ИТМО ВТ

Базы данных.

1. Введение

1. СУБД

Современные информационные системы

Основные требования:

- Надёжность.
- Масштабируемость.
- Удобство разработки нового функционала (обычно разработкой занимается большая команда!).
- Удобство поддержки и сопровождения.

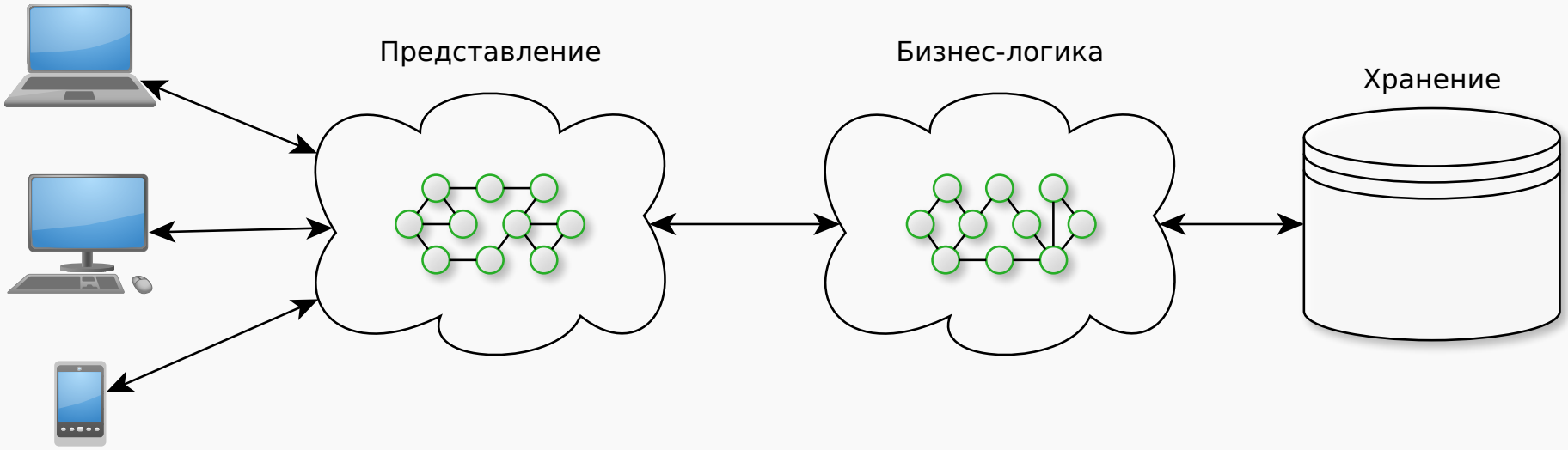
Следствия:

- ИС строится из отдельных «кубиков» (модулей, компонентов и т.д.).
- «Кубики» размещаются на нескольких уровнях.

ИТМО ВТ

Архитектура ИС

Типовая современная ИС:



ИТМО ВТ

Зачем нужны БД?

Почему нельзя «просто» использовать файлы?

ИТМО ВТ

Зачем нужны БД?

Просто хранить данные в файлах неудобно:

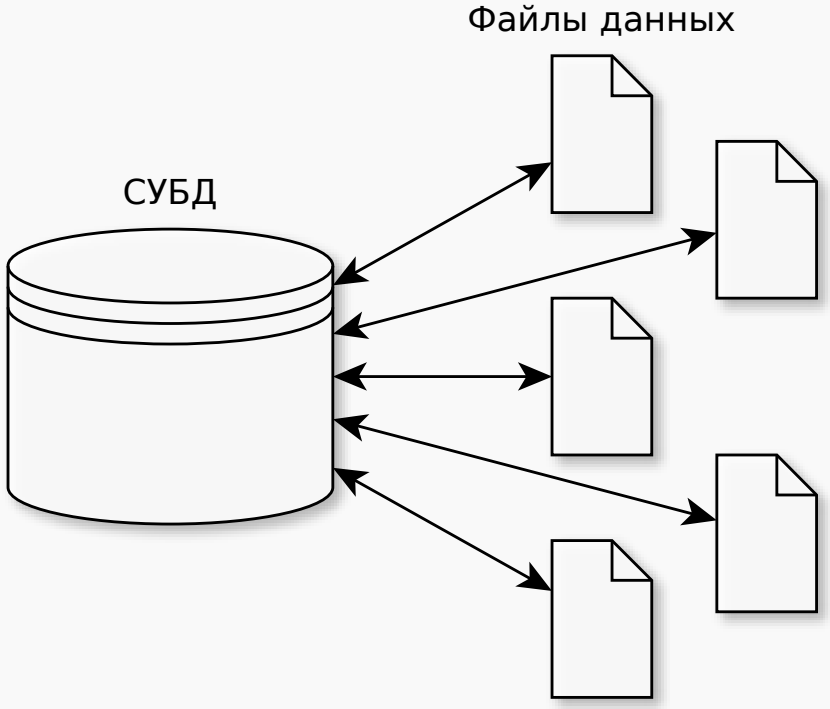
- Полное отсутствие гибкости (изменение структуры – изменение кода программы).
- Невозможность нормальной многопользовательской работы с данными.
- Вынужденная избыточность (проще создать еще одну копию данных, чем вносить изменения в десятки программ).

Решение — хранить данные и *метаданные* (данные о данных) вместе.

ИТМО ВТ

Формальное определение

База данных — это файлы, снабжённые описанием хранимых в них данных и находящиеся под управлением специальных программных комплексов, называемых "Системы управления базами данных" (СУБД).



ИТМО ВТ

Классификация СУБД

По степени распределённости:

- *локальные;*
- *распределённые.*

ИТМО ВТ

Классификация СУБД

По способу доступа к БД.

- *Файл-серверные* — данные находятся на файл-сервере, СУБД — на каждом клиентском компьютере. Примеры — M$ Access, dBase, FoxPro.
- *Клиент-серверные* — СУБД находятся на сервере вместе с данными. Примеры — Oracle, M$ SQL Server, Caché.
- *Встраиваемые* — СУБД встраивается в приложение, хранит только его данные и не требует отдельной установки. Примеры — SQLite, BerkeleyDB.

ИТМО ВТ

Классификация СУБД (продолжение)

По модели данных:

- *Иерархические* — данные представляются в виде дерева. Пример — LDAP / AD, реестр Windows.
- *Сетевые* — используют сетевую модель данных. Частный случай — графовые СУБД. Примеры — HypergraphDB, OrientDB.
- *Объектно-ориентированные* — используют ОО-модель данных. Пример — InterSystems Caché.
- *Реляционные и объектно-реляционные* — используют реляционную модель данных (возможно, с частичной поддержкой ООП). Примеры — Oracle, PostgreSQL.

