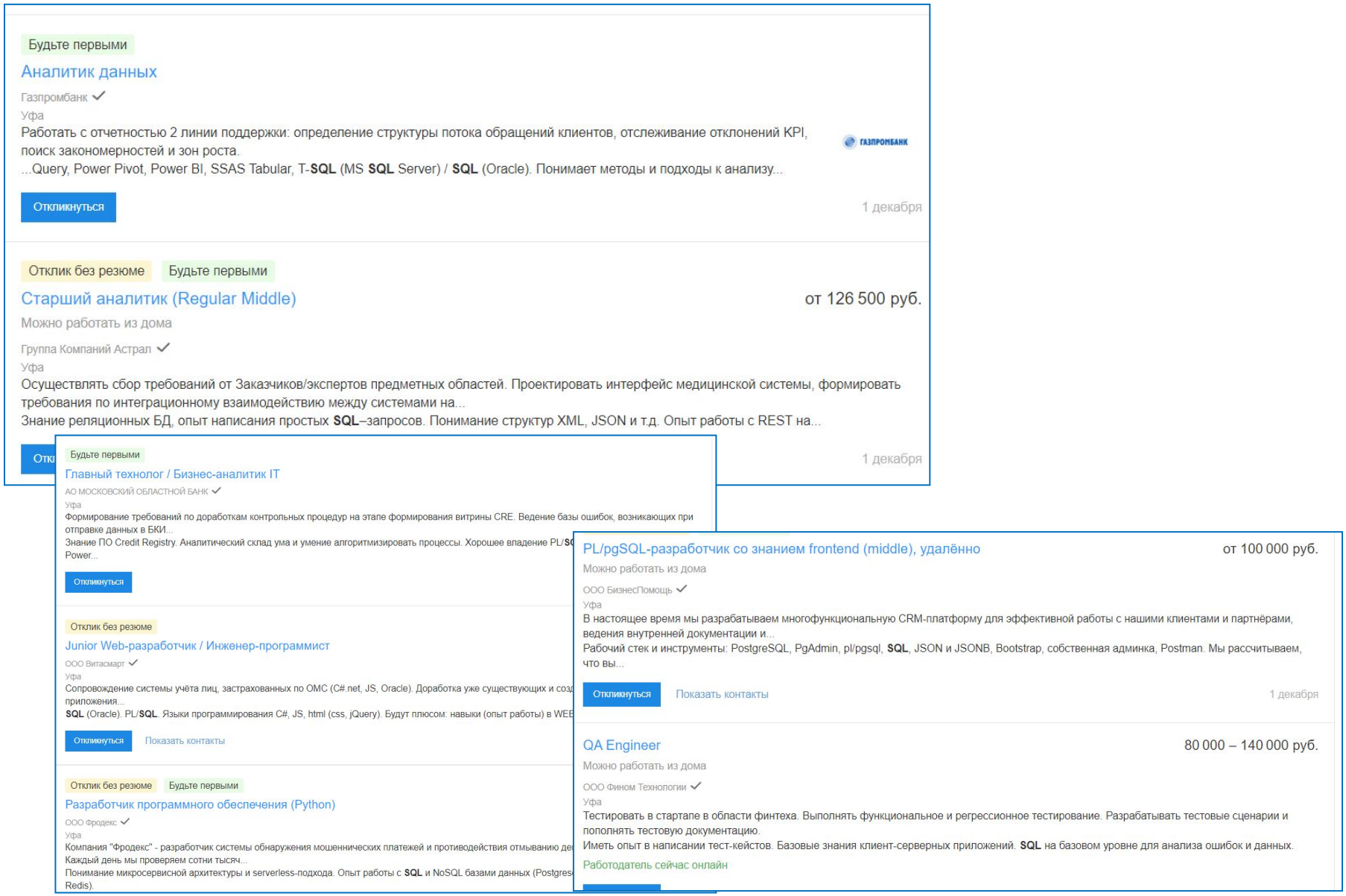








2) Почти в каждой вакансии упоминается SQL (Structured query language — «язык структурированных запросов»)



Будьте первыми

Аналитик данных

Газпромбанк

Уфа

Работать с отчетностью 2 линии поддержки: определение структуры потока обращений клиентов, отслеживание отклонений KPI, поиск закономерностей и зон роста.

...Query, Power Pivot, Power BI, SSAS Tabular, T-**SQL** (MS **SQL** Server) / **SQL** (Oracle). Понимает методы и подходы к анализу...

Откликнуться

1 декабря

Отклик без резюме Будьте первыми

Старший аналитик (Regular Middle)

от 126 500 руб.

Можно работать из дома

Группа Компаний Астрал

Уфа

Осуществлять сбор требований от Заказчиков/экспертов предметных областей. Проектировать интерфейс медицинской системы, формировать требования по интеграционному взаимодействию между системами на...

Знание реляционных БД, опыт написания простых **SQL**–запросов. Понимание структур XML, JSON и т.д. Опыт работы с REST на...

1 декабря

Будьте первыми

Главный технолог / Бизнес-аналитик IT

АО МОСКОВСКИЙ ОБЛАСТНОЙ БАНК

Уфа

Формирование требований по доработкам контрольных процедур на этапе формирования витрины CRE. Ведение базы ошибок, возникающих при отправке данных в БКИ...

Знание ПО Credit Registry. Аналитический склад ума и умение алгоритмизировать процессы. Хорошее владение PL/**SQ** Power...

Откликнуться

Отклик без резюме

Junior Web-разработчик / Инженер-программист

ООО Витасмарт

Уфа

Сопровождение системы учёта лиц, застрахованных по ОМС (C#.net, JS, Oracle). Доработка уже существующих и созд приложения...

SQL (Oracle). PL/**SQL**. Языки программирования C#, JS, html (css, jQuery). Будут плюсом: навыки (опыт работы) в WEB

Откликнуться Показать контакты

Отклик без резюме Будьте первыми

Разработчик программного обеспечения (Python)

ООО Фродекс

Уфа

Компания "Фродекс" - разработчик системы обнаружения мошеннических платежей и противодействия отмыванию де

Каждый день мы проверяем сотни тысяч...

Понимание микросервисной архитектуры и serverless-подхода. Опыт работы с **SQL** и NoSQL базами данных (Postgres Redis).

PL/pgSQL-разработчик со знанием frontend (middle), удалённо

от 100 000 руб.

Можно работать из дома

ООО БизнесПомощь

Уфа

В настоящее время мы разрабатываем многофункциональную CRM-платформу для эффективной работы с нашими клиентами и партнёрами, ведения внутренней документации и...

Рабочий стек и инструменты: PostgreSQL, PgAdmin, pl/pgsql, **SQL**, JSON и JSONB, Bootstrap, собственная админка, Postman. Мы рассчитываем, что вы...

Откликнуться Показать контакты

1 декабря

QA Engineer

80 000 – 140 000 руб.

Можно работать из дома

ООО Фином Технологии

Уфа

Тестировать в стартапе в области финтеха. Выполнять функциональное и регрессионное тестирование. Разрабатывать тестовые сценарии и пополнять тестовую документацию.

Иметь опыт в написании тест-кейстов. Базовые знания клиент-серверных приложений. **SQL** на базовом уровне для анализа ошибок и данных.

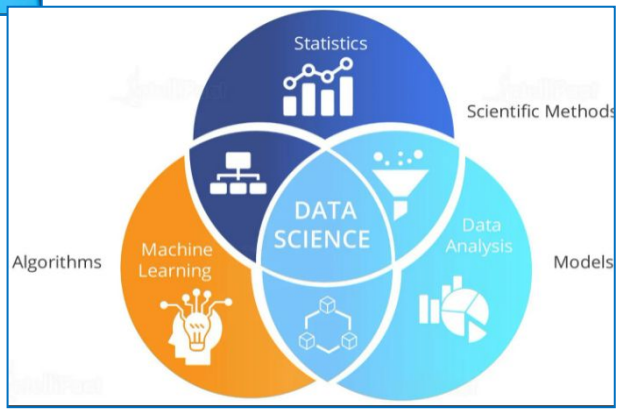
Работодатель сейчас онлайн

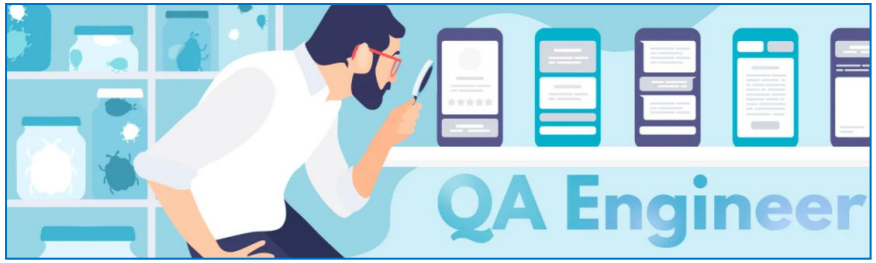
3) Базы данных позволяют хранить большие объемы данных

4) Возможность анализ накопленных данных.

Профессии, требующие знания БД

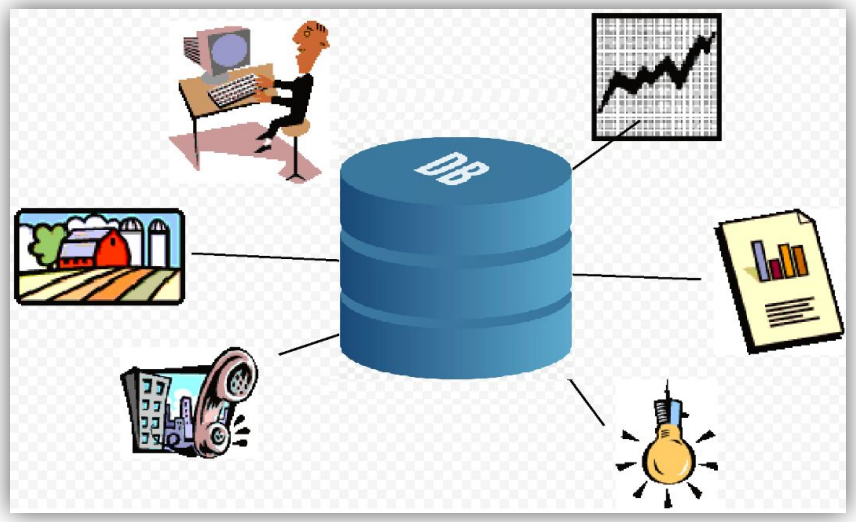
1. Разработчик программного обеспечения.
2. Аналитик данных (Data Analyst).
3. Data Scientist.
4. QA инженер (Quality Assurance обеспечение качества).
5. Project Manager.
6. DevOps.

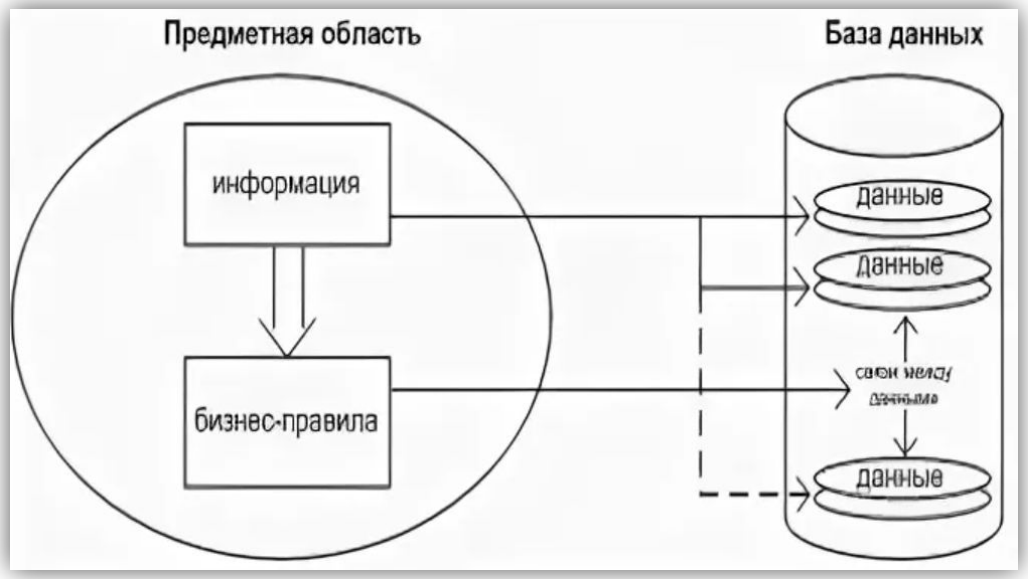




Основные понятия БД

- данные;
- предметная область;
- бизнес – правила.





Пример фрагмента предметной области «Муниципальная библиотека»

Правило: по абонементу, читатель может получить на дом не более пяти библиотечных документов на срок до 15 календарных дней. Через 15 дней после истечения срока пользования книгами и иными печатными материалами, если продление не осуществлялось, и через 30 дней, если продление было сделано, сотрудники библиотеки должны послать напоминание или сообщить читателю по телефону о необходимости их возврата в библиотеку.

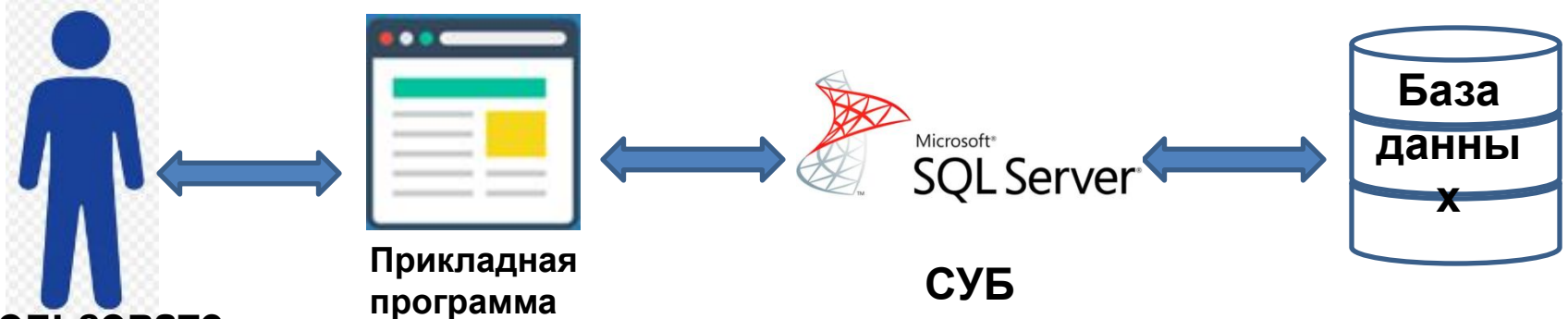
При регистрации читателя: паспортные и контактные данные.
При регистрации книги: входные данные о книге.
При выдачи книги: кому выдана, какая книга, дата выдачи и кол-во экземпляров.
При возврате книги: кто вернул, дата возврата и состояние книги

UML (Unified Modeling Language) диаграмма вариантов

Основные термины

База данных (БД) (Database, BD) – это организованная совокупность данных о некоторой предметной области, предназначенная для длительного хранения и постоянного применения.

Система управления базой данных (СУБД) (Database Management System, DBMS) – это программное обеспечение для работы с БД, т.е. совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.



Задачи, которые решает СУБД

1. Надежное хранение данных.
2. Быстрый поиск нужной информации.
3. Многопользовательский доступ.
4. Разграничение прав доступа.
5. Доступ к базе данных по сети.
6. Понятный для работы с данными язык SQL (Structured Query Language).

Например, нужно вывести фамилии и имена сотрудников проживающих в городе Уфа из таблицы Сотрудники:

SQLQuery1.sql -...OX\admin (53))*

```
SELECT Surname,Name,City
FROM Cooperator
WHERE City = 'Уфа'
```

Результаты | Сообщения

	Surname	Name	City
1	Иванов	Егор	Уфа
2	Егорова	Анастасия	Уфа
3	Федоровская	Анита	Уфа

Информационная система

Информационная система – это система, реализующая автоматизированный сбор, хранение, поиск, извлечение и модификацию данных и включающая технические средства обработки данных, программное обеспечение и соответствующий персонал.

Информационная система = БД + СУБД

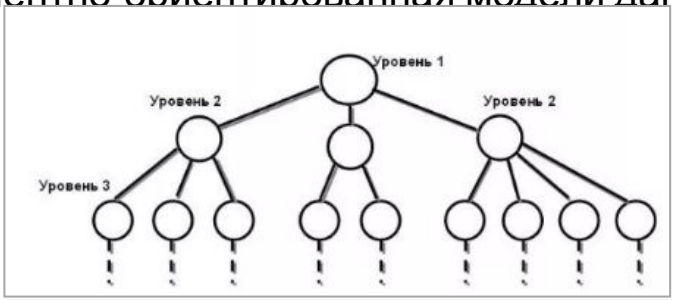
Классификация баз данных

1) По модели данных

Модель данных – это метод (принцип) логической организации данных, используемый СУБД.

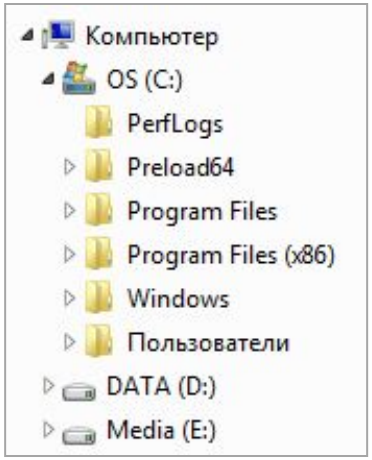
По способу установления связей между данными исторически сложились 3-и классические модели: ***иерархическая, сетевая, реляционная.***

Далее появились постреляционная, многомерная, объектно-ориентированная, объектно-реляционная, документно-ориентированная модели данных и др.

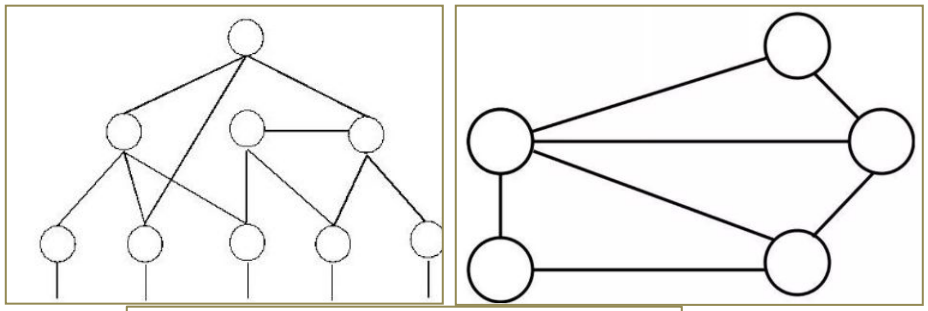


Графическое представление иерархической модели данных

Структура таблицы реляционной БД



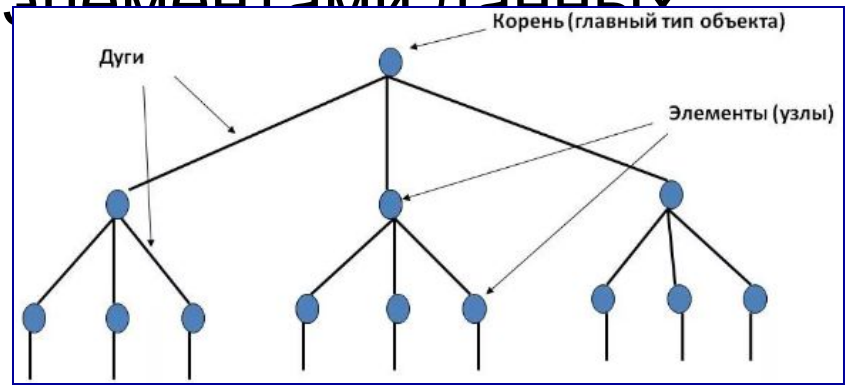
Иерархическая модель схожа по принципу построения с файловой системой

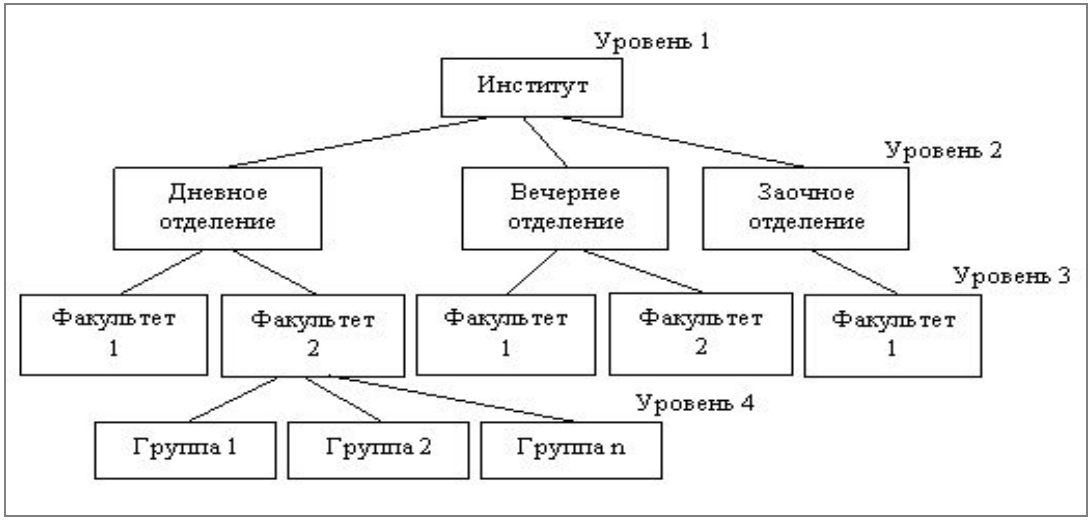


Графические представления сетевой модели данных

Иерархическая модель данных

Иерархическая модель данных имеет форму дерева с дугами-связями и узлами-элементами данных

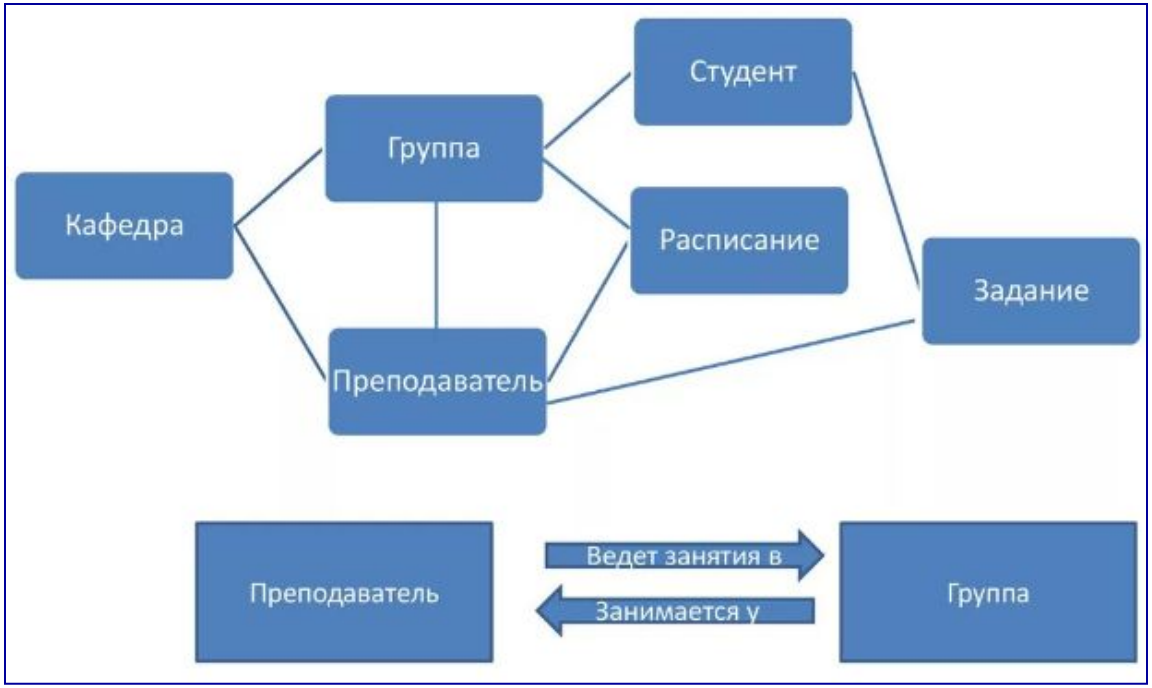




Сетевая модель данных

Сетевую модель данных можно рассматривать как расширенную версию иерархической модели.

Основное различие между иерархической и сетевой моделью состоит в том, что в сетевой модели запись может иметь связи со многими другими записями, а не только с одной родительской.



Реляционная модель данных

Реляционная база данных – это набор простых таблиц (отношений, сущностей), между которыми установлены связи с помощью ключей.

- Edgar Frank Codd.
- Основные концепции модели опубликована в 1970 г.
- Модель основывается на понятии «отношения» (Relation).

Запись - это строка таблицы.

Поле - это столбец таблицы.

Имя поля содержит название столбца вынесенное в заголовок.

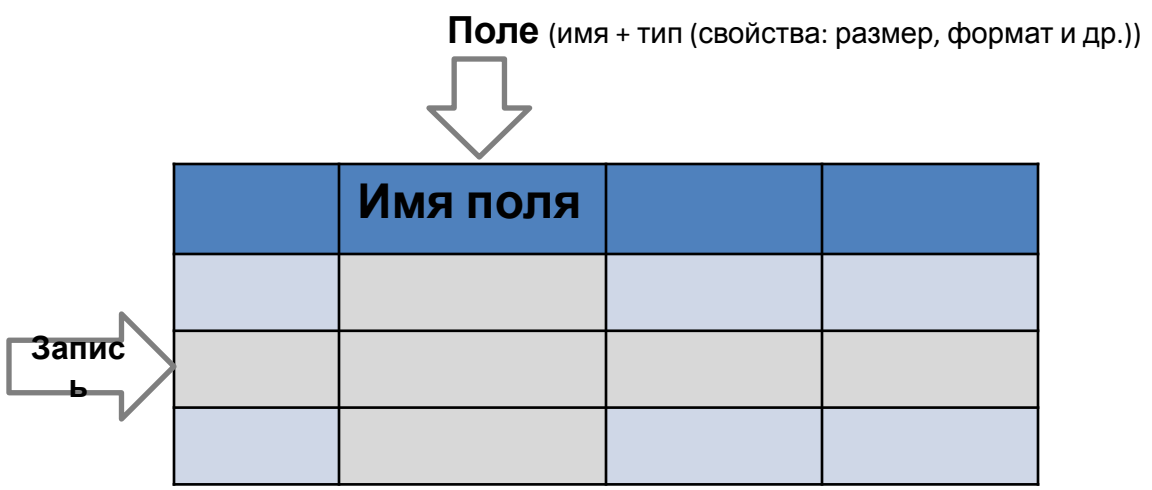


таблица «Предприятие»				
Код	**Наименование**	**Отрасль**	**Вид деятельности**	**Дата основания**
1	Уфимское моторостроительное производственное объединение	авиадвигателестроение	производство авиационных двигателей	01.01.1925
2	Туймазинский картонно-бумажный комбинат	целлюлозно-бумажная	производство и продажа бумаги	01.01.1962
3	Учалинский горно-обогатительный комбинат	горное дело	добыча и производства	01.01.1961

Атомарные значения полей

Фрагмент БД ВУЗа, таблица «Студент»								
Код	**Фамилия**	**Имя**	**Отчество**	**Год рождения**	**Пол**	**Индекс**	**Город**	**Адрес**
1	Иванов	Иван	Иванович	2001	мужской	450075	Уфа	бульвар Славы, 2, кв.12
2	Сидоров	Петр	Петрович	2001	мужской	450064	Уфа	Мира, 7, кв.10
3	Синицына	Инна	Петровна	2002	женский	450008	Уфа	Карла Маркса, 3, кв.4

Фрагмент БД Издательства, таблица «Подписчик»										
Код	**Фамилия**	**Имя**	**Отчество**	**Год рождения**	**Пол**	**Индекс**	**Город**	**Улица**	**Дом**	**Квартира**
1	Иванов	Иван	Иванович	2001	мужской	450075	Уфа	бульвар Славы	2	12
2	Сидоров	Петр	Петрович	2001	мужской	450064	Уфа	Мира	7	10
3	Синицына	Инна	Петровна	2002	женский	450008	Уфа	Карла Маркса	3	4

Документно-ориентированная модель данных

Коллекция "Товары"

```
{
    Id: 1,
    Артикул: "4678",
    Название: "Стол письменный",
    Исполнения: [
        {Материал: "Сосна",
         Цвет: "Беж"},
        {Материал: "ДСП",
         Цвет: "Белый"},
    ]
}
```

```
{
    Id: 2,
    Артикул: "7875",
    Название: "Плита ДСП",
    Габариты: {
      Ширина: 150,
      Длина: 200,
      Толщина: 1.5
    }
}
```

Реляционная модель данных

таб. Товар

ID_tovar (PK)	Артикул	Название
1	4678	Стол письменный
2	7875	Плита ДСП

таб.Исполнение товара

ID_tovar (FK)	ID_ispoln (PK)	Материал	Цвет
1	100	Сосна	Беж
1	101	ДСП	Белый

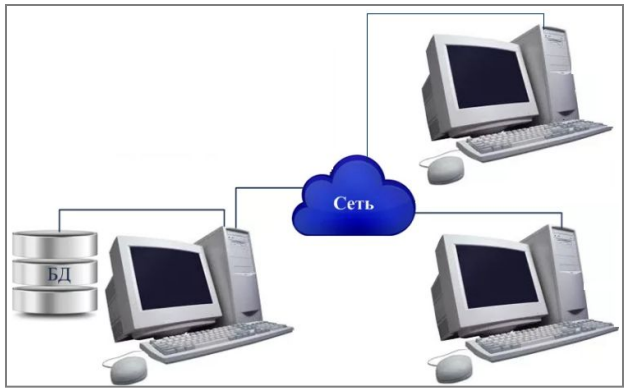
таб. Габариты товара

ID_tovar (FK)	ID_gabarit (PK)	Щирина	Длина	Толщина
2	1000	150	200	1.5

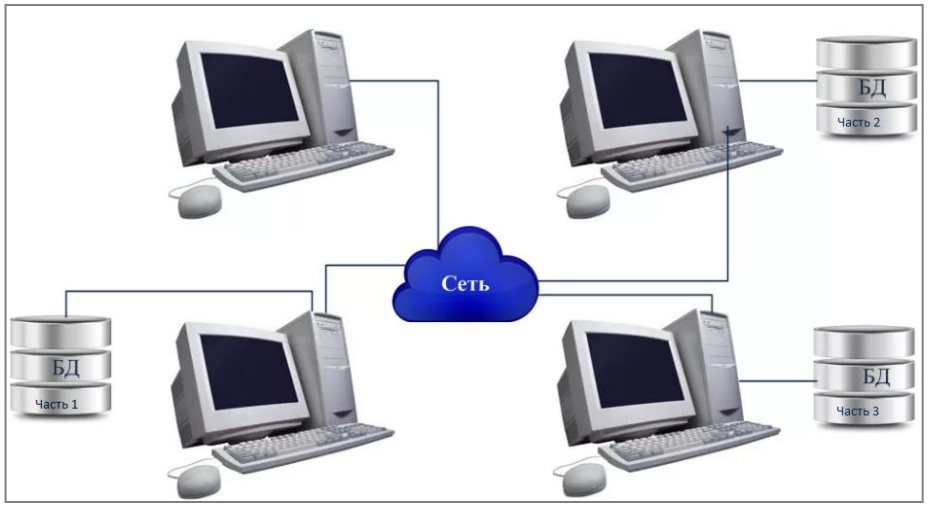
2) по способу хранения данных

Базы данных

Централизованные
(БД хранится на одном сервере)



Распределенные
(составные части единой БД хранятся на нескольких серверах, объединенных в сеть)



3) по способу доступа к БД

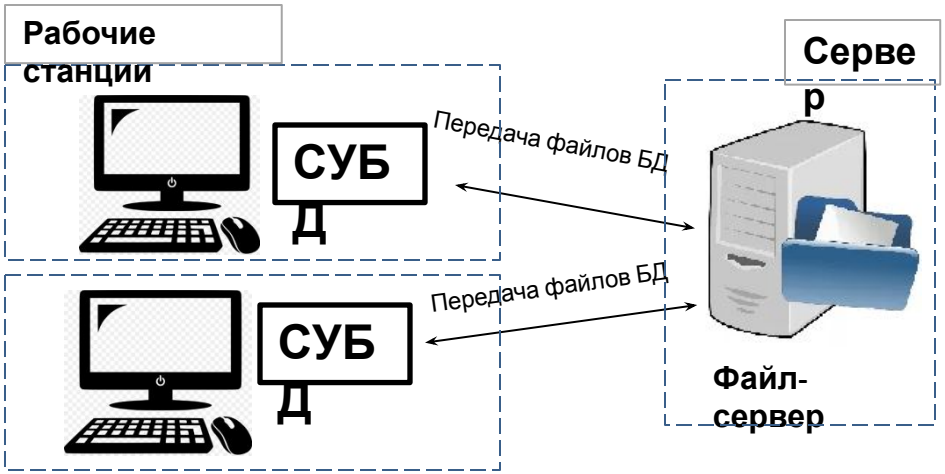
Базы данных

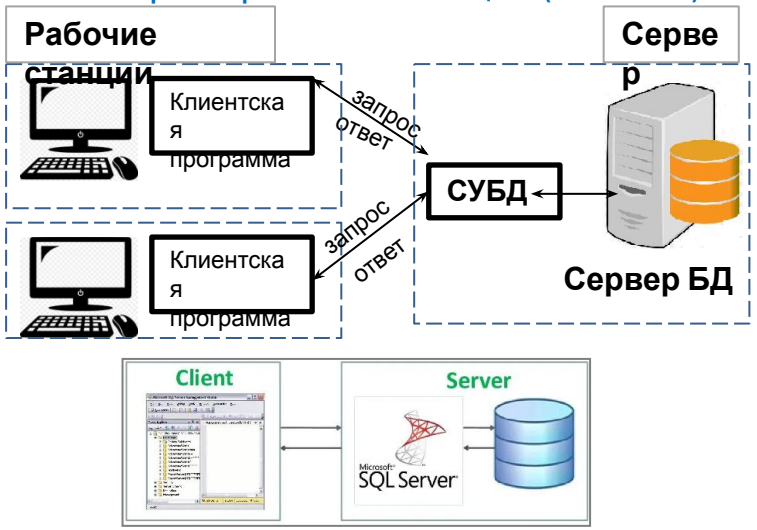
Локальные
(БД , СУБД и клиентские программы установлены на рабочей станции (PC))

Удаленные (сетевые)

Файл –серверные
(БД находится на сервере сети (файловом сервере), а СУБД и клиентские программы на рабочей станции)

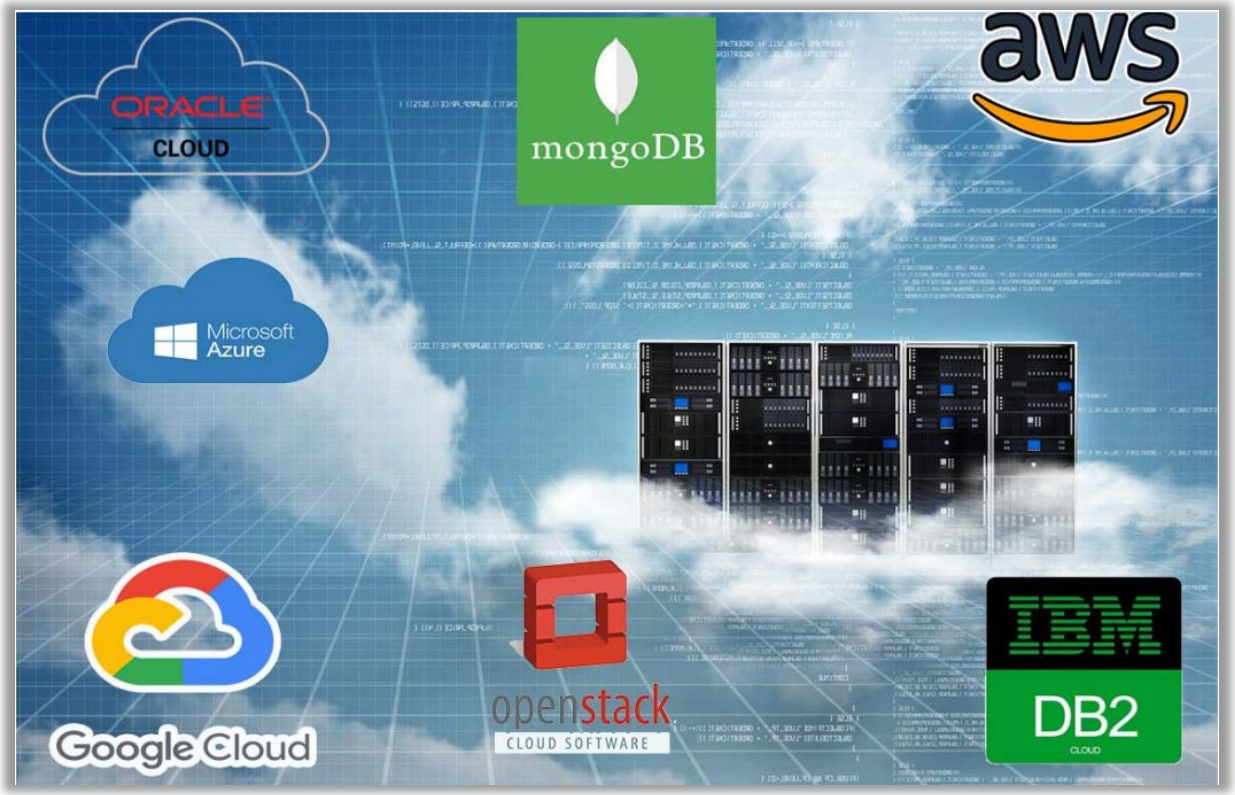
Клиент-серверные
(БД и СУБД находятся на сервере (сервер БД), а клиентские программы на рабочих станциях .
С рабочей станции (клиента) отправляются запросы на сервер (используется специальный язык запросов SQL), полученные результаты выводятся на экране рабочей станции (клиенте)





Облачные платформы

Облачные платформы предоставляют возможность разработки, выполнения приложений и хранения данных на серверах, расположенных в распределенных дата-центрах.



Главные законы об информации и информационной безопасности

149-ФЗ Об информационной безопасности — устанавливает основные права и обязанности, касающиеся информации и информационной безопасности.

152-ФЗ — описывает правила работы с персональными данными.

98-ФЗ — определяет, что относится к коммерческой тайне компаний.

68-ФЗ — дает определение электронной подписи и описывает, как и когда ее можно применять, какой юридической силой она обладает.

187-ФЗ — описывает правила защиты IT-инфраструктуры на предприятиях, работающих в сферах, критически важных для государства. К таким сферам относится здравоохранение, наука, оборона, связь, транспорт, энергетика, банки и некоторая промышленность.

Федеральная служба по техническому и экспортному контролю

Государственный орган

составляет

Государственный реестр сертифицированных средств защиты информации

Программа курса «Базы данных»

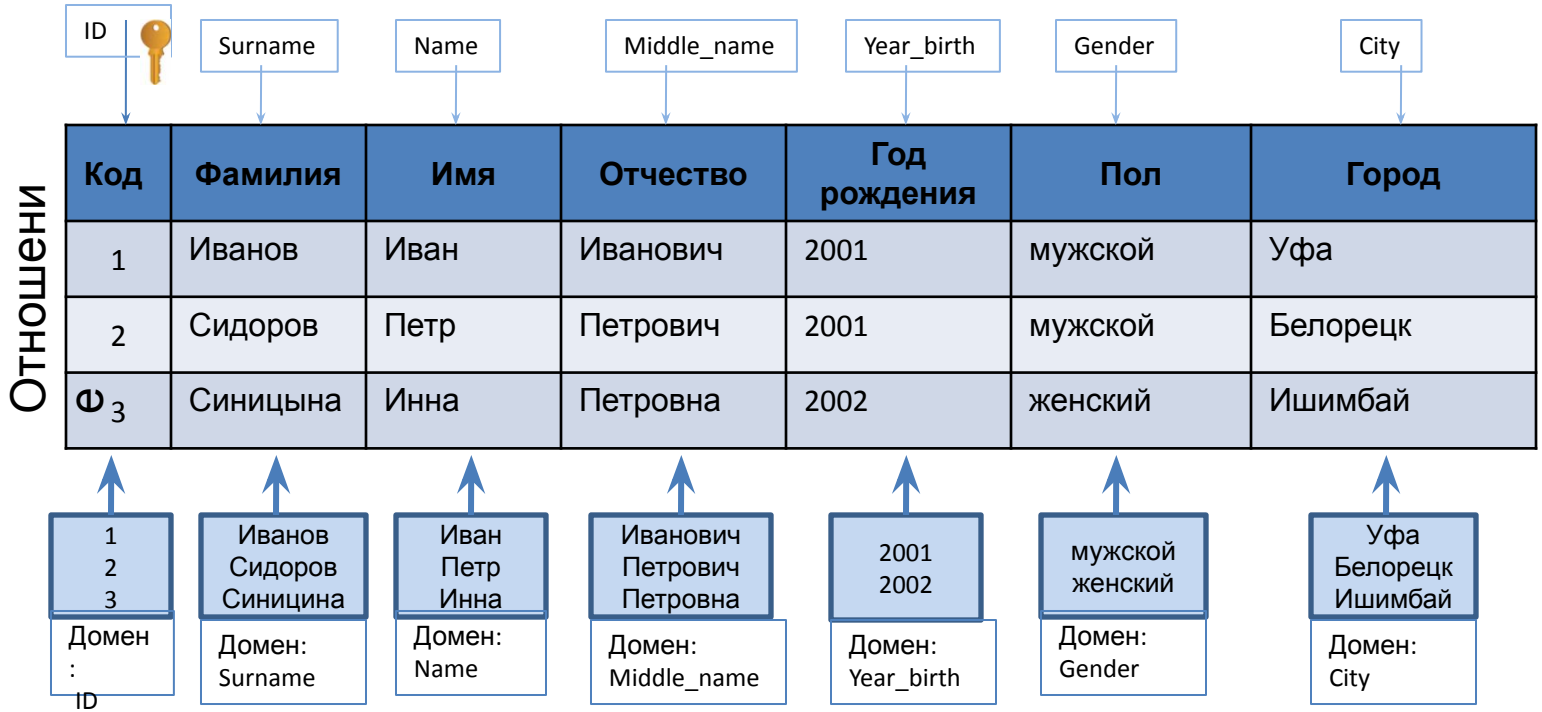
- освоите язык запросов доступа к базам данных;
- сможете обеспечить целостность данных в базах данных;
- научитесь оптимизировать запросы;
- научитесь проектировать базы данных;
- научитесь создавать приложение для работы с базами данных.

Примеры СУБД

1. MS Access, MS SQL Server от компании Microsoft Corporation.
2. Oracle, MySQL от компании Oracle Corporation.
3. PostgreSQL от компании PostgreSQL Global Development Group
   a) Postgres Pro от российской компании Postgres Professional.
4. и др.