2. Реляционные БД

ИТМО ВТ

Реляционные БД

Реляционная (relation — отношение, связь) или табличная модель данных была предложена в конце 60-х годов Эдгаром Коддом (IBM).

Реляционные БД (продолжение)

Ключевые особенности:

1) Данные хранятся в таблицах.
2) Каждая таблица состоит из строк, имеющих одну и ту же структуру, и имеет уникальное имя.

STUDENT

id	name	surname	gr_id
1	Григорий	Иванов	34
2	Григорий	Иванов	34
3	Иван	Сидоров	37

Реляционные БД (продолжение)

Ключевые особенности:

3) Строки имеют фиксированное число полей (столбцов) и значений (множественные поля и повторяющиеся группы недопустимы).

4) В каждой позиции таблицы на пересечении строки и столбца всегда имеется в точности одно значение или ничего.

STUDENT

id	name	surname	gr_id
1	Григорий	Иванов	34
2	Григорий	Иванов	34
3	Иван	Сидоров	37

Реляционные БД (продолжение)

5) Строки таблицы обязательно отличаются друг от друга хотя бы единственным значением.
6) Столбцам таблицы однозначно присваиваются имена.
7) В каждом из столбцов размещаются однородные значения данных (даты, фамилии, целые числа).

STUDENT

id	name	surname	gr_id
1	Григорий	Иванов	34
2	Григорий	Иванов	34
3	Иван	Сидоров	37

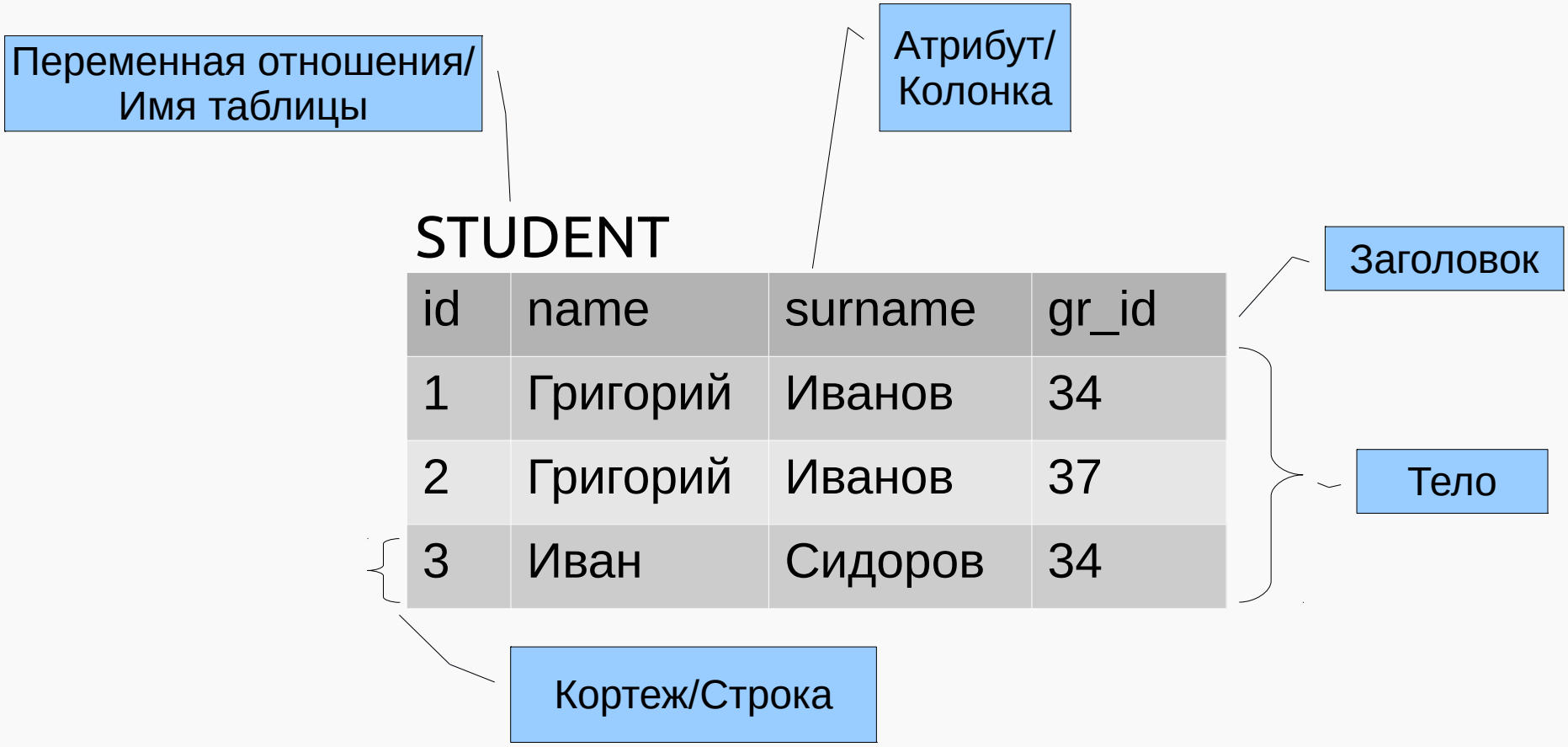
Реляционные БД (продолжение)

8) Содержание базы данных представляется в виде явных значений данных (не существует каких-либо специальных "связей" или указателей, соединяющих одну таблицу с другой).

9) При выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их содержанию.

10) Таблицы базы данных часто группируют в схемы.

Терминология



ИТМО ВТ

Ключи

- ***Ключ*** — минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности.
- ***Внешний ключ*** — множество атрибутов сущности для организации ссылок на экземпляры другой сущности (должен соответствовать *первичному* ключу в другой сущности)

ИТМО ВТ

Таблицы

STUDENT

id	name	surname	gr_id
1	Григорий	Иванов	34
2	Григорий	Иванов	33
3	Иван	Сидоров	37

GROUP

id	name	class_leader
34	P3100	**3**
37	P3112	**2**
354	R4230	**NULL**

ИТМО ВТ

Целостность данных

Целостность (от англ. integrity) — корректность данных в любой момент времени.

Выделяют три группы правил целостности.

- *Целостность по сущностям*.
- *Целостность по ссылкам*.
- *Целостность, определяемая пользователем.*

ИТМО ВТ

Целостность данных

- *Целостность по сущностям*. Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.

- *Целостность по ссылкам*. Значение внешнего ключа должно либо:
  - быть равным значению первичного ключа ассоциируемой сущности;
  - быть полностью неопределенным.

ИТМО ВТ

Целостность данных

- *Целостность, определяемая пользователем*:
  - уникальность тех или иных атрибутов;
  - диапазон значений (экзаменационная оценка от 2 до 5);
  - принадлежность набору значений (пол "М" или "Ж").

ИТМО ВТ

Сколько ошибок на слайде?

STUDENT

id	name	surname	gr_id
1	Григорий	Иванов	37
2	Иван	Сидоров	37
2	Петр	Петров	37
2	Иван	Сидоров	354

GROUP

id	name	class_leader
'Vasya'	P3100	1
37	P3112	2
354	R4230	4

ИТМО ВТ

Ответ

STUDENT

id	name	surname	gr_id
1	Григорий	Иванов	37
2	Иван	Сидоров	37
3	Петр	Петров	37
4	Иван	Сидоров	354

GROUP

id	name	class_leader
'Vasya'	P3100	1
37	P3112	2
354	R4230	4

3. Язык SQL

ИТМО ВТ

О языке

- *SQL (структурированный язык запросов)* — декларативный язык программирования.
- Применяется для получения, создания, модификации и управления данными в реляционной базе данных.
- Разработан в 1970-е гг. в компании IBM.

ИТМО ВТ

О языке

- Язык используется в большинстве реализаций реляционных БД.
- Стандартизирован:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL-99
  - SQL-2003
  - SQL-2006
  - SQL-2008
  - ...

ИТМО ВТ

Особенности языка

Каждое предложение SQL — это либо запрос данных из базы, либо обращение к базе данных, которое приводит к изменению данных в базе.

Различают следующие **типы запросов**:

- запросы на создание или изменение в базе данных новых или существующих объектов;
- запросы на получение данных;
- запросы на добавление новых данных (записей);
- запросы на удаление данных;
- обращения к СУБД.

ИТМО ВТ

Операторы SQL

Операторы определения данных *(Data Definition Language, DDL)*:

- CREATE создает объект БД (саму базу, таблицу, представление, пользователя и т. д.);
- ALTER изменяет объект;
- DROP удаляет объект.

ИТМО ВТ

Операторы SQL (продолжение)

Операторы манипуляции данными *(Data Manipulation Language, DML)*:

- SELECT считывает данные, удовлетворяющие заданным условиям;
- INSERT добавляет новые данные;
- UPDATE изменяет существующие данные;
- DELETE удаляет данные.

ИТМО ВТ

Операторы SQL (продолжение)

Операторы определения доступа к данным *(Data Control Language, DCL)*:

- GRANT предоставляет пользователю (роли) разрешения на определенные операции с объектом;
- REVOKE отзывает ранее выданные разрешения;

ИТМО ВТ

Оператор SELECT

Возвращает набор данных из БД, удовлетворяющих заданному условию.

Формат запроса:

SELECT список полей FROM список таблиц

WHERE условия…

ИТМО ВТ

Оператор SELECT

Основные ключевые слова:

- WHERE — используется для определения, какие строки должны быть выбраны или включены в GROUP BY.
- GROUP BY — используется для объединения строк с общими значениями в элементы меньшего набора строк.
- HAVING — используется для определения, какие строки после GROUP BY должны быть выбраны.
- ORDER BY — используется для определения, какие столбцы используются для сортировки результирующего набора данных.

ИТМО ВТ

Агрегатные функции

- COUNT(*)
- COUNT(A)
- SUM(A)
- AVG(A)
- MIN(A)
- MAX(A)

A — некоторый атрибут

ИТМО ВТ

Параметр GROUP BY

- Позволяет группировать строки и использовать агрегатные функций (MAX, SUM, AVG, …).
- Необходимо, чтобы в SELECT были заданы только требуемые в выходном потоке столбцы, перечисленные в GROUP BY и/или агрегированные значения.
- Пример использования:

  Запрос возвращает список партнеров с общей суммой продажи:

  SELECT Partner, SUM(SaleAmount) FROM Sales GROUP BY Partner;

ИТМО ВТ

Параметр HAVING

- Позволяет указывать условия на результат агрегатных функций (MAX, SUM, AVG, …).
- Аналогичен WHERE за исключением того, что строки отбираются не по значениям столбцов, а строятся из значений столбцов, указанных в GROUP BY, и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY.
- Необходимо, чтобы в SELECT были заданы только требуемые в выходном потоке столбцы, перечисленные в GROUP BY и/или агрегированные значения.
- Если параметр GROUP BY в SELECT не задан, HAVING применяется к «группе» всех строк таблицы, полностью дублируя WHERE.

ИТМО ВТ

Параметр HAVING

- Пример использования:

  возвращает список идентификаторов отделов, продажи которых превысили 1000 долларов вместе с суммами продаж:

  ```
  SELECT DeptID, SUM(SaleAmount) FROM Sales
  GROUP BY DeptID
  HAVING SUM(SaleAmount) > 1000;
  ```

ИТМО ВТ

Транзакции

Транзакция — группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными.

Операторы управления транзакциями *(Transaction Control Language, TCL)*:

- COMMIT применяет транзакцию;
- ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции;
- SAVEPOINT делит транзакцию на более мелкие участки.

ИТМО ВТ

Лекция 2:
создание Базы Данных

1. Построение БД

ИТМО ВТ

С чего начать построение БД?

Предметная область - часть реального мира, данные о которой необходимо разместить в базе данных (учебный процесс в университете, магазин и т.п.)

ИТМО ВТ

Пользовательский уровень

Обычный пользователь системы не знает о:

- фактическом размещении данных в памяти компьютера;
- механизмах для оптимизации операций с данными;
- обработке запроса и осуществлении поиска;

ИТМО ВТ

Пользовательский уровень

Как представляет предметную область (часть предметной области) пользователь?

- Пользователь обычно:
  - взаимодействует с частью системы;
  - имеет неполное представление о системе в целом;
- Сбор информации в виде: текста, диаграмм и т.д;

ИТМО ВТ

Инфологический уровень

- Инфологическая модель:
  - обобщенное представление предметной области;
  - собирается на основе анализа пользовательских представлений;
  - не зависит от «физического» хранилища;
  - есть стандартные средства для описания (например, ER-диаграмма);

ИТМО ВТ

Даталогическая модель

- Представление инфологической модели с учетом:
  - используемой модели данных;
  - могут появляться детали, относящиеся к СУБД (типы данных);

ИТМО ВТ

Физический уровень

- Реализация даталогической модели средствами СУБД:
  - зависит от особенностей конкретной СУБД.
  - описывается на языке, поддерживаемом СУБД.

- Пример: SQL-код описания таблиц:

  CREATE TABLE STUDENT (…)

ИТМО ВТ

Проектирование базы данных

1. Пользовательское представление о предметной области.
2. Инфологическая модель.
3. Физическая реализация.

Проектирование «сверху-вниз» (top-down approach)

2. Инфологическая модель

ИТМО ВТ

Основные элементы ИМ

- *Сущность* — класс объектов, фактов, явлений, предметов, элементы которых будут храниться в базе данных.
- *Экземпляр сущности* относится к конкретной вещи в наборе. Например, типом сущности (сущностью) может быть СТУДЕНТ , а экземпляром — Иван Иванов и т. д.
- *Атрибут* — важная характеристика (свойство) сущности, которой присваивается имя.
- *Связь* — ассоциирование двух или более сущностей, выражающая форму взаимодействия между ними.