Основные элементы реляционной БД

Термины реляционной модели	Термины «табличные» и языка SQL	Термины обработки данных
Отношение	Таблица	Файл
Кортеж	Строка	Запись
Атрибут	Столбец	Поле
Домен	Множество допустимых значений	Базовый или пользовательский тип данных (с условиями)
Мощность (кардинальность) отношения	Количество строк	Количество записей
Степень отношения	Количество столбцов	Количество полей



Ключи

Первичный ключ *(сокращенно РК - Primary Key)* – это поле (или совокупность полей), значения которого не могут повторяться.

таблица «Студент»						
Фамилия	**Имя**	**Отчество**	**Дата рождения**	**Пол**	**Номер зачетной книжки**	**Адрес**
Иванов	Иван	Иванович	01.01.2001	мужской	12345678	450075, Уфа, б-р Славы, 2
Сидоров	Петр	Петрович	02.02.2001	мужской	89123456	450900 д. Жилино, Семейная, 7
Синицына	Инна	Петровна	03.03.2002	женский	5678901	450008 Уфа, пл. Карла Маркса, 3

Первичный ключ

Простой, составной ключ

Простой первичный ключ состоит из одного поля.
Составной первичный ключ состоит из более чем одного поля.

таблица «Студент»						
Фамилия	**Имя**	**Отчество**	**Дата рождения**	**Пол**	**Номер зачетной книжки**	**Адрес**
Иванов	Иван	Иванович	01.01.2001	мужской	12345678	450075, Уфа, б-р Славы, 2
Сидоров	Петр	Петрович	02.02.2001	мужской	89123456	450900 д. Жилино, Семейная, 7
Синицына	Инна	Петровна	03.03.2002	женский	56789012	450008 Уфа, пл. Карла Маркса, 3

Составной первичный ключ

Простой первичный ключ

таблица «Студент»					
Фамилия	**Имя**	**Отчество**	**Дата рождения**	**Пол**	**Адрес**
Иванов	Иван	Иванович	01.01.2001	мужской	450075, Уфа, б-р Славы, 2
Сидоров	Петр	Петрович	02.02.2001	мужской	450900 д. Жилино, Семейная, 7
Синицына	Инна	Петровна	03.03.2002	женский	450008 Уфа, пл. Карла Маркса, 3

Ключи по способу задания

Логический (естественный) первичный ключ – поле, данные в котором логически связанны с информационным содержимым записи и уникально ее идентифицируют.

Логический первичный ключ

таблица «Пользователь»		
Имя	**E-mail**	**Пароль**
Иван	ivan@mail.ru	A@s123
Семен	semen@bk.ru	@A345s
Инна	Inna@yandex.ru	f5@As

Суррогатный (искусственный) первичный ключ - поле добавленное искусственным путем для однозначной идентификации записей.

Суррогатный первичный ключ

таблица «Пользователь»			
Код пользователя	**Имя**	**E-mail**	**Пароль**
5	Иван	ivan@mail.ru	A@s123
6	Семен	semen@bk.ru	@A345s
7	Инна	Inna@yandex.ru	f5@As

Ключи

Внешний ключ (сокращенно FK - Foreign Key) – поле (совокупность полей) таблицы, связанное с первичным ключом другой таблицы.

Первичный ключ (PK)

Главная таблица

Таблица «Город»

Код города	Название	Год основания	Площадь, км²
1	Уфа	1574	708
2	Москва	1147	2 561
3	Калининград	1255	224
NULL	Самара	1586	541

Ошибка!
Значение первичного ключа не может быть NULL

Первичный ключ (PK)

Внешний ключ (FK)

Подчиненная таблица

Таблица «Улица»

Код улицы	Код города	Название
100	1	бульвар Славы
101	1	Карла Маркса
102	3	Карла Маркса
103	2	Арбат
104	2	Тверская
105	3	Янтарная
106	4	Свердловская

Ошибка! Нет значения PK в главной таблице

Ключи

Потенциальный ключ (Candidate key) - простой или составной ключ, который уникально идентифицирует каждую запись таблицы.

Потенциальный ключ удовлетворяет требованиям уникальности и минимальности (не сократимости).

Если отношение имеет более одного потенциального ключа, то один из них рассматривается как первичный, остальные являются **альтернативными (Alternative key)**.

таблица «Пользователь»				
Код пользователя	**Имя**	**Логин**	**E-mail**	**Пароль**
5	Иван	Ivan	ivan@mail.ru	A@s123
6	Семен	Semen	semen@bk.ru	@A345s
7	Инна	Inna	Inna@yandex.ru	f5@As

Потенциальные ключи:

1) Код пользователя (например, первичный ключ).
2) Логин.
3) E-mail.
4) Имя + Пароль (если не задано условие на уникальность пароля).

Не потенциальные ключи (т.к. не отвечают требованию минимальности):

5) Имя + Логин.
6) Имя + E-mail.

Ограничения целостности данных

Целостностью данных можно назвать механизм поддержания соответствия базы данных предметной области.

Ограничения целостности – это требования предназначенные для предупреждения добавления в базу данных невозможных, невероятных данных, т.е. обеспечивают защиту от возможных ошибок пользователя.

Базовые требования обеспечения целостности:

1) **Ссылочную целостность (или целостность ссылок)** обеспечивается системой первичных и внешних ключей. Правило: значение внешнего ключа подчиненной таблицы должно соответствовать одному из значений первичного ключа главной таблицы или иметь значение *NULL*.

2) **Целостность сущностей**. Правило: любая таблица (отношение) должна иметь первичный ключ. Поле первичного ключа не должно содержать пустые значения *(NULL*).

3) **Целостность домена[1].** Правило: все значения некоторого поля должны принадлежат множеству допустимых значений.

Целостность домена обеспечивается заданием условий на значения (CHECK), запретом пустых значений (NOT NULL), заданием значения по умолчанию (DEFAULT) , хранимыми процедурами, триггерами.

Например, при создании структуры таблицы можно сразу указать возможное количество цифр в поле Номер телефона.

[1]Домен (в реляционной модели данных) − множество допустимых значений поля.

Задачи

1) Какие поля в таб.1 и таб.2 могут быть первичными ключами?

Определите названия ключей по типу и по способу задания.

Дата	Время	Температура воды	Температура воздуха	Скорость ветра	Волны
01.06.2019	9:00	19	29	8	2
26.07.2019	9:00	22	30	7	2
23.08.2019	10:00	24	36	9	3
23.08.2019	14:00	26	38	2	1

Фамилия	Имя	Адрес	Телефон
Иванов	Иван	г.Уфа, Айская,11	9271112233
Евдокимов	Петр	г.Сибай, Ленина,2	9167779900
Петров	Иван	г.Уфа, Мира,10	9064455612

Задачи

2) Какие из приведенных ниже полей могут стать простым естественным первичным ключом?

- фамилия;
- имя;
- номер и серия паспорта;
- номер дома;
- город проживания;
- адрес электронной почты;
- дата выполнения работы;
- марка стиральной машины.

3) Сколько строительных компаний в городе Москва?

4) Сколько строительных компаний в городе Уфа?

таблица «Город»

Код города	Название
1	Пермь
2	Москва
3	Челябинск
4	Уфа

таблица «Строительная

Код	Название	Код города
100	ЖилСтрой	3
101	Новострой	2
102	Трест №3	4
103	ИнвестСтрой	1
104	Кооператив №1	2
105	Трест№1	4
106	СтройКапитал	4

Задачи

5) Определите какие материалы отправлены в каждый из городов?

6) Сколько единиц огнеупорных кирпичей отправлено в каждый город?

7) Посчитайте общую стоимость материалов, отправленных в каждый город.

таблица «Город»

Код	Название
1	Пермь
2	Москва
3	Челябинск
4	Уфа

таблица «Материал»

Артикул	Название	Цена за единицу, руб
1234	Кирпич огнеупорный	10
4352	Кирпич облицовочный	8
1265	Опора бруса	100
763	Панель	1200
8412	Плитка настенная	50

таблица «Строительная компания»

Код	Название	Код города
100	ЖилСтрой	3
101	Новострой	2
102	Трест №3	4
103	ИнвестСтрой	1
104	Кооператив №1	2
105	Трест№1	4
106	СтройКапитал	4

таблица «Накладная»

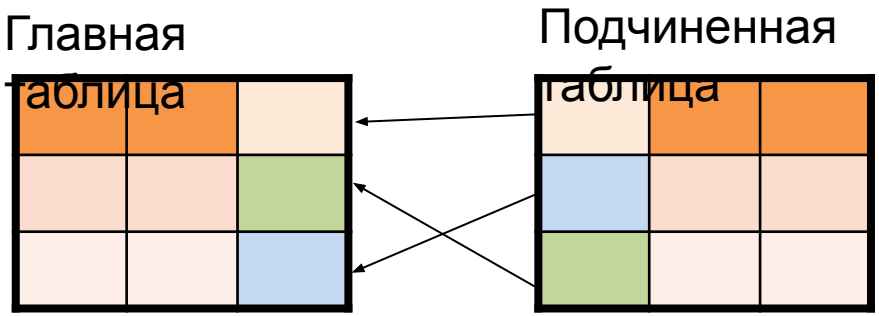
Накладная	Код компании	Артикул	Количество, ед.
912	101	1234	1000
751	100	1234	2000
784	102	8412	1000
132	100	763	100
542	104	1234	500
321	101	1265	100

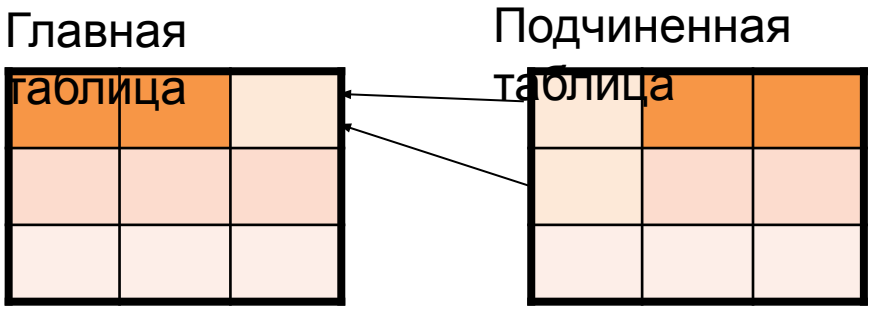
Виды связей между реляционными таблицами

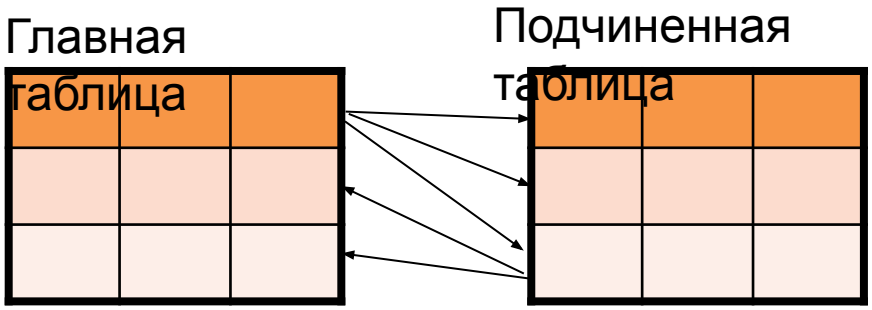
Реляционная база данных — это совокупность двумерных взаимосвязанных таблиц.

Виды связей между таблицами:

1) Один к одному (1:1, 1−1).
2) Один ко многим (1:M, *1* – ∞).
3) Многие ко многим (M:M, ∞ – ∞).







Вид связи один к одному

Связь **один к одному** означает, что одной записи в главной таблице соответствует только одна запись в подчиненной.

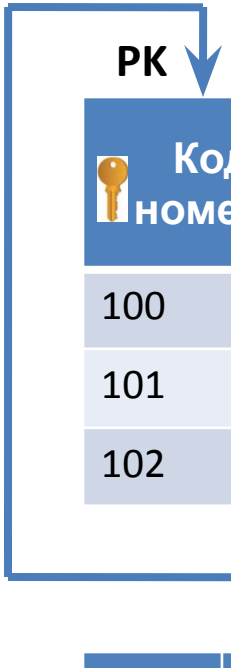


таблица «ИНН»

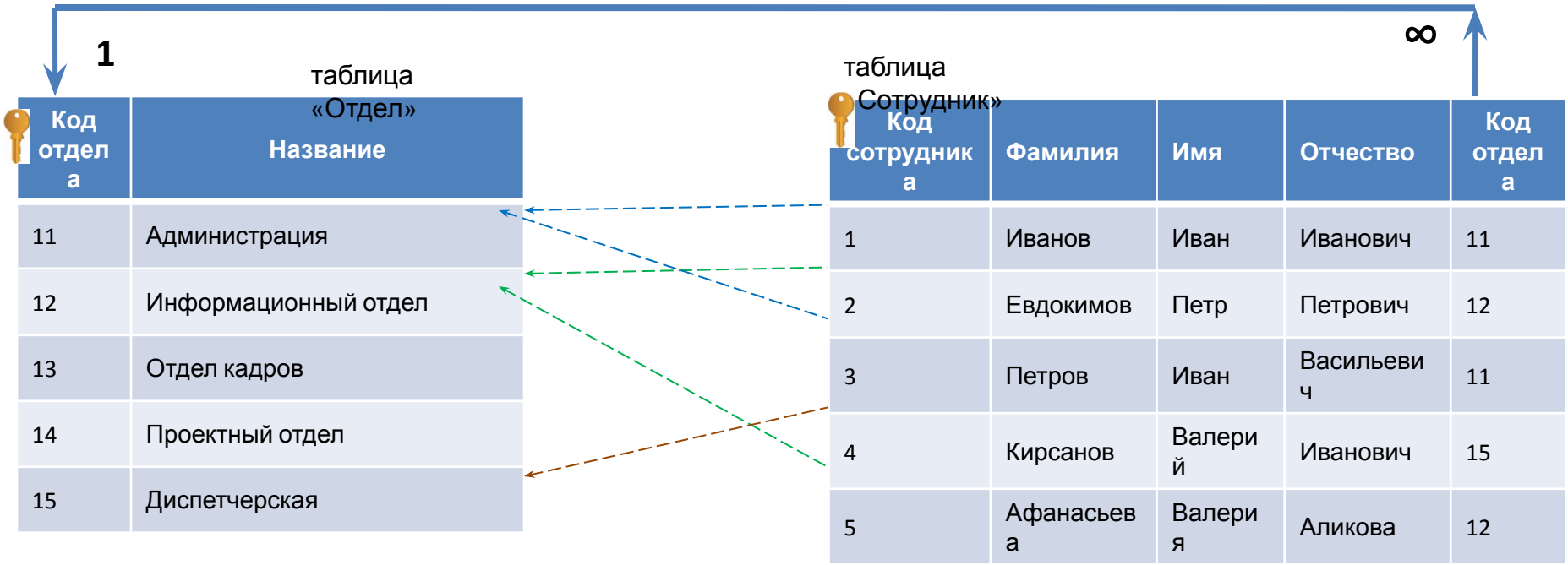
Код номера	Налоговый номер	Дата выдачи	Код подразделения	Серия и номер паспорта
100	100135	01.01.2020	021-345	1234 213212
101	101895	02.02.2020	891-091	4531 312456
102	102141	03.03.2020	120-321	4321 432165

таблица «Гражданин РФ»

Код	Код номера	Фамилия	Имя	Отчество	Дата рождения	Место рождения
10	100	Иванов	Иван	Иванович	21.01.2006	Уфа
11	101	Евдокимов	Петр	Петрович	12.02.2000	Сибай
12	102	Петров	Иван	Васильевич	14.03.2000	Ишимбай

Вид связи один ко многим

Связь **один ко многим** означает, что одной записи в главной таблице может соответствовать множество записей в подчиненной таблице.



Код отдела	Название
11	Администрация
12	Информационный отдел
13	Отдел кадров
14	Проектный отдел
15	Диспетчерская

Код сотрудника	Фамилия	Имя	Отчество	Код отдела
1	Иванов	Иван	Иванович	11
2	Евдокимов	Петр	Петрович	12
3	Петров	Иван	Васильевич	11
4	Кирсанов	Валерий	Иванович	15
5	Афанасьева	Валерия	Аликова	12

Вид связи многие ко многим

Связь **многие ко многим** означает, что одной записи в главной таблице может соответствовать множество записей в подчиненной, и наоборот.

таблица «Преподаватель»

Код преподавателя	Фамилия	Имя	Отчество
1	Иванов	Иван	Иванович
2	Евдокимов	Петр	Петрович
3	Петров	Иван	Васильевич

таблица «Дисциплина»

Код дисциплины	Название
11	Информатика
12	Программирование
13	Математический анализ

1

таблица «Преподаватель»

Код преподавателя	Фамилия	Имя	Отчество
1	Иванов	Иван	Иванович
2	Евдокимов	Петр	Петрович
3	Петров	Иван	Васильевич

1

таблица «Дисциплина»

Код дисциплины	Название
11	Информатика
12	Программирование
13	Математический анализ

∞ ∞

таблица «Семестровый учебный план»

Код	Код преподавателя	Код дисциплины	Семестр	Лекции	Практические занятия	Лабораторные занятия	СРС
1	1	11	1	12	6	24	30
2	1	12	1	16	4	24	64
3	2	13	1	38	34	4	140
4	3	13	2	54	44	4	116

Задачи

1) Какие виды связи заданы между таблицами? Определите отношения подчиненности между таблицами?

Фрагмент БД «Информация по парфюмерной продукции»

1

таблица «Бренд»

Код бренда	Бренд	Страна
100	Dior	Франция
101	Chanel	Франция
102	Antonio Banderas	Испания
103	Hugo Boss	Германия

∞

таблица «Каталог»

Код каталога	Код бренда	Название
201	100	J'adore
202	101	Mademoiselle
203	102	Secret Collection
204	103	Boss Number One

∞ ∞

таблица «Эфирное масло»

Код	Название
1	роза
2	мускус
3	жасмин
4	кедр

Задачи

2) Какие виды связи заданы между таблицами? Определите отношения подчиненности между таблицами?

Фрагмент БД «Дипломное проектирование»

1

таблица «Группа»

Код группы	Группа	Количество
1	БУС-20	20
2	БПО-20	24
3	БПОи-20	12

1

таблица «Кафедра»

Код кафедры	Кафедра	Количество
1	ЭЭП	16
2	ВТИК	32
3	АТПП	32

таблица

∞ ∞

Код	Фамилия	Имя	Отчество	Код группы	Код дип. руководителя
1	Тимошин	Антон	Валерьевич	2	1
2	Манников	Илья	Анатольевич	2	2
3	Ерохина	Анна	Витальевна	3	2

1

таблица «Дипломный руководитель»

∞

Код	Фамилия	Имя	Отчество	Код кафедры
1	Иванов	Иван	Иванович	1
2	Ершов	Петр	Петрович	2
3	Петров	Иван	Васильевич	2

Задачи

3) Какая из двух таблиц главная, какая подчинённая? Какой вид связи между таблицей 1 и таблицей 2?

Фрагмент БД «Административные единицы»

таблица «Город»

Код	Название	Площадь
1	Уфа	707
2	Ишимбай	108,52
3	Сибай	157

∞ — 1

таблица «Территориальное

Код города
1
1
2
2

Код	Название
101	Демский
102	Калининский
103	Шахтау
104	Уральский

4) Какая из двух таблиц главная, какая подчинённая? Какой вид связи между таблицей 1 и таблицей 2?

Фрагмент БД «География»

таблица

Код	Название	Площадь
1	Россия	146 748 590
2	Италия	301 340
3	Испания	505 990

1 — 1

таблица «Столица»

Код страны
1
2
3

Код	Название	Площадь
101	Москва	2 561
102	Рим	1 287
103	Мадрид	607

Поддержка целостности сущностей и целостности ссылок.

Синтаксис :

1) <*поле и тип данных*> **PRIMARY KEY**

2) FOREIGN KEY <*поле*> **REFERENCES** <*главная таблица*> [<*первичный ключ*>]

Пример, фрагмент БД «Дипломное проектирование»

table. Diploma_supervisor

Diploma_supervisor_Id	Surname	Name	Middle_name	Birthday
1	Иванов	Иван	Иванович	01.01.1900
2	Ершов	Петр	Петрович	01.02.1991
3	Петров	Иван	Васильевич	02.02.1995

CREATE TABLE Diploma_supervisor
(Diploma_supervisor_Id INTEGER **PRIMARY KEY IDENTITY(1,1)**,
Surname VARCHAR(20) **NOT NULL**,
Name VARCHAR(20) **NOT NULL**,
Middle_name VARCHAR(20),
Birthday DATE)

table. Student

Student_id	Surname	Name	Middle_name	Diploma_supervisor_Id
1	Тимошин	Антон	Валерьевич	1
2	Манников	Илья	Анатольевич	2
3	Ерохина	Анна	Витальевна	2

CREATE TABLE Student
(Student_id INTEGER **PRIMARY KEY IDENTITY(1,1),**
Surname VARCHAR(20) **NOT NULL**,
Name VARCHAR(20) **NOT NULL**,
Middle_name VARCHAR(20),
Diploma_supervisor_id INTEGER,
CONSTRAINT Key_ foreign
FOREIGN KEY (Diploma_supervisor_id) **REFERENCES** Diploma_supervisor (Diploma_supervisor_id))

ALTER TABLE Student
ADD CONSTRAINT Key_ foreign **FOREIGN KEY**(Diploma_supervisor_id) **REFERENCES** Diploma_supervisor (Diploma_supervisor_id)

Поддержка целостности данных при использовании команд UPDATE и DELETE

FOREIGN KEY () REFERENCES [[ ]
[**ON DELETE** {**NO ACTION** | SET NULL| CASCADE | SET DEFAULT}]
[ON UPDATE {NO ACTION | SET NULL| CASCADE | SET DEFAULT}]

table. Diploma_supervisor

Diploma_supervisor_id	Surname	Name	Middle_name	Birthday
1	Иванов	Иван	Иванович	01.01.1900
~~2~~	~~Ершов~~	~~Петр~~	~~Петрович~~	~~01.02.1991~~
3	Петров	Иван	Васильевич	02.02.1995

Ошибка

table. Student

Student_id	Surname	Name	Middle_name	Diploma_supervisor_id
1	Тимошин	Антон	Валерьевич	1
2	Манников	Илья	Анатольевич	2
3	Ерохина	Анна	Витальевна	2

CREATE TABLE **Student**
(Student_id INTEGER **PRIMARY KEY IDENTITY(1,1),**
Surname VARCHAR(20) **NOT NULL**,
Name VARCHAR(20) **NOT NULL**,
Middle_name VARCHAR(20),
Diploma_supervisor_id INTEGER,
CONSTRAINT Key_ foreign
FOREIGN KEY(Diploma_supervisor_id) **REFERENCES** Diploma_supervisor(Diploma_supervisor_id) **ON DELETE NO ACTION**)

Поддержка целостности данных при использовании команд UPDATE и DELETE

FOREIGN KEY ( ) REFERENCES [[ ]
[**ON DELETE** {NO ACTION | **SET NULL**|CASCADE | SET DEFAULT}]
[ON UPDATE {NO ACTION | SET NULL| CASCADE | SET DEFAULT}]

table. Diploma_supervisor

Diploma_supervisor_id	Surname	Name	Middle_name	Birthday
1	Иванов	Иван	Иванович	01.01.1900
2	Ершов	Петр	Петрович	01.02.1991
3	Петров	Иван	Васильевич	02.02.1995

table. Student

Student_id	Surname	Name	Middle_name	Diploma_supervisor_id
1	Тимошин	Антон	Валерьевич	1
2	Манников	Илья	Анатольевич	NULL
3	Ерохина	Анна	Витальевна	NULL

```
CREATE TABLE Student
(Student_id          INTEGER PRIMARY KEY IDENTITY(1,1),
Surname                  VARCHAR(20) NOT NULL,
Name                          VARCHAR(20) NOT NULL,
Middle_name              VARCHAR(20),
Diploma_supervisor_id          INTEGER,
CONSTRAINT Key_ foreign
FOREIGN KEY(Diploma_supervisor_id) REFERENCES Diploma_supervisor(Diploma_supervisor_id) ON DELETE SET NULL)
```

Поддержка целостности данных при использовании команд UPDATE и DELETE

FOREIGN KEY ( ) REFERENCES [[ ]
[**ON DELETE** {NO ACTION | SET NULL|**CASCADE** | SET DEFAULT}]
[ON UPDATE {NO ACTION | SET NULL| CASCADE | SET DEFAULT}]

table. Department

Dept_id	Department	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Отдел кадров	23431
~~103~~	~~Проектный отдел~~	~~45673~~

table. Cooperator

Cooperator_id	Surname	Name	Middle_name	Dept_id
1	Иванов	Иван	Иванович	100
2	Сидоров	Петр	Петрович	101
3	Синицына	Инна	Петровна	102
~~4~~	~~Егоров~~	~~Валерий~~	~~Игнатьевич~~	~~103~~
~~5~~	~~Воронина~~	~~Наталья~~	~~Игоревна~~	~~103~~

CREATE TABLE **Cooperator**
(Cooperator_id INTEGER **PRIMARY KEY IDENTITY(1,1),**
Surname VARCHAR(20) **NOT NULL**,
Name VARCHAR(20) **NOT NULL**,
Middle_name VARCHAR(20),
Dept_id INTEGER,
CONSTRAINT Key_ foreign
FOREIGN KEY(Dept_id) **REFERENCES** Department(Dept_id) **ON DELETE CASCADE**)

Поддержка целостности данных при использовании команд UPDATE и DELETE

FOREIGN KEY ( ) REFERENCES [[ ]
[**ON DELETE** {NO ACTION | SET NULL|CASCADE | **SET DEFAULT**}]
[ON UPDATE {NO ACTION | SET NULL| CASCADE | SET DEFAULT}]

table. Department

Dept_id	Department	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Отдел кадров	23431
~~103~~	~~Проектный отдел~~	~~45673~~
1000	Резерв	11108

table. Cooperator

Cooperator_id	Surname	Name	Middle_name	Dept_id
1	Иванов	Иван	Иванович	100
2	Сидоров	Петр	Петрович	101
3	Синицына	Инна	Петровна	102
4	Егоров	Валерий	Игнатьевич	~~103~~ 1000
5	Воронина	Наталья	Игоревна	~~1030~~ 1000

CREATE TABLE **Cooperator**
(Cooperator_id INTEGER **PRIMARY KEY,**
Surname VARCHAR(20) **NOT NULL**,
Name VARCHAR(20) **NOT NULL**,
Middle_name VARCHAR(20),
Dept_id INTEGER DEFAULT 1000,
CONSTRAINT Key_ foreign
FOREIGN KEY(Dept_id) **REFERENCES** Department(Dept_id) **ON DELETE SET DEFAULT**)

Язык SQL

SQL
Stuctured Query Language

Язык SQL, предназначенный для взаимодействия с данными в БД.

Появился в середине 70-х (первые публикации 1074 г.).

Первый принятый стандарт SQL/86 разработан Американским национальным институтом стандартов (ANSI) и одобрен Международной организаций по стандартизации (ISO) в 1986г.

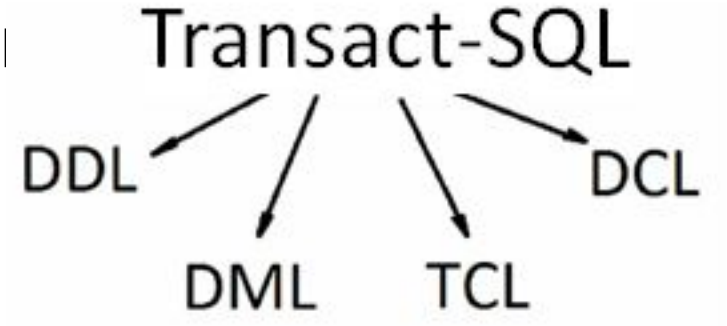
Последняя редакция стандарта: SQL:2016

В каждой СУБД применяется свой диалект языка.

Диалекты языка SQL (расширения SQL)

1) Transact-SQL (или T-SQL) — СУБД MS SQL Server (Microsoft)
2) Jet SQL – СУБД Access (Microsoft).
3) PL/SQL (Procedural Language/SQL) — СУБД Oracle (Oracle).
4) PL/pgSQL (Procedural Language/postgreSQL) — СУБД PostgreSQL (PostgreSQL Global Development Group).
5) и др.

Команды



Команды языка определения данных (DDL - Data Definition Language)	CREATE, ALTER, DROP, TRUNCATE
Команды языка манипулирования данными (DML - Data Manipulation Language)	INSERT, UPDATE, DELETE, SELECT
Команды языка управления транзакциями (TCL - Transaction Control Language)	SAVE TRANSACTION, COMMIT, ROLLBACK
Команды языка управления данными (DCL - Data Control Language)	GRANT, REVOKE, DENY

Команды языка определения данных (DDL - Data Definition Language)

Команда	Смысл	Действие
CREATE DATABASE	Создать БД	Создает новую базу данных, определив основные параметр для нее
ALTER DATEBASE	Изменить БД	Изменяет набор основных объектов в базе данных, ограничений, касающихся всей базу данных
DROP DATABESE	Удалить БД	Удаляет существующую БД, только в том случае, если есть права на выполнение данного действия
CREATE TABLE	Создать таблицу	Создает новую таблицу
ALTER TABLE	Изменить таблицу	Изменяет структуру существующей таблицы или ограничения целостности
DROP TABLE	Удалить таблицу	Удаляет таблицу из БД
TRUNCATE TABLE	Удалить данные из таблицы	Удаляет данные в таблице, но сохраняет ее структуру и индексы
CREATE VIEW	Создать представление	Создает виртуальную таблицу, соответствующую некоторому SQL-запросу
ALTER VIEW	Изменить представление	Изменяет ранее созданное представление
CREATE INDEX	Создать индекс	Создает индекс для некоторой таблицы с целью обеспечения быстрого доступа по атрибутам, входящим в индекс
ALTER INDEX	Изменить индекс	Изменяет существующий индекс таблицы или представления
DROP INDEX	Удалить индекс	Удаляет ранее созданный индекс
CREATE PROCEDURE	Создать процедуру	Создает хранимую процедуру и сохраняет как объект базы данных
ALTER PROCEDURE	Изменить процедуру	Изменяет ранее созданную процедуру
DROP PROCEDURE	Удалить процедуру	Удаляет процедуру из БД
CREATE FUNCTION	Создать функцию	Создает определенную пользователем функцию и сохраняет как объект базы данных
ALTER FUNCTION	Изменить функцию	Изменяет существующую функцию
DROP FUNCTION	Удалить функцию	Удаляет функцию из БД
CREATE TRIGGER	Создать триггер	Создает триггер, как предварительно определенное действие или последовательность действий, автоматически осуществляемых при выполнении операций обновления, добавления или удаления данных.
ALTER TRIGGER	Изменить триггер	Изменяет определение триггера
DROP TRIGGER	Удалить триггер	Удаляет ранее созданный триггер

Примеры применения команд DDL

Создание таблицы «Плата за электроэнергию»

```
CREATE TABLE Rent_for_light
(Id INT PRIMARY KEY IDENTITY(1,1),
Region  VARCHAR(20),
N_Month TINYINT CHECK (N_Month>0 AND N_Month<13),
Unit_cost REAL  /* плата за единицу электрической энергии, кВт*/
)
```

Table. Rent_for_light

Id	Region	N_Month	Unit_cost
NULL	*NULL*	*NULL*	*NULL*

```
ALTER TABLE Rent_for_light
ADD Regional_coefficient REAL
```

Table. Rent_for_light

Id	Region	N_Month	Unit_cost	Regional_coefficient
NULL	*NULL*	*NULL*	*NULL*	*NULL*

```
TRUNCATE TABLE Rent_for_light
```

```
DROP TABLE Rent_for_light
```

Команды языка манипулирования данными (DML - Data Manipulation Language)

Команда	Смысл	Действие
INSERT	Вставить строку	Вставляет одну или несколько строк в базовую таблицу. Допустимы модификации команды, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
UPDATE	Обновить строку	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации
DELETE	Удалить строку	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение команды согласуется с принципами поддержки целостности, поэтому эта команда не всегда может быть выполнена корректно, даже если синтаксически она записана правильно
SELECT	Выбрать строки	Команда, позволяющая сформировать результирующее отношение, соответствующее запросу

Примеры применения команд DML

INSERT INTO Rent_for_light
VALUES (‘Республика Башкортостан’, 1, 20, 0.1), (‘Республика Татарстан’, 1, 25, 0.1)

Id	Region	N_Month	Unit_cost	Regional_coefficient
1	Республика Башкортостан	1	20	0.1
2	Республика Татарстан	1	25	0.1

UPDATE Rent_for_light
SET Regional_coefficient = Regional_coefficient *2
WHERE Region =‘Республика Татарстан’

Id	Region	N_Month	Unit_cost	Regional_coefficient
1	Республика Башкортостан	1	20	0.1
2	Республика Татарстан	1	25	0.2

SELECT Region, Unit_cost
FROM Rent_for_light

Region	Unit_cost
Республика Башкортостан	20
Республика Татарстан	25

DELETE FROM Rent_for_light
WHERE Regional_coefficient>0.1

Id	Region	N_Month	Unit_cost	Regional_coefficient
1	Республика Башкортостан	1	20	0.1

Синтаксис SELECT

SELECT *column_name1, column_name2, …*

FROM *table_name*

WHERE *condition*

поля для вывода

таблица, данные которой нужно вывести

условие поиска, возвращает только строки , соответствующие этому условию

Id	Region	N_Month	Unit_cost	Regional_coefficient
1	Республика Башкортостан	1	20	0.1
2	Республика Татарстан	1	25	0.2

```
SELECT Region , N_Month, Unit_cost , Regional_coefficient
FROM Rent_for_light
WHERE Regional_coefficient>0.1
```

Region	N_Month	Unit_cost	Regional_coefficient
Республика Татарстан	1	25	0.2

Транзакция

Транзакция – это последовательность операций с данными, выполняющаяся как единое целое.

Транзакции повышают надежность баз данных.

Пример транзакции

таблица «Счет в банке»

Код	Номер счета	Баланс	Код_клиента (FK)
1	101	1000	1
2	102	2000	2
3	109	500	4

Нужно перевести от одного клиента банка (с номером счета 101) другому клиенту банка (с номером счета 109) денежную сумму в размере 500 руб.

1) **UPDATE** Bank_account
SET Balance = Balance - 500
WHERE Number_account = 101

2) **UPDATE** Bank_account
SET Balance = Balance + 500
WHERE Number_account = 109

Транзакции и целостность баз данных

Количество операций, входящих в транзакцию, может быть от одной и более. Разработчик решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций.

Транзакция обладает следующими важными свойствами (**ACID**), которые гарантируют правильность и надежность работы системы:

1) **A**tomicity (атомарность).

Каждая транзакция в БД должна быть выполнена полностью либо не выполнена совсем. Не допускается частичное выполнение.

2) **C**onsistency (согласованность).

Должно быть согласованное состояние БД до и после выполнения транзакции.

3) **I**solation (изолированность).

Результаты транзакции не должны быть видены другим транзакциям, пока она не завершиться.

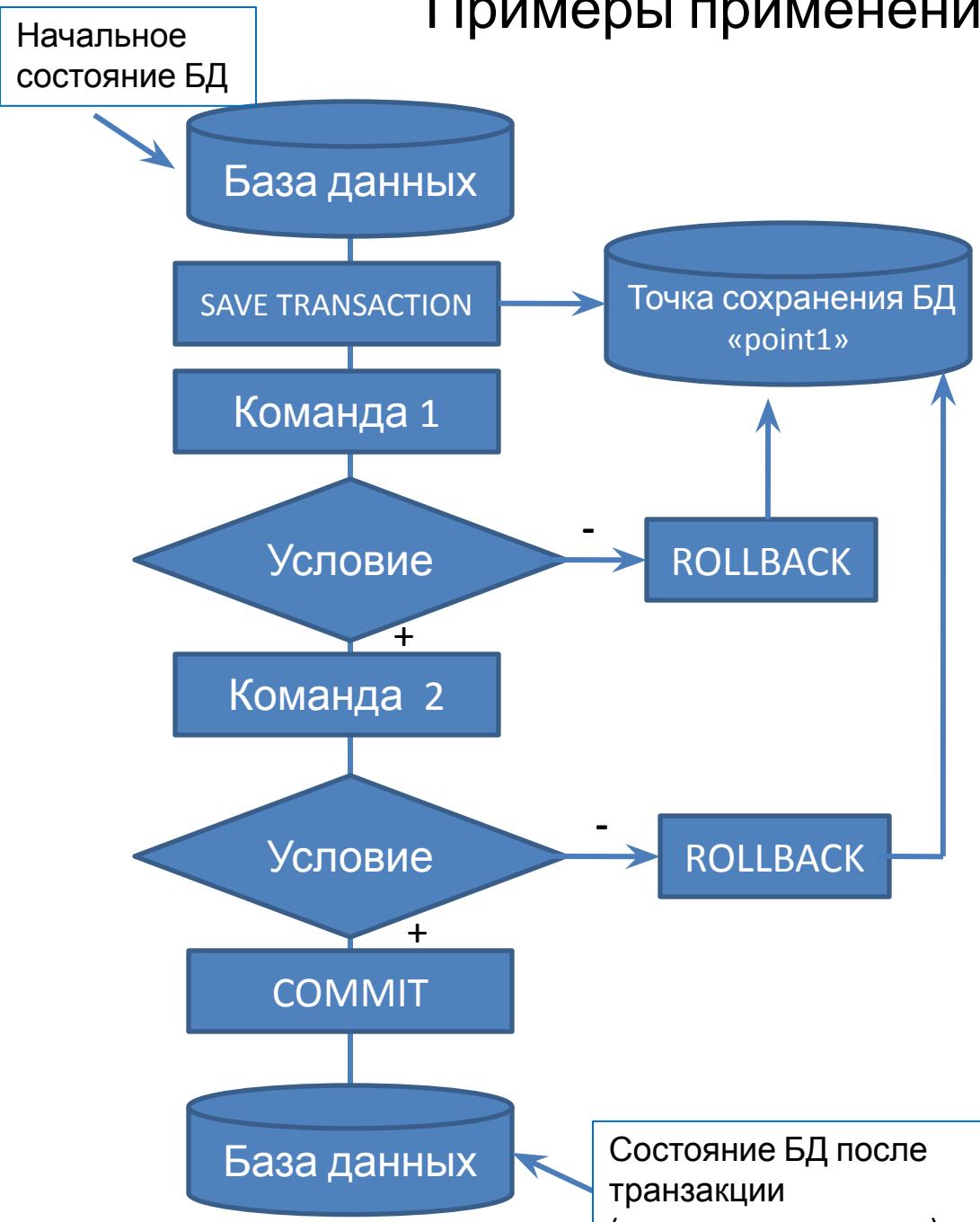
4) **D**urability (устойчивость, долговечность).

Изменения, внесенные в БД в результате выполнения транзакции должны быть зафиксированы.

Команды языка управления транзакциями (TCL - Transaction Control Language)

Команда	Смысл	Действие
SAVE TRANSACTION	Сохранить промежуточную точку выполнения транзакции	Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшим к нему вернуться
COMMIT	Завершить транзакцию	Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию
ROLLBACK	Прерывание транзакции	Отменить изменения, проведенные в ходе выполнения транзакции

Примеры применения команд TCL




```
BEGIN TRANSACTION
SAVE TRANSACTION point1
      UPDATE Bank_account
        SET Balance= Balance - 500
        WHERE Number_account = 101
      IF (@@error != 0)
        ROLLBACK TRANSACTION point1
      UPDATE Bank_account
        SET Balance= Balance + 500
        WHERE Number_account = 109
      IF (@@error != 0)
        ROLLBACK TRANSACTION point1
COMMIT
```

Команды языка управления данными (DCL - Data Control Language)

Команда	Смысл	Действие
GRANT	Предоставить права	Предоставить права доступа на ряд действий над некоторым объектом БД
REVOKE	Лишить права	Лишить прав доступа к некоторому объекту или некоторым действиям над объектом
DENY	Запретить доступ	Запретить доступ к объектам базы данных

Примеры применения команд DCL

GRANT SELECT ON Student **TO** User2;

REVOKE SELECT ON Student **TO** User2;

DENY CREATE DATABASE, CREATE TABLE TO User2.

Значение NULL

Значение NULL - универсальное значение, не зависимое от типа данных поля. Данное значение
свидетельствует об отсутствии значения у поля, это не то же самое, что число «0».
Поле с значением NULL - это поле, которое было оставлено пустым во время создания записи!

таблица «Информация об абитуриенте»

Код абитуриента	Фамилия	Имя	Отчество	Дата рождения	Номер телефона	…
101	Синицына	Инна	Петровна	01.01.2000	12345678910	
102	Егоров	Валерий	Игнатьевич	01.02.2001	*NULL*	
103	Воронина	Наталья	Игоревна	11.02.2001	34567890123	

таблица «Результат экзамена»

Код	Код абитуриента	Наименование экзамена	Результат	Номер свидетельства
1	101	ЕГЭ Математика	89	1234567
2	102	ЕГЭ Математика	90	2334556
3	102	ЕГЭ Физика	*NULL*	*NULL*
4	101	ЕГЭ Физика	*NULL*	*NULL*
5	101	ЕГЭ Русский язык	*NULL*	*NULL*

Использование значения NULL в условиях поиска

IS NULL – предикат, применяется для выявления равенства значения некоторого поля неопределенному значению (NULL).

IS NOT NULL– предикат, применяется для выявления неравенства значения некоторого поля неопределенному значению (NULL).

1) **SELECT** *column_names*
FROM *table_name*
WHERE *column_name* **IS NULL**;

2) **SELECT** *column_names*
FROM *table_name*
WHERE *column_name* **IS NOT NULL**;

Пример 1:

SELECT *
FROM Information
WHERE Telephone **IS NULL**;

Пример 2:

SELECT *
FROM Information
WHERE Telephone **IS NOT NULL**;

Ошибка: …. WHERE Telephone = NULL или …. WHERE Telephone = NOT NULL

т.к. любая операция сравнения с NULL (даже с самим с собой «NULL = NULL»), в результате сравнения выдает значение UNKNOWN (неизвестность).

Оператор SQL состоит из:

1) зарезервированных слов;
2) пользовательских названий.

Пользовательские названия могут быть идентификаторами или именами различных объектов базы данных.

Идентификаторы в Transact SQL должны состоять из символов алфавита, цифр или символа «_», начинаться с буквы и не могут содержать пробелы.

Возможно включение других символов (@, #, $ в СУБД SQL Server и #, $ в СУБД Oracle).

[ ] — идентификатор группировки слов в переменную.

Для обращения к таблице или полю таблицы можно указать составное имя:
Название_БД.имя_владельца.название_таблицы.название_поля или название_таблицы.название_поля

Каждая из этих характеристик отделяется от предыдущей точкой:
database.dbowner.table_name.column_name;
Промежуточное значения – имя владельца можно не указывать, если это не приводит к конфликтам имен.

Выбор базы данных для использования:
USE <название БД>
Например, use Sudent_Ivanov

Комментарии в языке Transact - SQL:
1. /*Текст комментария*/ –для записи многострочных комментариев.
2. --Текст комментария –для однострочных комментариев.