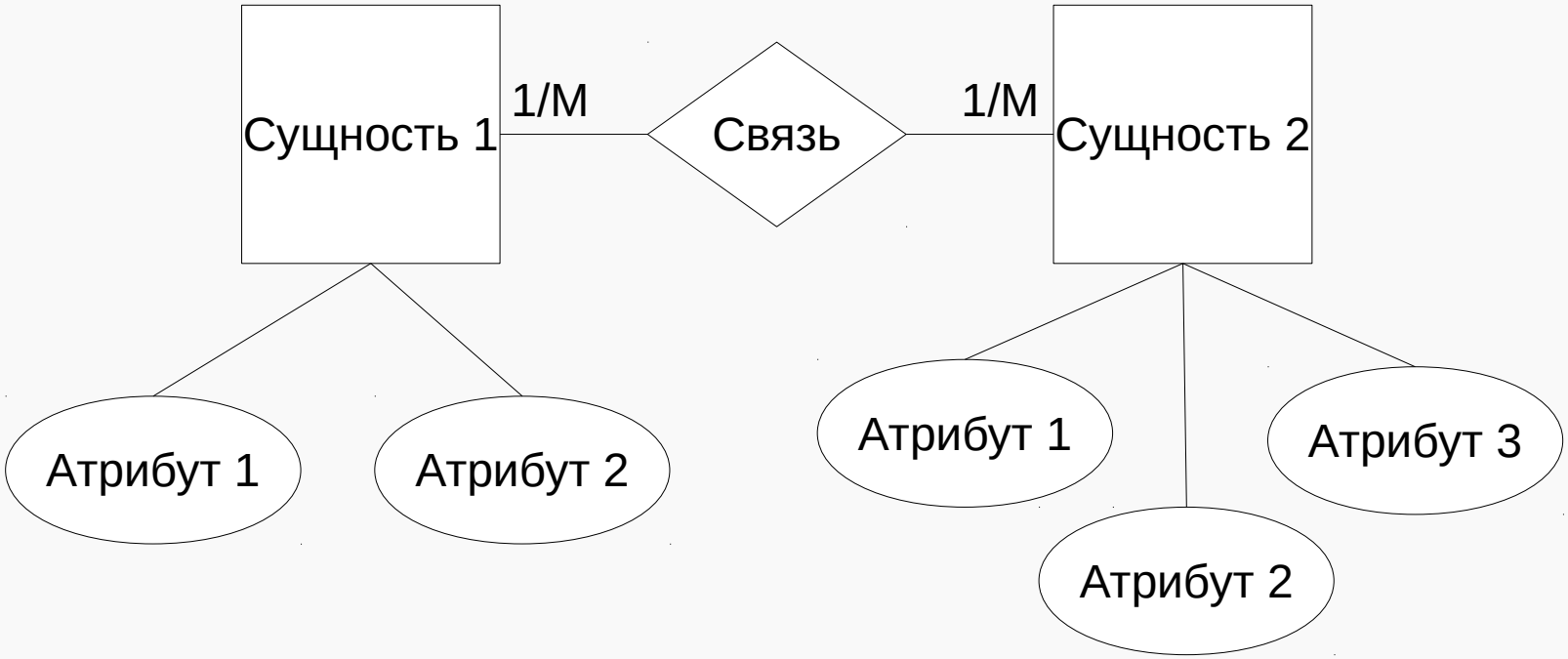
ИТМО ВТ

Модель «сущность-связь»

- Один из вариантов для построения инфологической модели: ER-диаграммы (Entity-Relationship - сущность-связь).
- Питер Чен (1976, IBM).

ИТМО ВТ

Изображение основных элементов



Над связями - степень связи (1 или M - "много")

ИТМО ВТ

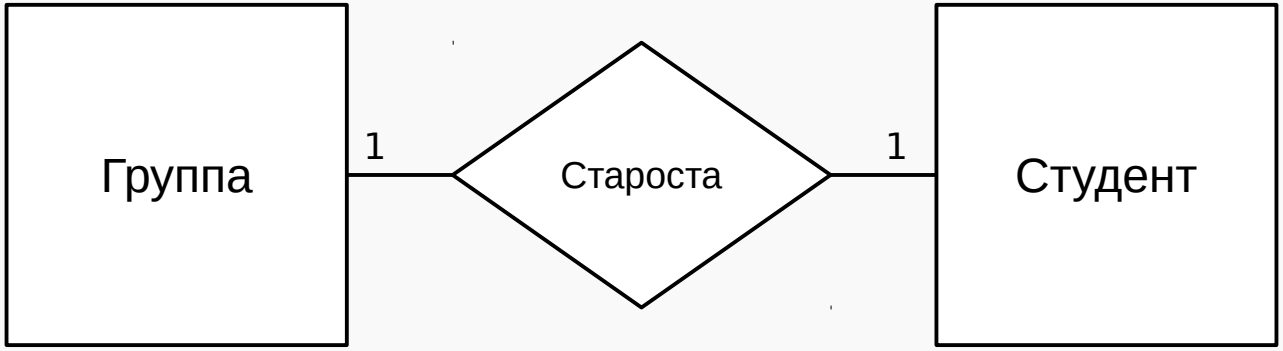
Связи между сущностями

4 (3) вида связей:

- "один-к-одному" (1:1)
- "один-ко-многим" (1:M)
- "многие-к-одному" (M:1)
- "многие-ко-многим" (M:M)

Один-к-одному (1:1)

В каждый момент времени каждому экземпляру первой сущности соответствует 1 или 0 экземпляров второй сущности:



ИТМО ВТ

Один-ко-многим (1:M)

Одному экземпляру первой сущности соответствуют 0, 1 или несколько экземпляров второй сущности;

Одному экземпляру второй сущности соответствует 0 или 1 экземпляров первой сущности;

ИТМО ВТ

Многие-к-одному (M:1)

Обратная связь к 1:M (рассматриваем со стороны второй сущности).

Связь сущностей Студент-Группа - M:1

ИТМО ВТ

Многие-ко-многим (M:M)

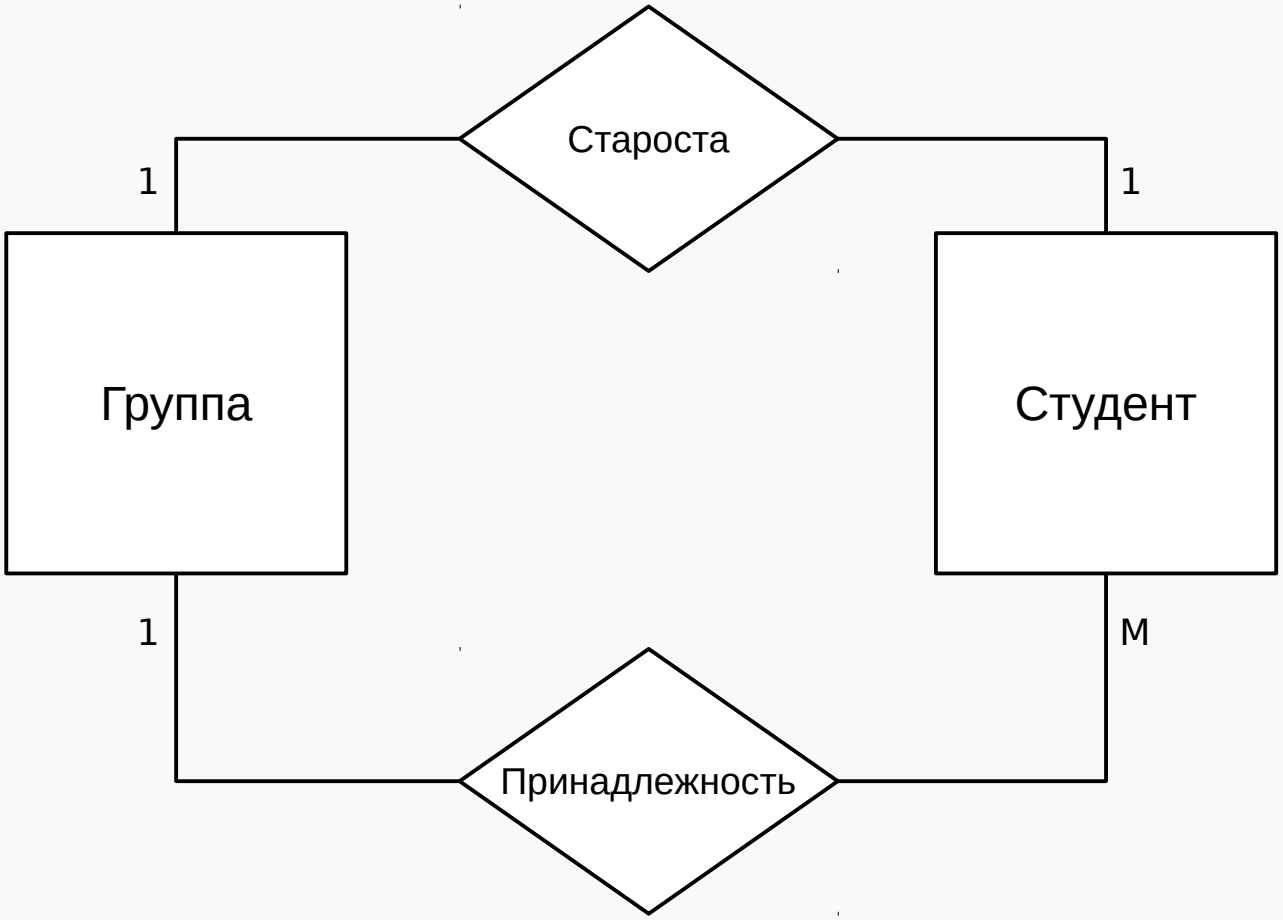
Одному экземпляру первой сущности соответствуют 0, 1 или несколько экземпляров второй сущности;

Одному экземпляру второй сущности соответствует 0, 1 или несколько экземпляров первой сущности;

ИТМО ВТ

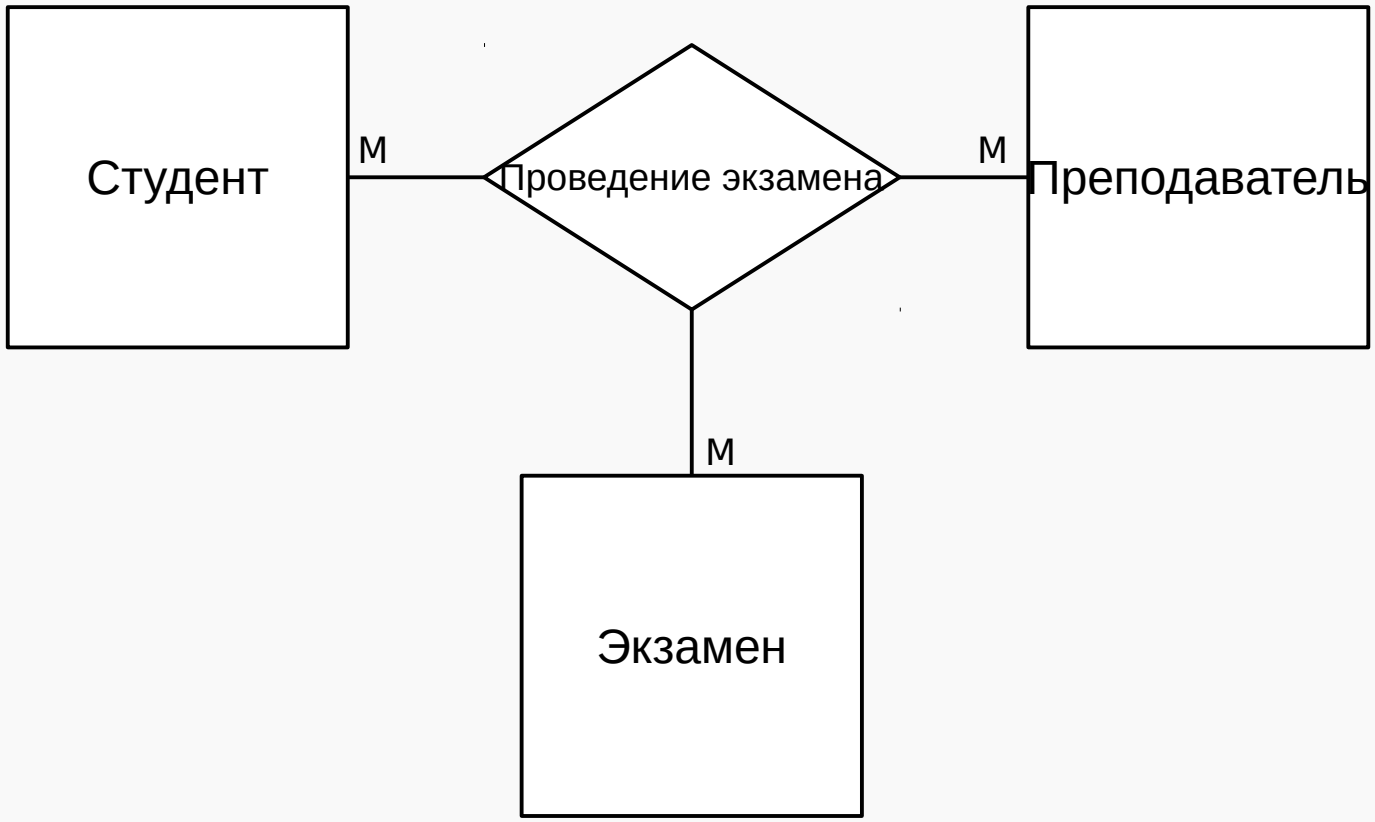
Сложные связи

Несколько связей между двумя сущностями;