Подзапросы SQL (вложенные SQL запросы)

Вложенный запрос (подзапрос или внутренний запрос) — это запрос, вложенный в другой запрос.

Подзапрос может использоваться:

- в инструкции SELECT;
- в инструкции FROM;
- в условии WHERE.

Подзапрос может быть вложен в инструкции SELECT, INSERT, UPDATE или DELETE, а также в другой подзапрос.

Пример структуры вложенного запроса:

```
SELECT <поле или список полей>
FROM <таблица или список таблиц>
WHERE [поле]|[значение] оператор_сравнения|логический_оператор (SELECT <поле>
                                                                FROM <таблица>)
```

Например, нужно узнать оценки сотрудника Иванова:

```
SELECT *
FROM Evaluation
WHERE Cooperator_id =(SELECT Cooperator_id
                      FROM Cooperator
                      WHERE Surname = 'Иванов')
```

Синтаксис оператора SELECT (продолжение)

SELECT [ALL | DISTINCT | TOP <число>[PERCENT]] <поле или список полей>
FROM <таблица или список таблиц>
[WHERE <условие выборки >]
[GROUP BY <поле или список полей >]
[HAVING <условие выборки для группы строк>]
[ORDER BY <поле_1> [ASC | DESC] [, <поле_n > [ASC | DESC]]]

Таблица «Продукция»

Код	Наименование	Производитель	Стоимость за ед.	Количество, шт.
1	Мяч	Torneo	999	10
2	Лыжи	Fischer	5900	2
3	Коньки	Nordway	999	5
4	Лыжи	Salomon	5000	8
5	Сноуборд	Termit	8900	4
6	Лыжи	Madshus	3600	3

Нужно вывести список наименований имеющейся продукции:
SELECT DISTINCT Product_name **AS** Наименование
FROM Product

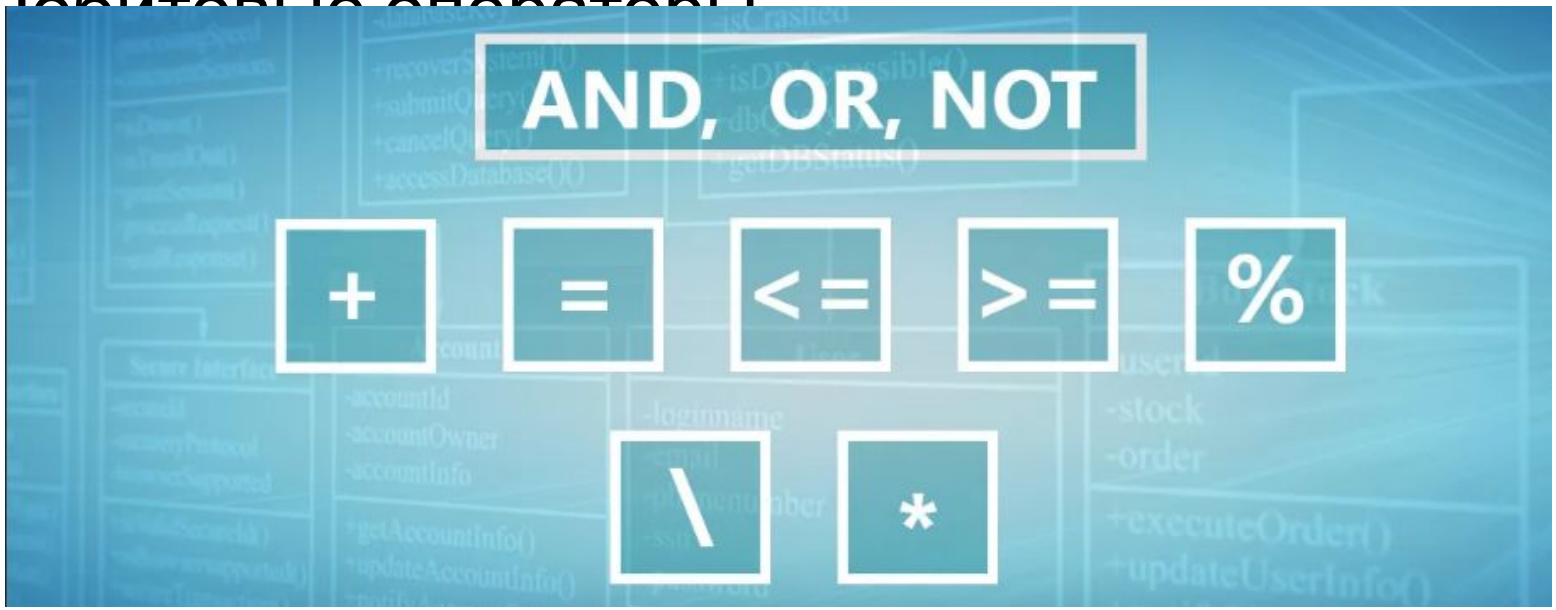
Наименование
Мяч
Лыжи
Коньки
Сноуборд

Нужно вывести первые три дорогие продукции:
SELECT TOP 3 *
FROM Product
ORDER BY Price **DESC**

ID	Product_name	Manufacturer	Price	Number
5	Сноуборд	Termit	8900	4
2	Лыжи	Fischer	5900	2
4	Лыжи	Salomon	5000	8

Операторы:

1. Арифметические операторы.
2. Операторы присваивания.
3. Операторы сравнения.
4. Логические операторы.
5. Унарные операторы.
6. Побитовые операторы



Арифметические операторы

Арифметические операторы выполняют математические операции над двумя значениями числовых типов данных или символьных.
Результатом выполнения любой арифметической операцию со значением ***NULL***,
всегда будет ***NULL***.

Арифметический оператор	Действие
+	Сложение
-	Вычитание
/	Деление
*	Умножение
%	Остаток от деления. Возвращает остаток от деления в виде целого числа

Арифметические операторы. Сложение

Table. Cooperator

ID	Surname	Name	Salary	…
1	Иванов	Иван	3000	
2	Сидоров	Петр	2500	
3	Синицына	Инна	5000	

Select Surname, Name, Salary+1000
From Cooperator

Surname	Name	Отсутствует имя столбца
Иванов	Иван	4000
Сидоров	Петр	3500
Синицына	Инна	6000

Table. Cooperator

ID	Surname	Name	Salary	Increase	…
1	Иванов	Иван	3000	1000,50	
2	Сидоров	Петр	2500	*NULL*	
3	Синицына	Инна	5000	2000	

Select Surname, Name, Salary+ Increase
From Cooperator

Surname	Name	Отсутствует имя столбца
Иванов	Иван	4000,50
Сидоров	Петр	*NULL*
Синицына	Инна	7000

Операторы присваивания

Оператор присваивания «=» присваивает значение переменной.

В качестве оператора для присваивания псевдонимов таблицам или заголовкам столбцов применяется ключевое слово **AS** (alias).

Table. Cooperator

ID	Surname	Name	Salary	…
1	Иванов	Иван	3000	
2	Сидоров	Петр	2500	
3	Синицына	Инна	5000	

UPDATE Cooperator
SET Salary=1000

ID	Surname	Name	Salary	…
1	Иванов	Иван	1000	
2	Сидоров	Петр	1000	
3	Синицына	Инна	1000	

Базовая таблица после обновления

Table. Cooperator

ID	Surname	Name	Salary	…
1	Иванов	Иван	3000	
2	Сидоров	Петр	2500	
3	Синицына	Инна	5000	

SELECT Surname **AS** Фамилия, Name **AS** Имя, Salary **AS** Стипендия
FROM Cooperator

Фамилия	Имя	Зарплата
Иванов	Иван	3000
Сидоров	Петр	2500
Синицына	Инна	5000

Результат выполненного запроса

Операторы сравнения

Операторы сравнения проверяют равенство или неравенство двух выражений. Результатом операции является булево значение – TRUE или FALSE.

СУБД сверяет все значения выбранного столбца с заданным и, если результат сравнения возвращает TRUE – выводит результат.

Оператор	Описание
=	если левый аргумент равен правому
>	если левый аргумент больше правого
<	если левый аргумент меньше правого
>=	если левый аргумент больше или равен правому
<=	если левый аргумент меньше или равен правому
<>	если левый аргумент не равен правому

Операторы сравнения

Table. Cooperator

ID	Surname	Name	Salary	…
1	Иванов	Иван	3000	
2	Сидоров	Петр	2500	
3	Синицына	Инна	5000	

```
SELECT Surname AS [Фамилия сотрудника], Name AS [Имя сотрудника]
FROM Cooperator
WHERE Salary >3000
```

Фамилия сотрудника	Имя сотрудника
Синицына	Инна

Table. Cooperator

ID	Surname	Name	Salary	…
1	Иванов	Иван	3000	
2	Сидоров	Петр	2500	
3	Синицына	Инна	5000	

```
SELECT Surname AS [Фамилия сотрудника], Name AS [Имя сотрудника]
FROM Cooperator
WHERE Salary >=3000
```

Фамилия сотрудника	Имя сотрудника
Иванов	Иван
Синицына	Инна

Побитовые операторы

Побитовые операторы выполняют побитовые действия над двумя выражениями с любым типом данных, относящихся к целочисленному.

Оператор	Название	Описание
&	Побитовое И	Если оба бита в определённой позиции равны 1, результат равен 1
|	Побитовое ИЛИ	Если хотя бы один бит в определённой позиции равен 1, результат равен 1
^	Побитовое исключающее ИЛИ	При разных значениях бит в определённой позиции результат равен 1.
~	Побитовое НЕ	Меняет значение бита в каждой позиции на противоположное.

Побитовое И

Id	Value_a	Value_b
1	2	3
2	4	8
3	2	*NULL*

0010 & 0011 = 0010

0100 & 1000 = 0000

Результат
2
0
NULL

SELECT Value_a **&** Value_b **AS** Результат
FROM Table1

Побитовое ИЛИ

Id	Value_a	Value_b
1	2	3
2	4	8

0010 | 0011 = 0011

0100 | 1000 = 1100

Результат
3
12

SELECT Value_a **|** Value_b **AS** Результат
FROM Table1

Побитовое «исключающее ИЛИ»

Id	Value_a	Value_b
1	2	3
2	4	8
3	2	*NULL*

0010 ^ 0011 = 0001

0100 ^ 1000 = 1100

Результат
1
12
NULL

SELECT Value_a **^** Value_b **AS** Результат
FROM Table1

Побитовое НЕ

Id	Value_a
1	2
2	4

~0010=1101

~0100 =1011

Результат
-3
-5

SELECT ~Value_a **AS** Результат
FROM Table1

Логические операторы

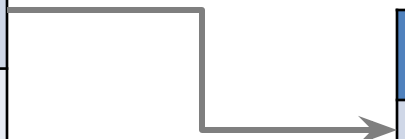
Логические операторы проверяют истину некоторого условия. Логические операторы возвращают булево значение **TRUE** или **FALSE**.

Логический оператор	Описание
AND	TRUE, если оба булевых выражения дают результат TRUE
OR	TRUE, если любое булево выражение равно TRUE
NOT	Обращает значение любого другого булева оператора
IN	TRUE, если операнд равен одному выражению из списка или одной или нескольким строкам, возвращаемым подзапросом
LIKE	TRUE, если операнд совпадает с шаблоном
BETWEEN	TRUE, если операнд находится внутри диапазона
EXISTS	TRUE, если подзапрос возвращает хотя бы одну строку
ALL	TRUE, если весь набор сравнений дает результат TRUE
ANY	TRUE, если хотя бы одно сравнение из набора дает результат TRUE
SOME	TRUE, если несколько сравнений из набора дают результат TRUE

Оператор AND (И)

Table. Cooperator

ID	Surname	Name	Salary	City	…
1	Иванов	Иван	*NULL*	Уфа	
2	Сидоров	Петр	2500	Ишимбай	
3	Синицына	Инна	5000	Уфа	
4	Федорова	Мария	0	Сибай	



Результат

Фамилия	Имя	Зарплата
Синицына	Инна	5000

Select Surname **AS** Фамилия, Name **AS** Имя, Salary **AS** Зарплата
From Cooperator
WHERE Salary>0 **AND** City =‘Уфа’

Оператор OR (ИЛИ)

Table. Cooperator

ID	Surname	Name	Salary	City	…
1	Иванов	Иван	*NULL*	Уфа	
2	Сидоров	Петр	2500	Ишимбай	
3	Синицына	Инна	5000	Уфа	
4	Федорова	Мария	0	Сибай	

Результат

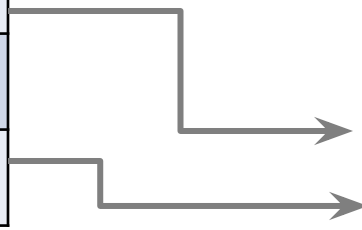
Фамилия	Имя	Зарплата	Город
Иванов	Иван	*NULL*	Уфа
Синицына	Инна	5000	Уфа
Федорова	Мария	0	Сибай

Select Surname **AS** Фамилия, Name **AS** Имя, Salary **AS** Зарплата, City **AS** Город
From Cooperator
WHERE City ='Сибай' **OR** City ='Уфа'

Оператор NOT (НЕ)

Table. Cooperator

ID	Surname	Name	Salary	City	…
1	Иванов	Иван	*NULL*	Уфа	
2	Сидоров	Петр	2500	Ишимбай	
3	Синицына	Инна	5000	Уфа	
4	Федорова	Мария	0	Сибай	



Результат

Фамилия	Имя	Зарплата	Город
Сидоров	Петр	2500	Ишимбай
Федорова	Мария	0	Сибай

Select Surname **AS** Фамилия, Name **AS** Имя, Salary **AS** Зарплата, City **AS** Город
From Cooperator
WHERE NOT City =‘Уфа’

Оператор IN

Операторы IN (равен любому из списка) и **NOT IN** (не равен ни одному из списка) используются для сравнения проверяемого значения поля с заданным списком. Этот список значений указывается в скобках справа от оператора IN.

Table. Cooperator

ID	Surname	Name	Salary	City	…
1	Иванов	Иван	*NULL*	Уфа	
2	Сидоров	Петр	2500	Ишимбай	
3	Синицына	Инна	5000	Уфа	
4	Федорова	Мария	0	Сибай	

Результат

Фамилия	Имя	Зарплата	Город
Иванов	Иван	*NULL*	Уфа
Синицына	Инна	5000	Уфа
Федорова	Мария	0	Сибай

```
Select Surname AS Фамилия, Name AS Имя, Salary AS Зарплата, City  AS Город
From Cooperator
WHERE City IN (‘Уфа’, ‘Сибай’)
```

Оператор NOT IN

Операторы IN (равен любому из списка) и **NOT IN** (не равен ни одному из списка) используются для сравнения проверяемого значения поля с заданным списком. Этот список значений указывается в скобках справа от оператора IN.

Table. Cooperator

ID	Surname	Name	Salary	City	…
1	Иванов	Иван	*NULL*	Уфа	
2	Сидоров	Петр	2500	Ишимбай	
3	Синицына	Инна	5000	Уфа	
4	Федорова	Мария	0	Сибай	

Результат

Фамилия	Имя	Зарплата	Город
Сидоров	Петр	2500	Ишимбай

```
Select Surname AS Фамилия, Name AS Имя, Salary AS Зарплата, City  AS Город
From Cooperator
WHERE NOT City IN (‘Уфа’, ‘Сибай’)
```

Оператор LIKE

Оператор **LIKE** просматривает строковые значения полей с целью определения, входит ли заданная в операторе подстрока (образец поиска) в символьную строку-значение проверяемого поля.

Можно применять шаблон искомого образца строки, использующий следующие символы:

- символ подчеркивания «_», определяет возможность наличия в указанном месте одного любого символа;
- символ «%» допускает присутствие в указанном месте проверяемой строки последовательности любых символов произвольной длины.

Если необходимо включить в образец символы «_» или «%» для этого с помощью ключевого слова **ESCAPE** нужно определить так называемый escape-символ, чаще для этой цели применяют символы "@" или "~".

Table. Cooperator

ID	Surname	Name	Salary	City	…
1	Иванов	Иван	*NULL*	Уфа	
2	Сидоров_	Петр	2500	Ишимбай	
3	Синицына	Инна	5000	Уфа	
4	Федорова	Мария	0	Сибай	

Результат

Фамилия	Имя	Зарплата
Синицына	Инна	5000

```
Select  Surname AS ‘Фамилия’, Name AS ‘Имя’, Salary AS ‘Зарплата’
From Cooperator
WHERE Surname LIKE ‘C%’ AND Surname NOT LIKE '%@_%' ESCAPE '@'
```

Оператор BETWEEN

Оператор BETWEEN используется для проверки условия вхождения значения поля в заданный интервал, то есть вместо списка значений атрибута этот оператор задает границы его изменения.

Table. Student

ID	Surname	Name	Salary	City	Birthday	…
1	Иванов	Иван	*NULL*	Уфа	01.01.2004	
2	Сидоров	Петр	2500	Ишимбай	20.01.2003	
3	Синицына	Инна	5000	Уфа	3.01.2003	
4	Федорова	Мария	0	Сибай	02.02.2004	

Результат

Surname	Name	City	Дата рождения
Иванов	Иван	Уфа	01.01.2004
Федорова	Мария	Сибай	02.02.2004

```
Select Surname AS Фамилия, Name AS Имя, City  AS Город, Birthday AS  [Дата рождения]
From Student
WHERE Birthday BETWEEN ’01.01.2004’ AND ’31.12.2004’
```

Table. Student

ID	Surname	Name	Salary	City	Birthday	…
1	Иванов	Иван	*NULL*	Уфа	01.01.2004	
2	Сидоров	Петр	2500	Ишимбай	20.01.2003	
3	Синицына	Инна	5000	Уфа	3.03.2003	
4	Федорова	Мария	0	Сибай	02.02.2004	

Результат

Surname	Name	City	Дата рождения
Сидоров	Петр	Ишимбай	20.01.2003

```
Select Surname AS Фамилия, Name AS Имя, City  AS Город, Birthday AS  [Дата рождения]
From Student
WHERE Birthday BETWEEN ’01.01.2003’ AND ’31.01.2003’
```

Оператор ANY (SOME)

Table. Student

Student_Id	Surname	Name	City	Birthday	…
1	Иванов	Иван	Уфа	01.01.2004	
2	Сидоров	Петр	Ишимбай	20.01.2003	
3	Синицына	Инна	Уфа	3.01.2003	
4	Федорова	Мария	Сибай	02.02.2004	

Table. Subject

Subj_Id	Name	Hour	…
1	Базы Данных	60	
2	Информатика	40	
3	Дискретная математика	60	

Задание: Вывести студентов у которых есть тройки, т.е. есть хотя бы одна тройка

```
SELECT Surname, Name, Birthday
FROM Student s
WHERE Student_Id=ANY(SELECT Student_Id
                     FROM Exam_mark e
                WHERE s.Student_Id=e.Student_Id AND Mark=3)
-----------------------------------------------
SELECT Surname, Name, Birthday
FROM Student s
WHERE Student_Id IN(SELECT Student_Id
                          FROM Exam_mark e
                    WHERE s.Student_Id=e.Student_Id AND Mark=3)
```

Table. Exam_mark

ID	Student_Id	Mark	Subj_Id	…
1	1	3	1	
2	1	4	2	
3	2	5	2	

Результат

Surname	Name	Birthday
Иванов	Иван	01.01.2004

Задание: Вывести студентов у которых нет троек , т.е. нет ни одной тройки

```
SELECT Surname, Name, Birthday
FROM Student
WHERE NOT Student_Id=ANY(SELECT Student_Id
                         FROM Exam_mark
WHERE Mark=3)
        AND Student_Id IN(SELECT Student_Id
FROM
Exam_mark )
```

Результат

Surname	Name	Birthday
Сидоров	Петр	20.01.2003

Оператор ALL

Table. Student

Student_Id	Surname	Name	City	Birthday	…
1	Иванов	Иван	Уфа	01.01.2004	
2	Сидоров	Петр	Ишимбай	20.01.2003	
3	Синицына	Инна	Уфа	3.01.2003	
4	Федорова	Мария	Сибай	02.02.2004	

Table. Subject

Subj_Id	Name	Hour	…
1	Базы Данных	60	
2	Информатика	40	
3	Дискретная математика	60	

Table. Exam_mark

ID	Student_Id	Mark	Subj_Id	…
1	1	3	1	
2	1	4	2	
3	2	5	2	
4	3	3	2	
5	2	5	1	

Задание: Вывести студентов у которых все оценки отличные, т.е. найти отличников

```
SELECT Surname, Name, Birthday
FROM Student s
WHERE 5=All(SELECT  Mark
      FROM Exam_mark e
      WHERE s.Student_Id=e.Student_Id)
AND s.Student_Id IN(SELECT Student_Id
            FROM Exam_mark)
```

Результат

Surname	Name	Birthday
Сидоров	Петр	20.01.2003

Задание: Вывести студентов у которых все оценки выше троек

```
SELECT Surname, Name, Birthday
FROM Student s
WHERE 3<All(SELECT Mark
      FROM Exam_mark e
      WHERE s.Student_Id=e.Student_Id)
      AND s.Student_Id IN(SELECT Student_Id
                  FROM Exam_mark )
```

Результат

ID	Surname	Name	Birthday
2	Сидоров	Петр	20.01.2003

Оператор EXISTS

Оператор **EXISTS** возвращает TRUE, если подзапрос возвращает хотя бы одну строку.

Задание: Вывести студентов которые сдавали экзамен

```
SELECT Surname, Name
FROM Student s
WHERE EXISTS (SELECT *
              FROM Exam_mark e                    WHERE
s.Student_Id=e.Student_Id)
```

Задание: Вывести студентов, которые не сдавали экзамены

```
SELECT Surname, Name
FROM Student s
WHERE  NOT EXISTS (SELECT *
                   FROM Exam_mark e
WHERE s.Student_Id=e.Student_Id)
```

Задание: Вывести студентов у которых есть хоть одна тройка

```
SELECT Surname, Name
FROM Student s
WHERE EXISTS (SELECT e.Student_Id
                    FROM Exam_mark e
          WHERE s.Student_Id=e.Student_Id AND e.Mark=3
          GROUP BY e.Student_Id)
```

Задание: Вывести студентов у которых все оценки больше тройки

```
SELECT Surname, Name
FROM Student s
WHERE EXISTS (SELECT e.Student_Id
                     FROM Exam_mark  e
WHERE s.Student_Id=e.Student_Id
               GROUP BY e.Student_Id
               HAVING MIN(e.Mark)> 3)
```

Унарные операторы

Унарные операторы выполняют операцию над одним выражением любого типа, относящимся к числу.

Оператор	Действие
+	Числовое значение становится положительным
-	Числовое значение становится отрицательным
~	Побитовое НЕ. Меняет значение бита в каждой позиции на противоположное.

Примеры, унарные операторы

Id	Value_a
1	2
2	4

Результат
-2
-4

SELECT -Value_a **AS** Результат
FROM Table1

Id	Value_a
1	2
2	4

~0010=1101

~0100 =1011

Результат
-3
-5

SELECT ~Value_a **AS** Результат
FROM Table1

Приоритет операторов

1. () – выражения в скобках.
2. +, -, ~ – унарные операторы.
3. *, /, % – арифметические операторы.
4. +, - – арифметические операторы.
5. =, >, <, >=, <=, <> – операторы сравнения.
6. ^ (побитное исключающее ИЛИ), & (побитное И), | (побитное ИЛИ).
7. NOT.
8. AND.
9. ALL, ANY, SOME,BETWEEN, IN, LIKE, OR.
10. = присваивание значения переменной.

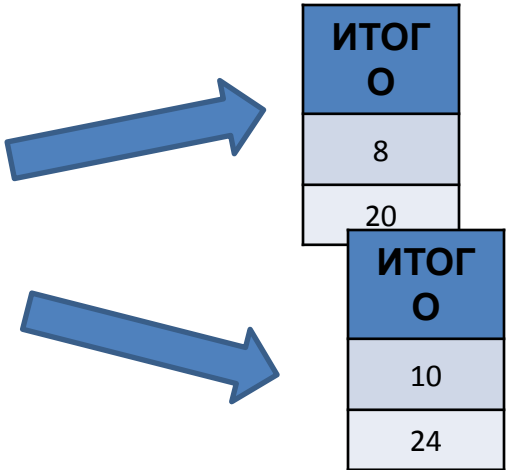
tab. Table1

Id	a	b
1	2	3
2	4	8

Пример. Вычислим сумму

```
SELECT a + b * 2 AS ИТОГО
FROM Table1
```

```
SELECT ( a + b) * 2 AS
ИТОГО
FROM Table1
```



Задачи

Table. Product

ID	Product_name	Manufacturer	Price	Number	Date_of_delivery
1	Мяч	Torneo	1000	10	11.01.2021
2	Лыжи	Fischer	5900	1	23.12.2020
3	Коньки	Nordway	1499	5	10.12.2020
4	Лыжи	Salomon	5000	8	1.12.2020
5	Сноуборд	Termit	8900	4	18.11.2020
6	Лыжи	Madshus	3000	3	10.11.2020

Какие данные будут получены в результате выполнения запросов? Какие операторы применялись в запросах?

1) SELECT *
FROM Product
WHERE Price<5000 OR NOT(Date_of_delivery<’20.12.2020’)

2) SELECT Product_name, Price*Number
FROM Product
WHERE Date_of_delivery BETWEEN ’20.12.2020’ AND ’01.02.2021’

3) SELECT *
FROM Product
WHERE NOT (Number>5) AND Product_name LIKE ‘_ы%’

4) SELECT Product_name, Price*Number
FROM Product
WHERE (Price*Number)>6000 AND Number IN(1,2,3)

Агрегатные функции

Агрегатные функции выполняют вычисления над значениями группы строк

Функция	Описание
SUM(поле таблицы)	Возвращает сумму значений
AVG(поле таблицы)	Возвращает среднее значение
MIN(поле таблицы)	Возвращает минимальное значение
MAX(поле таблицы)	Возвращает максимальное значение
COUNT(поле таблицы)	Возвращает количество записей без учета значения NULL
COUNT(*)	Возвращает количество записей с учетом значения NULL

Общая структура запроса с агрегатной (или агрегатными) функциями и GROUP BY:

SELECT [<поле или список полей группировки>,] <агрегатная функция 1> [, <агрегатная функция n>]
FROM <таблица>
[**GROUP BY** <поле или список полей группировки>]
[**HAVING** <условие выборки для группы строк>]

Агрегатная функция SUM

Table. Product

ID	Product_name	Manufacturer	Price	Number
1	Мяч	Torneo	999	10
2	Лыжи	Fischer	5900	2
3	Коньки	Nordway	1499	5
4	Лыжи	Salomon	5000	8
5	Сноуборд	Termit	8900	*NULL*
6	Лыжи	Madshus	3600	3

Посчитать сколько всего продукции в магазине:

```
SELECT SUM(Number) AS Сумма
FROM Product
```

Сумма
28

Посчитать на какую сумму, по каждому наименованию, имеется продукции в магазине:

```
SELECT Product_name AS 'Наименование продукции', SUM(Price* Number) AS Сумма
FROM Product
GROUP BY Product_name
```

```
WHERE Price IS NOT NULL AND Number IS NOT NULL
```

Наименование	Сумма
Мяч	9990
Лыжи	62600
Коньки	7495
Сноуборд	NULL

(5900*2)+(5000*8)+(3600*3)

Агрегатная функция AVG

Table. Product

ID	Product_name	Manufacturer	Price	Number
1	Мяч	Torneo	999	10
2	Лыжи	Fischer	5900	2
3	Коньки	Nordway	1499	5
4	Лыжи	Salomon	5000	8
5	Сноуборд	Termit	8900	*NULL*
6	Лыжи	Madshus	*NULL*	3

Вычислить среднею цену продукции в магазине:

```
SELECT AVG(Price) AS 'Средняя цена'
FROM Product
```

Средняя цена
4316

Вывести продукцию, средняя цена которых по каждому наименованию превышает значение 1000:

```
SELECT Product_name AS 'Наименование продукции', AVG(Price) AS 'Средняя цена'
FROM Product
GROUP BY Product_name
HAVING AVG(Price) >1000
```

Наименование	Средняя цена
Лыжи	5450
Коньки	1499
Сноуборд	8900

(5900+5000)/2

Агрегатные функции MAX, MIN

Table. Product

ID	Product_name	Manufacturer	Price	Number
1	Мяч	Torneo	999	10
2	Лыжи	Fischer	5900	2
3	Коньки	Nordway	1499	2
4	Лыжи	Salomon	5000	8
5	Сноуборд	Termit	8900	4
6	Лыжи	Madshus	*NULL*	3

Вывести информацию о продукции, которой меньше всего на складе:

```
SELECT *
FROM Product
WHERE Number =(SELECT MIN(Number)
               FROM Product)
```

ID	Product_name	Manufacturer	Price	Number
2	Лыжи	Fischer	5900	2
3	Коньки	Nordway	1499	2

Вывести максимальную и минимальную цену по каждому наименованию продукции, а также посчитать разницу в цене между ними:

```
SELECT Product_name AS 'Наименование продукции', MIN(Price) AS 'Минимальная цена', MAX(Price) AS 'Максимальная цена',  (MAX(Price) - MIN(Price)) AS 'Разница'
FROM Product
GROUP BY  Product_name
ORDER BY   2
```

Наименование	Минимальная цена	Максимальная цена	Разница
Мяч	999	999	0
Коньки	1499	1499	0
Лыжи	5000	5900	900
Сноуборд	8900	8900	0

Агрегатная функция COUNT

Table. Product

ID	Product_name	Manufacturer	Price	Number	Date_of_delivery
1	Мяч	Torneo	999	10	01.02.2021
2	Лыжи	Fischer	5900	2	02.02.2021
3	Коньки	Nordway	1499	5	21.01.2021
4	Лыжи	Salomon	5000	8	20.01.2021
5	Сноуборд	Termit	8900	4	11.01.2021
6	Лыжи	Madshus	*NULL*	3	11.01.2021

Варианты применения функции:

1) COUNT(поле)
2) COUNT(DISTINCT поле)
3) COUNT(*)

Посчитать количество различных наименований продукции:

```
SELECT COUNT(DISTINCT Product_name) AS 'Количество наименований'
FROM Product
```

Количество наименований
4

Посчитать количество поставок в январе 2021 года:

```
SELECT COUNT(DISTINCT Date_of_delivery) AS 'Количество поставок'
FROM Product
WHERE Date_of_delivery BETWEEN '01/01/2021'AND '31/01/2021'
```

Количество поставок
3

```
SELECT COUNT(Price) AS 'Количество без учета NULL', COUNT(*) AS 'Количество с учетом NULL'
FROM Product
```

Количество без учета NULL	Количество с учетом NULL
5	6

Многотабличные запросы

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321

```
SELECT coop.Coop_id , coop.Surname, coop.Name, dep.Dept_id ,dep.Name
FROM Cooperator coop, Department dep
```

Неверный результат

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	100	Администрация
3	Синицына	Инна	100	Администрация
1	Иванов	Иван	101	Информационный отдел
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел

Многотабличные запросы

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321

```
SELECT coop.Surname, coop.Name, dep.Dept_id ,dep.Name
FROM Cooperator coop, Department dep
WHERE coop.Dept_id =dep.Dept_id
```

Верный результат

Surname	Name	Dept_id	Name
Иванов	Иван	100	Администрация
Сидоров	Петр	101	Информационный отдел
Синицына	Инна	101	Информационный отдел

Многотабличные запросы, оператор соединения JOIN

Ключевое слово **JOIN** в SQL используется при построении запросов на выборку, обновление и удаление.
JOIN позволяет соединить поля из нескольких таблиц в одну таблицу вывода.
Соединение временное и целостность таблиц не нарушает.

Существует следующие типы оператора **JOIN**:

1. INNER JOIN (INNER обычно опускается);
2. OUTER JOIN (OUTER обычно опускается)
    1) LEFT JOIN;
    2) RIGHT JOIN;
    3) FULL JOIN.
3. CROSS JOIN.

Синтаксис:

1. JOIN: <левая_таблица> JOIN <правая_таблица> ON <условия_соединения>
2. LEFT JOIN: <левая_таблица> LEFT JOIN <правая_таблица> ON <условия_соединения>
3. RIGHT JOIN: <левая_таблица> RIGHT JOIN <правая_таблица> ON <условия_соединения>
4. FULL JOIN: <левая_таблица> FULL JOIN <правая_таблица> ON <условия_соединения>
5. CROSS JOIN: <левая_таблица> CROSS JOIN <правая_таблица>

Оператор соединения INNER JOIN

Задание: нужно вывести сотрудников и названия отделов за которыми они закреплены.

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101
4	Егоров	Валерий	12.01.1991	*NULL*
5	Воронина	Наталья	23.02.1993	103

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Проектный отдел	23431
103	Отдел кадров	45673

```
SELECT coop.Coop_id , coop.Surname, coop.Name, dep. Dept_id ,dep.Name
FROM Cooperator coop JOIN Department dep ON coop.Dept_id =dep.Dept_id
```

Результат

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел
5	Воронина	Наталья	103	Отдел кадров

Оператор соединения LEFT OUTER JOIN

Задание: нужно вывести данные по всем сотрудникам, чтобы сформировать список, например, для начисления ЗП. В список должны быть включены все сотрудники, даже если они на текущий момент не закреплены к конкретному отделу.

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101
4	Егоров	Валерий	12.01.1991	*NULL*
5	Воронина	Наталья	23.02.1993	103

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Проектный отдел	23431
103	Отдел кадров	456[illegible]

```
SELECT coop.Coop_id , coop.Surname, coop.Name, dep.Dept_id ,dep.Name
FROM Cooperator coop LEFT JOIN Department dep ON coop.Dept_id =dep.Dept_id
```

Результат

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел
4	Егоров	Валерий	*NULL*	*NULL*
5	Воронина	Наталья	103	Отдел кадров

Оператор соединения RIGHT OUTER JOIN

Задание: выяснить, какие отделы еще не сформированы, т.е. определить отделы, в которых еще нет сотрудников.

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101
4	Егоров	Валерий	12.01.1991	*NULL*
5	Воронина	Наталья	23.02.1993	103

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Проектный отдел	23431
103	Отдел кадров	456[illegible]

```
SELECT Coop_id , coop.Surname, coop.Name, dep.Dept_id ,dep.Name
FROM Cooperator coop RIGHT JOIN Department dep ON coop.Dept_id =dep.Dept_id
```

Результат

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел
NULL	*NULL*	*NULL*	102	Проектный отдел
5	Воронина	Наталья	103	Отдел кадров

Оператор соединения FULL OUTER JOIN

Задание: нужно получить все данные по сотрудникам и все данные по имеющимся отделам.

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101
4	Егоров	Валерий	12.01.1991	*NULL*
5	Воронина	Наталья	23.02.1993	103

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Проектный отдел	23431
103	Отдел кадров	456[illegible]

```
SELECT Coop_id , coop.Surname, coop.Name, dep.Dept_id ,dep.Name
FROM Cooperator coop FULL JOIN Department dep ON coop.Dept_id =dep.Dept_id
```

Результат

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел
4	Егоров	Валерий	*NULL*	*NULL*
5	Воронина	Наталья	103	Отдел кадров
NULL	*NULL*	*NULL*	102	Проектный отдел

Оператор соединения CROSS JOIN

Задание: нужно вывести все возможные варианты пошива моделей одежды с имеющимися материалами.

Table. Fabric

Fabric_id	Color	Kind_fabric
1	синий	драп
2	желтый	хлопок

Table. model_of_clothes

Model_id	Model_of_clothes
1	юбка
2	брюки

```
Select *
From Fabric CROSS JOIN Model_of_clothes
```

Результат

Fabric_id	Color	Kind_fabric	Model_id	Model_of_clothes
1	синий	драп	1	юбка
2	желтый	хлопок	1	юбка
1	синий	драп	2	брюки
2	желтый	хлопок	2	брюки

Типы данных MS SQL Server

Числовые типы данных:

BIT: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.

TINYINT: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.

SMALLINT: хранит числа от –32 768 до 32 767. Занимает 2 байта

INT: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.

BIGINT: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.

DECIMAL[(p[, s])] и NUMERIC[(p[, s])]: числа с фиксированной точностью и масштабом. При использовании максимальной точности числа могут принимать значения в диапазоне

от -10^38+1 до 10^38-1.

Синонимами типа DECIMAl по стандарту ISO является тип DEC(p, s).

Тип NUMERIC функционально эквивалентен типу DECIMAl.

p (точность) − максимальное общее число хранимых десятичных разрядов. Это число включает символы слева и справа от десятичной запятой. Точность должна быть значением в диапазоне от 1 до максимум 38. Точность по умолчанию составляет 18.

s (масштаб) − максимальное число хранимых десятичных разрядов справа от десятичной запятой. Это число отнимается от p для определения максимального количества цифр слева от десятичной запятой. Масштаб должен иметь значение от 0 до p и может быть указан только при заданной точности. По умолчанию масштаб принимает значение 0, поэтому 0 <= s <= p. Максимальный размер хранилища зависит от точности.

В зависимости от количества чисел после запятой переменная типа Decimal может занимать от 5 до 17 байт.

Точность	Байты хранилища
1–9	5
10–19	9
20–28	13
29–38	17

чения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу DECIMAL(10,4).

MONEY: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу DECIMAL(19,4).

FLOAT: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части.

Может иметь форму опредения в виде FLOAT(n), где n представляет число бит, которые используются для хранения десятичной части числа (мантиссы). По умолчанию n = 53.

Значение n	Точность	Объем памяти
1-24	7 цифр	4 байта
25-53	15 знаков	8 байт

т 4 байта. Эквивалентен типу FLOAT(24).

Типы данных MS SQL Server

Типы данных, представляющие дату и время:

DATE: ГГГГ-ММ-ДД. Хранит даты от 1 января 0001 года до 31 декабря 9999 года. Занимает 3 байта.

DATETIME: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.

DATETIME2: ГГГГ-ММ-ДД чч:мм:сс[.доли секунды], хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999.

Занимает от 6 до 8 байт в зависимости от точности времени.

Может иметь форму DATETIME2(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

SMALLDATETIME: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.

DATETIMEOFFSET: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

TIME: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.

Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд

Типы данных MS SQL Server

Строковые типы данных:

CHAR: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.

VARCHAR: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX).

Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

В отличие от типа CHAR если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столбце будет сохранено именно пять символов.

NCHAR: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15)

NVARCHAR: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта.

Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.

Еще два типа **TEXT** и **NTEXT** являются устаревшими и поэтому их не рекомендуется использовать. Вместо них применяются VARCHAR и NVARCHAR соответственно.

Типы данных MS SQL Server

Бинарные типы данных:

BINARY: хранит бинарные данные в виде последовательности от 1 до 8 000 байт.

VARBINARY: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до 2^31–1 байт при использовании значения MAX (VARBINARY(MAX)).

Еще один бинарный тип - тип IMAGE является устаревшим, и вместо него рекомендуется применять тип VARBINARY.

Остальные типы данных:

UNIQUEIDENTIFIER: уникальный идентификатор GUID (по сути строка с уникальным значением), который занимает 16 байт.

TIMESTAMP: некоторое число, которое хранит номер версии строки в таблице. Занимает 8 байт. В новых версиях СУБД заменен на rowversion.

CURSOR: представляет набор строк.

HIERARCHYID: представляет позицию в иерархии.

SQL_VARIANT: может хранить данные любого другого типа данных T-SQL.

XML: хранит документы XML или фрагменты документов XML. Занимает в памяти до 2 Гб.

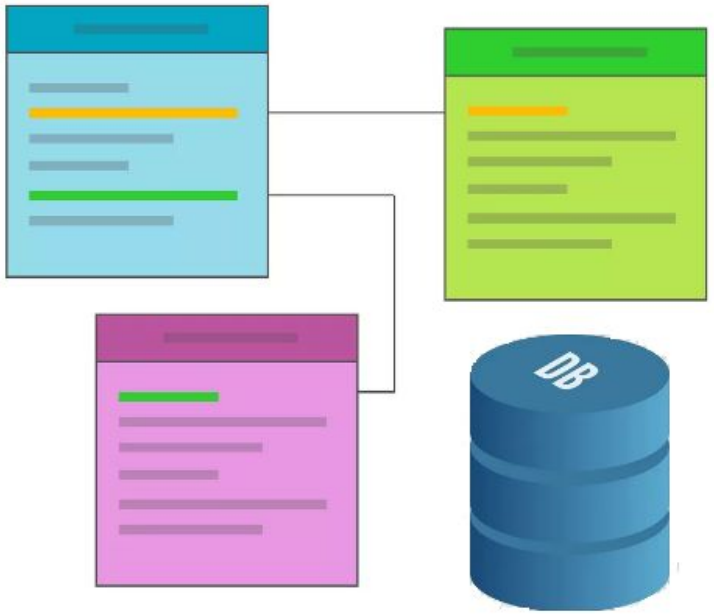
TABLE: представляет определение таблицы.

GEOGRAPHY: хранит географические данные, такие как широта и долгота.

GEOMETRY: хранит координаты местонахождения на плоскости.

Проектирование баз данных

Проектирование баз данных — процесс разработки схемы базы данных и определения необходимых ограничений целостности.



Основные задачи:

1) Сохранить необходимые данные о конкретной предметной области.
2) Получить данные по всем необходимым запросам.
3) Сократить избыточность дублирования данных.
4) Обеспечить целостности данных.

Проблемы, возникающие при проектировании БД

Нужно добавить новый отдел, а сотрудников пока не набрали=> **аномалия добавления**.

Т.к.чтобы добавить новый отдел без сотрудника нужно будет присвоить значение NULL в соответствующей строке поля *Табельный номер сотрудника*, но так как поле *Табельный номер сотрудника* является первичным ключом отношения, СУБД отклонит попытку добавления такой записи.

При изменении названия отдела или номера телефона => **аномалия модификации**.

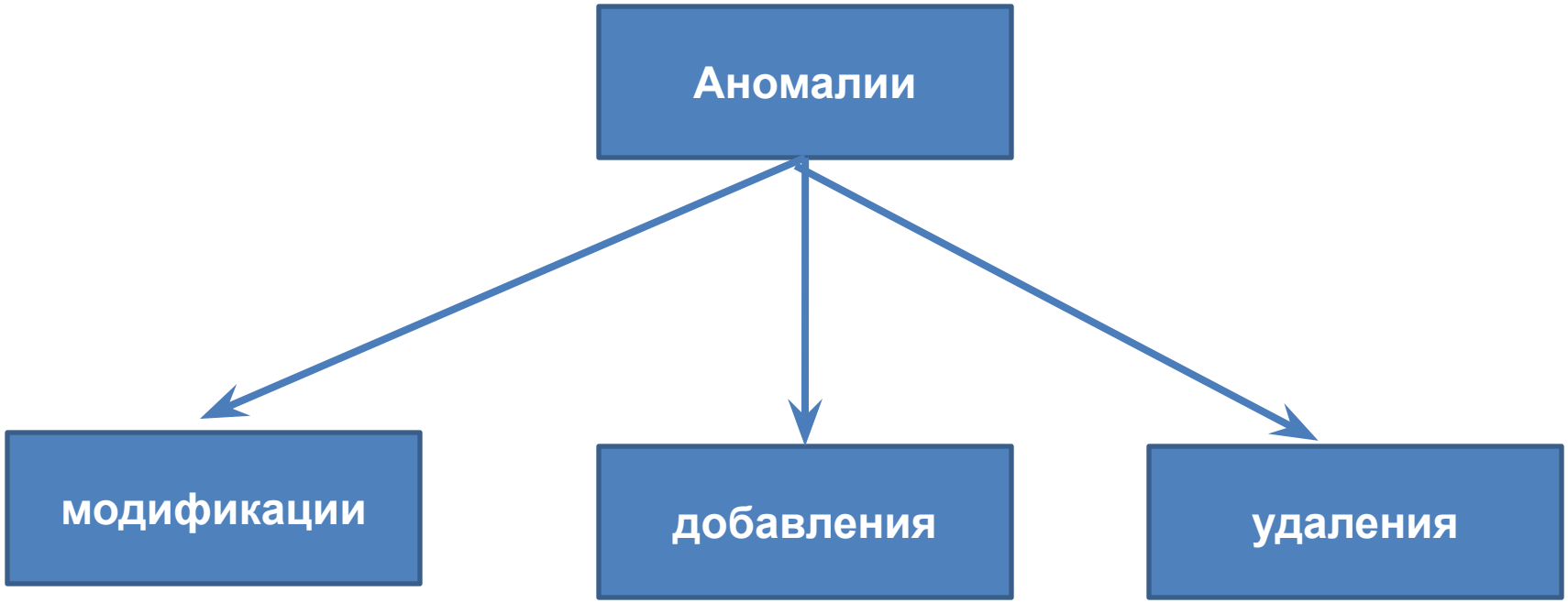
Если в отделе работает всего один сотрудник и он увольняется => **аномалия удаления**.

Таблица «Сотрудник отдела»

Табельный номер сотрудника (PK)	**Фамилия**	**Имя**	**Отчество**	**Отдел**	**Телефон отдела**
1	Иванов	Иван	Иванович	Администрация	12345
2	Сидоров	Петр	Петрович	Информационный отдел	54321
3	Синицына	Инна	Петровна	Отдел кадров	23431
4	Егоров	Валерий	Игнатьевич	Проектный отдел	45673
5	Воронина	Наталья	Игоревна	Отдел кадров	23431

Аномалии в таблицах БД

При неправильно спроектированной схеме реляционной БД могут возникнуть аномалии при выполнении операций модификации, добавления, удаления данных.



Решение проблемы

Декомпозиция

таблица «Сотрудник отдела»

Табельный номер сотрудника (PK)	Фамилия	Имя	Отчество	Отдел	Телефон отдела
1	Иванов	Иван	Иванович	Администрация	12345
2	Сидоров	Петр	Петрович	Информационный отдел	54321
3	Синицына	Инна	Петровна	Отдел кадров	23431
4	Егоров	Валерий	Игнатьевич	Проектный отдел	45673
5	Воронина	Наталья	Игоревна	Отдел кадров	23431

таблица «Сотрудник»

Код (PK)	Табельный номер сотрудника	Фамилия	Имя	Отчество	Код отдела (FK)
1	1	Иванов	Иван	Иванович	100
2	2	Сидоров	Петр	Петрович	101
3	3	Синицына	Инна	Петровна	102
4	4	Егоров	Валерий	Игнатьевич	103
5	5	Воронина	Наталья	Игоревна	102

таблица «Отдел»

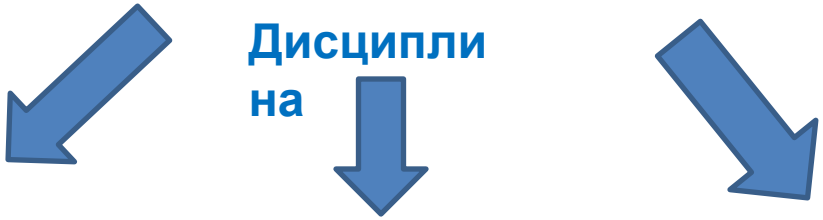
Код отдела (PK)	Отдел	Телефон отдела
100	Администрация	12345
101	Информационный отдел	54321
102	Отдел кадров	23431
103	Проектный отдел	45673

Проектирование баз данных

Нормализация – это процесс преобразования отношения в состояние, обеспечивающее лучшие условия выборки, добавления, изменения и удаления данных.

Цель: устранение избыточности данных в базе данных.

Не допускается наличие в таблице полей, названия которых входят в одно множество допустимых значений => не эффективная таблица.



Код	Семестр	Математика, часы	Физика, часы	Физическая культура, часы
11	1	60	46	42
12	2	62	36	32
13	3	54	56	40

Пример, таблица «Учебный план ВУЗа»

Избыточность данных – это дублирование данных, содержащихся в базе

Проектирование баз данных

таблица «Учебный план ВУЗа»

Код	Семестр	Математика, часы	Физика, часы	Физическая культура, часы
11	1	60	46	42
12	2	62	36	32
13	3	54	56	40

Код	Дисциплина	Часы	Семестр
11	Математика	60	1
12	Физика	62	1
13	Физическая культура	64	1
14	Математика	46	2
15	Физика	36	2
16	Математика	56	3
17	Физика	42	3
18	Физическая культура	32	2
19	Физическая культура	40	3

Вопрос: какие аномалии могут возникнуть в данной таблице?

Появляется избыточность данных => что можно предпринять? (см. следующий слайд)

Код	Дисциплина	Часы	Семестр
11	Математика	60	1
12	Физика	62	1
13	Физическая культура	64	1
14	Математика	46	2
15	Физика	36	2
16	Математика	56	3
17	Физика	42	3
18	Физическая культура	32	2
19	Физическая культура	40	3

таблица «Учебный план ВУЗа»

Вопросы:
1) Сколько таблиц будет получено в результате декомпозиции. С какими полями?
2) Какой вид связи будет между таблицами?

Декомпозиция

Код (PK)	Дисциплина
11	Математика
12	Физика
13	Физическая культура

таблица «Дисциплина»

Код (PK)	Код Дисциплины (FK)	Часы	Семестр
1	11	60	1
2	12	62	1
3	13	64	1
4	11	46	2
5	12	36	2
6	11	56	3
7	12	42	3
8	13	32	2
9	13	40	3

таблица «Учебный план ВУЗа»

Нормальные формы

- Эдгар Кодд.
- Нормальные формы:
  1) Первая нормальная форма (1NF, 1НФ).
  2) Вторая нормальная форма (2NF, 2НФ).
  3) Третья нормальная форма (3NF, 3НФ).
  4) Нормальная форма Бойса — Кодда (BCNF).
  5) Четвёртая нормальная форма (4NF).
  6) Пятая нормальная форма (5NF).
  7) Доменно-ключевая нормальная форма (DKNF).
  8) Шестая нормальная форма (6NF).

Первая нормальная форма (1НФ)

Определение. Отношение находится в 1НФ тогда и только тогда, когда все его атрибуты содержат только простые неделимые (атомарные) значения.

таблица «Счет в банке»

Номер отделения	Адрес отделения	Фамилия менеджера	Имя менеджера	Фамилия клиента	Имя клиента	Адрес клиента	Номер счета	Тип счета	Остаток на счете
11	Садовая, 3	Игнатюк, Тимошин	Иван, Тимур	Иванов	Иосиф	Пушкина, 10	1111, 2222	депозит	1000, 100000
11	Садовая, 3	Игнатюк	Иван	Сидоров	Семен	Кирова, 5	3333, 4455	текущий, депозит	2000, 10000
22	Кольцевая, 6	Федосевич	Федор	Петрова	Полина	Проспект, 7	4466	текущий	1000

таблица «Счет в банке»

Номер отделения	Адрес отделения	Фамилия менеджера	Имя менеджера	Фамилия клиента	Имя клиента	Адрес клиента	Номер счета	Тип счета	Остаток на счете
11	Садовая, 3	Игнатюк	Иван	Иванов	Иосиф	Пушкина, 10	1111	депозит	1000
11	Садовая, 3	Тимошин	Тимур	Иванов	Иосиф	Пушкина, 10	2222	депозит	100000
11	Садовая, 3	Игнатюк	Иван	Сидоров	Семен	Кирова, 5	3333	текущий	2000
11	Садовая, 3	Игнатюк	Иван	Сидоров	Семен	Кирова, 5	4455	депозит	10000
22	Кольцевая, 6	Федосевич	Федор	Петрова	Полина	Проспект, 7	4466	текущий	1000

Вторая нормальная форма (2НФ)

Определение. Отношение находится во 2НФ тогда и только тогда, когда соответствует 1НФ и не ключевые атрибуты *полностью зависят* от всего первичного ключа.

таблица «Счет в банке»

Номер отделения	Адрес отделения	Фамилия менеджера	Имя менеджера	Фамилия клиента	Имя клиента	Адрес клиента	Номер счета	Тип счета	Остаток на счете
11	Садовая, 3	Игнатюк	Иван	Иванов	Иосиф	Пушкина, 10	1111	депозит	1000
11	Садовая, 3	Тимошин	Тимур	Иванов	Иосиф	Пушкина, 10	2222	депозит	100000
11	Садовая, 3	Игнатюк	Иван	Сидоров	Семен	Кирова, 5	3333	текущий	2000
11	Садовая, 3	Игнатюк	Иван	Сидоров	Семен	Кирова, 5	4455	депозит	10000
22	Кольцевая, 6	Федосевич	Федор	Петрова	Полина	Проспект, 7	4466	текущий	1000

Декомпозиция

Таблица «Банк»

Код (РК)	Номер отделения	Адрес отделения	Фамилия менеджера	Имя менеджера
1	11	Садовая, 3	Игнатюк	Иван
2	11	Садовая, 3	Тимошин	Тимур
3	22	Кольцевая, 6	Федосевич	Федор

Код (РК)	Фамилия клиента	Имя клиента	Адрес клиента
1	Иванов	Иосиф	Пушкина, 10
2	Сидоров	Семен	Кирова, 5
3	Петрова	Полина	Проспект, 7

Таблица «Клиент банка»

Таблица «Банковский счет»

Код (РК)	Номер счета	Тип счета	Остаток на счете	Код клиента (FK)	Код Банка (FK)
100	1111	депозит	1000	1	1
101	2222	депозит	100000	1	2
102	3333	текущий	2000	2	1
103	4455	депозит	10000	2	1
104	4466	текущий	1000	3	3

Третья нормальная форма (3НФ)

Определение. Отношение находится в 3НФ тогда и только тогда, когда соответствует 2 НФ и все не ключевые атрибуты взаимно независимы.

Таблица «Банк»

Код (PK)	**Номер отделения**	**Адрес отделения**	**Фамилия менеджера**	**Имя менеджера**
1	11	Садовая, 3	Игнатюк	Иван
2	11	Садовая, 3	Тимошин	Тимур
3	22	Кольцевая, 6	Федосевич	Федор

Таблица «Банка»

Код отделения (PK)	**Номер отделения**	**Адрес отделения**
4	11	Садовая, 3
5	12	Кольцевая, 6

Таблица «Менеджер»

Код (PK)	**Код отделения (FK)**	**Фамилия менеджера**	**Имя менеджера**
1	4	Игнатюк	Иван
2	4	Тимошин	Тимур
3	5	Федосевич	Федор

таблица «Информация о счете»

Номер отделения	Адрес отделения	Фамилия менеджера	Имя менеджера	Фамилия клиента	Имя клиента	Адрес клиента	Номер счета	Тип счета	Остаток на счете
11	Садовая, 3	Игнатюк, Тимошин	Иван, Тимур	Иванов	Иосиф	Пушкина, 10	1111, 2222	депозит	1000, 100000
11	Садовая, 3	Игнатюк	Иван	Сидоров	Семен	Кирова, 5	3333, 4455	текущий, депозит	2000, 10000
22	Кольцевая, 6	Федосевич	Федор	Петрова	Полина	Проспект, 7	4466	текущий	1000

Таблица

Код отделения (PK)	Номер отделения	Адрес отделения
4	11	Садовая, 3
5	12	Кольцевая, 6

Таблица «Менеджер»

Код (PK)	Код отделения (FK)	Фамилия менеджера	Имя менеджера
1	4	Игнатюк	Иван
2	4	Тимошин	Тимур
3	5	Федосевич	Федор

Таблица «Банковский счет»

Код (PK)	Номер счета	Тип счета	Остаток на счете	Код клиента (FK)	Код менеджера (FK)
100	1111	депозит	1000	1	1
101	2222	депозит	100000	1	2
102	3333	текущий	2000	2	1
103	4455	депозит	10000	2	1
104	4466	текущи	1000	3	3

Таблица «Клиент банка»

Код (PK)	Фамилия клиента	Имя клиента	Адрес клиента
1	Иванов	Иосиф	Пушкина, 10
2	Сидоров	Семен	Кирова, 5
3	Петрова	Полина	Проспект, 7

Таблица «Грузовые морские перевозки»

Номер судна	Название	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия	Ф.И.О. капитана	Дата рождения	Вид судна	Грузоподъемность, тонны
1234567	Japan Bear	1101W	01.02.2019	NSP	12.03.2019	RAN	Иванов И.И.	01.01.1980	Сухогруз	500
7890123	Korea Bear	1102W	02.02.2020	VYP	10.03.2020	DXB	Сидоров С. С.	02.01.1980	Ролкер	1000
2345678	China Bear	1103W	03.03.2020	VVO	11.04.2020	ADL	Петров П.П.	01.01.1970	Универсал	1500
1234567	Japan Bear	1104W	04.04.2020	NSP	18.05.2020	RAN	Володин Е. И.	01.01.1975	Сухогруз	500

Задание:

1) Определите название ключа по типу и по способу задания.
2) Определите к какой нормальной форме нужно привести исходную таблицу. И что для этого нужно сделать?

Номер судна	Название	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия	Фамилия капитана	Имя капитана	Отчество капитана	Дата рождения	Вид судна	Грузоподъемность, тонны
1234567	Japan Bear	9201W	01.02.2019	NSP	12.03.2019	RAN	Иванов	Иван	Иванович	01.01.1980	Сухогруз	500
7890123	Korea Bear	9202W	02.02.2020	VYP	10.03.2020	DXB	Сидоров	Семен	Семенович	02.01.1980	Ролкер	1000
2345678	China Bear	9203W	03.03.2020	VVO	11.04.2020	ADL	Петров	Петр	Петрович	01.01.1970	Танкер	1500
1234567	Japan Bear	9204W	04.04.2020	NSP	18.05.2020	RAN	Володин	Евгений	Иванович	01.01.1975	Сухогруз	500

Таблица «Грузовые морские перевозки»

Номер судна	Название	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия	Фамилия капитана	Имя капитана	Отчество капитана	Дата рождения	Вид судна	Грузоподъемность, тонны
1234567	Japan Bear	9201W	01.02.2019	NSP	12.03.2019	RAN	Иванов	Иван	Иванович	01.01.1980	Сухогруз	500
7890123	Korea Bear	9202W	02.02.2020	VYP	10.03.2020	DXB	Сидоров	Семен	Семенович	02.01.1980	Ролкер	1000
2345678	China Bear	9203W	03.03.2020	VVO	11.04.2020	ADL	Петров	Петр	Петрович	01.01.1970	Танкер	1500
1234567	Japan Bear	9204W	04.04.2020	NSP	18.05.2020	RAN	Иванов	Иван	Иванович	01.01.1980	Сухогруз	500

Задание:

1) Определите к какой нормальной форме нужно привести исходную таблицу.
2) Определите структуру новых таблиц (сколько таблиц и какие поля) и как они будут связанны. Определите ключевые поля.

2 НФ

Таблица «Информация о судне»

Код (PK)	Номер судна	Название	Вид судна	Грузоподъемность, тонны
1	1234567	Japan Bear	Сухогруз	500
2	7890123	Korea Bear	Ролкер	1000
3	2345678	China Bear	Танкер	1500

Вопрос:
Нужно ли полученные таблицы приводить к следующей нормальной форме?

таблица «Информация о рейсе»

Код (PK)	Код судна (FK)	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия	Фамилия капитана	Имя капитана	Отчество капитана	Дата рождения
100	1	9201W	01.02.2019	NSP	12.03.2019	RAN	Иванов	Иван	Иванович	01.01.1980
101	2	9202W	02.02.2020	VYP	10.03.2020	DXB	Сидоров	Семен	Семенович	02.01.1980
102	3	9203W	03.03.2020	VVO	11.04.2020	ADL	Петров	Петр	Петрович	01.01.1970
103	1	9204W	04.04.2020	NSP	18.05.2020	RAN	Иванов	Иван	Иванович	01.01.1980

Таблица «Морские перевозки»

Код (PK)	Код судна (FK)	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия	Фамилия капитана	Имя капитана	Отчество капитана	Дата рождения
100	1	9201W	01.02.2019	NSP	12.03.2019	RAN	Иванов	Иван	Иванович	01.01.1980
101	2	9202W	02.02.2020	VYP	10.03.2020	DXB	Сидоров	Семен	Семенович	02.01.1980
102	3	9203W	03.03.2020	VVO	11.04.2020	ADL	Петров	Петр	Петрович	01.01.1970
103	1	9204W	04.04.2020	NSP	18.05.2020	RAN	Иванов	Иван	Иванович	01.01.1980

таблица «Капитан»

Код (PK)	Фамилия капитана	Имя капитана	Отчество капитана	Дата рождения
11	Иванов	Иван	Иванович	01.01.1980
12	Сидоров	Семен	Семенович	02.01.1980
13	Петров	Петр	Петрович	01.01.1970

таблица «Информация о рейсе»

Код (PK)	Код судна (FK)	Код капитана (FK)	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия
100	1	11	9201W	01.02.2019	NSP	12.03.2019	RAN
101	2	12	9202W	02.02.2020	VYP	10.03.2020	DXB
102	3	13	9203W	03.03.2020	VVO	11.04.2020	ADL
103	1	11	9204W	04.04.2020	NSP	18.05.2020	RAN

Результат приведения исходной таблицы к 3НФ

Таблица «Грузовые морские перевозки»

Номер судна	Название	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия	Ф.И.О. капитана	Дата рождения	Вид судна	Грузоподъемность, тонны
1234567	Japan Bear	1101W	01.02.2019	NSP	12.03.2019	RAN	Иванов И.И.	01.01.1980	Сухогруз	500
7890123	Korea Bear	1102W	02.02.2020	VYP	10.03.2020	DXB	Сидоров С. С.	02.01.1980	Ролкер	1000
2345678	China Bear	1103W	03.03.2020	VVO	11.04.2020	ADL	Петров П.П.	01.01.1970	Универсал	1500
1234567	Japan Bear	1104W	04.04.2020	NSP	18.05.2020	RAN	Володин Е. И.	01.01.1975	Сухогруз	500

Процесс нормализации

Таблица «Информация о судне»

Код (PK)	Номер судна	Название	Вид судна	Грузоподъемность, тонны
1	1234567	Japan Bear	Сухогруз	500
2	7890123	Korea Bear	Ролкер	1000
3	2345678	China Bear	Танкер	1500

таблица «Капитан»

Код (PK)	Фамилия капитана	Имя капитана	Отчество капитана	Дата рождения
11	Иванов	Иван	Иванович	01.01.1980
12	Сидоров	Семен	Семенович	02.01.1980
13	Петров	Петр	Петрович	01.01.1970

таблица «Информация о рейсе»

Код (PK)	Код судна (FK)	Код капитана (FK)	Номер рейса	Дата погрузки	Порт погрузки	Дата прибытия	Порт прибытия
100	1	11	9201W	01.02.2019	NSP	12.03.2019	RAN
101	2	12	9202W	02.02.2020	VYP	10.03.2020	DXB
102	3	13	9203W	03.03.2020	VVO	11.04.2020	ADL
103	1	11	9204W	04.04.2020	NSP	18.05.2020	RAN

Проектирование баз данных

Проектирование базы данных осуществляется в три этапа:

1) концептуальное проектирование (инфологическое);

2) логическое проектирование (даталогическое);

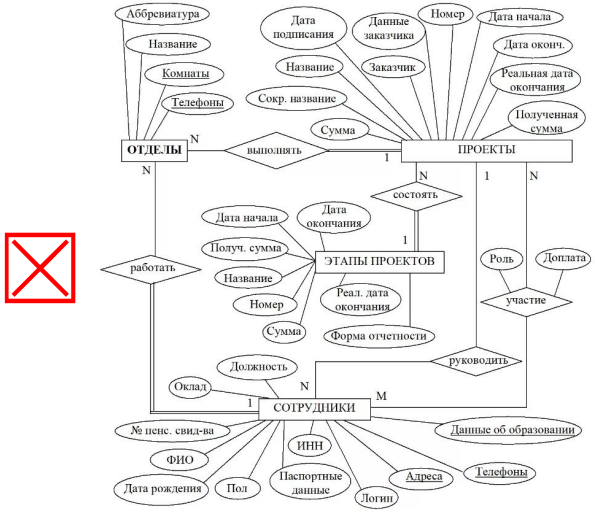
3) физическое проектирование.

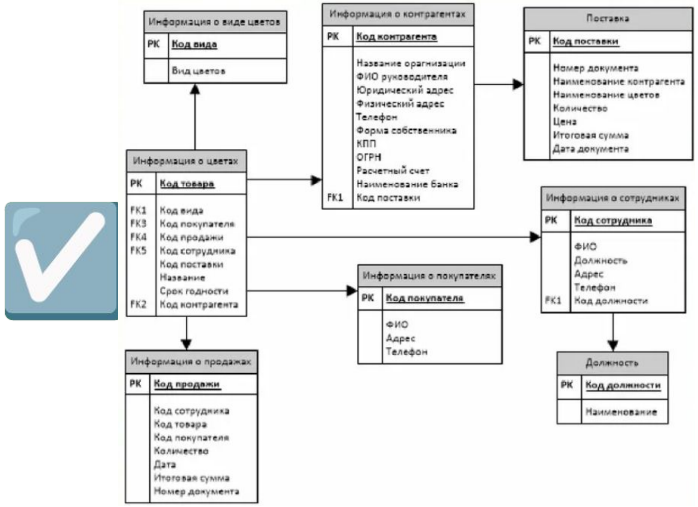
Концептуальное (инфологическое) проектирование

Цель – построение независимой от СУБД информационной структуры путем объединения требований пользователей.

Задачи:

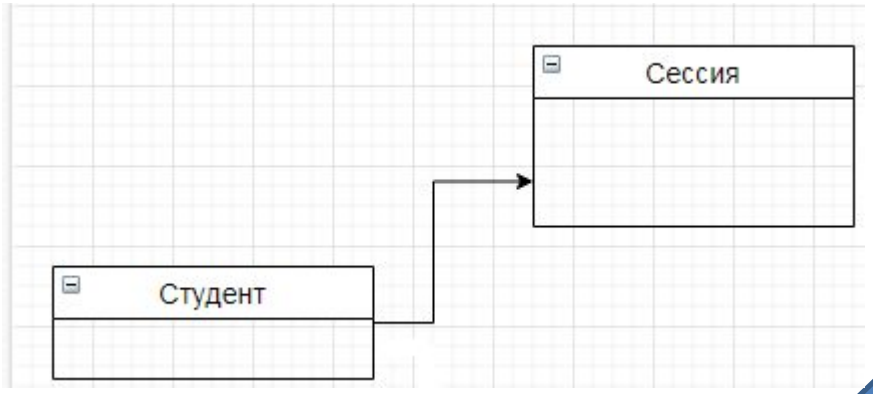
1. Определение сущностей и их документирование.
2. Задание связей между сущностями и их документирование.
3. Создание ER-модели (*entity-relationship diagram, ERD*) предметной области.
4. Определение атрибутов и их документирование.
5. Определение значений атрибутов и их документирование.
6. Определение первичных ключей для сущностей и их документирование.
7. Обсуждение концептуальной модели данных с конечными пользователями.



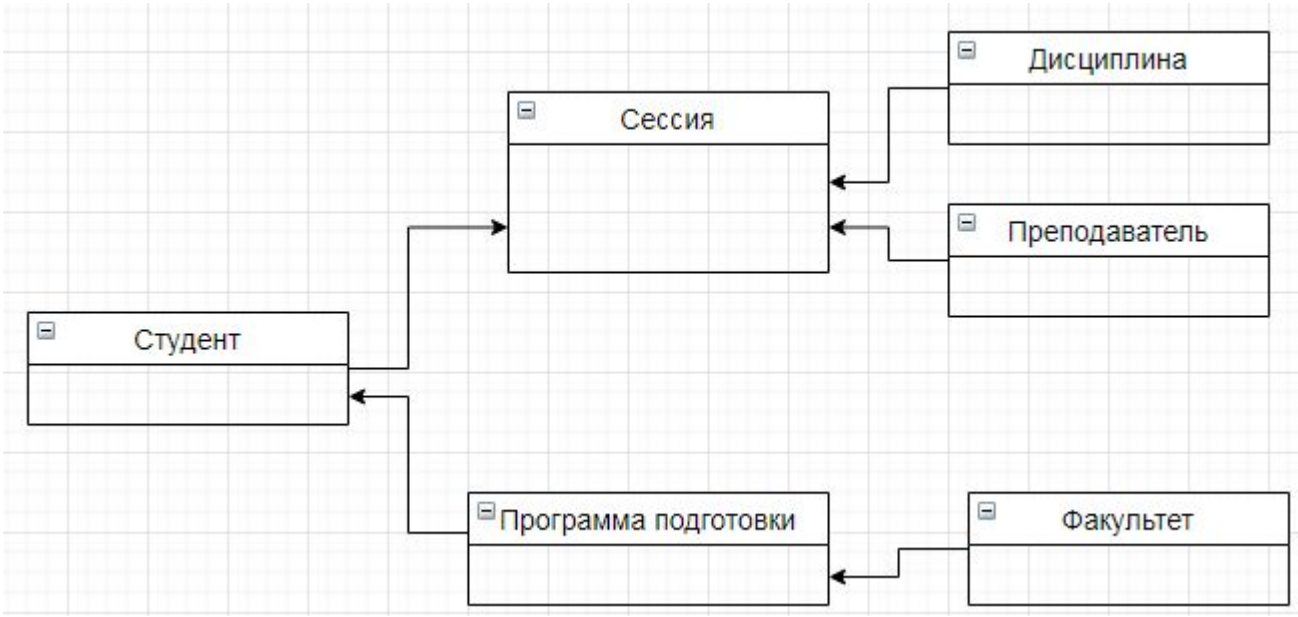


CASE системы (англ. computer-aided software engineering)— набор инструментов и методов программной инженерии для проектирования программного обеспечения.
IDEF1X (IDEF1 Extended) — Data Modeling — методология построения реляционных структур (баз данных), относится к типу методологий «Сущность-взаимосвязь» (ER — Entity-Relationship) и, как правило, используется для моделирования реляционных баз данных.
Erwin - программа для проектирования и документирования баз данных.
UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения.

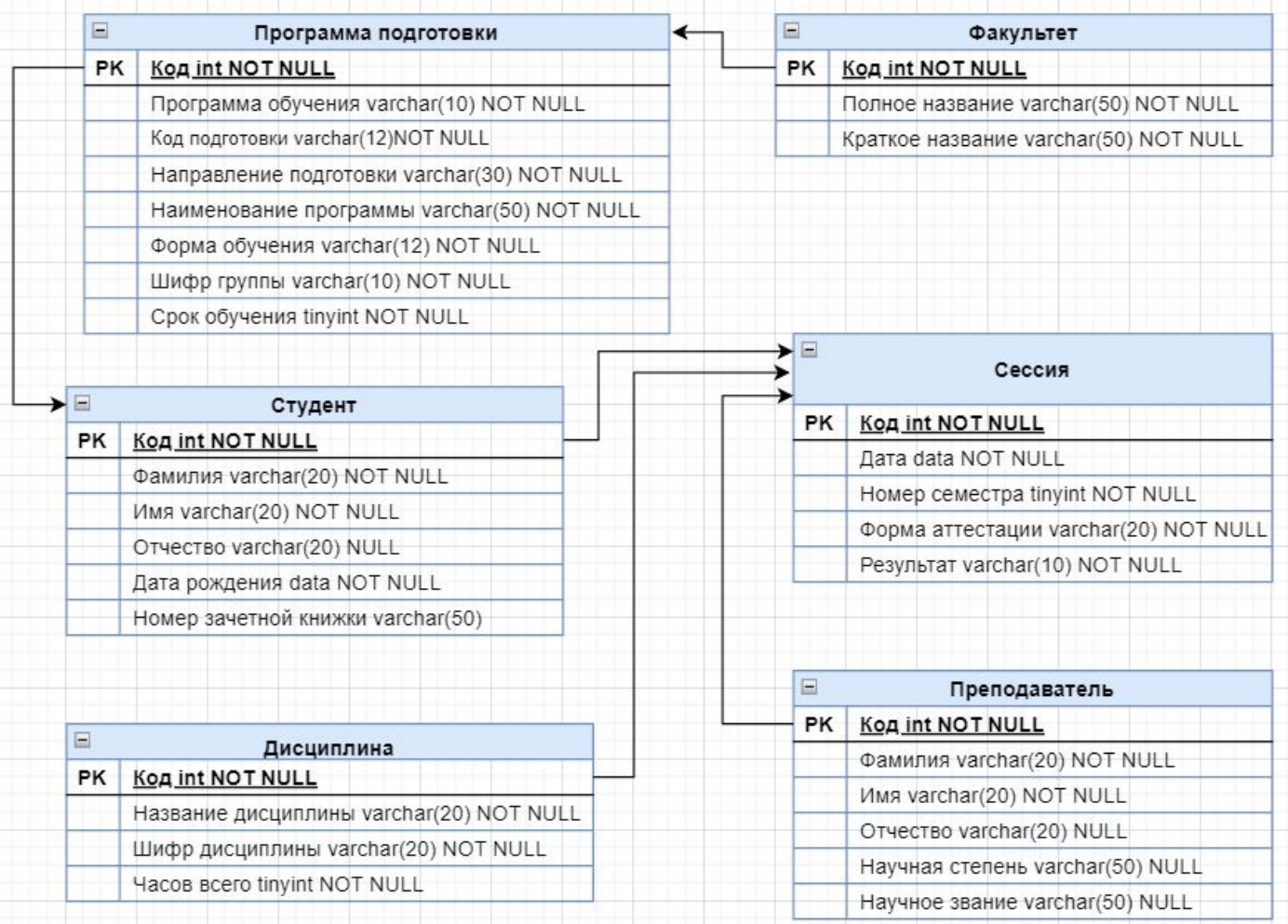
Концептуальное проектирование, пример, предметная область «Успеваемость студентов».



Уточнение



Концептуальное проектирование



Примеры значений для заполнения.

Программа обучения:
бакалавриат,
магистратура,
специалитет,
аспирантура
Форма обучения: очная, заочная, очно-заочная
Шифр группы:
БПО-18

Код и Направление подготовки (специальности):
09.03.01 Информатика и вычислительная техника;
07.03.04 Управление в технических системах
Наименование программы:
Программное обеспечение средств вычислительной техники и автоматизированных систем;
Системы и средства автоматизации технологических процессов.

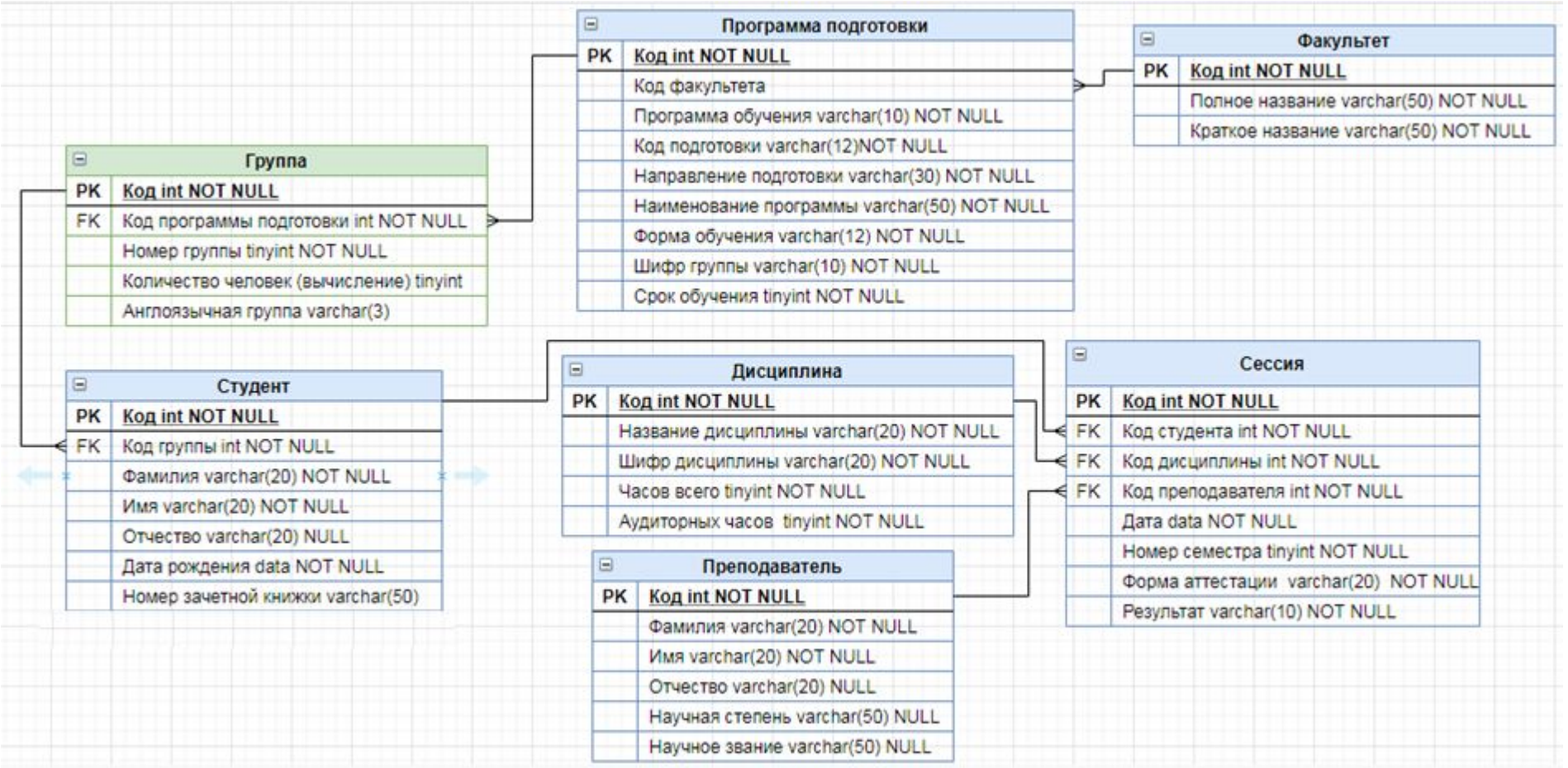
Логическое (даталогическое) проектирование

Цель – создание схемы базы данных на основе конкретной модели данных.

Задачи:

1. Выбор модели данных.
2. Определение вида связей между таблицами.
3. Раскрытие отношения "многие-ко-многим".
4. Нормализация таблиц.
5. Проверка логической модели данных на предмет возможности выполнения всех транзакций, предусмотренных пользователями.
6. Определение требований поддержки целостности данных и их документирование.
7. Создание окончательного варианта логической модели данных и обсуждение его с пользователями.

Логическое проектирование



Физическое проектирование

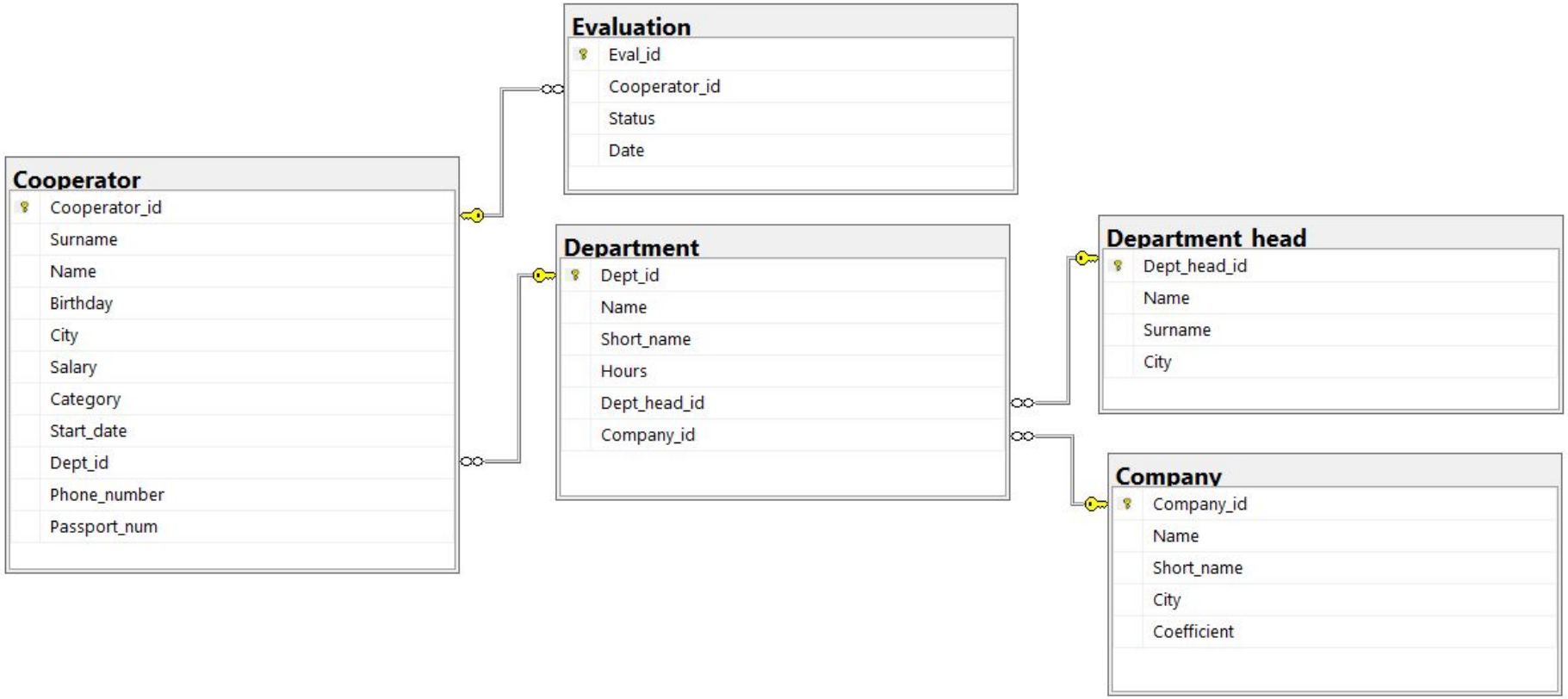
Цель – создание схемы базы данных для конкретной СУБД.

Задачи:

1. Проектирование таблиц базы данных средствами выбранной СУБД.
2. Реализация бизнес-правил в среде выбранной СУБД и их документирование.
3. Проектирование физической организации базы данных.
4. Разработка стратегии защиты базы данных.
5. Организация мониторинга функционирования базы данных и ее настройка.

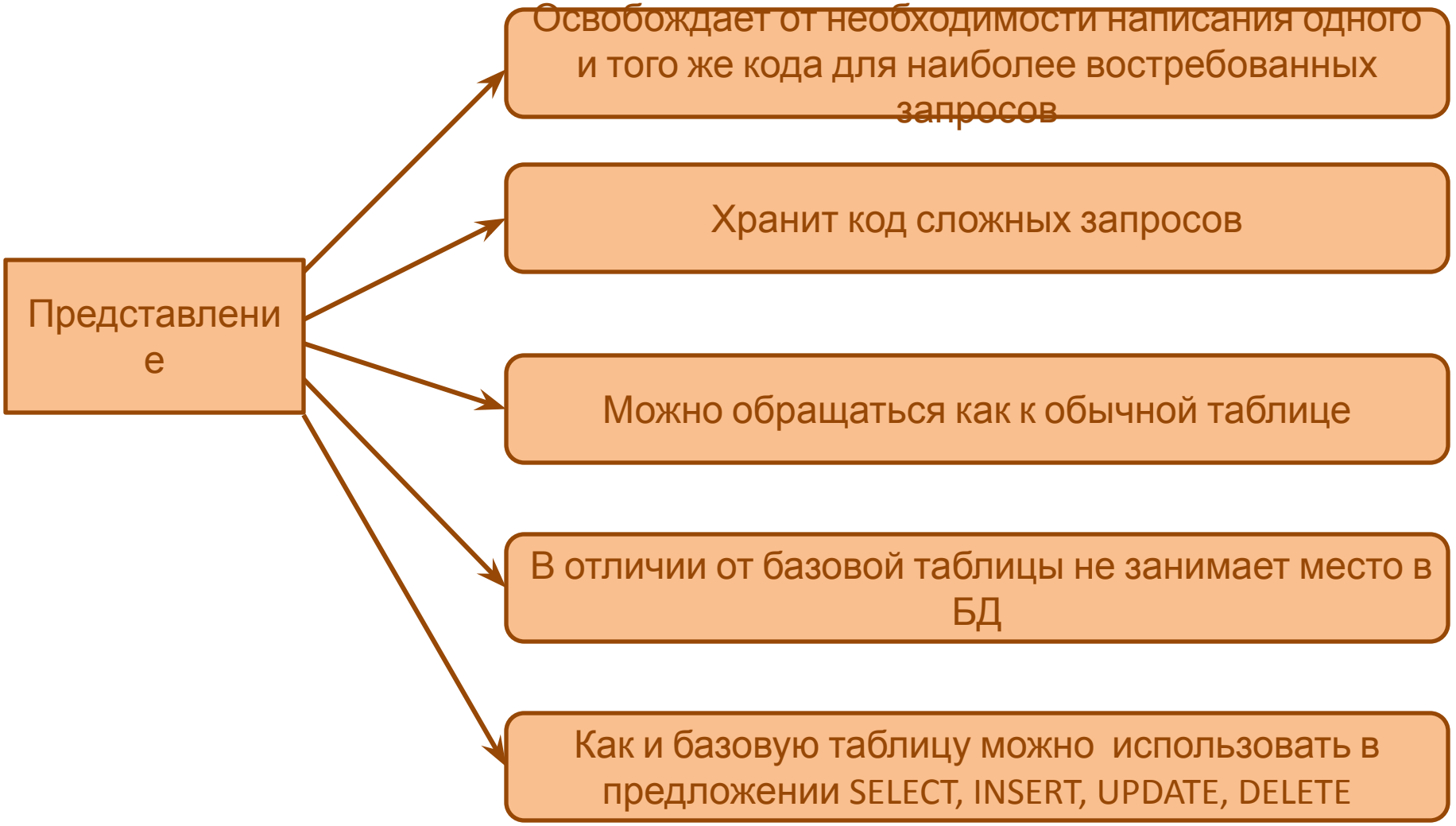
СХЕМА БАЗЫ ДАННЫХ

Схема данных – графическое отображение логической структуры базы данных в СУБД.