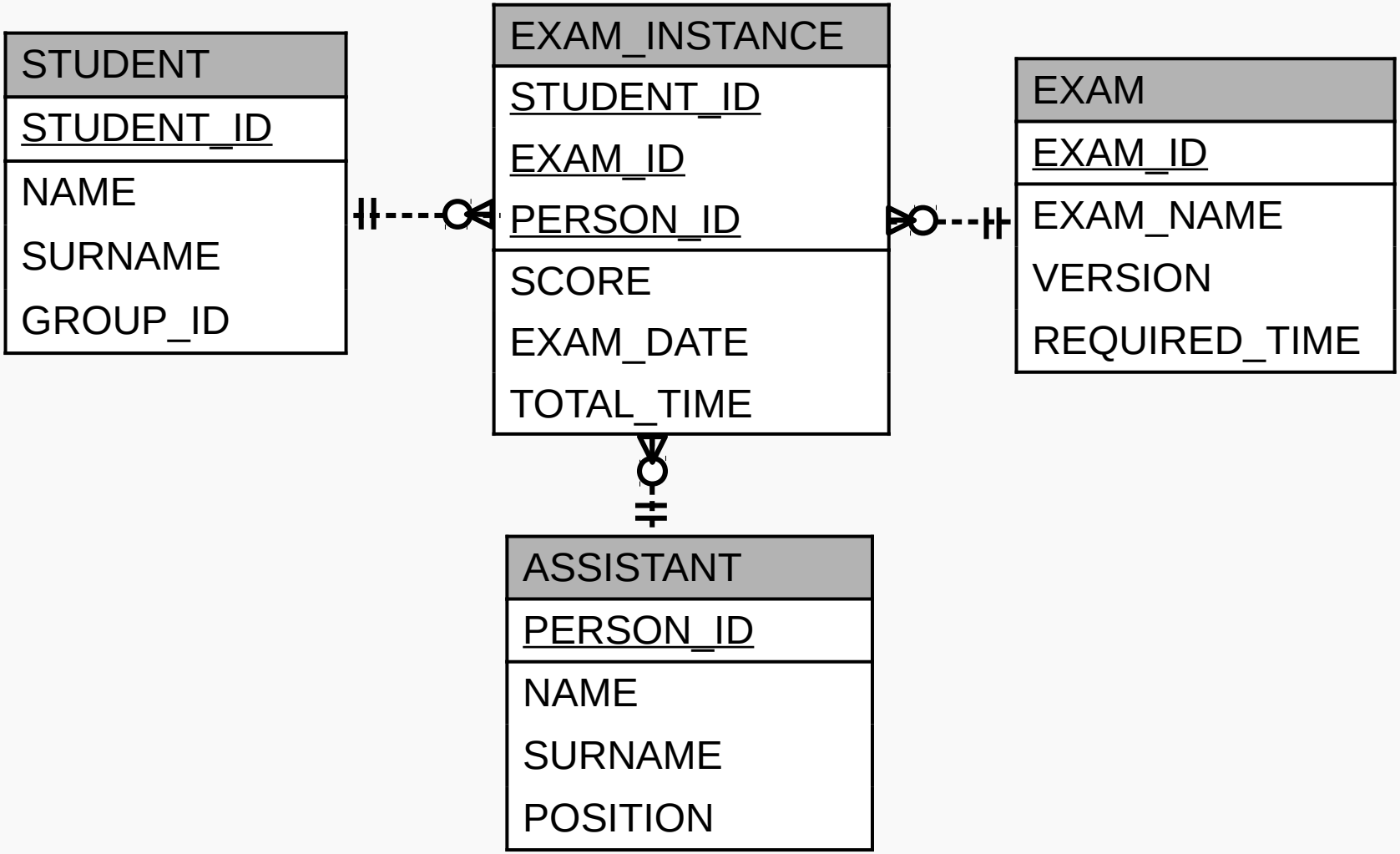
ИТМО ВТ

Тернарные связи



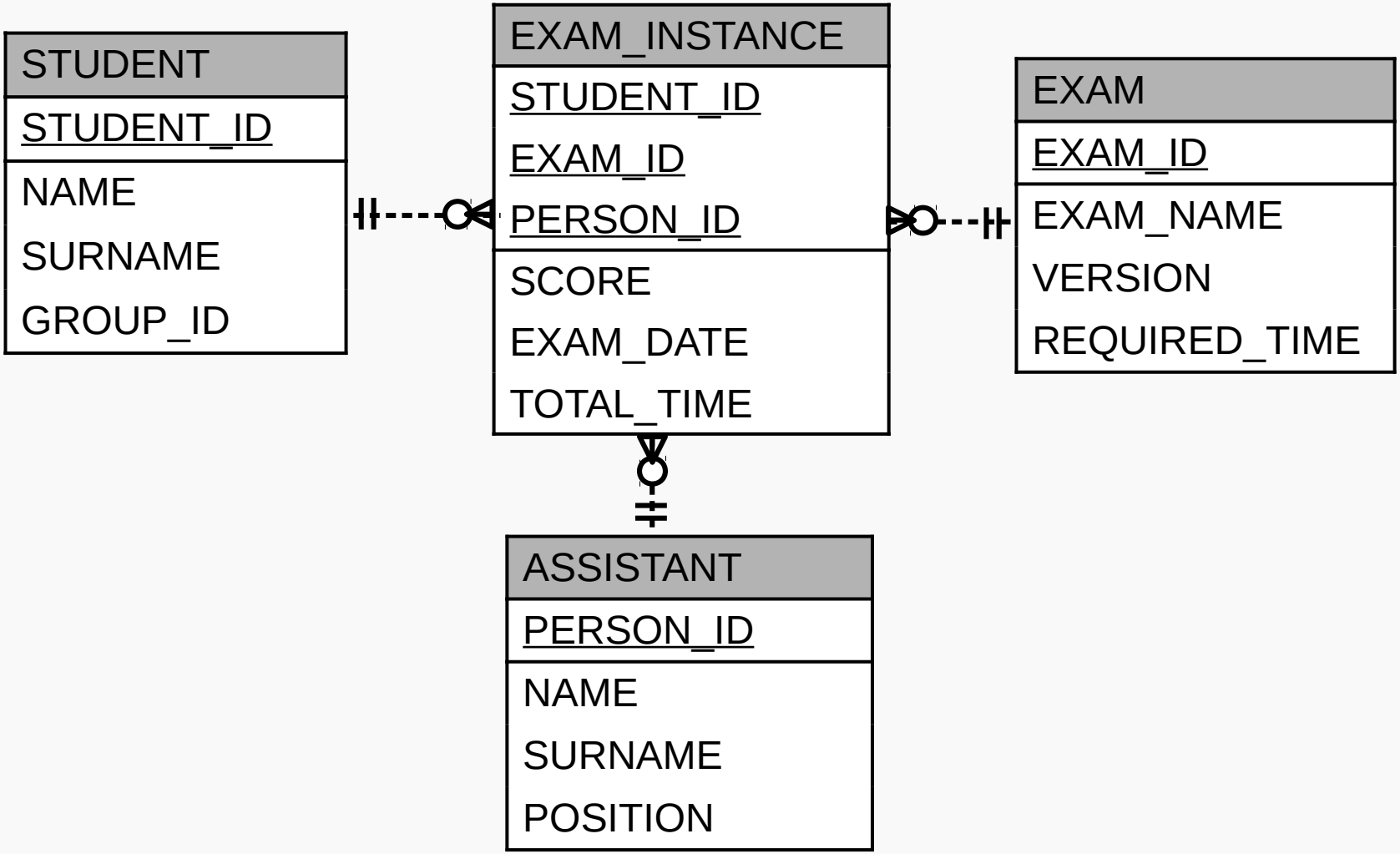
ИТМО ВТ

Дополненное представление



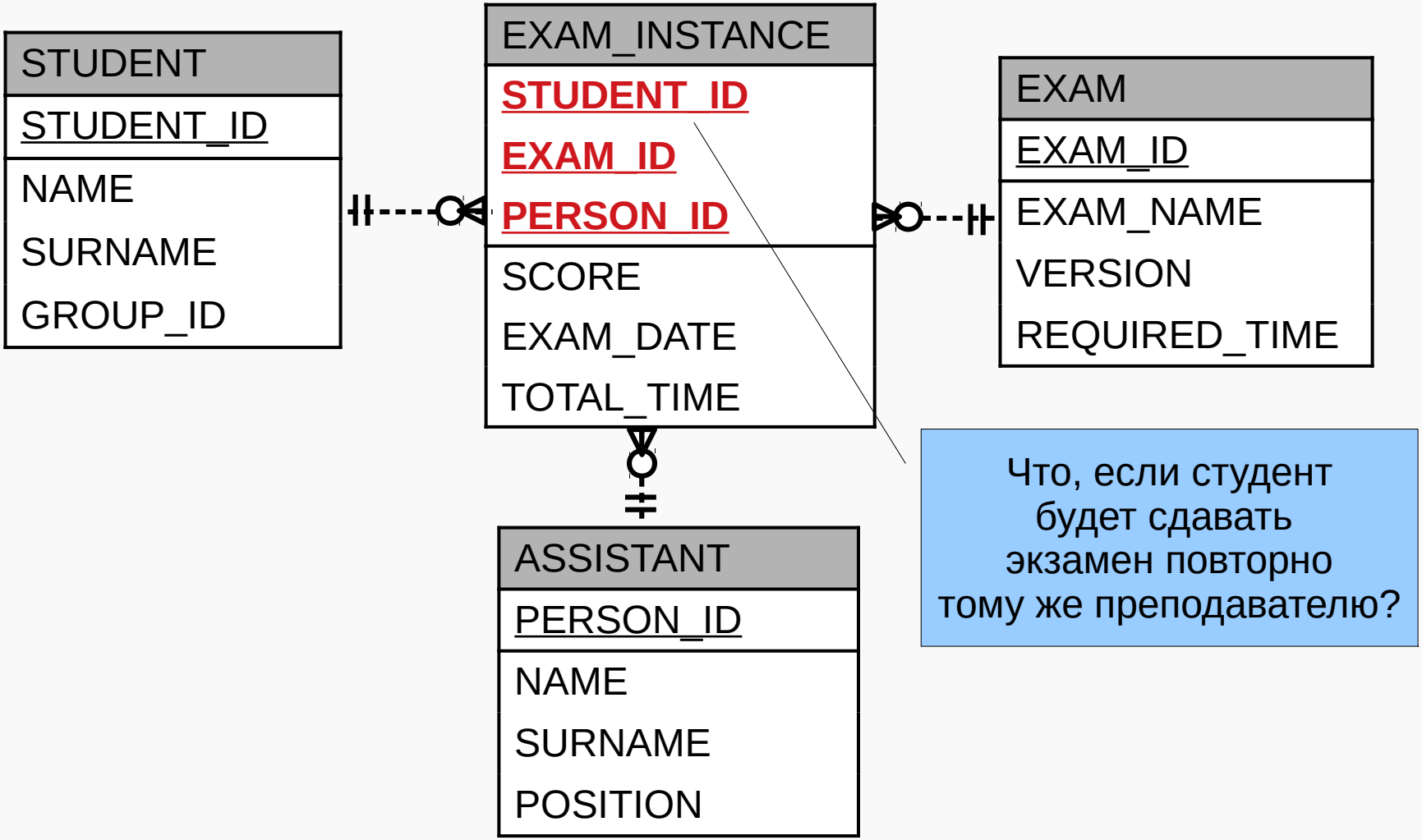
ИТМО ВТ

Какие недостатки в модели?



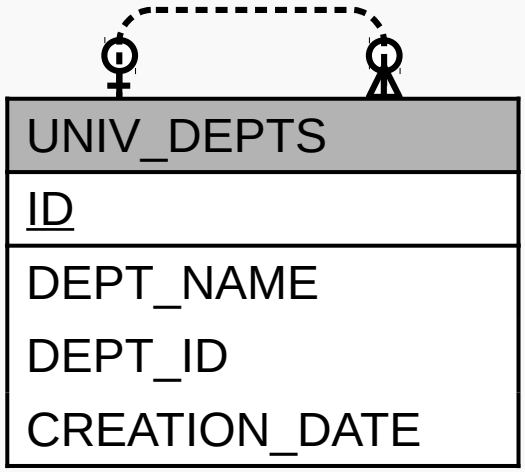
ИТМО ВТ

Какие недостатки в модели?



ИТМО ВТ

Рекурсивная связь



ИТМО ВТ

Классификация сущностей

3 класса сущностей:

- стержневые
- ассоциативные
- характеристические

ИТМО ВТ

Классификация сущностей

- **Стержневая сущность** (стержень) — независимая, базовая сущность (Студент, Группа)
- **Ассоциативная сущность** (ассоциация) — связь вида "многие-ко-многим" ("*-ко-многим" и т. д.) между двумя или более сущностями
- **Характеристическая сущность** (характеристика) — связь вида "многие-к-одной" или "одна-к-одной" между двумя сущностями (частный случай ассоциации). Цель характеристики - описание или уточнении некоторой другой сущности.