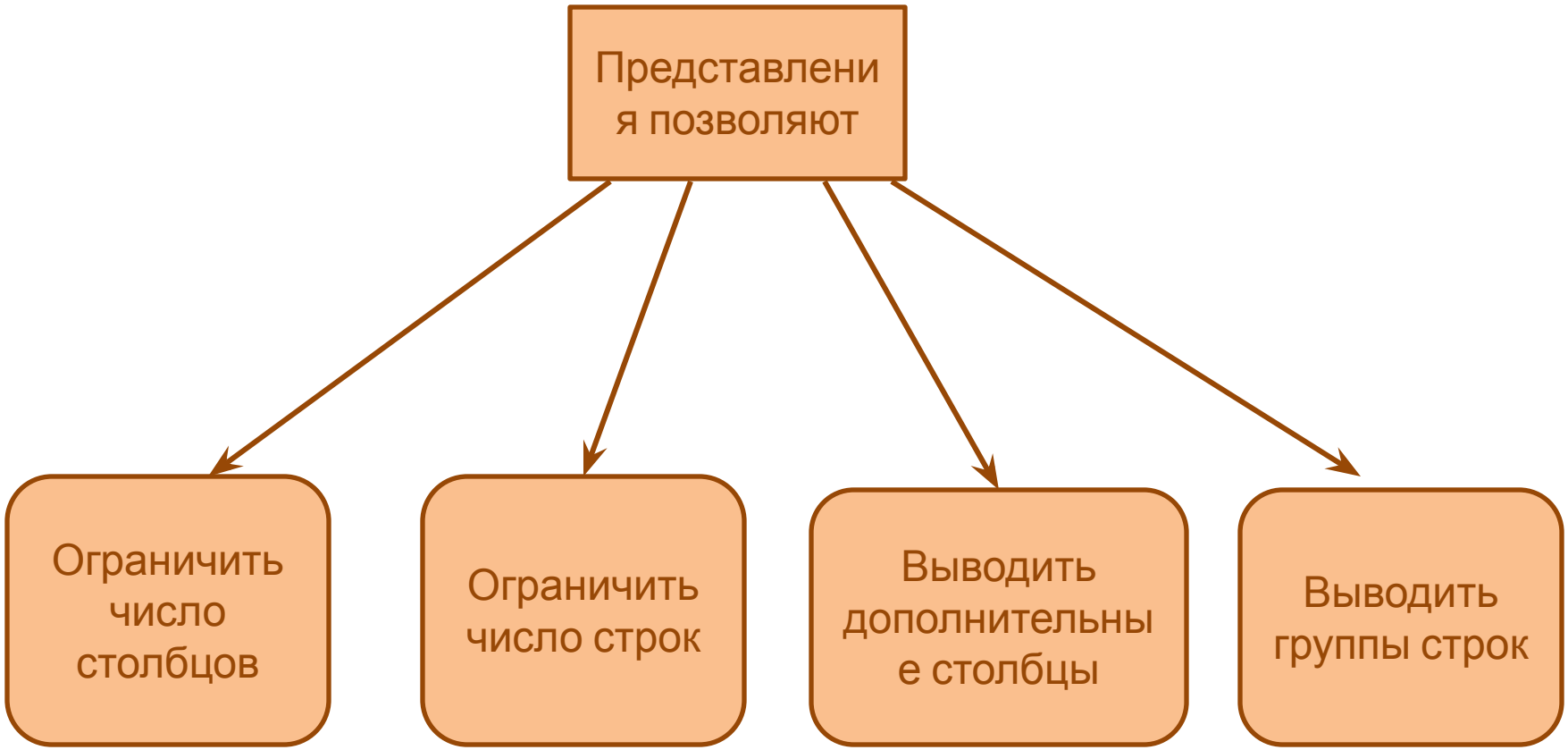


Представления/VIEW в SQL (Виртуальные таблицы)

Представления/VIEW или **виртуальная таблица** – это поименованная таблица, получаемая в результате выполнения команды SELECT с возможным изменением названия полей.



Представления/VIEW в SQL (Виртуальные таблицы)



Представления/VIEW (Виртуальные таблицы)

Синтаксис:

[CREATE|ALTER] VIEW <view_name> [(column_list)]

[WITH {ENCRYPTION | SCHEMABINDING| VIEW_METADATA}]

AS

SELECT <column_names>

FROM <table_names>

[WHERE condition] [WITH CHECK OPTION]

Опция WITH CHECK OPTION (в условии) указывает, что должна осуществляться проверка заданного условия при выполнении команды INSERT или UPDATE к представлению (виртуальной таблице). Если условие, заданное в описании представления, возвращает TRUE, то команда INSERT или UPDATE выполняется и происходит добавление записи или соответственно обновление записи в базовой таблице, иначе выполнение команды отклоняется.

Представления/VIEW (Виртуальные таблицы)

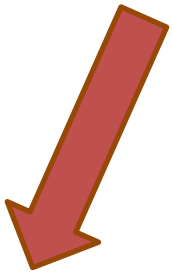
Например, нужно создать представление для просмотра информации о студентах из города Уфа:

```
CREATE VIEW Students_from_Ufa
AS
SELECT *
FROM Student
WHERE City = 'Уфа‘
```

Обращение к представлению **Students_from_Ufa**:

```
SELECT *
FROM Students_from_Ufa
```

Представления таблиц и представления столбцов



CREATE VIEW New_tab1_stud **AS**
SELECT *
FROM Student

Вызов представления:
SELECT *
FROM New_tab1_stud

CREATE VIEW New_tab2_stud (new_surname, new_name, new_stipend, new_kurs, new_city, new_ birthday, new_univ_id) **AS**
SELECT Surname, Name, Stipend, Kurs, city, Birthday, Univ_id
FROM Student

Вызов представления:
SELECT *
FROM New_tab2_stud

Представление с вложенным запросом, представление с группировкой

Задание: создайте представление, которое представит информацию по сотрудникам, работающим в отделе «IT».

```
CREATE VIEW Cooperator_IT
AS
SELECT  Surname, Name, Phone_number, Start_date, Category
FROM Cooperator
WHERE Dept_id IN (SELECT Dept_id
            FROM Department
            WHERE Name = 'IT')
```

Задание: создайте представление, которое представит отделы, входящие в состав компании с максимальным коэффициентом.

```
CREATE VIEW Company_max_coefficient
AS
SELECT Full_name, MAX(Coefficient)
FROM Department JOIN Company ON Department.Company_id =
Company.Company_id
GROUP BY Full_name
```

Представления с командами модификации

Создание представления Student_view:

```
CREATE VIEW Student_view
AS
SELECT Surname, Name, City, Birthday
FROM Student
```

```
UPDATE Student_view
SET City = 'Уфа'
WHERE Student_id = 6
```

```
*INSERT INTO Student_view
  VALUES ('Иванов', 'Иван', NULL, NULL)
```

```
DELETE FROM Student_view
WHERE Surname= 'Иванов' AND Name='Иван‘
```

Вызов представления Student_view:

```
SELECT *
FROM Student_view
```

** данные будут добавлены, если поле первичного ключа в таблице Student является автоинкрементным, иначе нужно добавить в представлении поле первичного ключа Student_Id и при вставке новой записи, добавить значение для этого поля*

Применение команд модификации к представлениям, скрывающим поля

Создадим таблицу «Издательство»:

```
CREATE TABLE Publishing_house
(Id_publ INT PRIMARY KEY IDENTITY (1, 1),
Name_publ VARCHAR(30) CHECK(Name_publ NOT LIKE '%[-^:\|.,/=><"@#?№&*+0-9a-zA-Z~%]%‘ESCAPE ‘~'),
City VARCHAR(20) CHECK(City NOT LIKE '%[^:\|.,/=><"@#?№&*+0-9a-zA-Z~%]%‘ESCAPE ‘~'),
Address VARCHAR(30) CHECK(Address  NOT LIKE '%[-^:\|.,/=><"@#?№&*+0-9a-zA-Z~%]%‘ESCAPE ‘~'),
Telephone VARCHAR(12) CHECK(Telephone LIKE '+[1-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
Raiting TINYINT CHECK(Raiting BETWEEN 1 AND 10))
```

Создадим представление Publishing_house_view:

```
CREATE VIEW Publishing_house_view
AS
SELECT  Name_publ, City, Raiting, Telephone
FROM Publishing_house
```

```
*INSERT INTO Publishing_house_view
 VALUES ('Эксмо-АСТ' , 'Москва', 8, NULL)
```

```
UPDATE Publishing_house_view
SET Raiting = Raiting+1
WHERE Name_publ= 'Эксмо-АСТ'
```

** данные будут добавлены, если поле первичного ключа в таблице Publishing_house является автоинкрементным*

```
DELETE FROM Publishing_house_view
WHERE Name_publ = 'Эксмо-АСТ'
```

Вызов представления Publishing_house_view:

```
SELECT *
FROM Publishing_house_view
```

Представления/VIEW, скрывающие строки и команды модификации

Создание представления Publishing_house_view_more_than_5 :

```
CREATE VIEW Publishing_house_view_more_than_5
AS
SELECT  Name_publ, City, Raiting, Telephone
FROM Publishing_house
WHERE Raiting>5;

INSERT INTO Publishing_house_view_more_than_5
VALUES ('Лань',  'Санкт-Петербург', 4, NULL);

UPDATE Publishing_house_view_more_than_5
SET Raiting = 3
WHERE Name_publ= 'Эксмо -АСТ'
```

При вызов представления, добавленная и обновленная записи не будут выведены, поскольку в этих записях в поле Raiting, значения не соответствуют условию заданному при создании представления:

```
SELECT *
FROM Publishing_house_view_more_than_5
```

Решением проблемы может стать добавление опции **WITH CHECK OPTION:**

```
CREATE VIEW Publishing_house_view_more_than_5
AS
SELECT  Name_publ, City, Raiting, Telephone
FROM Publishing_house
WHERE Raiting>5 WITH CHECK OPTION;
```

Агрегатные функции в представлениях

Представление для просмотра успеваемости:

```
CREATE VIEW Total_day
AS
SELECT Surname, Name, COUNT(Subj_id) AS Subj_quantity,
                AVG(CAST(Mark as float)) AS Mark_avg
FROM Exam_mark e, Student s
WHERE e.Student_id =s. Student_id
GROUP BY Surname, Name;
```

Увидеть данные можно с помощью простого запроса к представлению:

```
SELECT *
FROM Total_day;
```

```
CREATE VIEW Student_Subject  AS
SELECT Surname, Subj_name, Mark
FROM Student a, Exam_mark b, Subject c
WHERE a.Student_id = b.Student_id AND b.Subj_id = c.Subj_id
```

Создав представление Student_Subject , можно будет узнать, следующую информацию:

```
SELECT Surname
FROM Student_Subject
WHERE Subj_name = 'Базы
данных';
```

```
SELECT Subj_name
FROM Student_Subject
WHERE Surname =
'Иванов';
```

Ограничения в представлениях:

В представлении можно применять сортировку **ORDER BY** если указать оператор **OFFSET**.

Представление для просмотра сколько сотрудников однофамильцев есть в каждом отделе:

```
CREATE VIEW Count_cooperator_homonym
AS
SELECT d.Name 'Отдел', c.Surname 'Фамилия', COUNT(c.Surname) 'Количество'
FROM Cooperator c, Department d
WHERE c.Dept_id=d.Dept_id
GROUP BY d.Name, c.Surname
HAVING COUNT(c.Surname) >1
ORDER BY 3 DESC
OFFSET 0 ROWS
   FETCH NEXT 10 ROWS ONLY
```

Модифицируемые и немодифицируемые представления

Модифицируемое представление — это представление, относительно которого можно применить команды модификации UPDATE/INSERT/DELETE.

Критерии модифицируемости представления:

1) Содержит первичный ключ базовой таблицы, если ключ не автоинкрементный.
2) Не содержит агрегатные функции.
3) Не содержит ключевое слово **DISTINCT**.
4) Нет **GROUP BY** или **HAVING** в определении представления.
5) Нет подзапросов.
6) Может быть использовано в другом представлении, но это представление должно быть также модифицируемыми.
7) В определении представления нет полей вывода константы или выражения значений.
8) Если в описание представления более одной базовой таблицы => команды модификации могут быть применены только к полям одной из них.

Пример, модифицируемого представления с оператором соединения JOIN

Создадим таблицу «Книга» для хранения данных о книге:

```
CREATE TABLE Book
(Id_book INT PRIMARY KEY IDENTITY (1, 1),
Name_book VARCHAR(30) CHECK(Name_book NOT LIKE '%[-^:\|.,/=><"@#?№&*+0-9~%]%' ESCAPE '~'),
Surname_author VARCHAR(30) CHECK(Surname_author NOT LIKE '%[-^:\|.,/=><"@#?№&*+0-9a-zA-Z~%]%' ESCAPE '~'),
Name_author VARCHAR(30) CHECK(Name_author NOT LIKE '%[-^:\|.,/=><"@#?№&*+0-9a-zA-Z~%]%' ESCAPE '~'),
Publication_date date DEFAULT Getdate(),
Id_publ INT FOREIGN KEY REFERENCES Publishing_house)
```

Создадим представление для просмотра информации о книге и в каком издательстве была издана:

```
CREATE VIEW Publishing_house_view (Name_publ, City, Name_book,Surname_author,Name_author,
Publication_date,Id_publ)
AS
SELECT Name_publ, City, Name_book,Surname_author,Name_author, Publication_date, b.Id_publ
FROM Publishing_house p JOIN Book b ON p.Id_publ=b.Id_publ
```

Команда добавления данных не выполнится:

```
INSERT INTO Publishing_house_view (Name_publ, City, Name_book,Surname_author,Name_author, Publication_date, Id_publ)
VALUES ('Азбука-Аттикус', Null, 'Школа волшебства','Энде','Михаэль', '01.10.2021')
```

Команды добавления данных выполнятся успешно:

```
INSERT INTO Publishing_house_view (Name_publ, City)
VALUES ('Феникс', 'Ростов-на-Дону')/*запись добавиться в таблицуPublishing_house*/
```

и

```
INSERT INTO Publishing_house_view (Name_book,Surname_author,Name_author, Publication_date, Id_publ )
VALUES ('Базы данных','Иванов','Иван', '01.04.2020', 3) /*запись добавиться в таблицу Book*/
```

Чтобы добавленные записи увидеть через представление Publishing_house_view нужно добавить значение внешнего ключа в таблицу Book на соответствующее издательство.

Id_publ	Name_publ	City	Telephone	Raiting
1	Эксмо-АСТ	Москва	NULL	8
2	Феникс	Ростов-на-Дону	NULL	3
3	Лань	Санкт-Петербург	NULL	4

Id_book	Name_book	Surname_author	Name_author	Publication_date	Id_publ
1	Математика	Иванов	Иван	2020-04-01	2
2	Базы данных	Иванов	Иван	2020-05-01	2
4	Физика	Сидоров	Инакентий	2020-10-09	3

Вызов представления:

```
Select*
From Publishing_house_view
```

Результаты | Сообщения

	Name_publ	City	Name_book	Surname_author	Name_author	Publication_date
1	Феникс	Ростов-на-Дону	Математика	Иванов	Иван	2020-04-01
2	Феникс	Ростов-на-Дону	Базы данных	Иванов	Иван	2020-05-01
3	Лань	Санкт-Петербург	Физика	Сидоров	Инакентий	2020-10-09

Команда обновления данных выполнится:

```
UPDATE Publishing_house_view
SET Name_book='Молекулярная физика'
WHERE Id_publ =3
```

Результаты | Сообщения

	Name_publ	City	Name_book	Surname_author	Name_author	Publication_date	Id_publ
1	Феникс	Ростов-на-Дону	Математика	Иванов	Иван	2020-04-01	2
2	Феникс	Ростов-на-Дону	Базы данных	Иванов	Иван	2020-05-01	2
3	Лань	Санкт-Петербург	Молекулярная физика	Сидоров	Инакентий	2020-10-09	3

Команда удаления данных не выполнится:

```
DELETE FROM Publishing_house_view
WHERE Id_publ =7
```

Сообщения

```
Сообщение 4405, уровень 16, состояние 1, строка 2
Невозможно обновить представление или функцию "Publishing_house_view", так как изменение влияет на несколько базовых таблиц.
```

Команды изменения описания представления, удаления представления

ALTER VIEW <имя представления>
описание представления

DROP VIEW <имя представления>

Например, добавим условие в описании представления:

```
ALTER VIEW Publishing_house_view
AS
SELECT Name_publ,Name_book,Surname_author,Name_author
FROM Publishing_house p JOIN Book b ON p.Id_publ=b.Id_publ
WHERE Publication_date BETWEEN '01.01.2020' AND '01.05.2020'
```

Удалим ранее созданное представление:

```
DROP VIEW Publishing_house_view
```

Условный оператор CASE

Оператор CASE позволяет осуществить проверку условий и возвратить в зависимости от выполнения того или иного условия тот или иной результат.

Синтаксис простого CASE выражения:

```
CASE <input_expression>
    WHEN <when_expression> THEN <result_expression> [ ...n ]
    [ ELSE <else_result_expression>]
END
```

Синтаксис поискового CASE выражения:

```
CASE
    WHEN <boolean_expression> THEN <result_expression> [ ...n ]
    [ ELSE <else_result_expression>]
END
```

Пример с условным оператором CASE

Премировать сотрудников: IT-отдела - 50% от ЗП;
Бухгалтерии - 40% от ЗП;
Остальных отделов по 30% от ЗП.

```
SELECT d.Name 'Отдел ',Surname 'Фамилия',Salary 'Зарплата',
                    CASE d.Name -- проверяемое значение
                          WHEN 'IT' THEN '50%'
                          WHEN 'Бухгалтерия' THEN '40%'
                          ELSE '30%'
                    END 'Надбавка к ЗП',
               Salary/100*CASE d.Name
                          WHEN 'IT' THEN 50
                          WHEN 'Бухгалтерия' THEN 40
                          ELSE 30
                    END 'Премия‘
FROM Cooperator c, Department d
WHERE c.Dept_id=d.Dept_id
ORDER BY d.Name
```

Результаты | Сообщения

	Отдел	Фамилия	Зарплата	Надбавка к ЗП	Премия
1	IT	Иванов	10000	50%	5000
2	IT	Сидоров	11000	50%	5500
3	Бухгалтерия	Петров	15000	40%	6000
4	Бухгалтерия	Тимохина	18000	40%	7200
5	Отдел кадров	Егорова	12000	30%	3600

Пример с условным оператором CASE

Вывести средние оценки по студентам и расшифровать значения оценок

```
SELECT Surname, ROUND(AVG(CAST(Mark AS Float)),2)'Средняя оценка', [Расшифровка
оценки] =
CASE
     WHEN AVG(Mark)>4.5 THEN 'отл'
     WHEN AVG(Mark)>3.5 AND AVG(Mark)<=4.5 THEN 'хор'
     WHEN AVG(Mark)>=3 AND AVG(Mark)<=3.5 THEN 'удов'
     WHEN AVG(Mark)<3 THEN 'неуд'
     --ELSE 'без расшифровки'
END
FROM Exam_marks e, Student s
WHERE e.Student_Id=s.Student_Id
GROUP BY e.Student_Id,Surname
```

Результаты | Сообщения

	Фамилия	Средняя оценка	Расшифровка оценки
1	Иванов	3,67	хор
2	Сидоров	5	отл

Пример с условным оператором CASE

Посчитать сколько на каждом курсе оценок со значениями 2, 3, 4, 5

```
SELECT Surname, Name, Kurse 'Номер курса',
                    SUM(CASE WHEN Mark=2 THEN 1 ELSE 0 END) 'Количество 2',
                    SUM(CASE WHEN Mark=3 THEN 1 ELSE 0 END) 'Количество 3',
                    SUM(CASE WHEN Mark=4 THEN 1 ELSE 0 END) 'Количество 4',
                    SUM(CASE WHEN Mark=5 THEN 1 ELSE 0 END) 'Количество 5'
FROM Student s, Exam_mark e
WHERE s.STUDENT_ID=e.Student_Id
GROUP BY Kurse, Surname, Name
```

Результаты | Сообщения

	Surname	Name	Номер курса	Количество 2	Количество 3	Количество 4	Количество 5
1	Ерофеев	Игорь	1	0	0	1	2
2	Иванов	Иван	1	0	1	0	1
3	Сидоров	Александр	2	0	0	0	1

Объявление переменных в языке Transact-SQL

Задать имя и тип переменной в Transact-SQL можно с помощью оператора **DECLARE**.

Синтаксис:

```
DECLARE @local_variable data_type   data _type
```

Например,

```
DECLARE @Surname VARCHAR(20)
DECLARE @Surname VARCHAR(20), @Order_date DATE
```

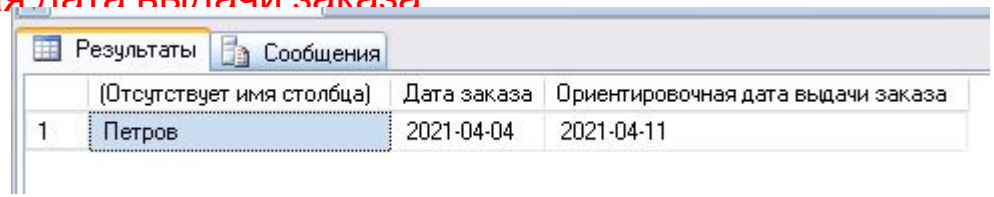
С помощью **SET** можно присвоить переменной некоторое значение:

Синтаксис:

```
SET @local_variable
```

Например,

```
DECLARE @Surname VARCHAR(20), @Order_date DATE, @Count_day TINYINT
SET @Surname ='Петров'
SET @Order_date = GETDATE()
SET @Count_day=7
SELECT @Surname 'Фамилия', @Order_date 'Дата заказа', DATEADD(d,@Count_day,
@Order_date) 'Ориентировочная дата выдачи заказа'
```



	(Отсутствует имя столбца)	Дата заказа	Ориентировочная дата выдачи заказа
1	Петров	2021-04-04	2021-04-11

PRINT в языке Transact-SQL

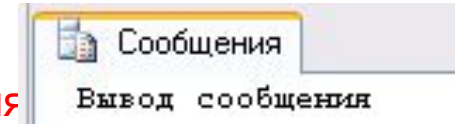
Оператор **PRINT** возвращает сообщение.

Синтаксис:

```
PRINT <msg_str|@local_variable|string_expr>
```

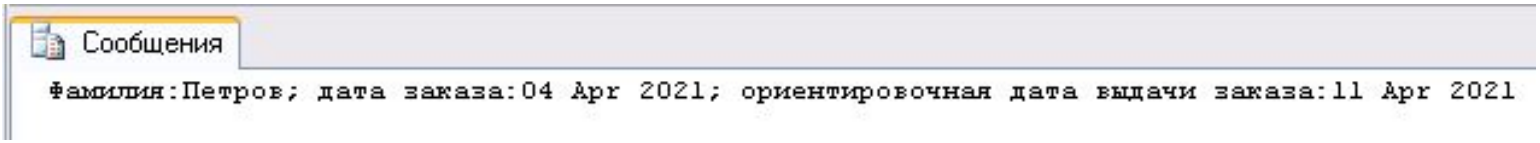
Пример 1:

```
PRINT 'Вывод сообщения
```



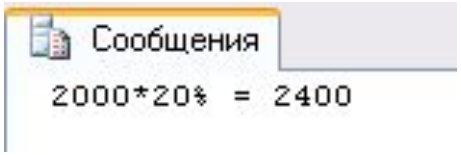
Пример 2:

```
DECLARE @Surname VARCHAR(20), @Order_date DATE, @Count_day TINYINT
SET @Surname ='Петров'
SET @Order_date = GETDATE()
SET @Count_day=7
PRINT 'Фамилия:'+@Surname+'; дата заказа:'+CONVERT(VARCHAR(12),@Order_date,106)+
'; ориентировочная дата выдачи заказа:'+
CONVERT(VARCHAR(12),DATEADD(d,@Count_day,@Order_date),106)
```



Пример 3:

```
DECLARE @stipend_increase FLOAT
SET @stipend_increase=(SELECT MIN(Stipend) FROM Student)
PRINT CAST(@stipend_increase AS VARCHAR(5))+'*20% = '+ CAST(@stipend_increase*1.2 AS VARCHAR(5))
```



Условный оператор IF...ELSE

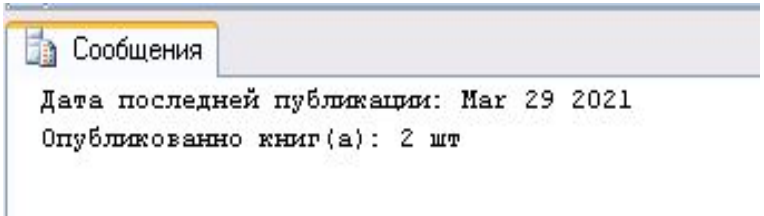
Конструкция **IF**...**ELSE** используется для наложения условий, определяющих, какие операторы T-SQL нужно выполнить.

Синтаксис:

```
IF Boolean_expression
        {sql_statement | BEGIN … statement_block … END}
[ ELSE {sql_statement | BEGIN … statement_block … END}]
```

Пример, определим были ли публикации в текущем месяце:

```
DECLARE @last_data DATE, @count SMALLINT
 SELECT @last_data = MAX(Publication_date),
      @count = SUM(CASE WHEN DATEDIFF(MONTH, Publication_date, GETDATE()) < 1 THEN 1 ELSE 0 END)
FROM Book
 IF @count > 0
   BEGIN
      PRINT 'Дата последней публикации: ' + CONVERT(VARCHAR, @last_data,100)
      PRINT 'Опубликовано книг(а): ' + CONVERT(NVARCHAR, @count) + 'шт'
   END
ELSE
   PRINT 'Нет публикаций'
```



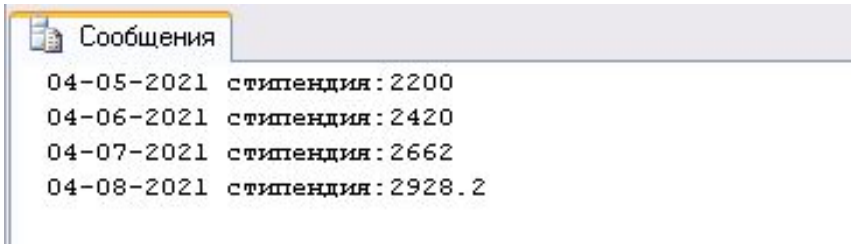
Цикл в языке Transact-SQL

Синтаксис:

```
WHILE Boolean_expression
{sql_statement|statement_block}
[ BREAK]
{sql_statement|statement_block}
[CONTINUE}
END
```

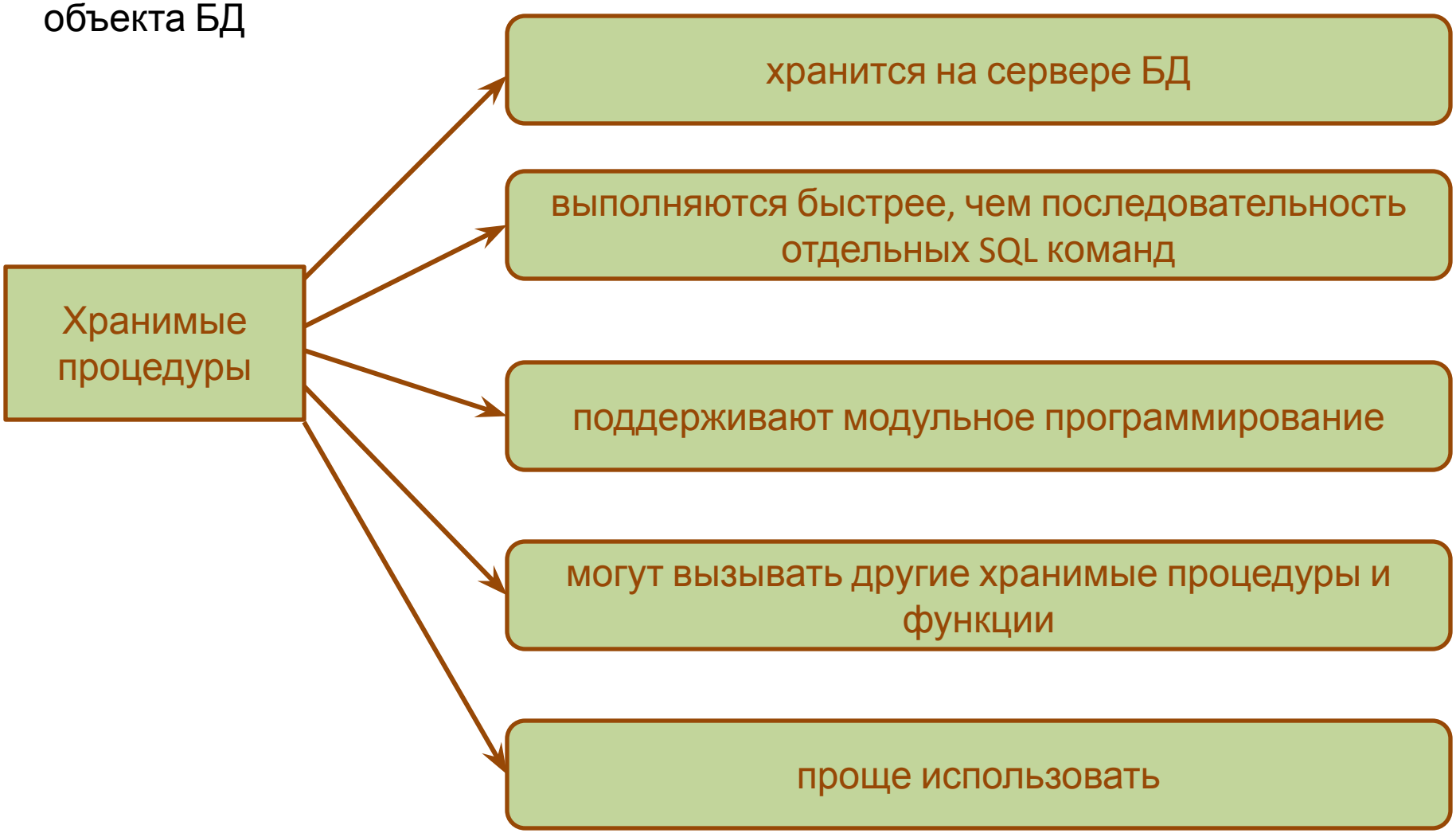
Пример, представить, какой будет стипендия если ежемесячно увеличивать на 10%:

```
DECLARE @count_month INT,  @start_date DATE, @min_stipend  FLOAT, @stipend_increase FLOAT
SET @count_month=4
SET @start_date = GETDATE()
SET @min_stipend=(SELECT MIN(Stipend) FROM Student)
SET @stipend_increase=@min_stipend
WHILE @count_month>0
BEGIN
    SET @count_month=@count_month-1
    SET @start_date = DATEADD(month, 1, @start_date)
    SET @stipend_increase =@stipend_increase*1.1
     --SELECT @start_date 'Дата',@stipend_increase 'Увеличенная стипендия'
     PRINT CONVERT(VARCHAR(12),@start_date,105)+ ' стипендия:‘
                                                +CAST(@stipend_increase AS VARCHAR(12))
END
```



Хранимые процедуры в Transact−SQL

Хранимая процедура (Stored Procedure) – это именованный набор команд языка Transact-SQL, хранящийся на сервере в качестве самостоятельного объекта БД



Хранимые процедуры в Transact–SQL

Хранимые процедуры в Transact−SQL

Синтаксис создания (изменения) процедуры:
{CREATE|ALTER} PROCEDURE|PROC procedure_name [; number]
[{@name_paramet data_type} **[VARYING] [=DEFAULT][OUTPUT]]**[,...n]
[WITH {RECOMPILE|ENCRYPTION|RECOMPILE, ENCRYPTION}]
[FOR REPLICATION]
AS

Параметр **procedure_name** задает имя хранимой процедуры.
Параметр **@name_parametr** является параметром процедуры (формальным аргументом), тип данных определяется data_type.
VARYING — указывает результирующий набор, поддерживаемый в качестве выходного параметра.
DEFAULT определяет значение по умолчанию для соответствующего параметра процедуры. Значением по умолчанию также может быть NULL.
Опция **OUTPUT** указывает, что параметр процедуры является возвращаемым.
Опция **WITH RECOMPILE** задается если требуется компилировать хранимую процедуру при каждом ее вызове.
WITH ENCRYPTION задает шифрование исходного текста инструкции CREATE PROCEDURE.
FOR REPLICATION указывает, что процедура создается для репликации. Процедура, созданная с параметром FOR REPLICATION, используется в качестве фильтра и выполняется только в процессе репликации. Параметры не могут быть объявлены, если указан параметр FOR REPLICATION.

Хранимые процедуры в Transact–SQL

Синтаксис вызова:
[EXECUTE|EXEC] имя_процедуры [;номер]
[[@имя_параметра=]{значение | @имя_переменной}
[OUTPUT]|[DEFAULT]][,...n]

Синтаксис удаления процедуры:
DROP PROCEDURE {имя_процедуры}
[,...n]

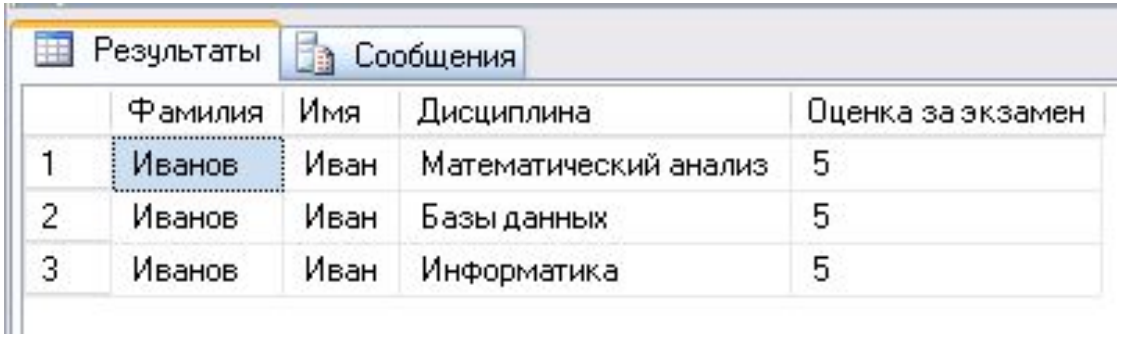
Пример создания процедуры без параметров

Создадим процедуру для получения названий экзаменов и оценок, полученных студентом Ивановым:

```
CREATE Procedure ExamResults
AS
SELECT Surname AS 'Фамилия', Name AS 'Имя', Subj_Name AS 'Дисциплина', Mark AS
'Оценка за экзамен'
FROM Student INNER JOIN Exam_marks ON Student.Student_Id=Exam_marks.Student_Id
            INNER JOIN Subject ON Exam_marks.Subj_Id = Subject.Subj_Id
WHERE Student.Surname = 'Иванов'
```

Вызов процедуры:

```
EXECUTE ExamResults
```



	Фамилия	Имя	Дисциплина	Оценка за экзамен
1	Иванов	Иван	Математический анализ	5
2	Иванов	Иван	Базы данных	5
3	Иванов	Иван	Информатика	5

Пример создания процедуры для вычисления значений поля *Cost_product*

Table. Product

ID_product (PK)	Name_product	Cost_product	Date_of_manufacture
1	Шоколад	NULL	20.01.2021
2	Мармелад	NULL	21.01.2021

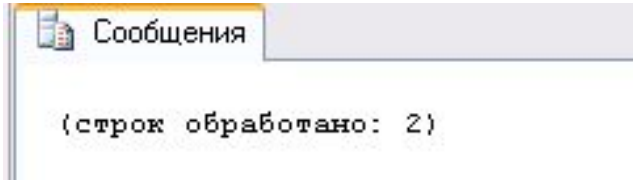
Table. Ingredient

ID_ingredient (PK)	Name_ingredient
11	какао-бобы
12	орехи
13	патока

Table. Composition

ID_Composition (PK)	ID_ingredient (FK)	Id_product (FK)	Cost_unit	Count_unit
1	11	1	900	2
2	12	1	300	1
3	13	2	200	1

```
CREATE PROCEDURE Cost_product
AS
Update Product
SET Cost_product=(SELECT SUM(Cost_unit*Count_unit)
        FROM Composition
        WHERE Product.Id_product= Composition.Id_product
        GROUP BY Id_product)
```



Вызов процедуры для расчета значений в поле Cost_product таблицы Product:

EXEC Cost_product или Cost_product

Пример создания процедуры с входным параметром

Создать процедуру для увеличения размера стипендии у студентов со средней оценкой больше 4-х. Процент увеличения стипендии входной параметр:

```
CREATE PROCEDURE Reduce_stipend
@Persent float
AS
UPDATE Student
SET Stipend = Stipend*(100+@percent)/100
WHERE Student_Id IN (SELECT Student_Id
                     FROM Exam_marks
                          WHERE Mark IS NOT NULL AND Mark!=0
                               GROUP BY Student_Id
                    HAVING AVG(Mark)>4)
```

Обращения к процедуре и задание процента увеличения стипендии:

```
EXEC Reduce_stipend 50
```

Пример создания процедуры с входными параметрами

Создать процедуру для выдачи списка студентов, получивших оценку, значение которой может передаться через входной параметр по дисциплине, название которой также может передаваться при вызове процедуры. По умолчанию выводить фамилии студентов, получивших оценку «5» по дисциплине «Базы данных»:

```
CREATE PROCEDURE Subject_procedure
@Subject VARCHAR(30)= 'Базы данных', @Mark TINYINT=5
AS
SELECT Surname AS 'Фамилия', Subj_Name AS 'Дисциплина', Mark AS 'Оценка за экзамен'
FROM Student s INNER JOIN EXAM_MARKS e ON s.Student_Id = e.Student_Id
        INNER JOIN SUBJECT sj ON e.Subj_Id = sj.Subj_Id
WHERE Subj_Name = @Subject AND Mark = @Mark
```

Варианты вызова процедуры

Subject_procedure:

1) EXEC Subject_procedure

	Фамилия	Дисциплина	Оценка за экзамен
1	Сидоров	Базы данных	5
2	Фролов	Базы данных	5

2) EXEC Subject_procedure @Mark = 3

	Фамилия	Дисциплина	Оценка за экзамен
1	Иванов	Базы данных	3

3) EXEC Subject_procedure @Subject = 'Операционные системы'

	Фамилия	Дисциплина	Оценка за экзамен
1	Ерофеев	Операционные системы	5

4) EXEC Subject_procedure @Subject = 'Операционные системы', @Mark =4

	Фамилия	Дисциплина	Оценка за экзамен
1	Сидоров	Операционные системы	4

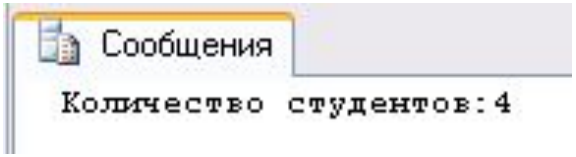
Пример создания процедуры с входными и выходным параметрами

Создать процедуру для подсчета числа студентов, сдававших экзамен по дисциплине у преподавателя фамилия и имя которого передаются через входные параметры

```
CREATE Procedure StudentNum
@Num SMALLINT OUTPUT,
@Lecture_surname VARCHAR(20),
@Lecture_name VARCHAR(20)
AS
SELECT @Num =COUNT(DISTINCT Student_Id)
FROM Exam_marks e INNER JOIN Subject s ON e.Subj_Id = s.Subj_Id
                  INNER JOIN Semester_plan sp ON s.Subj_Id =sp.Subj_Id
                  INNER JOIN Lecturer l ON sp.Lecturer_Id = l.Lecturer_Id
WHERE Surname = @Lecture_surname  AND Name=@Lecture_name
```

Вызов процедуры:

```
DECLARE @Result SMALLINT
EXECUTE StudentNum @Result OUTPUT, 'Сидорова', 'Елена'
PRINT 'Количество студентов:'+CAST(@Result AS VARCHAR(10))
```



Получение информации о процедурах

Системные процедуры:

1) sp_help

Синтаксис: **sp_help** proc1

Например,
sp_help StudentNum

Результаты | Сообщения

	Name	Owner	Type	Created_datetime
1	StudentNum	dbo	stored procedure	2021-04-08 21:35:23.430

	Parameter_name	Type	Length	Prec	Scale	Param_order	Collation
1	@Num	smallint	2	5	0	1	NULL
2	@Lecture_surname	varchar	20	20	NULL	2	Cyrillic_General_CI_AS
3	@Lecture_name	varchar	20	20	NULL	3	Cyrillic_General_CI_AS

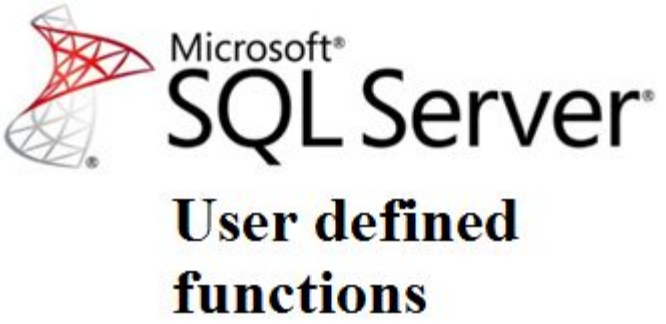
2) sp_helptext

Синтаксис: **sp_helptext** proc1

Например,
sp_helptext StudentNum

	Text
1	CREATE Procedure StudentNum
2	@Num smallint OUTPUT,
3	@Lecture_surname VARCHAR(20),
4	@Lecture_name VARCHAR(20)
5	AS
6	SELECT @Num = COUNT(DISTINCT Student_Id)
7	FROM EXAM_MARKS e INNER JOIN Subject s ON e.Subj_Id = s.Subj_Id
8	INNER JOIN Semester_plan sp ON s.Subj_Id =sp.Subj_Id
9	INNER JOIN Lecturer l ON sp.Lecturer_Id = l.Lecturer_Id
10	WHERE Surname = @Lecture_surname AND Name=@Lecture_name

Пользовательские функции (User Defined Functions, UDF) в Transact-SQL



Функция, определенная пользователем – программа выполняющая действия, например, вычисления и возвращающая результат этого действия в виде значения. Функция может получать параметры или не получать и возвращать скалярное значение или таблицу.

```
CREATE PROCEDURE Count_student_proc
@count SMALLINT OUTPUT
AS
SELECT @count=COUNT(DISTINCT e.Student_Id)
FROM Student s, Exam_marks e
WHERE s.Student_Id=e.Student_Id
GROUP BY e.Student_Id
HAVING AVG(Mark)>=4.75 AND MIN(Mark)>3
```

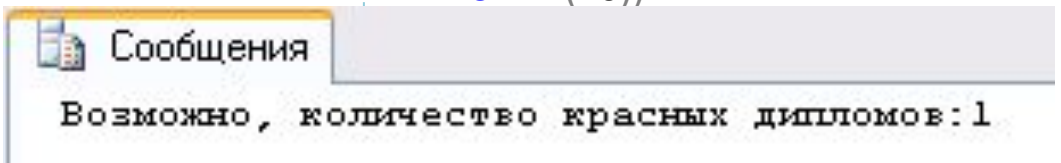
Вызов процедуры:

```
DECLARE @result SMALLINT
EXECUTE Count_student_proc @result OUTPUT
PRINT 'Возможно, количество красных
дипломов :' +CAST(@result AS VARCHAR(10))
```

```
CREATE FUNCTION dbo.Count_student_func()
RETURNS SMALLINT
AS
BEGIN
DECLARE @count SMALLINT
SELECT @count=COUNT(DISTINCT e.Student_Id)
FROM Student s, Exam_marks e
WHERE s.Student_Id=e.Student_Id
GROUP BY e.Student_Id
HAVING AVG(Mark)>=4.75 AND MIN(Mark)>3
RETURN @count
END
```

Вызов функции:

```
PRINT 'Возможно, количество красных
дипломов:' +CAST(dbo.Count_student_func() AS
VARCHAR(10))
```



Пользовательские функции (User defined functions, UDF) Transact-SQL

Поддерживаются следующие типы пользовательских функций:

1) **Скалярная функция (Scalar)** – похожа на встроенную функцию, возвращает одно значение. Значение может иметь различный формат данных.

2) **Функция, возвращающая таблицу (Inline)–** возвращает результирующий набор данных (таблицу), полученный при выполнении SELECT.

3) **Многооператорная функция (Multi-statement) –** возвращает таблицу созданную одним или несколькими операторами Transact-SQL, чем напоминает хранимые процедуры. В отличие от процедур, на такие функции можно ссылаться в WHERE как на объект просмотра

Скалярные функции (Scalar) в Transact-SQL

```
{CREATE | ALTER} FUNCTION [владелец.]  имя_функции
([{@имя_параметра скалярный_тип_данных  [=DEFAULT]}[,...n]])
RETURNS скалярный_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING} [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN скалярное выражение
END
```

Скалярные функции (Scalar) в Transact-SQL

Например, создание скалярной функции для подсчета количества студентов, у которых дата рождения входит в заданный период:

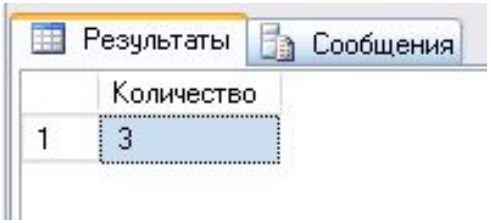
```
CREATE FUNCTION dbo.Count_student
  (@data_start DATETIME, @data_end DATETIME)
RETURNS INT
AS
BEGIN
DECLARE @count INT
SET @count=(SELECT COUNT(Student_Id)
          FROM Student
          WHERE Birthday BETWEEN @data_start and @data_end)
RETURN @count
END
```

Вызов функции:

```
DECLARE @kol INT
SET @kol=dbo.Count_student('01.01.1996', '31.12.1997')
SELECT @kol 'Количество'
```

или

```
SELECT dbo.Count_student('01.01.1996', '31.12.1997') 'Количество
'
```



Табличные функции (Inline)в Transact-SQL

```
{CREATE | ALTER } FUNCTION [владелец.] имя_функции
([{ @имя_параметра скалярный_тип_данных
[=DEFAULT]}[,...n]])
RETURNS TABLE
[ WITH {ENCRYPTION | SCHEMABINDING} [,...n]]
[AS]
RETURN [(] SELECT_оператор [)]
```

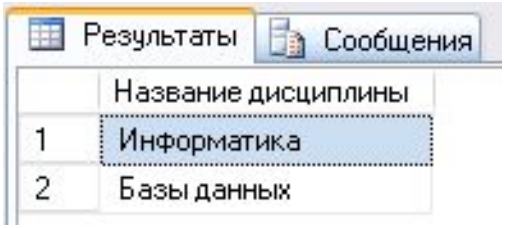
Табличные функции (Inline)в Transact-SQL

Пример, функции табличного типа для определения двух дисциплин с наибольшими оценками.

```
CREATE FUNCTION [dbo].[subj_name]()
RETURNS TABLE
AS
RETURN (
        SELECT TOP 2 sj.Subj_Name, MAX(Mark) as max_mark
         FROM Exam_mark e  INNER JOIN Subject sj ON sj.Subj_Id=e.Subj_Id
        GROUP BY sj.Subj_Name
        ORDER BY MAX(Mark) DESC
        )
```

Вызов функции:

```
SELECT Subj_Name 'Название дисциплины'

FROM dbo.subj_name()
```



Многооператорные функции (Multi-statement)в Transact-SQL

```
{CREATE | ALTER }FUNCTION [владелец.] имя_функции
([{ @имя_параметра скалярый_тип_данных [=DEFAULT]}[,...n]])
RETURNS @имя_параметра TABLE  <определение_таблицы>
[WITH {ENCRYPTION | SCHEMABINDING} [,...n]]
[AS]
BEGIN
<тело_функции>
RETURN
END
```

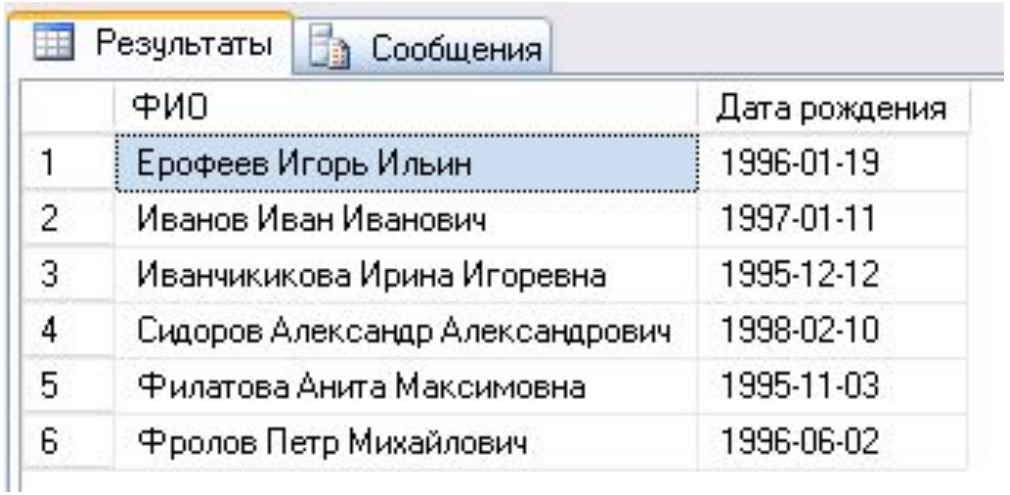
Многооператорные функции (Multi-statement)в Transact-SQL

Пример. Функция (типа multi-statement), которая выведет список студентов и их даты рождения:

```
CREATE FUNCTION [dbo].[get_FIO_student]()
RETURNS  @fio_sudent TABLE (ФИО varchar(50), [Дата рождения] DATE)
AS
 BEGIN
        INSERT @fio_sudent
        SELECT Surname+' '+Name+' '+Middle_name, Birthday
        FROM Student
        ORDER BY Surname
RETURN
 END
```

Вызов функции get_FIO_student:

```
SELECT *
FROM get_FIO_student()
```



	ФИО	Дата рождения
1	Ерофеев Игорь Ильин	1996-01-19
2	Иванов Иван Иванович	1997-01-11
3	Иванчикикова Ирина Игоревна	1995-12-12
4	Сидоров Александр Александрович	1998-02-10
5	Филатова Анита Максимовна	1995-11-03
6	Фролов Петр Михайлович	1996-06-02

Удаление пользовательской функции в Transact−SQL

Синтаксис удаления функции:
DROP FUNCTION {[владелец.] имя_функции} [,...n]]

Пример создания табличной функции

Создание функции для расчета количества различных оценок и определения их среднего значения по каждому студенту.

```
CREATE FUNCTION dbo.Avg_mark()
RETURNS TABLE
AS
RETURN(
      SELECT Student_Id, AVG(Mark) AVG_mark,
                  SUM(CASE WHEN Mark=2 THEN 1 ELSE 0 END) Count_2,
                  SUM(CASE WHEN Mark=3 THEN 1 ELSE 0 END) Count_3,
                  SUM(CASE WHEN Mark=4 THEN 1 ELSE 0 END) Count_4,
                  SUM(CASE WHEN Mark=5 THEN 1 ELSE 0 END) Count_5
      FROM Exam_mark GROUP BY Student_Id
      )
```

Вызов функции:

```
SELECT *
FROM dbo.Avg_mark()
```

Результаты | Сообщения

	Student_Id	AVG_mark	Count_2	Count_3	Count_4	Count_5
1	35	5	0	0	0	2
2	38	4,75	0	1	0	7
3	39	3	0	1	0	0
4	77	3,5	1	0	0	1

Пример вызова функции из хранимой процедуры

Создание процедуры для определения кандидатов на получение красного диплома. Название группы передается через входной параметр. Процедура вызывает ранее созданную функцию **dbo.Avg_mark()**.

```
CREATE PROCEDURE Information_red_diploma
@group VARCHAR(5)
AS
SELECT Surname 'Фамилия', Name 'Имя', AVG_mark 'Средний балл', Комментарий=
       (CASE
         WHEN Count_2=0 AND Count_3=0 AND AVG_mark>=4.75
              THEN 'может быть красный диплом'
          WHEN Count_2=0 AND Count_3>0 AND AVG_mark>=4.75
              THEN 'нужно исправить 3-ки, количесво:'+CAST(Count_3 AS VARCHAR(1))
        ELSE ' '
       END)
FROM Student,[Group], dbo.Avg_mark()
WHERE [Group].Id_Group=Student.Id_group AND Student.Student_Id =dbo.Avg_mark.Student_Id
                              AND Name_group=@group
```

Вызов процедуры:

```
Information_red_diploma 'БПО'
```

Результаты | Сообщения

	Фамилия	Имя	Средний балл	Комментарий
1	Иванов	Иван	5	может быть красный диплом
2	Сидоров	Александр	4,75	нужно исправить 3-ки, количесво:1
3	Фролов	Петр	4	

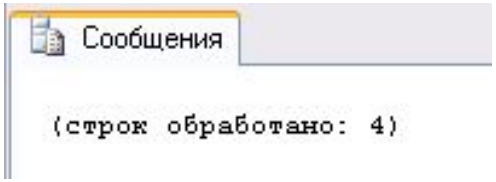
Пример вызова функции из хранимой процедуры

Создание процедуры для увеличения стипендии тем студентом у которых ср.оценка больше или равна 4.75 и нет троек. Значение процента увеличения стипендия передается через входной параметр. Процедура вызывает ранее созданную функцию **dbo.Avg_mark()**.

```
CREATE PROCEDURE Update_stipend
@Persent FLOAT
AS
UPDATE Student
SET Stipend=(CASE
        WHEN AVG_mark>=4.75 AND Count_2=0 AND Count_3=0
            THEN Stipend*(100+@Persent)/100
         ELSE Stipend
         END)
FROM Student, dbo.Avg_mark()
WHERE Student.Student_Id =dbo.Avg_mark.Student_Id
```

Вызов процедуры:

```
Update_stipend 10
```



Транзакции и целостность баз данных

Варианты завершения транзакции

Фиксация транзакции

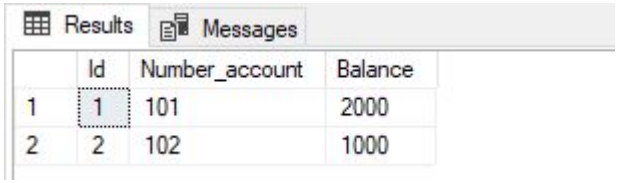
сохранение изменений БД, выполненных в процессе выполнения транзакции

Откат транзакции

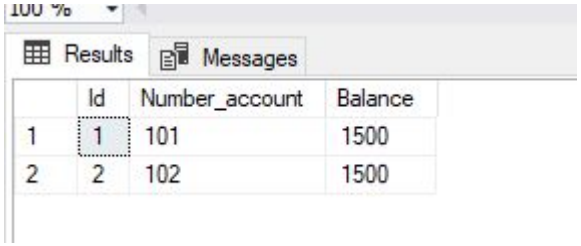
отмена всех изменений БД, сделанных в процессе выполнения транзакции

Пример

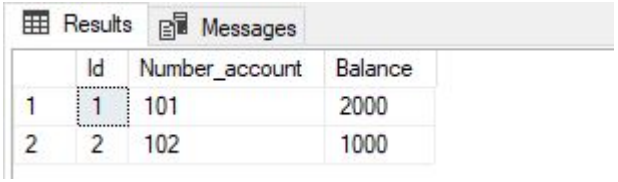
```
CREATE TABLE Bank_account
(Id INT PRIMARY KEY IDENTITY(1,1),
Number_account decimal(10,0),
Balance float)

INSERT INTO Bank_account
VALUES(101, 2000), (102, 1000)

BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance - 500
WHERE Number_account = 101

UPDATE Bank_account
SET Balance = Balance + 500
WHERE Number_account = 102

--COMMIT

-- ROLLBACK
```



	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000



	Id	Number_account	Balance
1	1	101	1500
2	2	102	1500



	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

Транзакции и целостность баз данных

Количество операций, входящих в транзакцию, может быть от одной и более. Разработчик решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций.

Транзакция обладает следующими важными свойствами (**ACID**), которые гарантируют правильность и надежность работы системы:

1) **A**tomicity (атомарность).
Каждая транзакция в БД должна быть выполнена полностью либо не выполнена совсем. Не допускается частичное выполнение.

2) **C**onsistency (согласованность).
Должно быть согласованное состояние БД до и после выполнения транзакции.

3) **I**solation (изолированность).
Результаты транзакции не должны быть видены другим транзакциям, пока она не завершиться.

4) **D**urability (устойчивость, долговечность).
Изменения, внесенные в БД в результате выполнения транзакции должн

Модели одновременного конкурентного доступа

В многопользовательской среде предусмотрены две модели обновления данных в базе данных: оптимистичный параллелизм и пессимистичный параллелизм.

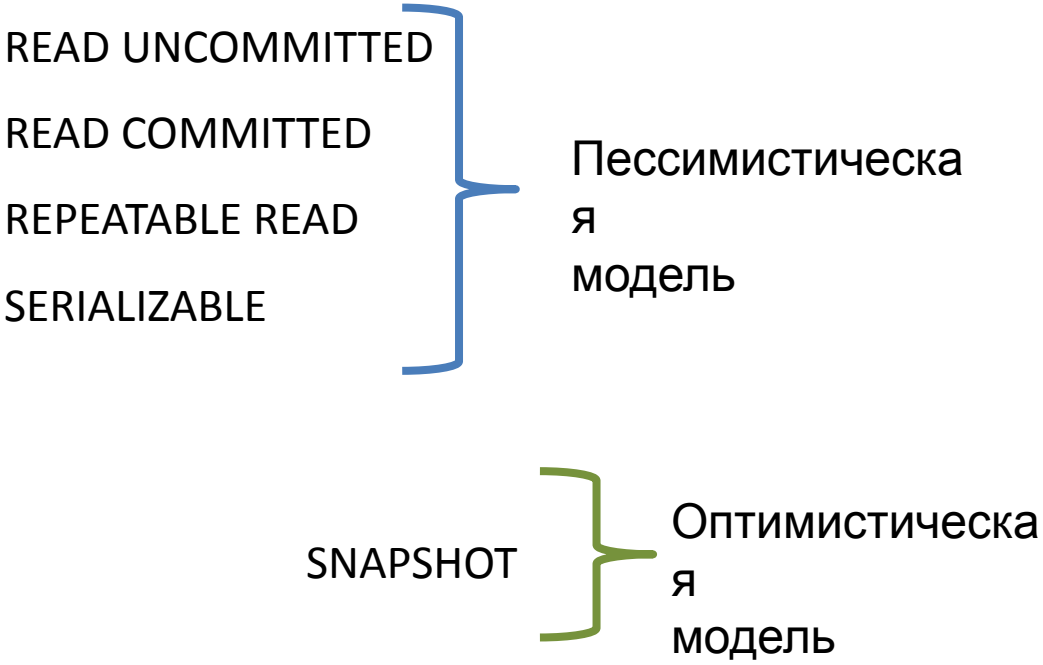
Пессимистичная модель предусматривает блокировку строк в источнике данных, что позволяет запретить другим пользователям модификацию данных, которая может отрицательно повлиять на работу текущего пользователя. Эта модель в основном используется в таких средах, где наблюдается интенсивная конкуренция по доступу к данным.

Оптимистичная модель не блокируют строку при ее чтении. Когда пользователю требуется обновить строку, приложение должно определить, не изменялась ли эта строка другим пользователем после того, как она была открыта для чтения. Эта модель обычно используется в средах, характеризующихся низкой конкуренцией за данные.

Уровни изоляции транзакций

Уровни изоляции транзакций — это мера степени успешной изоляции транзакций данных от возможности изменения другими транзакциями.

SET TRANSACTION ISOLATION LEVEL



Пессимистическая модель основана на блокировках.
Оптимистическая модель основана на управлении версиями строк.

Уровень изоляции READ UNCOMMITTED

Results | Messages

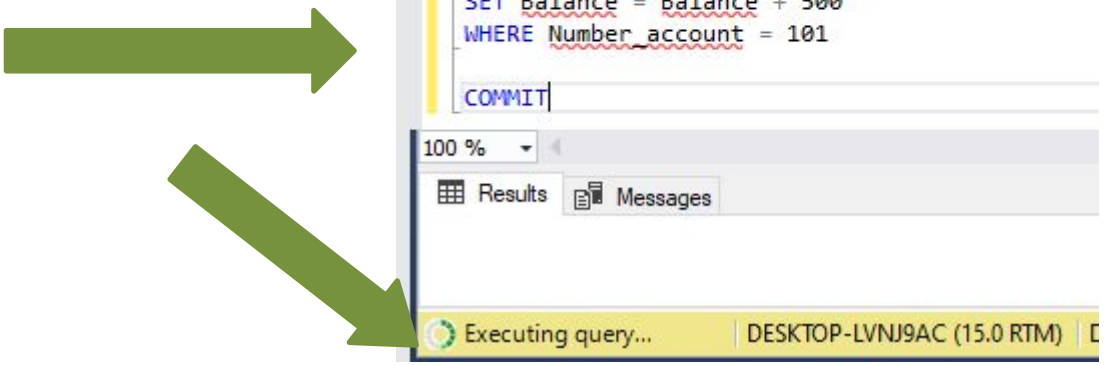
	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Клиент 1 выполняет транзакцию:

BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance - 500
WHERE Number_account = 101

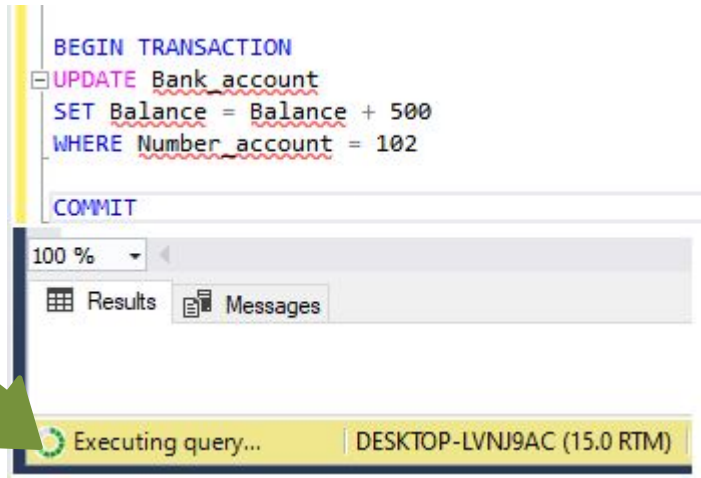
COMMIT



В это время Клиент 2 начинает выполнять транзакцию для другого счета:

BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance - 500
WHERE Number_account = 102

COMMIT



Уровень изоляции READ UNCOMMITTED. Проблема уровня «Грязное» чтение (dirty read)

Results | Messages

	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Сотрудником выполняется транзакция зачисления суммы денег на счет 102, но потом выполняет команду ROLLBA(

```
BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance + 1000
WHERE Number_account = 102
```

Results | Messages

	Id	Number_account	Balance
1	2	102	1000

Results | Messages

	Id	Number_account	Balance
1	2	102	2000

Results | Messages

	Id	Number_account	Balance
1	2	102	1000

ROLLBACK

Во время выполнения предыдущей транзакции, до выполнения команды ROLLBACK, владелец счета 102 проверяет свой баланс:

```
SELECT *
FROM Bank_account
WHERE Number_account=102
```

Results | Messages

	Id	Number_account	Balance
1	2	102	2000

Владелец счета 102 решив, что средств достаточно пытается совершить транзакцию для снятия наличных в размере 2000, но на счете 102 такой суммы нет.

Владелец счета 102 видел баланс в еще не завершенной транзакции (до COMMIT или ROLLBACK)

Уровень изоляции READ COMMITTED

Results | Messages

	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

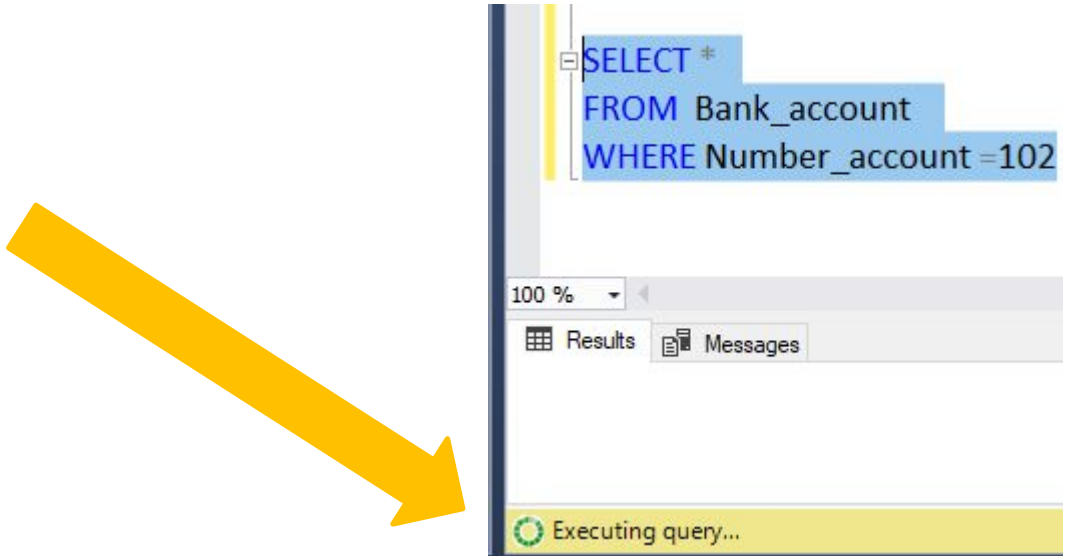
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

Сотрудником выполняется транзакция зачисления суммы денег на счет 102:

BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance + 1000
WHERE Number_account = 102
ROLLBACK

Владелец счета 102 не может узнать свой баланс, пока в предыдущей транзакции не выполнена команда COMMIT или ROLLBACK:

SELECT *
FROM Bank_account
WHERE Number_account=102



Уровень изоляции READ COMMITTED.

Проблема: неповторяющееся чтение (non-repeatable read)

Results | Messages

	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION
SELECT *
FROM Bank_account
WHERE Number_account =102

-- выполняются другие команды транзакции
SELECT *
FROM Bank_account
WHERE Number_account =102

COMMIT
```

Results | Messages

	Id	Number_account	Balance
1	2	102	2000

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION
SELECT *
FROM Bank_account
WHERE Number_account =102

-- выполняются другие команды транзакции
SELECT *
FROM Bank_account
WHERE Number_account =102

COMMIT
```

Results | Messages

	Id	Number_account	Balance
1	2	102	1500

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance - 500
WHERE Number_account = 101

COMMIT
```

Messages

```
(1 row affected)
```

Уровень изоляции REPEATABLE READ

Results | Messages

	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION
SELECT *
FROM Bank_account
WHERE Number_account =102

-- выполняются другие команды транзакции
SELECT *
FROM Bank_account
WHERE Number_account =102

COMMIT
```

Results | Messages

	Id	Number_account	Balance
1	2	102	2000

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance- 500
WHERE Number_account = 102

COMMIT
```

100 % | Results | Messages | Executing query...

Уровень изоляции REPEATABLE READ.

Проблема: фантомное чтение (phantom reads)

Results | Messages

	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION
SELECT count(Id)
FROM Bank_account
-- выполняются другие команды транзакции
SELECT count(Id)
FROM Bank_account
COMMIT
```

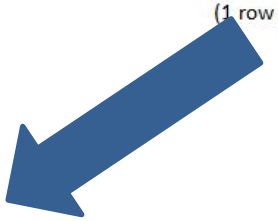
Results | Messages

(No column name)
5

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION
INSERT INTO Bank_account
VALUES (105, 5000)
COMMIT
```

Messages

(1 row affected)



```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION
SELECT count(Id)
FROM Bank_account
-- выполняются другие команды транзакции
SELECT count(Id)
FROM Bank_account
COMMIT
```

Results | Messages

(No column name)
6

Уровень изоляции SERIALIZABLE

Results | Messages

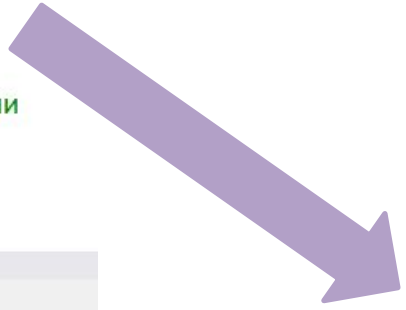
	Id	Number_account	Balance
1	1	101	2000
2	2	102	1000
3	3	103	2000

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION
SELECT count(Id)
FROM  Bank_account
-- выполняются другие команды транзакции
SELECT count(Id)
FROM  Bank_account
COMMIT
```

Results | Messages

(No column name)
6



```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION
INSERT INTO  Bank_account
VALUES (106, 6000)
COMMIT
```

100 %

Results | Messages

Executing query...

Проблемы одновременного доступа

При одновременном изменении данных различными пользователями возможно возникновение следующих проблем:

Уровни изоляции	Возникающие проблемы		
	«Грязное» чтение (dirty read)	Неповторяющееся чтение (non-repeatable read)	Фантомное чтение (phantom reads)
READ UNCOMMITTED	+	+	+
READ COMMITTED		+	+
REPEATABLE READ			+
SERIALIZABLE			

Скорость

Надежность

Уровень изоляции SNAPSHOT

Разрешение на уровни СУБД:

ALTER DATABASE DATABASE_NAME SET ALLOW_SNAPSHOT_ISOLATION ON

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

```
ALTER DATABASE [Example BD for transactions] SET ALLOW_SNAPSHOT_ISOLATION ON
SET TRANSACTION ISOLATION LEVEL SNAPSHOT

BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102
COMMIT
```

Results | Messages

	Id	Number_account	Balance
1	2	102	1500

```
ALTER DATABASE [Example BD for transactions] SET ALLOW_SNAPSHOT_ISOLATION ON
SET TRANSACTION ISOLATION LEVEL SNAPSHOT

BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102
-- выполняются другие команды транзакции
SELECT *
FROM  Bank_account
WHERE Number_account =102
COMMIT
```

Results | Messages

	Id	Number_account	Balance
2	2	102	1500

```
BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance - 1000
WHERE Number_account = 102
COMMIT
```

Messages

(1 row affected)

Уровень изоляции SNAPSHOT

Разрешение на уровни СУБД:

ALTER DATABASE DATABASE_NAME SET ALLOW_SNAPSHOT_ISOLATION ON

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

```
BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102

UPDATE Bank_account
SET Balance = Balance+ 1000
WHERE Number_account = 102
COMMIT
```

Results | Messages

Id	Number_account	Balance
2	102	500

```
BEGIN TRANSACTION
UPDATE Bank_account
SET Balance = Balance - 1000
WHERE Number_account = 102
COMMIT
```

Messages

(1 row affected)

```
ALTER DATABASE [Example BD for transactions] SET ALLOW_SNAPSHOT_ISOLATION ON
SET TRANSACTION ISOLATION LEVEL SNAPSHOT

BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102

UPDATE Bank_account
SET Balance = Balance+ 1000
WHERE Number_account = 102
COMMIT
```

Messages

Msg 3960, Level 16, State 5, Line 9

Транзакция в режиме изоляции моментального снимка прервана из-за конфликта обновлений. Невозм

Уровень изоляции SNAPSHOT

Разрешение на уровни СУБД:

ALTER DATABASE DATABASE_NAME SET ALLOW_SNAPSHOT_ISOLATION ON

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

```
ALTER DATABASE [Example BD for transactions] SET ALLOW_SNAPSHOT_ISOLATION ON
SET TRANSACTION ISOLATION LEVEL SNAPSHOT

BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102

UPDATE Bank_account
SET Balance = Balance+ 1000
WHERE Number_account = 102
COMMIT
```

Results | Messages

Id	Number_account	Balance
2	102	3000

```
SET TRANSACTION ISOLATION LEVEL  SNAPSHOT

BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102

UPDATE Bank_account
SET Balance = Balance - 500
WHERE Number_account = 102
COMMIT
```

100 % | Results | Messages

	Id	Number_account	Balance
1	2	102	3000

```
BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102

UPDATE Bank_account
SET Balance = Balance+ 1000
WHERE Number_account = 102

SELECT *
FROM  Bank_account
WHERE Number_account =102
COMMIT
```

Results | Messages

Id	Number_account	Balance
2	102	4000

```
BEGIN TRANSACTION
SELECT *
FROM  Bank_account
WHERE Number_account =102

UPDATE Bank_account
SET Balance = Balance - 500
WHERE Number_account = 102
COMMIT
```

100 % | Results | Messages

Executing query...

Журнал транзакций (Write-Ahead Logging (WAL))

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого ***протокола Write Ahead Log - WAL***). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Если приложение выполняет инструкцию ROLLBACK или ядро СУБД обнаруживает ошибку, например, потерю связи с клиентом, записи журнала используются для отката изменений, сделанных незавершенной транзакцией.

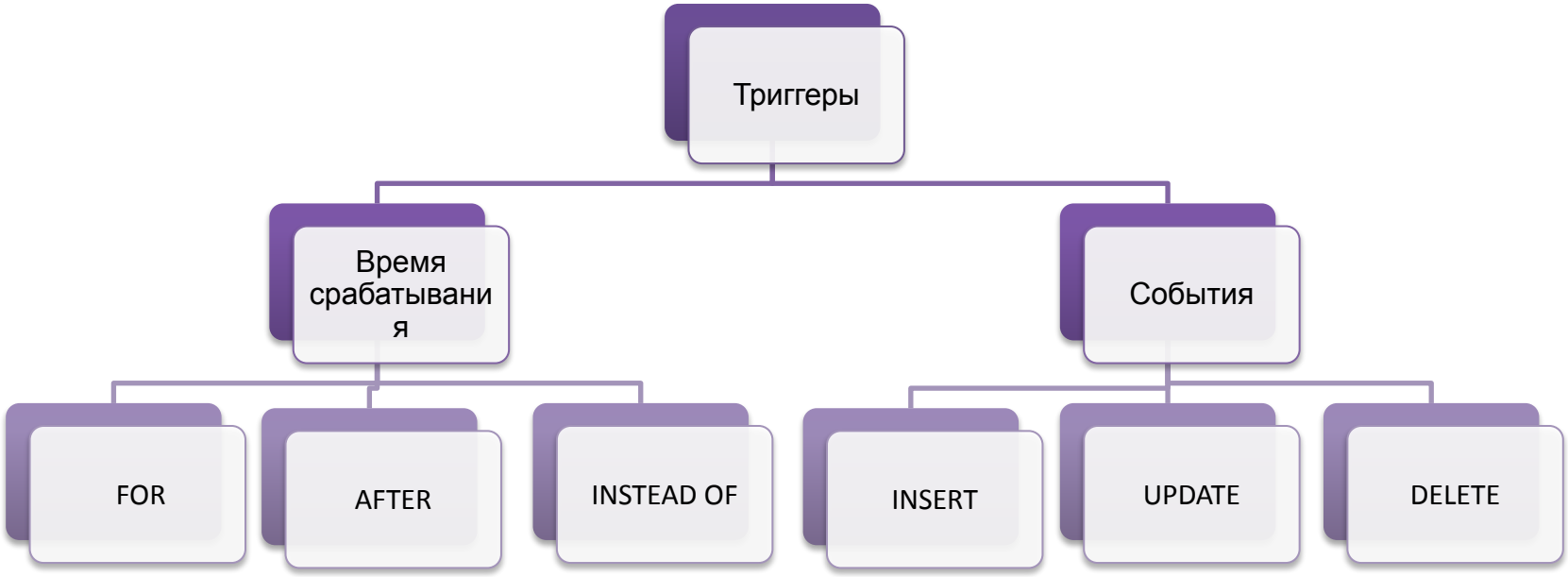
Если на сервере происходит сбой, базы данных могут остаться в состоянии, когда часть изменений не переписана из буферного кэша в файлы данных, но в них имеются изменения, совершенные незаконченными транзакциями. Когда экземпляр SQL Server будет запущен, он выполнит восстановление каждой базы данных. Каждое изменение, записанное в журнале, которое, возможно, не было записано в файлы данных, накатывается[1]. Чтобы сохранить целостность базы данных, будет также произведен откат каждой незавершенной транзакции, найденной в журнале транзакций.

1 накат (раскрутка)заключается во внесении в сохраненную копию БД результатов всех завершенных транзакций. При этом транзакции не обрабатываются повторно, а производятся изменения в БД согласно записям в журнале

Триггеры (Triggers) в Transact–SQL

Триггер –это специальный вид хранимой процедуры, который выполняется автоматически при наступлении определенного события (действия) внутри БД.

Триггеры (Triggers) в Transact−SQL



События:
INSERT – определяет действия, которые будут выполняться после добавления новой записи в таблицу;
UPDATE – определяет действия, которые будут выполняться после изменения записи таблицы;
DELETE – определяет действия, которые будут выполняться после удаления записи из таблиц.

Время срабатывания:
FOR – срабатывает до изменения данных;
AFTER – срабатывает после изменения данных;

Синтаксис создания/изменения триггера

Основной формат команды:
{CREATE | ALTER} TRIGGER [имя_триггера]
ON имя_таблицы
{FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE] [,] [DELETE]}
[WITH ENCRYPTION]
AS SQL_операторы

Или используя предложение **IF UPDATE**:
CREATE TRIGGER [имя_триггера]
ON имя_таблицы
{FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE]}
[WITH ENCRYPTION]
AS
IF UPDATE (имя_столбца)
[{AND |OR} UPDATE (имя_столбца)…]
SQL_операторы;

Триггеры и виртуальные таблицы deleted и inserted

Таблицы **deleted** и **inserted** создаются автоматически.

Таблицы **deleted** и **inserted** всегда имеют такую же структуру, что и у таблицы, на которую установлен триггер.

Содержимое таблиц **inserted** и **deleted** при разных событиях:
1) Если триггер срабатывает на событие **INSERT**: таблица **inserted** содержит копию строк, которые должны быть вставлены в таблицу.

2) Если триггер срабатывает на событие **DELETE**: удаляемые строки помещаются в таблицу **deleted**.

3) Если триггер срабатывает на событие **UPDATE**: обновление происходит в два этапа – удаление и вставка. Исходные строки помещаются в таблицу **deleted**, а новые данные помещаются в таблицу **inserted.**

Для получения информации о количестве строк, которое будет изменено при успешном завершении *триггера*, можно использовать функцию **@@ROWCOUNT.** Функция **@@ROWCOUNT** возвращает количество строк, обработанных последней командой.
Для отмены всех изменений, которые внес пользователь, необходимо использовать команду **ROLLBACK TRANSACTION**. Для фиксации изменений применяется команда **COMMIT TRANSACTION**.

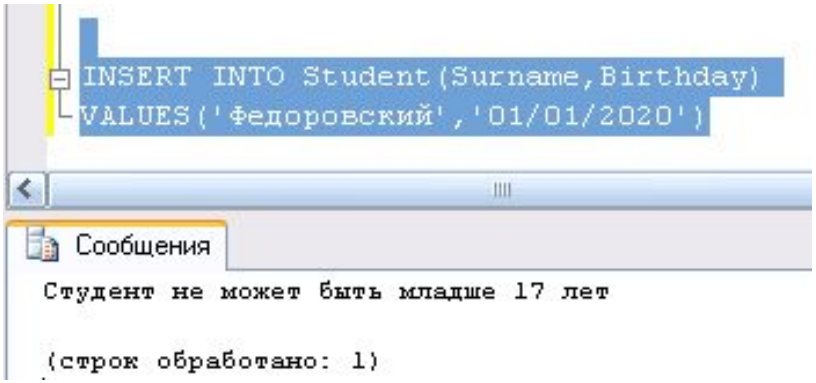
Синтаксис удаления триггера

DROP TRIGGER {имя_триггера} [,...n].

Реализовать ограничение на вводимое значение

Создание триггера проверки возраста студента при добавления записи в таблицу Student. Триггер выполняет вместо команды добавление.

```
CREATE TRIGGER Age_verification
ON Student
INSTEAD OF INSERT
AS
BEGIN
    IF(SELECT DATEDIFF(YEAR, Birthday, GETDATE()) FROM INSERTED)>=17
      INSERT INTO Student
      SELECT Stipend, Surname, Name, Middle_name, Birthday, City, Univ_Id, Id_group, data_input, Kurse
      FROM inserted
    ELSE
      PRINT 'Студент не может быть младше 17 лет'
END
```



Триггер сохраняющий дату последнего изменения записи в таблице Student

Ссоздание триггер сохраняющего дату последнего изменения записи в таблице Student. Для написания подобного триггера в таблице должно быть поле, в которое будет записываться дата и время внесенного изменения. В примере это поле Update_time. Если такого поля в таблице Student нет, тогда нужно добавить.

```
CREATE TRIGGER Update_time_in_the_column
ON Student
AFTER UPDATE
AS
DECLARE @Id INT
BEGIN
SET NOCOUNT ON
      SELECT @Id=Student_Id
      FROM inserted
            UPDATE Student
            SET Update_time=GETDATE()
            WHERE Student_Id=@Id
END
```

Триггер запрещающий изменение значения поля Mark и Exam_Date

```
CREATE TRIGGER Disallow_update_Mark_Exam_Date
ON Exam_mark
AFTER UPDATE
AS
IF UPDATE (Mark) OR UPDATE(Exam_Date)
BEGIN
    ROLLBACK TRANSACTION
    RAISERROR('нельзя изменять значение оценки  или даты
экзамена',16,1)
END
```

RAISERROR – системная функция, позволяющая создавать сообщение об ошибке и возвращать его как сообщение об ошибке сервера вызывающему приложению .
Синтаксис:
RAISERROR (сообщение или номер ошибки, степень_серьезности, состояние)

Триггер, не допускающий добавление более трех записей о результатах сдачи по одной и той же дисциплине одним и тем же студентом

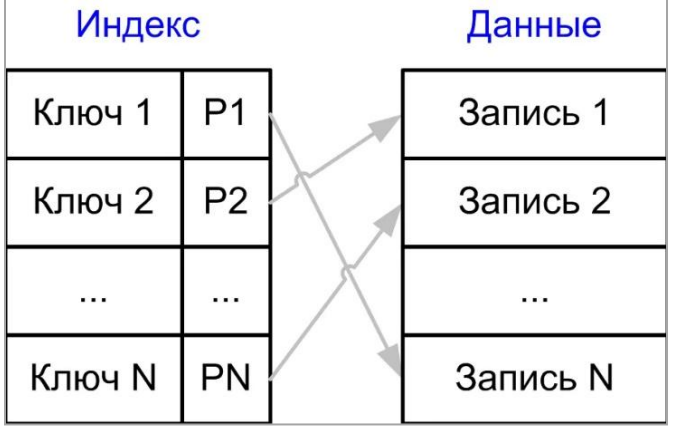
```
CREATE TRIGGER Count_re_examination
ON Exam_mark
AFTER INSERT
AS
DECLARE @Student_Id INT, @Subj_Id INT
BEGIN
      SELECT @Student_Id =Student_Id, @Subj_Id= Subj_Id
      FROM inserted
      IF (SELECT COUNT(Mark)
          FROM Exam_mark
          WHERE Student_Id=@Student_Id AND Subj_Id= @Subj_Id
           GROUP BY Student_Id, Subj_Id)>3
      BEGIN
                ROLLBACK TRANSACTION
           RAISERROR('не допускается более трех пересдач',16,1)
      END
END
```

Триггер перемещает удаленную запись из таблицы Student в архивную таблицу Archive_delete

```
CREATE TRIGGER Insert_in_table_Archive_delete
ON Student
AFTER DELETE
AS
BEGIN
    INSERT INTO Archive_delete
    SELECT *
    FROM deleted
END
```

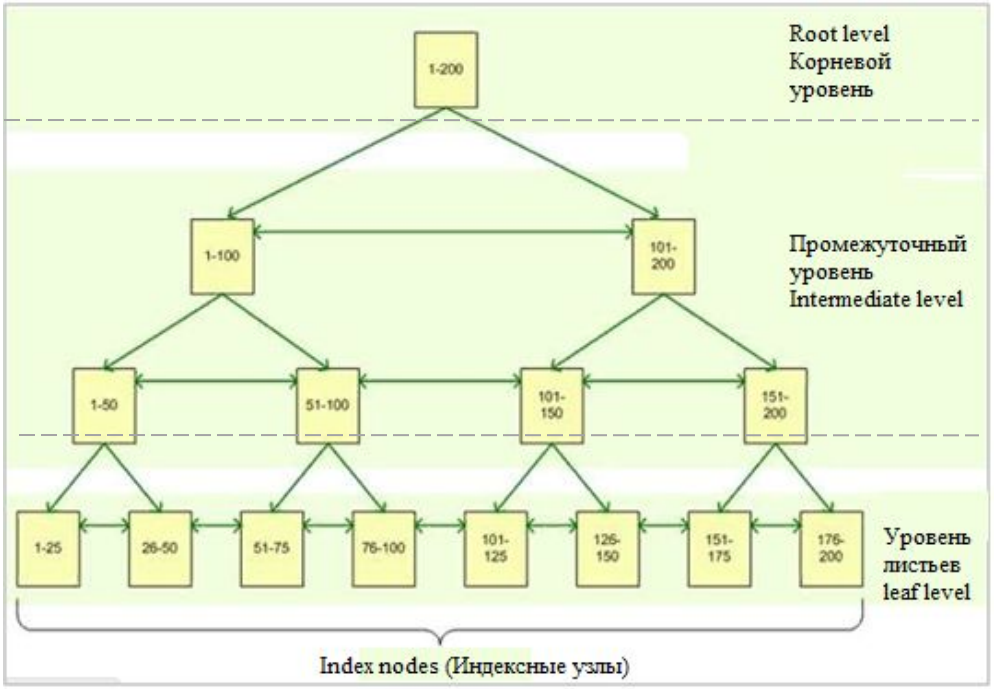
Индексы

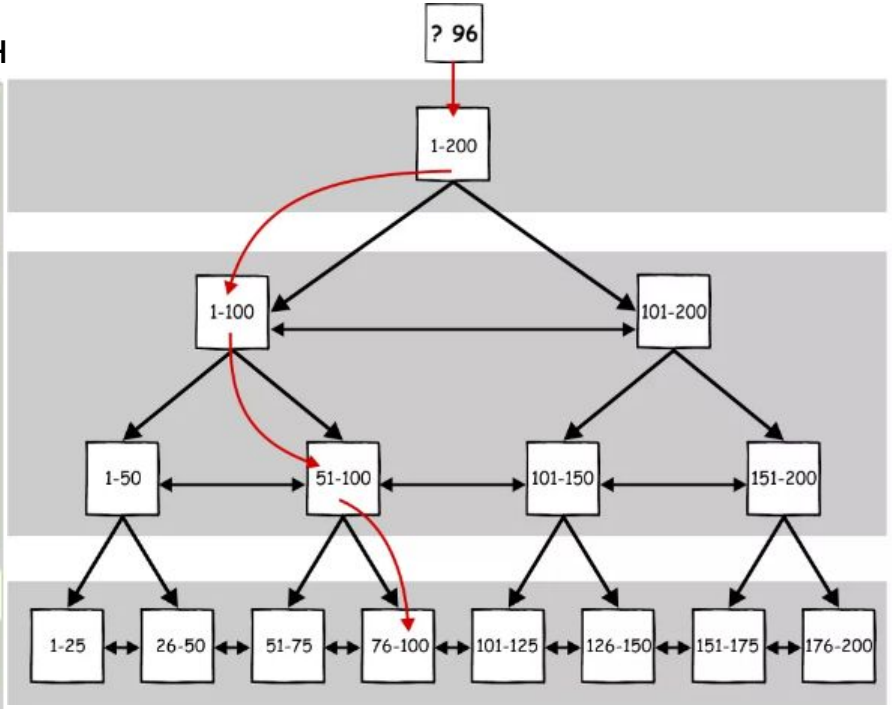
- Индекс (Index)- это объект базы данных, создаваемый для повышения производительности выборки данных.
- Индекс создается для поля (полей) таблицы и обеспечивает быстрый доступ к данным этой таблицы за счет упорядочения данных поля (полей) по значению.



Возможные варианты поиск с помощью индекса:
1) точного значения;
2) интервала значений;
3) значений нескольких полей.

В SQL Server индексы хранятся в виде B-деревьев (B-tree, Balanced tree). «B» означает сбалансирован

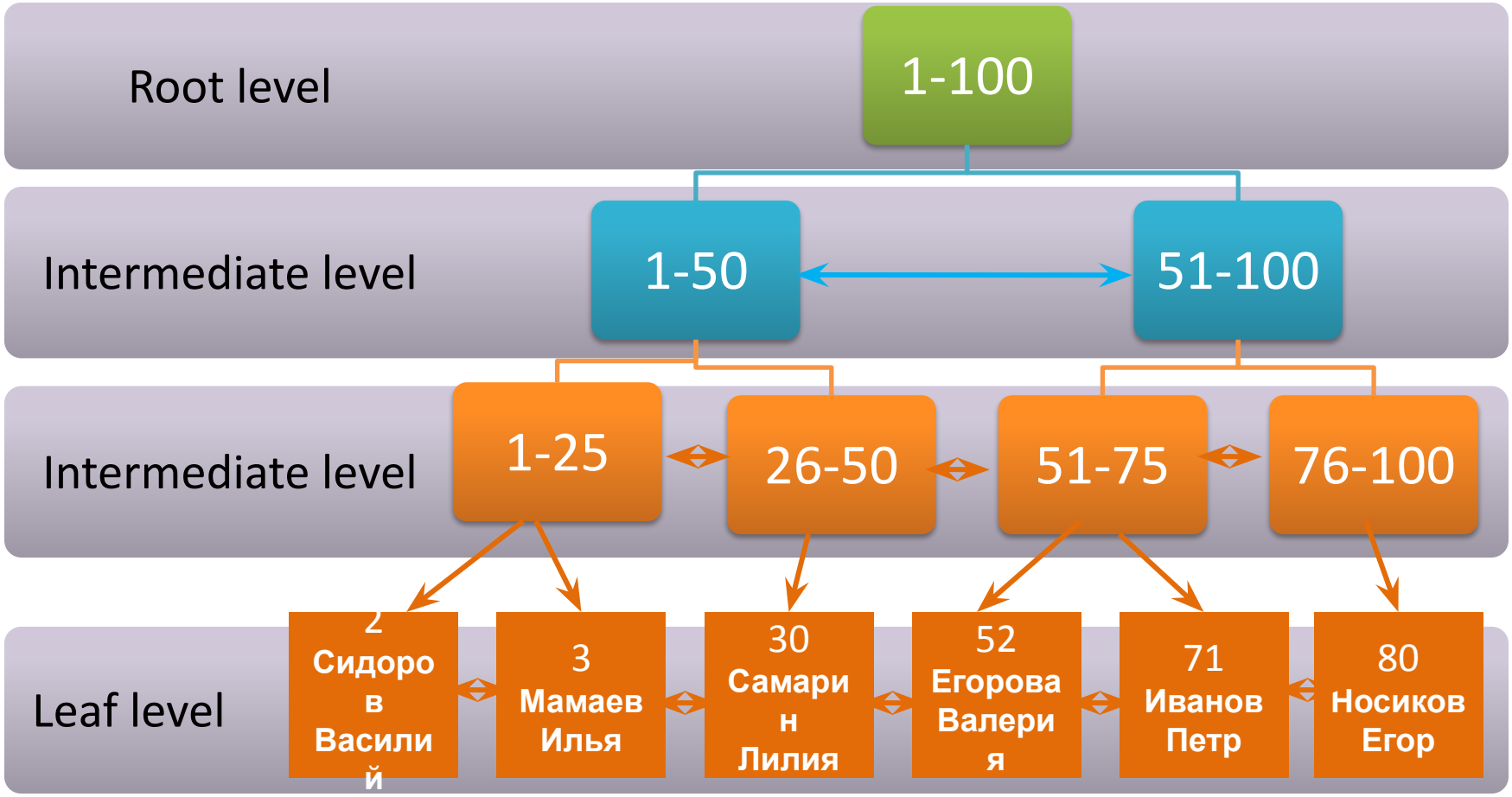




Структура сбалансированного в дерева

На примере таблицы «Студент»

```
SELECT *
FROM Student
WHERE Id_student = 71
```



Способы задания индекса

1) Автоматическое создание индекса при добавлении ограничения первичного ключа **PRIMARY KEY**.

Например,

```
CREATE TABLE Student
(ID_student INT PRIMARY KEY,
Surname VARCHAR(30),
Birthday DATA)
```

2) Автоматическое создание индекса при задании ограничения **UNIQUE**.

Например,

```
CREATE TABLE Student
(ID_student INT PRIMARY KEY,
Surname VARCHAR(30),
Series_nomber_pas VARCHAR(20),
UNIQUE(Series_nomber_pas ))
```

3) Добавление индекса при помощи команды **CREATE INDEX**.

Например,

```
CREATE INDEX  Index_table_Student
ON Student(Series_nomber_pas)
```

Создание индексов в MS SQL Server

Команда создания индекса для таблицы **CREATE INDEX** имеет следующий синтаксис:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX index_name ON table_name (column1 [ASC |
      DESC] ,...)
        [ INCLUDE ( column_name [ ,... ] ) ]
 [WITH
       [FILLFACTOR=n] [[, ] PAD_INDEX = {ON | OFF}]
       [[, ] DROP_EXISTING = {ON | OFF}]
       [[, ] SORT_IN_TEMPDB = {ON | OFF}]
       [[, ] IGNORE_DUP_KEY = {ON | OFF}]
       [[, ] ALLOW_ROW_LOCKS = {ON | OFF}]
       [[, ] ALLOW_PAGE_LOCKS = {ON | OFF}]
       [[, ] STATISTICS_NORECOMPUTE = {ON | OFF}]
       [[, ] ONLINE = {ON | OFF}]] [ON file_group | "default "]
```

Значения параметров по умолчанию:
Параметр **NONCLUSTERED** – создает некластеризованный индекс.
Параметр **ASC** сортировка по возрастанию значений столбца (column1).
Параметр **FILLFACTOR** имеет значение по умолчанию **n=0** => страницы узлов индекса заполняются полностью, а каждая из промежуточных страниц содержит свободное место для одной записи.
Параметр **WITH PAD_INDEX** задает заполнение индекса, по умолчанию выключен.
Параметр **SORT_IN_TEMPDB** указывает, следует ли хранить временные результаты сортировки в базе данных tempdb , по умолчанию выключен.
Параметр **IGNORE_DUP_KEY** задает ответ на ошибку, когда операция вставки пытается вставить повторяющиеся значения ключа в уникальный индекс. IGNORE_DUP_KEY применяется только к операциям вставки после создания или перестроения индекса.
Параметр **STATISTICS_NORECOMPUTE** определяет состояние автоматического перерасчета статистики указанного индекса, по умолчанию выключен.
Параметр **DROP_EXISTING** задает возможность удалить и перестроить существующий кластеризованный или некластеризованный индекс с измененными спецификациями столбцов и сохранить то же имя для индекса, по умолчанию выключен.
Параметр **ONLINE** указывает, доступны ли базовые таблицы и связанные с ними индексы для запросов и изменения данных во время операции индексирования, по умолчанию выключен.

Увидеть индексы в таблице

EXECUTE sp_helpindex name_table

Пример, EXECUTE sp_helpindex Student

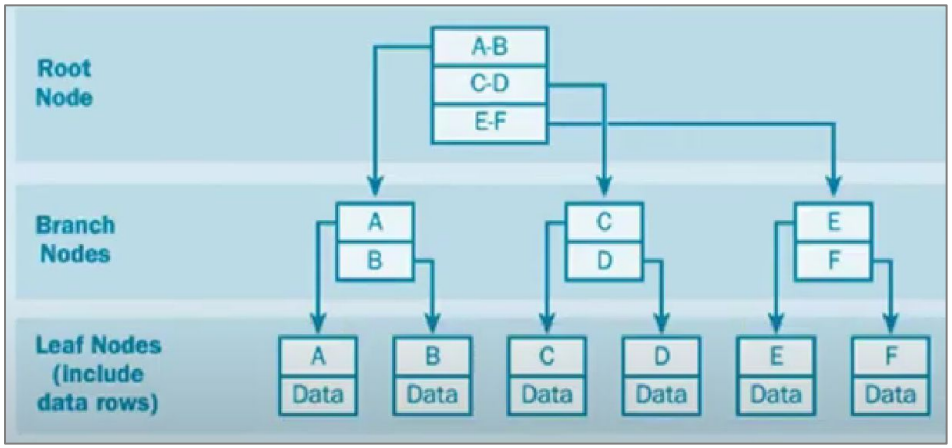
SELECT *
FROM dbo.Student

Кластеризованный и некластеризованный индекс

Существую два типа индексов:

1) Кластеризованный индексы (Clustered index).

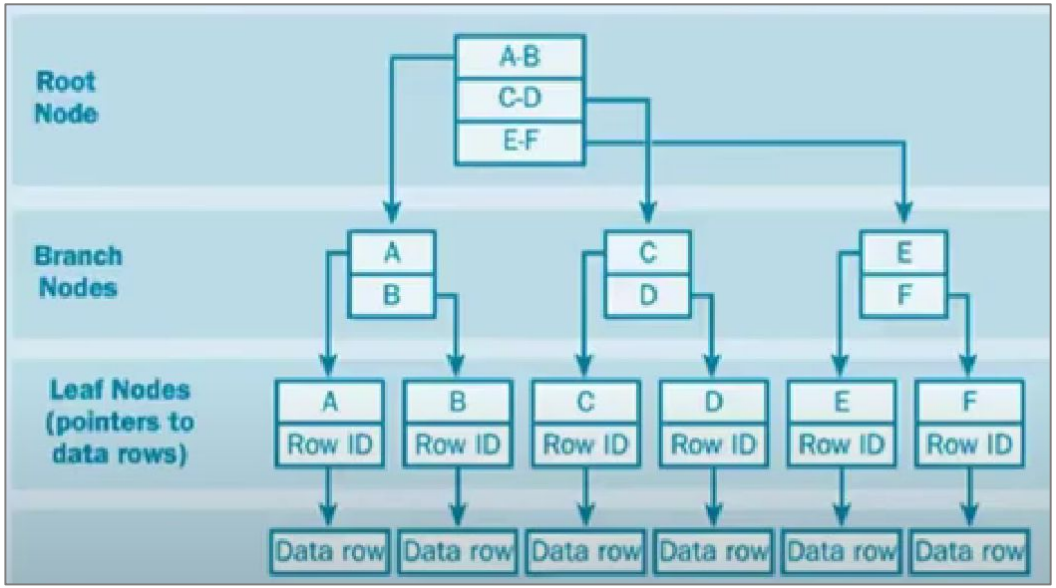
При определении кластеризованного индекса в таблице (представлении) физическое расположение данных перестраивается в соответствии со структурой индекса, т.е. выполняется сортировка по полю индекса.



2) Некластеризованный индекс (Nonclustered index).

При определении некластеризованного индекса в таблице (представлении), физическая структура данных не перестраивается, а только организуются ссылки на соответствующие записи.

Если таблица не имеет кластеризованного индекса, тогда на уровне листьев находится идентификатор строки (RID - Row Identifier), который состоит из: адреса файла, адреса физического блока (страницы), смещения строки в странице.



В SQL Server таблицу без кластеризованного индекса называется кучей (heap).

Если кластеризованного индекса нет, тогда некластеризованный индекс хранит на уровне листьев ссылки на записи кластеризованного индекса или на записи из кучи.

Пример, работа кластеризованного индекс

Id	Name	Birthday
1	Иван	01.01.2000
2	Мария	02.02.2000
3	Екатерина	02.04.2001
4	Петр	22.06.2001
5	Инга	12.04.2000
6	Мария	12.10.2002

Ключевые различия между кластеризованными и неклстеризованными индексами

Сравнение показателей	Кластеризованный индекс	Некластеризованный индекс
Порядок хранения	Определяет порядок хранения строк в таблице в целом.	Определяет порядок хранения строк в таблице с помощью отдельной физической структуры.
Количество индексов, разрешенных на одну таблицу	Только один кластерный индекс	Несколько некластеризованных индексов
Доступ к данным	Быстрее	Более медленный по сравнению с кластерным индексом
Дополнительное дисковое пространство	Не нужно	Требуется хранить индексы отдельно

Результаты тестов кластеризованного и некластеризованного индексов на объем занимаемого пространства и производительность

Созданы две таблицы testtable (одно из полей column int PRIMARY KEY) и testtable2 (одно из полей column int PRIMARY KEY NONCLUSTERED)

Этап 1. Таблицы были заполнены данными (< 5000000 строк).

Таблица	Индекс	Использовано (КБ)	Зарезервировано (КБ)	Число строк
testtable	PK__testtabl__357D0D3E3D086A66	257952	257992	4999999
testtable2	HEAP	256992	257032	4999999
testtable2	PK__testtabl__357D0D3F2CBA35D8	89432	89608	4999999

Этап 2. В каждую таблицу были добавлены еще 1000000 записей.

Таблица	Индекс	Использовано (КБ)	Зарезервировано (КБ)	Число строк
testtable	PK__testtabl__357D0D3E3D086A66	263128	263176	5099999
testtable2	HEAP	262392	262472	5099999
testtable2	PK__testtabl__357D0D3F2CBA35D8	91216	91272	5099999

Тип индекса	ЦП (мс)	Операций чтения	Операций записи	Длительность (мс)
Кластеризованный	3500	304919	654	11288
Куча	3890	406083	904	11438

Результаты тестов кластеризованного и некластеризованного индексов на объем занимаемого пространства и производительность

Этап 3. Из таблиц удалили 1000000 записей, затем добавили столько же записей.

Наименование таблицы	Наименование индекса	Использовано (КБ)	Зарезервировано (КБ)	Число строк
testtable	PK__testtabl__357D0D3E3D086A66	268304	268360	4199999
testtable2	HEAP	262392	262472	4199999
testtable2	PK__testtabl__357D0D3F2CBA35D8	93008	93064	4199999

Тип индекса	ЦП (мс)	Операций чтения	Операций записи	Длительность (мс)
Кластеризованный	3562	304859	653	10334
Куча	4973	422142	7053	13042

Запрос -> Оптимизатор запросов -> План выполнения запроса

Чтобы проанализировать производительность запроса и улучшить ее, необходимо сразу несколько инструментов: оптимизатор запросов, планы выполнения и хранилище запросов.

Оптимизатор запросов — один из наиболее важных компонентов, анализирует запрос и определяет наиболее эффективный способ доступа к необходимым данным. Входные данные для оптимизатора запросов включают запрос, схему базы данных (определение таблиц и индексов) и статистику базы данных.

План выполнения — это результат работы оптимизатора запросов.

Хранилище запросов дает представление о выборе плана выполнения и производительности. Это упрощает решение проблем с производительностью, помогая быстро находить различия в производительности, вызванные изменениями в плане выполнения. Хранилище запросов собирает данные, такие как журнал запросов, планы, статистику выполнения и статистику ожидания.

Инструкция создания хранилища запросов:

```
ALTER DATABASE Название_БД SET QUERY_STORE = ON;
```

Оптимизация БД и СУБД

Основных рекомендации по оптимизации БД и СУБД:

1) перестройка/реорганизация индексов;
2) очистка процедурного кэша;
3) обновление статистики.

Статистика

Статистика – это статистические сведения о распределении значений в одном или нескольких столбцах таблицы или индексированного представления.

Команда для просмотра информации о статистике:
DBCC SHOW_STATISTICS (' название таблицы', 'имя индекса, статистики или столбца ')

Системная процедура для обновления статистики:
sp_updatestats

Команда обновления статистики для определенной таблицы:
UPDATE STATISTICS название_таблицы

Перестройка индекса/реорганизация индекса

Команды в SQL Server для борьбы с фрагментацией индексов:

ALTER INDEX REBUILD / REORGANIZE

Рекомендуется:

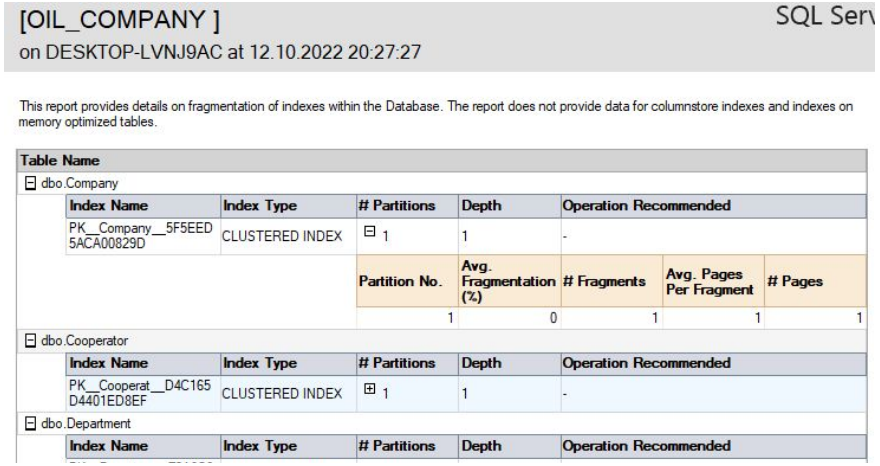
1) Если степень фрагментации менее 5%, тогда не нужно перестраивать или реорганизовывать индекс;
2) Если степень фрагментации от 5 до 30%, лучше выполнять реорганизацию индекса;
3) Если степень фрагментации более 30%, лучше выполнять перестроение индекса.

Отображение сведений о фрагментации таблицы:

DBCC SHOWCONTIG ('название таблицы');

Можно также посмотреть отчет о физическом состоянии индексов базы данных для этого правой кнопкой кликаем по БД–> Reports –> Standard Reports –> Index Physical Statistics

Отчет будет содержать информацию о состоянии индексов с рекомендациями Microsoft относительно их обслуживания


[OIL_COMPANY]
SQL Serv
on DESKTOP-LVNJ9AC at 12.10.2022 20:27:27

This report provides details on fragmentation of indexes within the Database. The report does not provide data for columnstore indexes and indexes on memory optimized tables.

Table Name						
⊟ dbo.Company						
Index Name	Index Type	# Partitions	Depth	Operation Recommended		
PK__Company__5F5EED5ACA00829D	CLUSTERED INDEX	⊟ 1	1	-		
		Partition No.	Avg. Fragmentation (%)	# Fragments	Avg. Pages Per Fragment	# Pages
		1	0	1	1	1
⊟ dbo.Cooperator						
Index Name	Index Type	# Partitions	Depth	Operation Recommended		
PK__Cooperat__D4C165D4401ED8EF	CLUSTERED INDEX	⊞ 1	1	-		
⊟ dbo.Department						
Index Name	Index Type	# Partitions	Depth	Operation Recommended		

Очистка кэш плана

Синтаксис для SQL Server:
DBCC FREEPROCCACHE

Подавление вывода всех регулярных информационных сообщений:
DBCC FREEPROCCACHE WITH NO_INFOMSGS

Очистка кэш плана для текущей базы данных:
USE Имя_БД;
GO
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;

Подробнее в технической документации

Оптимизация структуры таблиц

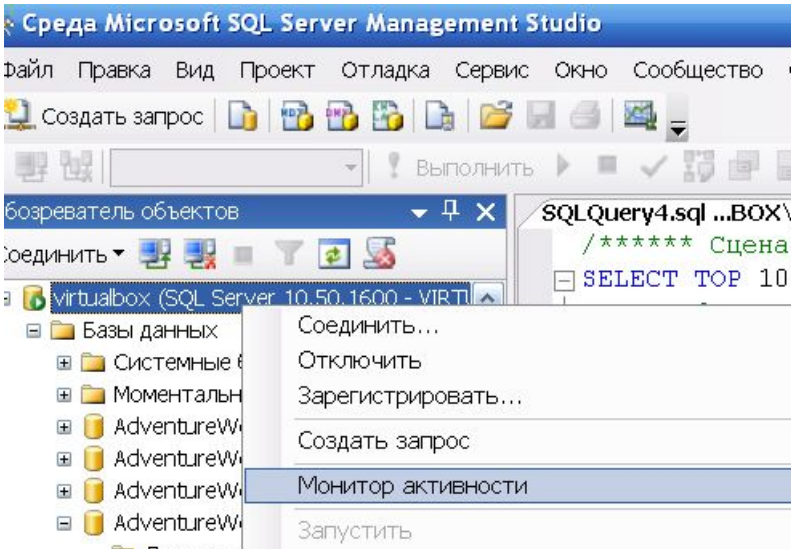
Рекомендации:

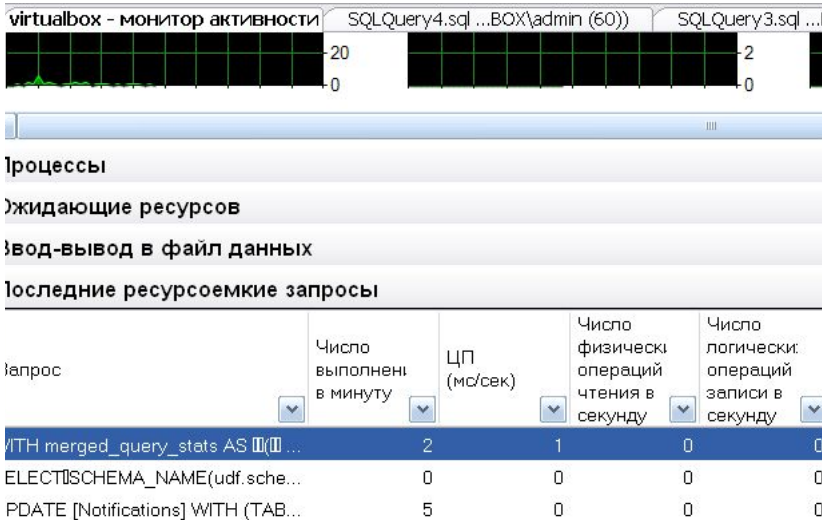
1) При создании таблиц выбирать выбирайте самый маленький из допустимых типов данных.
2) Выбирать тип данных VARCHAR или NVARCHAR, вместо TEXT.
3) Хранить изображений в БД нежелательно, можно в таблице хранить путь к файлу (локальный путь или URL).

Оптимизация запросов. Как найти проблемные запросы?

Activity Monitor – это утилита, позволяющая оценивать активность пользователей приложения или сети. Показывает текущее состояние SQL Server, осуществляемые на момент проверки процессы и то, как они отражаются на производительности СУБД. Activity Monitor выглядит как окно с несколькими вкладками. Администратор базы данных может открыть такие панели:

1) **Processes (процессы).** На этой панели отражаются все активные процессы и подробная информация по ним. В Processes также можно запустить скрипт, который автоматически анализирует выбранный процесс.
2) **Resource Waits (ожидающие ресурсы).** На этой панели отображается, какие ресурсы необходимы СУБД для выполнения заданных функций. В перечень ресурсов входит объем оперативной памяти и сервера, сети, компиляция и др. В этой же панели администратор базы данных может просмотреть общий и средний промежуток времени ожидания ресурсов.
3) **Data File I/O (ввод-вывод данных).** На этой панели отражаются все операции, связанные с внесением изменений в файлы БД, а также полная информация об этих файлах.
4) **Recent Expensive Queries (последние ресурсоемкие запросы).** На этой панели отражаются те запросы, которые были выполнены в течение ближайших 30 секунд, и обработка которых затребовала наибольшего числа ресурсов. В некоторых версиях SQL Server эта панель называется Activity Expensive Queries (активные ресурсоемкие запросы).

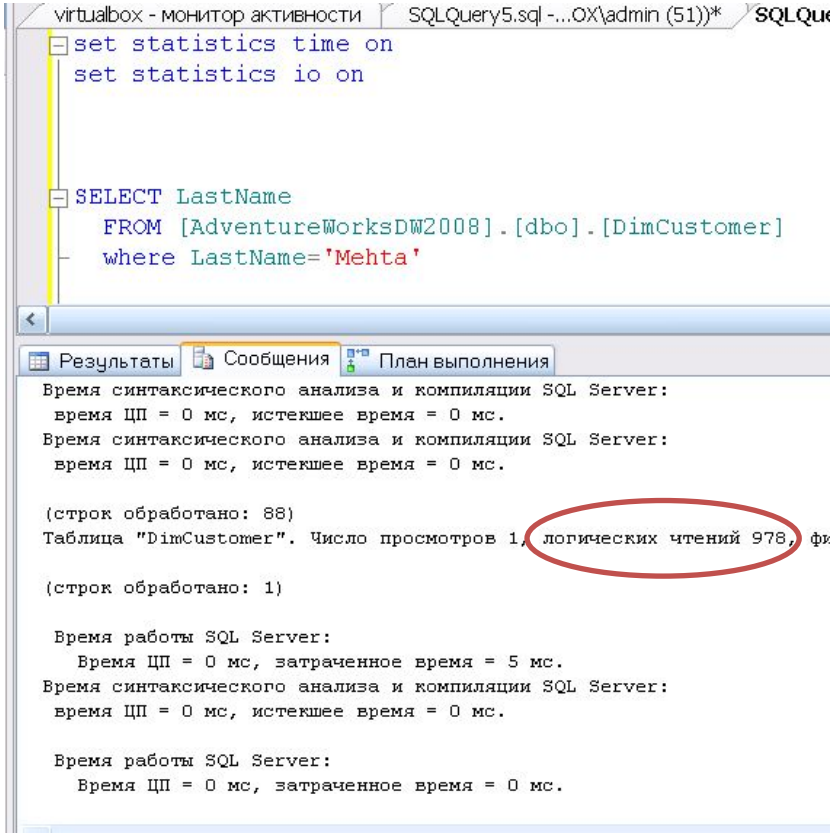




Как найти проблемные запросы? (SET STATISTICS IO, SET STATISTICS TIME)

Инструкция позволяет получить информацию о времени выполнения запроса:
SET STATISTICS TIME {ON | OFF}

Инструкция предписывает SQL Server предоставить отчет о реальной активности ввода-вывода при выполнении запроса:
SET STATISTICS IO {ON | OFF }



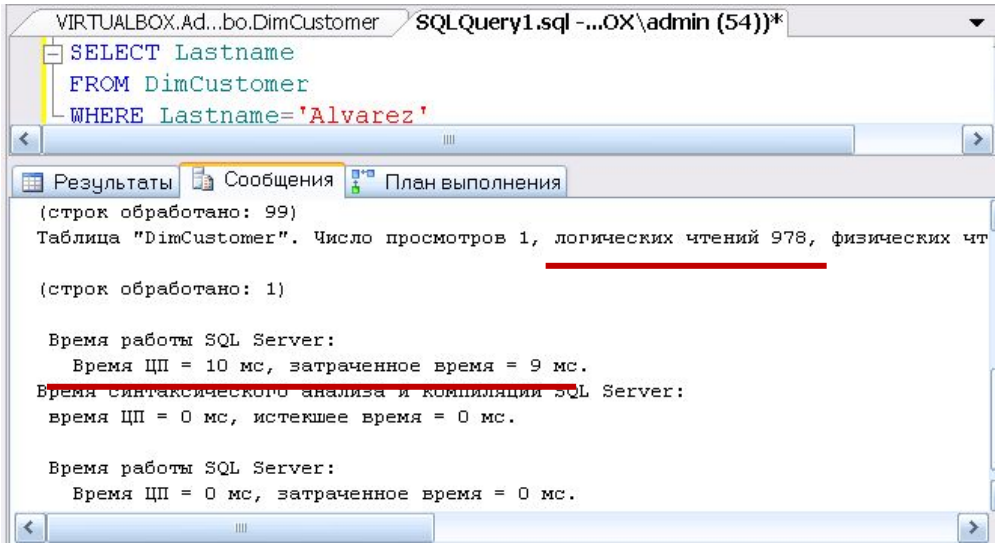
Оптимизация запросов

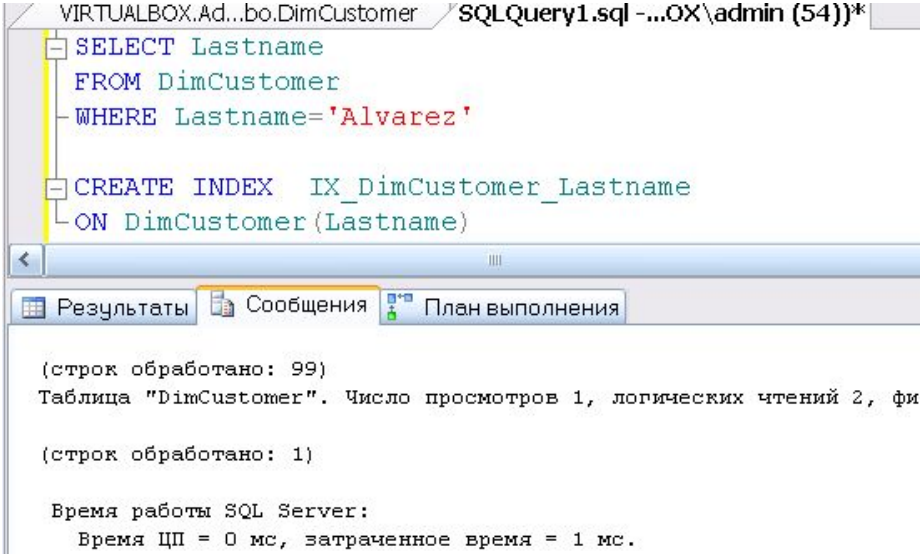
Рекомендации:

1) Кроме тех случаев, когда индекс создается автоматически, индексы можно добавить к полям, которые часто участвуют в запросах в разделе WHERE если применяется оператор сравнения «=».
2) Запросы с не колерующими подзапросами выполняются быстрее, чем с коррелирующими => по возможности писать не коррелирующие подзапросы вместо коррелирующих.
3) Не использовать * в SELECT. Не нужно извлекать данных больше чем нужно.
4) Если две или более таблицы часто участвую вместе в соединение, тогда для столбцов-соединения создайте соответствующие индексы (если они не были созданы БД по умолчанию, например, при добавлении ограничения Primary key).
5) Применять только тот тип JOIN, который вернет необходимые данные в контексте задания, без каких-либо дублей или лишней информации.
6) Применять сортировку строк только в случае необходимости.
7) Использовать как можно меньше столбцов в группировке. По возможности лучше использовать WHERE вместо HAVING.
8) Если в WHERE условие состоит из нескольких операторов AND, то условия должны располагаться в порядке возрастания вероятности истинности данного условия. Для OR наоборот.
9) Используйте IN вместо OR.
10) При выполнении соединения таблиц: соединение больших таблиц с маленькими менее оптимально, чем соединение маленьких таблиц с большими.
11) По возможности не применять конструкции DISTINCT, LIKE '%...' особенно на больших данных.
12) Использовать в запросе поиск по каждому индексу отдельно, т.е. применить быстрый поиск по каждому индексному полю

Индексы для ускорения поиска

Кроме тех случаев, когда индекс создается автоматически, индексы можно добавить к полям, которые часто участвуют в запросах в разделе WHERE если применяется оператор сравнения «=».





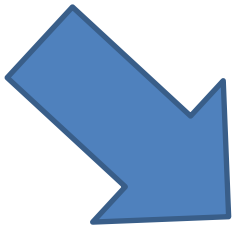
Индексы для ускорения поиска (покрывающий индекс)

VIRTUALBOX.Ad...bo.DimCustomer | SQLQuery1.sql -...OX\admin (54))*

```
SELECT Lastname,FirstName
FROM DimCustomer
WHERE Lastname='Alvarez'

CREATE INDEX  IX_DimCustomer_Lastname
ON DimCustomer(Lastname)
```

Результаты | Сообщения | План выполнения

```
(строк обработано: 99)
Таблица "DimCustomer". Число просмотров 1, логических чтений 314, физических

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 0 мс, затраченное время = 58 мс.
```



VIRTUALBOX.Ad...bo.DimCustomer | SQLQuery1.sql -...OX\admin (54))*

```
SELECT Lastname,FirstName
FROM DimCustomer
WHERE Lastname='Alvarez'

CREATE INDEX  IX_DimCustomer_Lastname_FirstName
ON DimCustomer(Lastname) INCLUDE(FirstName)
```

Результаты | Сообщения | План выполнения

```
(строк обработано: 99)
Таблица "DimCustomer". Число просмотров 1, логических чтений 4, физиче

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 0 мс, затраченное время = 4 мс.
Время синтаксического анализа и компиляции SQL Server:
```

Не коррелирующий подзапрос

```
SELECT c.*
From DimCustomer c
WHERE c.GeographyKey IN (SELECT g.GeographyKey
                         From DimGeography g)
```

Результаты | Сообщения | План выполнения

```
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.

 Время работы SQL Server:
   Время ЦП = 0 мс, затраченное время = 0 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 5 мс, истекшее время = 5 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упрежд
Таблица "DimCustomer". Число просмотров 1, логических чтений 978,
Таблица "DimGeography". Число просмотров 1, логических чтений 15,

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 241 мс, затраченное время = 666 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.
```

Обычно не коррелирующие подзапросы применяются в запросах, в которых значение определённого столбца сравнивается со значением, возвращаемым подзапросом, в запросах с IN, ALL и ANY.

Добавили некластерный индекс для поля GeographyKey в таблице DimCustomer

```
SELECT c.*
From DimCustomer c
WHERE c.GeographyKey IN (SELECT g.GeographyKey
                         From DimGeography g)

--create index IX_DimCustomer_GeographyKey on DimCustomer(GeographyKey)
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 0 мс, истекшее время = 0 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упреждающи
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтений 0, упрежд
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтений 0, упрежд

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 210 мс, затраченное время = 506 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.
```

Коррелирующий подзапрос

```
SELECT c.*
From DimCustomer c
WHERE EXISTS (SELECT g.GeographyKey
                      From DimGeography g
                      WHERE g.GeographyKey=c.GeographyKey)
```

Результаты | Сообщения | План выполнения

```
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 1 мс.

 Время работы SQL Server:
   Время ЦП = 0 мс, затраченное время = 0 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упреждающих
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтений 0, упрежде
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтений 0, упрежде

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 231 мс, затраченное время = 509 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.
```

Коррелирующий подзапрос содержит ссылку на таблицу, которая была объявлена во внешнем запросе.

Добавили некластерный индекс для поля GeographyKey в таблице DimCustomer

```
SELECT c.*
From DimCustomer c
WHERE EXISTS (SELECT g.GeographyKey
                      From DimGeography g
                      WHERE g.GeographyKey=c.GeographyKey)

--create index IX DimCustomer GeographyKey on DimCustomer(GeographyKey)
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 0 мс, истекшее время = 0 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упреждающих чтений
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтений 0, упреждающих чте
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтений 0, упреждающих чте

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 230 мс, затраченное время = 475 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 1 мс.
```

Оператор JOIN (выборка из всех столбцов)

```
SELECT c.*
From DimCustomer c JOIN DimGeography g
                   ON g.GeographyKey=c.GeographyKey
```

Результаты | Сообщения | План выполнения

```
 Время работы SQL Server:
   Время ЦП = 0 мс, затраченное время = 0 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 2 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упреждающих чтений
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтений 0, упреждающих чте
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтений 0, упреждающих чте

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 230 мс, затраченное время = 481 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.
```

Добавили некластерный индекс для поля GeographyKey в таблице DimCustomer

```
SELECT c.*
From DimCustomer c JOIN DimGeography g
                   ON g.GeographyKey=c.GeographyKey


--create index IX_DimCustomer_GeographyKey on DimCustomer(GeographyKey)
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 0 мс, истекшее время = 0 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упреждающи
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтений 0, упрежда
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтений 0, упрежда

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 210 мс, затраченное время = 481 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.

 Время работы SQL Server:
```

Оператор JOIN (выборка данных не из всех столбцов таблиц)

```
SELECT c.*
From DimCustomer c JOIN DimGeography g
                        ON g.GeographyKey=c.GeographyKey


--create index IX_DimCustomer_GeographyKey on DimCustomer(GeographyKey)
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 0 мс, истекшее время = 0 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, упреждающих
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтений 0, упрежд
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтений 0, упрежд

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 210 мс, затраченное время = 481 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.

 Время работы SQL Server:
```

Выборка данных из определенного количества столбов быстрее чем выборка данных из всех столбцов => не нужно извлекать данных больше чем нужно.

```
SELECT c.FirstName, c.LastName, g.City
 From DimCustomer c JOIN DimGeography g
                        ON g.GeographyKey=c.GeographyKey
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 5 мс, истекшее время = 5 мс.

(строк обработано: 18484)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0
Таблица "DimCustomer". Число просмотров 1, логических чтений 978, физических чтен
Таблица "DimGeography". Число просмотров 1, логических чтений 15, физических чтен

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 100 мс, затраченное время = 294 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.

 Время работы SQL Server:
```

Поиск по каждому индексному полю отдельно

```
SELECT GeographyKey, CustomerAlternateKey
 From DimCustomer
 WHERE   GeographyKey=5 or CustomerAlternateKey='1'
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 0 мс, истекшее время = 0 мс.

(строк обработано: 80)
Таблица "DimCustomer". Число просмотров 1, логических чтений 258, фи

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 0 мс, затраченное время = 70 мс.
Время синтаксического анализа и компиляции SQL Ser
 время ЦП = 0 мс, истекшее время = 0 мс.
```

Если оба поля выборки имеют не кластеризованные индексы

```
SELECT GeographyKey, CustomerAlternateKey
 From DimCustomer
 WHERE   GeographyKey=5
 UNION ALL
 SELECT GeographyKey, CustomerAlternateKey
 From DimCustomer
 WHERE   GeographyKey<>5 AND CustomerAlternateKey='1'
```

Результаты | Сообщения | План выполнения

```
 время ЦП = 7 мс, истекшее время = 7 мс.

(строк обработано: 80)
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, уп
Таблица "DimCustomer". Число просмотров 2, логических чтений 77, физических чтений 0,
Таблица "Worktable". Число просмотров 0, логических чтений 0, физических чтений 0, уп

(строк обработано: 1)

 Время работы SQL Server:
   Время ЦП = 10 мс, затраченное время = 41 мс.
Время синтаксического анализа и компиляции SQL Server:
 время ЦП = 0 мс, истекшее время = 0 мс.
```

Зачем нужна оптимизация запросов?

Запросы — это критически важный компонент для повышения общей производительности базы данных.

Неправильные запросы выполняются, пока не истечет время ожидания, они расходуют такие ресурсы, как ЦП и память. Это мешает доступу к критически важной бизнес-информации. Даже один неправильный запрос может вызвать серьезные проблемы с производительностью базы данных.

Рекомендации для увеличения производительности:

Для увеличения производительности, то есть для быстрого выполнения запросов, следует помнить некоторые правила составления строк запросов:

1) **Избегать NOT** - команды отрицания выполняются в несколько этапов, что увеличивает нагрузку на сервер.

2) **Избегать LIKE** - этот оператор сравнения применяет более мягкие шаблоны сравнения, чем оператор =, что увеличивает необходимое число этапов фильтрации.

3) **Применять точные шаблоны поиска** - применение подстановочных символов увеличивает время выполнения запроса, так как для проверки всех вариантов подстановки требуется дополнительные ресурсы сервера.

4) **Избегать ORDER** - команда сортировки требует упорядочивания строк таблицы вывода, что задерживает получение результата.

5) **Не возвращайте клиенту большее количество столбцов или строк**, чем действительно необходимо (Не используй * в Select).

6) **Как можно раньше отфильтруйте данные**. Не нужно выполнять большой тяжелый подзапрос для всех строк таблицы. Сначала отфильтруйте нужные строки.

Оптимизация запросов SELECT

1) Не читайте больше данных, чем надо. Не используйте * Не возвращайте клиенту большее количество столбцов или строк, чем действительно необходимо.

2) Если ваше приложение позволяет пользователям выполнять запросы, но вы не можете отсечь лишние тысячи возвращаемых строк, используйте оператор TOP внутри инструкции SELECT.

3) Как можно раньше отфильтруйте данные. Не нужно выполнять большой тяжелый подзапрос для всех строк таблицы. Сначала отфильтруйте нужные строки.

4) При объявлении полей всегда следует использовать размер, который нужен, и не выделять лишние байты про запас.

5) Если сортируете по дате создания записи, то попробуйте сортировать просто по id (первичный ключ с identity(1,1)).

Корректно используйте JOIN

1) Если Вы имеете две или более таблиц, которые часто соединяются, тогда столбцы, используемые для соединения должны иметь соответствующий индекс.

2) Для лучшей производительности, столбцы, используемые в соединения должны иметь одинаковые типы данных. И если возможно, это должны быть числовые типы данных, вместо символьных типов.

3) Избегайте соединения таблицы по столбцам с малым числом уникальных значений. Если столбцы, используемые при соединении, имеют мало уникальных значений, то SQL сервер будет просматривать всю таблицу, даже если по данному столбцу существует индекс. Для наилучшей производительности соеденение таблиц должно производится по столбцам с уникальными индексами.

Оптимизация WHERE в запросе SELECT

1) Если where состоит из условий, объединенных **AND**, они должны располагаться в порядке возрастания вероятности истинности данного условия. Чем быстрее мы получим false в одном из условий - тем меньше условий будет обработано и тем быстрее выполняется запрос.

2) Если where состоит из условий, объединенных **OR**, они должны располагаться в порядке уменьшения вероятности истинности данного условия. Чем быстрее мы получим true в одном из условий - тем меньше условий будет обработано и тем быстрее выполняется запрос.

3) Исопльзуйте IN вместо OR. Операция IN работает гораздо быстрее, чем серия OR. Запрос *"... WHERE column1 = 5 OR column1 = 6"* медленнее чем *"...WHERE column1 IN (5, 6)"*.

4) Используйте Exists вместо Count >0 в подзапросах. Используйте where exists (select id from t1 where id = t.id) вместо where count(select id from t1 where id=t.id) > 0

5) LIKE. Эту операцию следует использовать только при крайней необходимости, потому что лучше и быстрее использовать поиск, основанный на full-text индексах.

Советы по оптимизации хранимых процедур и SQL пакетов

1) Для обработки данных используйте хранимые SQL процедуры.

Когда хранимая процедура выполняется в первый раз (и у нее не определена опция WITH RECOMPILE), она оптимизируется, для нее создается план выполнения запроса, который кешируется SQL сервером. Если та же самая хранимая процедура вызывается снова, она будет использовать кешированный план выполнения запроса, что экономит время и увеличивает производительность.

2) Всегда включайте в ваши хранимые процедуры инструкцию "SET NOCOUNT ON". Если Вы не включите эту инструкцию, тогда каждый раз при выполнении запроса SQL сервер отправит ответ клиенту, указывающему число строк, на которые воздействует запрос.

Рекомендации для увеличения производительности: временные таблицы для больших таблиц, табличные переменные - для малых (меньше 1000).

Если требуется хранить промежуточные данные в таблицах, то используйте табличные переменные (@t1) для малых таблиц, а временные таблицы (#t1) - для больших. Временные таблицы бывают локальные (#)и глобальные (##)

Например, создадим локальную временную таблицу:

При определении временной таблицы имеет смысл проверить на существование:

```
IF OBJECT_ID('tempdb..# ProductSummary ') IS NOT NULL begin
   DROP TABLE #ProductSummary
End

CREATE TABLE #ProductSummary
(ProdId INT primary key IDENTITY(1,1),
ProdName NVARCHAR(20),
Price Dec(8,2))

INSERT INTO #ProductSummary
VALUES ('Nokia 8', 18000),
('iPhone 8', 56000)

SELECT * FROM #ProductSummary

-- Очистка временной таблицы
TRUNCATE TABLE #ProductSummary
```

Временная таблица храниться физически в tempdb. Временная таблица удаляется автоматически после завершения последнего сеанса с ее

Обобщенные табличные выражения

Кроме временных таблиц MS SQL Server позволяет создавать обобщенные табличные выражения (common table expression или CTE), которые являются производными от обычного запроса и в плане производительности являются более эффективным решением, чем временные. Обобщенное табличное выражение задается с помощью ключевого слова WITH:

```
WITH OrdersInfo AS
(
   SELECT ProductId,
      SUM(ProductCount) AS TotalCount,
      SUM(ProductCount * Price) AS TotalSum
   FROM Orders
   GROUP BY ProductId
)

SELECT *
FROM OrdersInfo
```

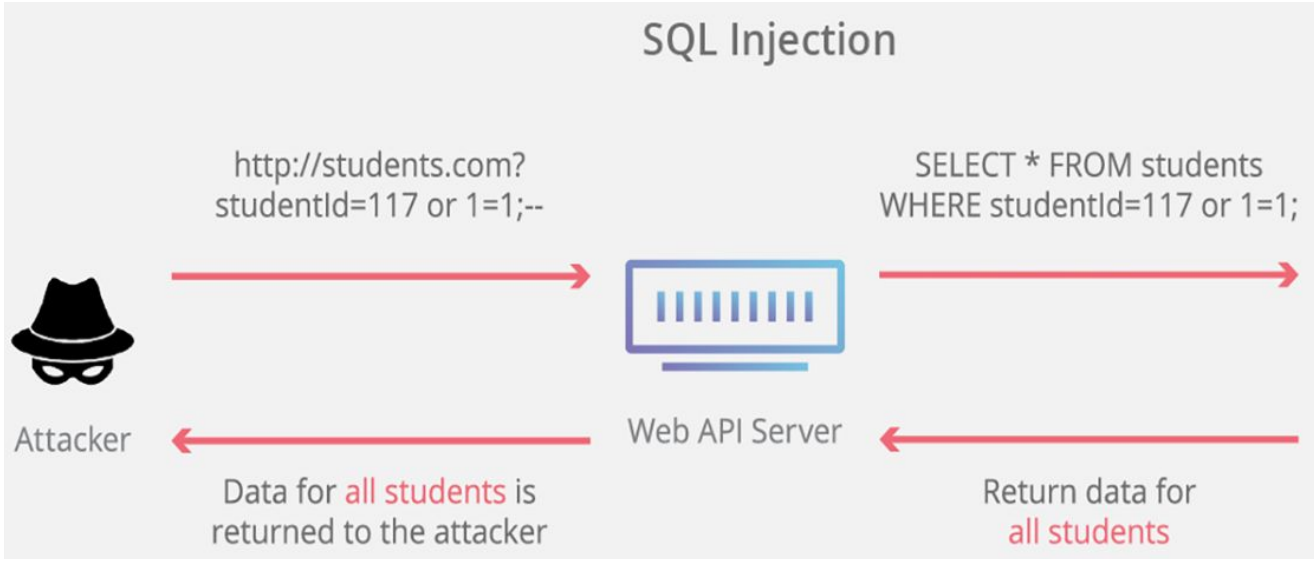
Обобщённые табличные выражения были добавлены в SQL для упрощения сложных длинных запросов, особенно с множественными подзапросами. Их главная задача – улучшение читабельности, простоты написания запросов и их дальнейшей поддержки. Это происходит за счёт сокрытия больших и сложных запросов в созданные именованные выражения, которые потом используются в основном запросе.

Лучшие кандидаты на установку индекса

1) Это поля, по которым идет JOIN
2) Поля связи, участвующие в подзапросах
3) Поля, по которым идет фильтрация в WHERE
4) Поля, по которым выполняется сортировка

SQL инъекция (SQL injection - SQLi)

SQL инъекция – это уязвимость веб-безопасности, которая позволяет злоумышленнику вмешиваться в запросы, которые приложение делает к своей базе данных для манипулирования базой данных и получения доступа к потенциально ценной информации.



Проблемы, вызванные SQL Инъекцией:

1. **Утечка данных:** Злоумышленники могут получить доступ к конфиденциальным данным, таким как пароли, кредитные карты или личная информация.
2. **Изменение данных:** Атакующие могут изменять данные в базе данных, что может привести к разрушению целостности данных.
3. **Отказ в обслуживании:** SQL инъекция может вызвать перегрузку сервера и отказ в обслуживании, так как запросы выполняются в бесконечном цикле.

Возможные SQL инъекции (SQL внедрения)

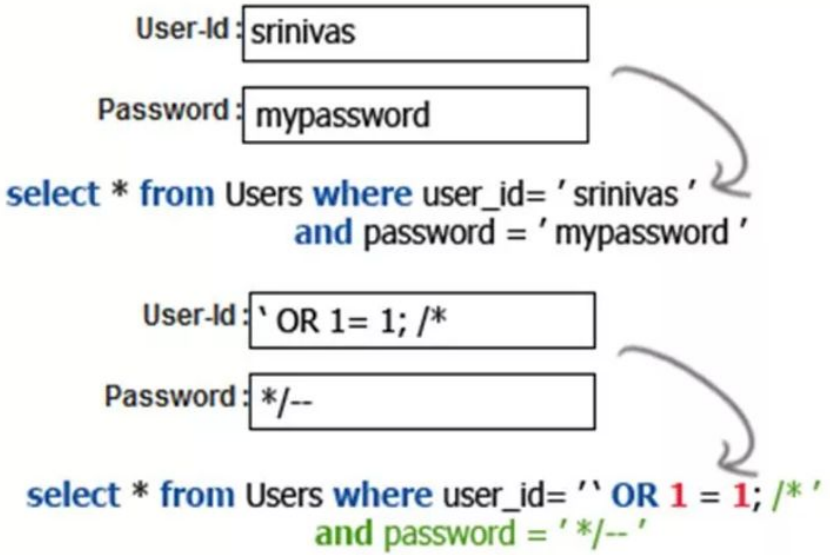
Наиболее простые:

1. Добавление к WHERE дополнительное условие, которое заведомо вернет истинный ре

Условие 1=1 всегда будет истинным, и база данных воспримет его как команду вернуть в ответе все, что находится в таблице! Таким образом, этот SELECT-запрос запрашивает все данные на всех сотрудников.

2. Присоединение к запросу результатов другого запроса. Делается это через оператор

3. Закомментирование части запро



Способы обнаружения SQL инъекции

Существует несколько способов обнаружения SQL инъекций:

1. **Использование специализированных инструментов.** Существуют различные инструменты, такие как SQLMap, DBeaver, Acunetix, Burp Suite и другие, которые автоматически сканируют веб-приложения и ищут уязвимости, включая SQL инъекции.
2. **Проверка наличия специфических символов.** SQL инъекции обычно осуществляются путем вставки специальных символов в пользовательский ввод, чтобы обмануть фильтры и выполнить нежелательные SQL-запросы. Проверка наличия таких символов, таких как одинарные кавычки ('), двойные кавычки ("), точки с запятой (;) и других, может помочь обнаружить возможную SQL инъекцию.
3. **Проверка неправильного поведения при обработке пользовательского ввода.** Если веб-приложение не корректно обрабатывает пользовательский ввод, это может привести к возможной SQL инъекции. Например, если система не экранирует или не фильтрует пользовательский ввод, это может сигнализировать о наличии уязвимости.
4. **Попытка найти ошибки в сообщениях об ошибках или логах.** Часто, когда веб-приложение сталкивается с ошибкой SQL-запроса, она может вернуть сообщение об ошибке или записать ее в логи. Анализирование таких сообщений или логов может помочь выявить наличие SQL инъекции.
5. **Тестирование наличия уязвимости с помощью специально сформированных запросов.** Использование инструментов для создания и отправки специально сформированных запросов веб-приложению может помочь выявить наличие уязвимостей в защите от SQL инъекций.

Важно отметить, что эти методы могут помочь выявить наличие SQL инъекций, но не гарантируют 100% точности. Рекомендуется проводить комплексное тестирование безопасности веб-приложений с использованием различных методов и инструментов.

Возможные решения:

1. **Использование подготовленных запросов:** используйте параметризованные запросы, чтобы передавать данные в базу данных, вместо конкатенации строк. Это помогает избежать инъекций.
2. **Экранизация данных:** экранируйте специальные символы, такие как одинарные кавычки, перед внесением данных в SQL-запросы.
3. **Ограничение привилегий БД:** минимизируйте привилегии пользователя базы данных, чтобы ограничить возможность выполнения опасных операций.
4. **Обновление и мониторинг безопасности:** регулярно обновляйте и мониторьте ваше ПО, чтобы устранить уязвимости, которые могут быть использованы для атак.
5. **Валидация ввода:** проводите строгую валидацию всех пользовательских входных данных, чтобы предотвратить внесение вредоносного кода.
6. **Использование файрвола приложений:** защитите вашу систему с помощью WAF (Web Application Firewall), который может обнаруживать и блокировать SQL инъекции.

SQL инъекция - серьезная угроза, и предупреждение этой атаки требует сочетания технических и образовательных мероприятий.

Службы Microsoft SQL Server

Наименование службы	Назначение
Database Engine	Управление реляционными БД
Analysis Services	Управление OLAP – кубами и интеллектуальный анализ данных
Integration Services	Поддержка решений по извлечению, преобразованию и загрузке данных
Reporting Services	Управление отчетами, построенными на основе SQL – запросов к реляционным БД
Full-Taxt Search	Управление полнотекстовым поиском
SQL Server Agent	Автоматизация административных задач
SQL Server Browser	Управление соединениями

Database Engine

Database Engine является ядром системы управления реляционной БД.

Может быть установлено несколько экземпляров службы Database Engine.

Один экземпляр Database Engine может быть службой по умолчанию (с именем MS SQL SERVER), другие экземпляры должны иметь уникальные имена.

Каждый экземпляр службы Database Engine требует отдельной инсталляции, конфигурации и настройки безопасности.

Один Database Engine может обеспечить доступ к нескольким БД.

Системные базы данных

Системная БД	Назначение
master	Хранит все системные данные Database Engine, а также информацию о других БД
msdb	Используется службами SQL Server Agent (выполнение заданий по расписанию), Database Mail (формирование уведомлений по электронной почте), а также хранит информацию о резервном копировании БД
tempdb	Пространство для временных объектов Database Engine и пользовательских временных таблиц. БД пересоздается при каждой перезагрузке
model	Шаблон, используемый при создании всех БД, управляемых экземпляром Database Engine
resource	БД, используемая только для чтения. Содержит системные объекты экземпляра Database Engine. Файлы БД являются скрытыми и не отображаются в MS SQL Server

Утилиты Microsoft SQL Server:

- SQL Server Management Studio.
- SQL Server Books Online.
- SQLCMD Microsoft.
- SQL Configuration Manager.

Система управления базами данных

Основные функции СУБД:

1) управление данными во внешней памяти (на дисках);
2) управление данными в оперативной памяти с использованием дискового кэш;
3) журнализация изменений, резервное копирование и восстановление базы данных
после сбоев;
4) поддержка языков БД (язык определения данных, язык манипулирования данными и т.д.).

Система управления базами данных

Обычно современная СУБД содержит следующие компоненты:

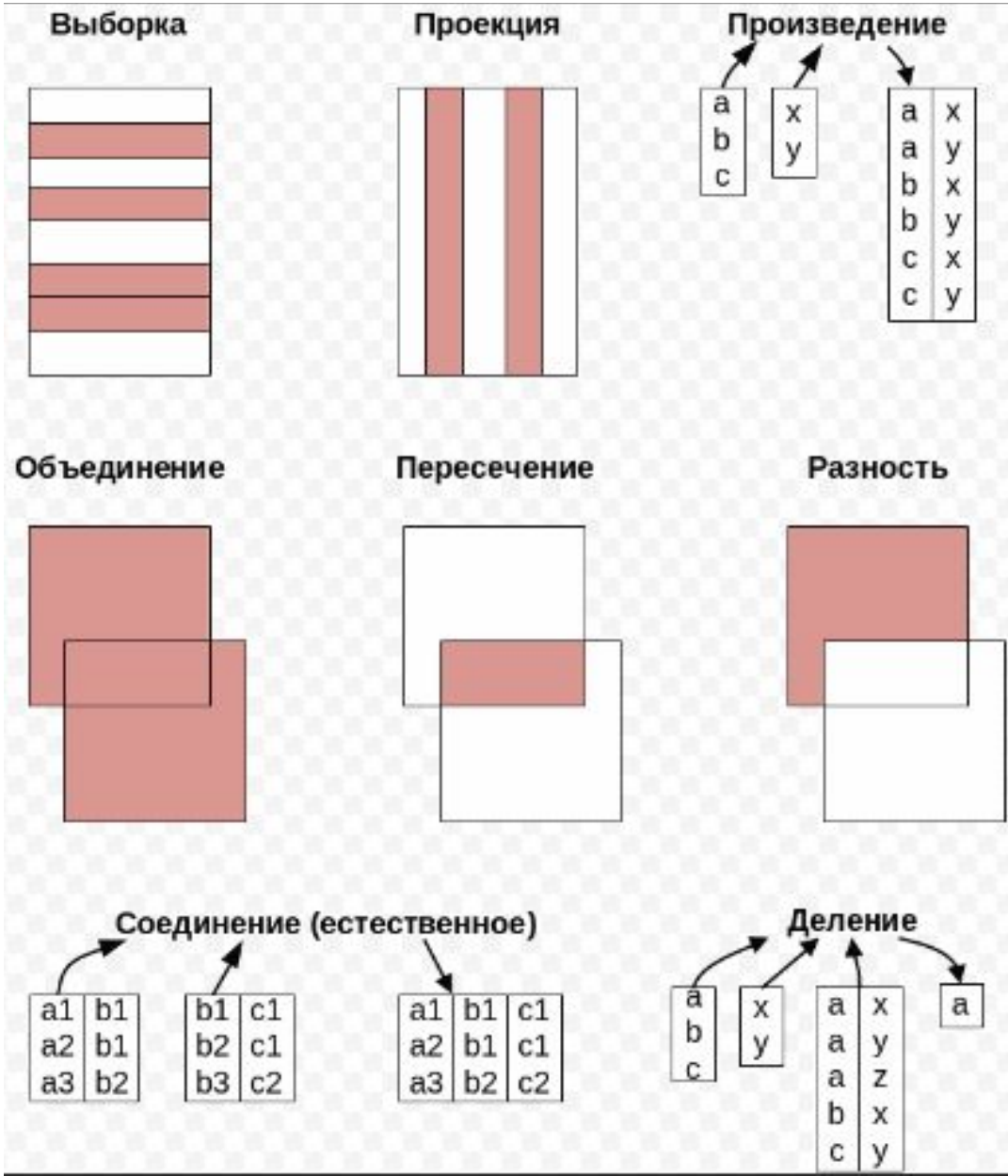
1) ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;
2) процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;
3) подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;
4) сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

Реляционная алгебра

Реляционная алгебра состоит из операций над множествами (атрибутами и кортежами).

Операции реляционной алгебры также называют реляционными операциями.

Основные операции реляционной алгебры: объединение, пересечение, разность, декартово произведение, деление, проекция, выборка (селекция), соединение.



Операция реляционной алгебры: проекция

Операция проекции (унарная операция) выделяет атрибуты только из указанных доменов.

$\pi_{a1,\ldots,an}(R)$ где $a_1,\ldots,a_n$ — список полей, подлежащих выборке.

таблица «Отдел продаж"				
Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Сергеевна	26	91
Фомин	Егор	Гришин	38	80

$\pi_{Возраст, Вес}$(Отдела продаж)

Возраст	Вес
38	80
27	65
30	80
26	91

Эквивалентный SQL-запрос:

```
SELECT DISTINCT Возраст, Вес
FROM [Отдел продаж]
```

Операция реляционной алгебры: объединение (union)

Операция объединения над двумя отношениями, создает новое отношение, состоящее из всех кортежей исходных отношений. Общие для исходных отношений кортежи в новом отношении не повторяются (не дублируются).

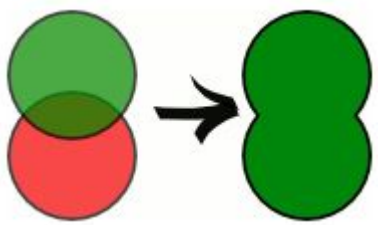


таблица «Отдел продаж"

Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Сергеевна	26	91
Фомин	Егор	Гришин	38	80

таблица «Отдел кадров"

Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Кузьмин	Виктор	Васильевич	40	77
Василькова	Марфа	Макарова	32	64
Фомин	Егор	Гришин	38	80

Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Скрнгеевна	26	91
Фомин	Егор	Гришин	38	80
Кузьмин	Виктор	Васильевич	40	77
Василькова	Марфа	Макарова	32	64

Эквивалентный SQL-запрос:

```
SELECT Фамилия, Имя,
Отчество,
Возраст, Вес
FROM [Отдел продаж]
UNION
SELECT Фамилия, Имя,
Отчество,
Возраст, Вес
```

Операция реляционной алгебры: выборка (селекция) (selection)

Операция выборки производится над кортежами одного отношения (унарная операция).

Результат выборки - новое отношение, состоящее из котежей исходного отношения, удовлетворяющих заданному условию.

как $\sigma_{a\theta b}(R)$ или $\sigma_{a\theta v}(R)$, где:

a, b — имена атрибутов;

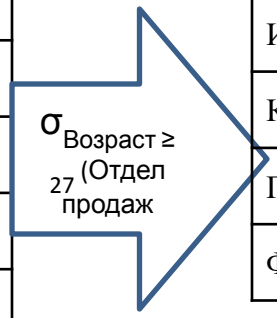
θ — оператор сравнения из множества {<; ≤; =; ≥; >};

v — константа;

R — отношение.

Таблица «Отдел продаж"

Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Скрнгеевна	26	91
Фомин	Егор	Гришин	38	80



Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Фомин	Егор	Гришин	38	80

Эквивалентный SQL-запрос: SELECT *
FROM [Отдел продаж]
WHERE Возраст >= 27

Операция реляционной алгебры: пересечение (intersection)

Операция пересечения над двумя отношениями, создает новое отношение, состоящее из кортежей, принадлежащих обоим исходным отношениям.

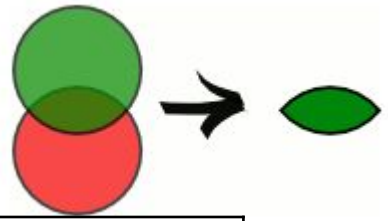
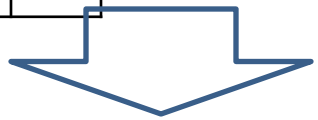


таблица «Отдел продаж"				
Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Сергеевна	26	91
Фомин	Егор	Гришин	38	80

таблица «Отдел кадров"				
Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Кузьмин	Виктор	Васильевич	40	77
Василькова	Марфа	Макарова	32	64
Фомин	Егор	Гришин	38	80



Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Фомин	Егор	Гришин	38	80

Эквивалентный SQL-запрос:

```
SELECT Фамилия, Имя, Отчество,
Возраст, Вес
FROM [Отдел продаж]
INTERSECT
SELECT Фамилия, Имя, Отчество,
Возраст, Вес
FROM [Отдел кадров]
```

Операция реляционной алгебры: разность

Операция разность над двумя отношениями создает новое отношение, состоящее из кортежей, принадлежащих первому отношению и не принадлежащих второму.

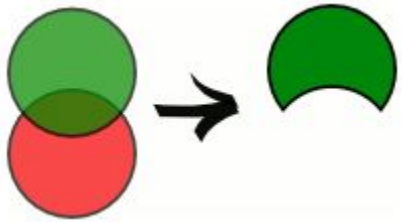
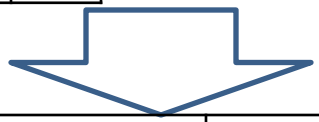


таблица «Отдел продаж"				
Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Сергеевна	26	91
Фомин	Егор	Гришин	38	80

таблица «Отдел кадров"				
Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Кузьмин	Виктор	Васильевич	40	77
Василькова	Марфа	Макарова	32	64
Фомин	Егор	Гришин	38	80



Фамилия	**Имя**	**Отчество**	**Возраст**	**Вес**
Иванов	Илья	Инакентевич	38	80
Костина	Марина	Мунирован	27	65
Петров	Исак	Михайлович	30	80
Сидорова	Кира	Сергеевна	26	91

Эквивалентный SQL-запрос:

```
SELECT Фамилия, Имя,
Отчество, Возраст, Вес
FROM [Отдел продаж]
EXCEPT
SELECT Фамилия, Имя,
Отчество, Возраст, Вес
FROM [Отдел кадров]
```

Операция реляционной алгебры: произведение (декартовое произведение) (cartesian product)

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321

произведение (декартовое произведение)

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	100	Администрация
3	Синицына	Инна	100	Администрация
1	Иванов	Иван	101	Информационный отдел
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел

Эквивалентный SQL-запрос:

```
SELECT Coop_id, coop.Surname, coop.Name, dep.Dept_id ,dep.Name
FROM Cooperator coop, Department dep
```

Операция реляционной алгебры: соединение (join)

Операция соединения над двумя отношениями создает новое отношение, состоящее из всех атрибутов исходных отношений и объединяющее только те кортежи исходных отношений в которых значения общих атрибутов совпадают.

Table Cooperator

Coop_id	Surname	Name	Birthday	Dept_id
1	Иванов	Иван	01.01.1990	100
2	Сидоров	Петр	01.03.1995	101
3	Синицына	Инна	21.05.1990	101
4	Егоров	Валерий	12.01.1991	*NULL*
5	Воронина	Наталья	23.02.1993	103

Table Department

Dept_id	Name	Telephone
100	Администрация	12345
101	Информационный отдел	54321
102	Проектный отдел	23431
103	Отдел кадров	45673

Coop_id	Surname	Name	Dept_id	Name
1	Иванов	Иван	100	Администрация
2	Сидоров	Петр	101	Информационный отдел
3	Синицына	Инна	101	Информационный отдел
5	Воронина	Наталья	103	Отдел кадров

Эквивалентный SQL-запрос:

SELECT Coop_id , coop.Surname, coop.Name**,** dep. Dept_id ,dep.Name
FROM Cooperator coop**,** Department dep
WHERE coop.Dept_id =dep.Dept_id

Выполняется через операции декартова произведения и выборки.

Операция реляционной алгебры: деление (division)

Операция деления над двумя отношениями создает новое отношение, состоящее из
всех картежей первого отношения для которых присутствуют все комбинации значений из второго отношения. Выполняется через вычитание, декартово произведение и проекцию

таблица. Мульфильм			
№	**Код_канала**	**Название_мультфильма**	**Название_канала**
1	1	The Simpsons	RenTV
2	1	The Simpsons	2x2
3	1	The Simpsons	CTC
4	2	Family Guy	RenTV
5	2	Family Guy	2x2
6	*NULL*	Duck Tales	CTC
7	*NULL*	Duck Tales	2x2

таблица. Канал	
Код_канала	**Название_канала**
1	RenTV
2	2x2

Код_мультфильма	**Название_мультфильма**
1	The Simpsons
2	Family Guy

Эквивалентный SQL-запрос привести затруднительно, возможен следующий вариант:

```
SELECT *
FROM Мультфильмы INNER JOIN Каналы
```

Подход NoSQL

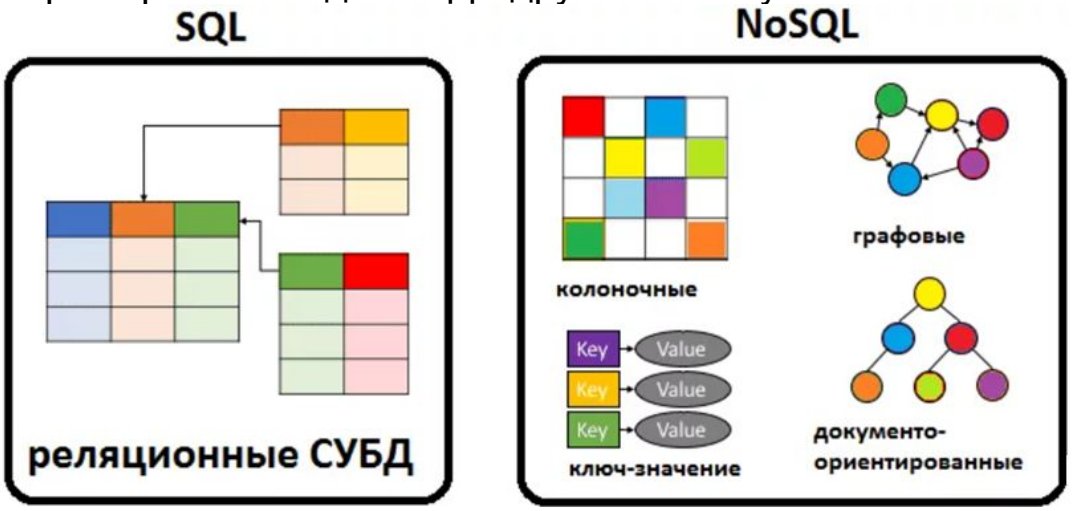
Термин NoSQL обозначает нереляционные базы данных, которые хранят данные в формате, отличном от реляционных таблиц.

Термин NoSQL принято переводить, как «не только SQL», поскольку некоторые из этих баз данных поддерживают запросы, совместимые с SQL.

В базах данных NoSQL применяются модели данных, оптимизированные под решение конкретных задач.

Базы данных NoSQL из-за высокой масштабируемости и высокой доступности используются в веб-приложениях реального времени, больших данных, в онлайн-играх, в проектах Интернет вещей, социальных сетях, поисковых системах, приложениях для онлайн-рекламы.

У каждой БД NoSQL есть свои правила для работы с данными, а также языки, соответственно быстро перейти от одной БД к другой не получиться.



Big Data или **большие данные** — это структурированные или неструктурированные массивы данных большого объема. Их обрабатывают при помощи специальных автоматизированных инструментов, чтобы использовать для статистики, анализа, прогнозов и принятия решений.

Интернет вещей (IoT, Internet of Things) — объединение разных устройств в общую сеть, в которой они могут собирать информацию, обрабатывать ее и обмениваться данными между собой, с человеком и серверами в дата-центре или облаке.

Модель данных «Ключ-значение»

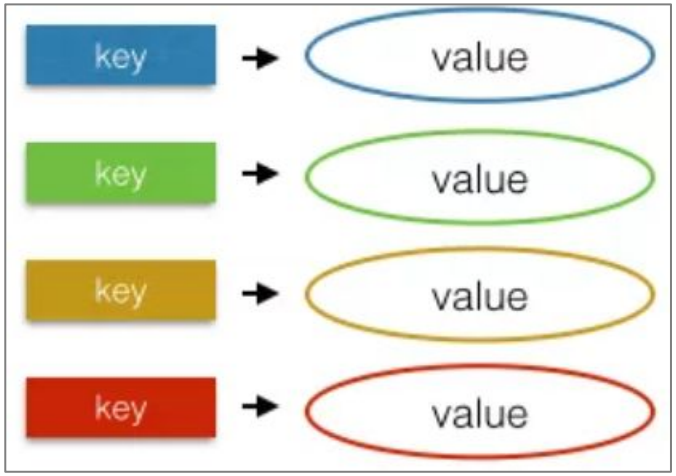
Большинство БД поддерживают только самые простые операции запроса, вставки и удаления. Чтобы частично или полностью изменить значение, приложение всегда перезаписывает существующее значение целиком. В большинстве реализаций атомарной операцией считается чтение или запись одного значения. Запись больших значений занимает относительно долгое время.

Поиск по значениям отсутствует, есть только по ключу/ключам. Все сведения о схеме поддерживаются и применяются на уровне приложения.

Главные плюсы: масштабируемость, простота.

Основные недостатки: не поддерживаются связи между объектами, в основном запросы только с поиском по ключу.

Возможно применить для хранения изображений, сессий, счетчиков посещений или просмотров, в игровых и рекламных приложениях, в проектах интернет вещей и т.д.



Key	Value (Opaque)
User:2:friends	{23, 76, 233, 11}
User:2:settings	Theme: dark, cookies: false
User:3:friends	[234, 3466, 86, 55]

в ключе могут хранится данные и настройки учетной записи

Интернет вещей (IoT, Internet of Things) — объединение разных устройств в общую сеть, в которой они могут собирать информацию, обрабатывать ее и обмениваться данными между собой, с человеком и серверами в дата-

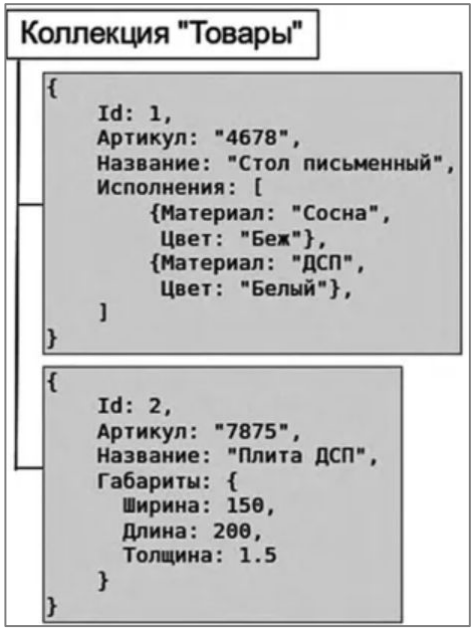
Документно-ориентированная модель данных

Возможно реализовать большую вложенность и сложность структуры данных, чем в БД «ключ-значение» (например, документ вложенный в документ, вложенный в документ). Данных хранятся в Json документах.

Основные достоинства: возможно хранение слабоструктурированных объектов; быстрое выполнение запросов.

Основные недостатки: при запросе выводиться весь документ, даже если необходимо было какое-то одно значение, что сказывается на производительности; медленный процесс обновления данных (данные могут храниться распределено на нескольких серверах); возможно дублирование данных.

Возможно применение для каталогов, пользовательские профилей, в CMS-системах, издательском деле и документальном поиске.



CMS (Content Management System) — это система управления, движок, платформа или конструктор, который позволяет управлять содержимым сайта.

Колоночная модель данных

Основная идея колоночной модели данных — это хранение данных не по строкам, как в реляционных таблицах, а по колонкам. Это означает, что с точки зрения SQL-клиента данные представлены как обычно в виде таблиц, но физически эти таблицы являются совокупностью колонок, каждая из которых по сути представляет собой таблицу из одного поля.

Достоинства: возможность хранить большое количество данных с большим количеством атрибутов, скорость выполнения запросов.

Недостатки: медленная запись, не поддерживают транзакции.

Колоночные СУБД применяются как правило в аналитических системах.

CustomerID	Column Family: Identity
001	First name: Mu Bae Last name: Min
002	First name: Francisco Last name: Vila Nova Suffix: Jr.
003	First name: Lena Last name: Adamcyz Title: Dr.

CustomerID	Column Family: Contact Info
001	Phone number: 555-0100 Email: someone@example.com
002	Email: vilanova@contoso.com
003	Phone number: 555-0120

Семейство колонок

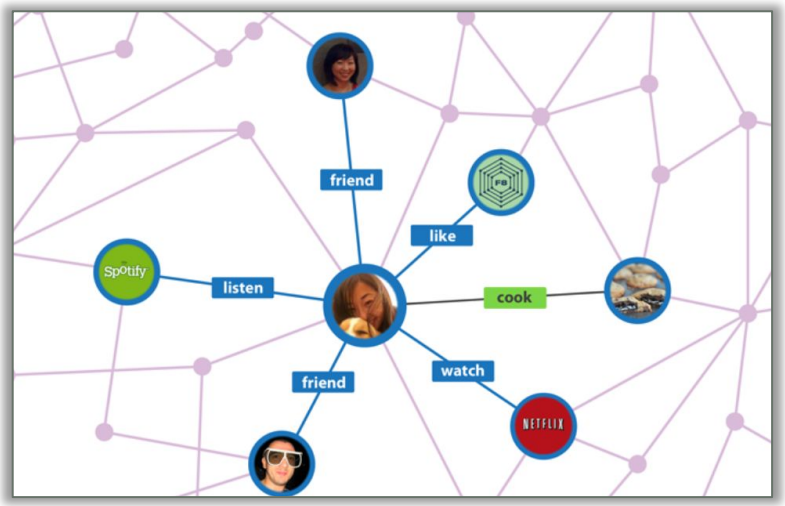
Реляционная таблица

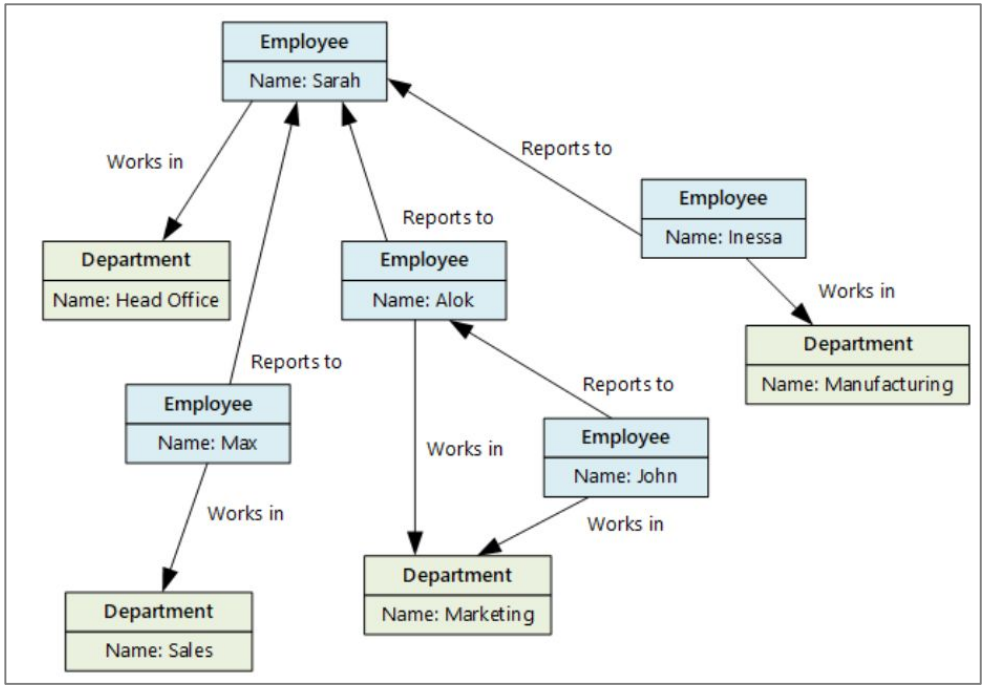
CustomerID	First name	Last name	Suffix	Title	Phone Number	Email
001	Mu Bae	Min			555-0100	someone@example.com
002	Francisco	Vila Nova	Jr.			
003	Lena	Adamcyz		Dr.	555-0120	vilanova@contoso.com

Графовая модель данных

Графовая модель данных основана на узлах и рёбрах, представляющих взаимосвязанные данные (например, отношения между людьми в социальной сети), они упрощают хранение и навигацию по сложным отношениям.

Возможно применение в задачах, ориентированных на связи: социальные сети, выявление мошенничества, маршруты общественного транспорта, дорожные карты и т.п.





Узлы: сотрудники и отделы.
Ребра: определяют отношения подчинения и отдел, в котором работает каждый сотрудник.
Стрелки: показывают направление связей.

Основные черты

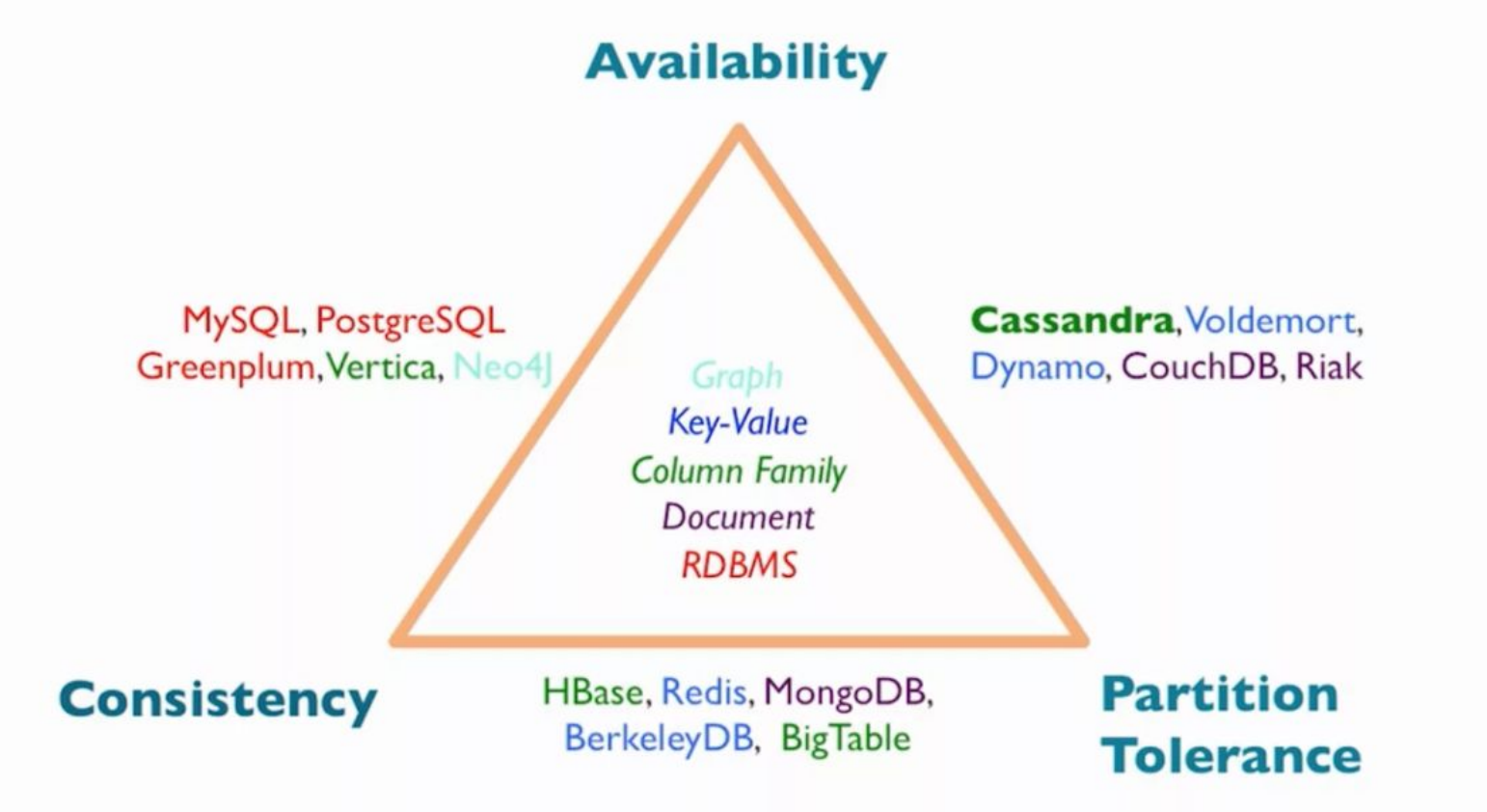
Традиционные реляционные СУБД основаны на принципах ACID:

- Atomicity - атомарность
- Consistency - согласованность
- Isolation – изолированность
- Durability - надежность

СУБД с подходом NoSQL основаны на принципах BASE:

- Basic Availability - базовая доступность — каждый запрос гарантированно завершается (успешно или безуспешно).
- Soft State - гибкое состояние — состояние системы может изменяться со временем, даже без ввода новых данных, для достижения согласования данных.
- Eventual Consistency - согласованность в конечном счете — данные могут быть некоторое время рассогласование, но приходят к согласованию через некоторое время.

Теорема CAP (или теорема Брюера)



Примеры NoSQL СУБД

1. Документно-ориентированные: CouchDB (Couchbase), MongoDB (MongoDB).
2. Колоночные: Cassandra (Apache Software Foundation), ClickHouse (Яндекс).
3. Графовые: Neo4j (Neo Technology), OrientDB (Orient Technologies LTD).
4. Ключ – значение: Tarantool (VK), Redis (Redis Labs), Oracle NOSQL Datebase (Oracle), Amazon DynamoDB (Amazon).

Источники информации

- К.Дж. Дейт. Введение в системы баз данных. Восьмое издание. – М.: Вильямс, 2005. – 1328 С.
- Голицина О.Л. Базы данных. - Изд. «ФОРУМ», 2009. - 400 с.
- Дунаев В.В. Базы данных. Язык SQL для студента. - СПб.: БХВ-Петербург, 2007. - 320 с.
- Советов Б.Я. Базы данных. – Изд. «Высшая школа», 2007. - 463 с.
- Харрингтон Д. Проектирование объектно-ориентированных баз данных. Год 2007. ”Лань” Электронная библиотечная система
- Техническая документация Microsoft: https://docs.microsoft.com/ru-ru/
- Интерактивные учебник по SQL: http://www.sql-tutorial.ru/ru/
- Национальная библиотека им. Н. Э. Баумана: https://ru.bmstu.wiki
- К.Ю. Поляков и т.д.