ИТМО ВТ

Ключ

- ***Ключ*** — минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности.
- ***Суррогатный ключ*** - автоматически сгенерированный атрибут для однозначной идентификации экземпляра сущности в рамках данной сущности.

3. Создание реляционной БД

ИТМО ВТ

Объекты базы данных

- **Объекты** базы данных - таблицы, представления, процедуры, триггеры.
- Для работы с объектами БД используется **SQL**:
  - Предложения манипуляции данными (Data Manipulation Language, **DML**)
  - Предложения определения данных (Data Definition Language, **DDL**)
  - Предложения определения доступа к данным (Data Control Language, **DCL**)

ИТМО ВТ

Составляющие языка SQL

- Предложения;
- Идентификаторы (имена);
- Константы;
- Операторы;
- Зарезервированные и ключевые слова;

ИТМО ВТ

Предложения SQL

Предложение — команда, которая состоит из ключевых и зарезервированных слов, определяемых пользователем в соответствии с синтаксическими правилами:

```
SELECT * FROM STUDENTS;
```

ИТМО ВТ

Идентификаторы

- Имена объектов баз данных (определенные пользователем или системные).
- Создается при определении объекта.
- Используется для обращения к объекту:

```
SELECT * FROM STUDENTS;
```

ИТМО ВТ

Константы

Любые значения, которые не являются идентификаторами или ключевыми словами:

- числовые значения: 9999, 5E6
- строки символов: 'Пример строки'
- значения, связанные с представлением времени (дата и время): 06-03-2017 10:06:54
- булевы значения: TRUE

ИТМО ВТ

Операторы

- Служат для обозначения действия над одним или несколькими выражениями.

- Делятся на категории: арифметические, логические, присваивания, сравнения и тд.

- SELECT * FROM STUDENTS WHERE AGE > 19;

Зарезервированные и ключевые слова

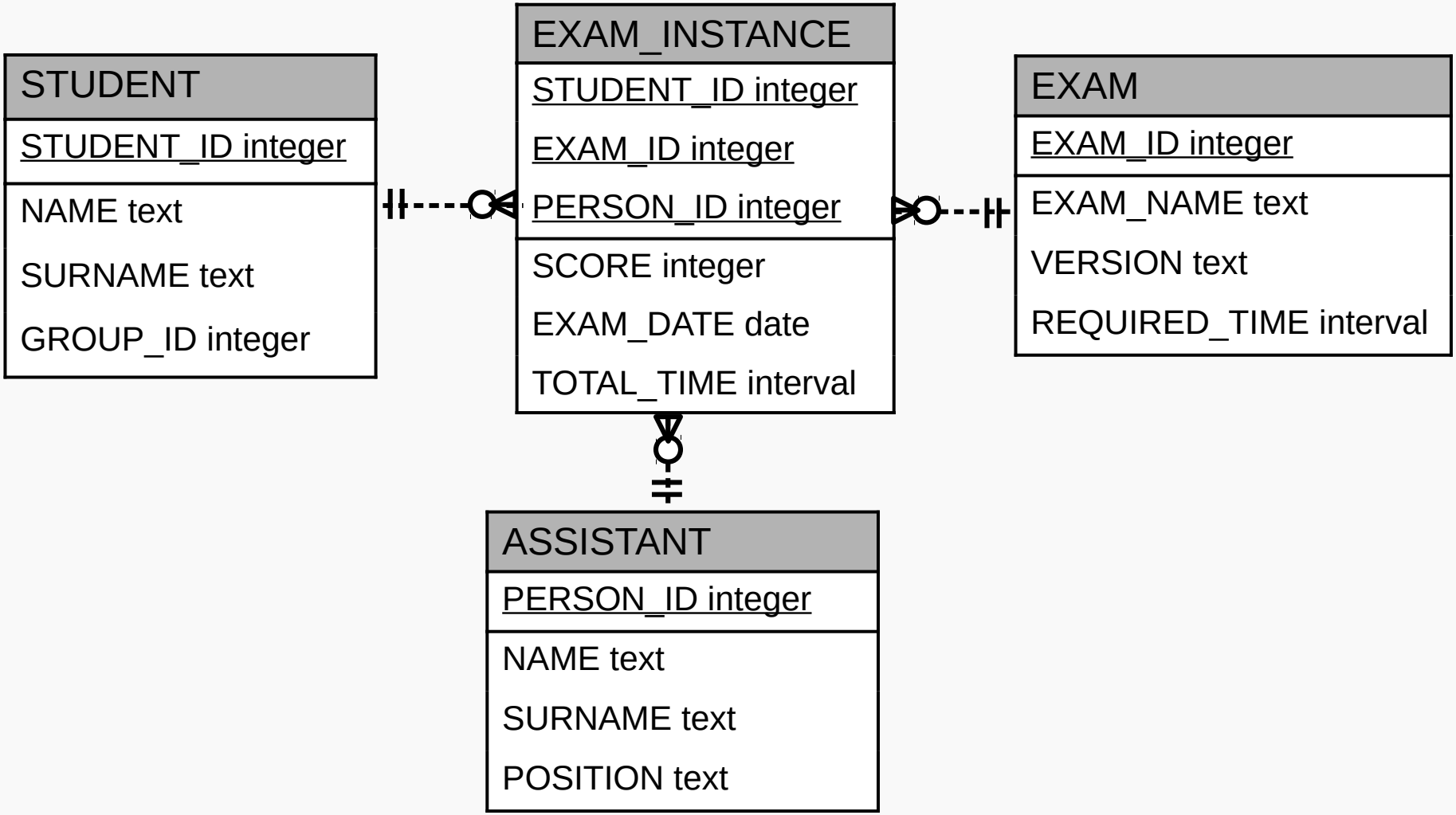
Слова и фразы для задания конструкций и использования возможностей языка SQL;

```
SELECT * FROM STUDENTS WHERE AGE > 19;
```

4. Работа с таблицами

ИТМО ВТ

Даталогическая модель



ИТМО ВТ

DDL

Операторы определения данных *(Data Definition Language, DDL)*:

- CREATE создает объект БД (саму базу, таблицу, представление, пользователя и т. д.);
- ALTER изменяет объект;
- DROP удаляет объект.

ИТМО ВТ

Создание таблиц

Таблицы:

- *Базовые* — в действительности существующие, хранящиеся в физической памяти машины.

- *Виртуальные* — представления, курсоры, неименованные таблицы (таблицы, которые не существуют постоянно в базе данных).

Создание базовой таблицы

Используется предложение CREATE TABLE:

```
CREATE TABLE STUDENTS (
    STUD_ID integer,
    STUD_NAME text,
    BIRTH_DATE date
);
```

Удаление базовой таблицы

Используется предложение DROP TABLE:

DROP TABLE *STUDENTS;*

Изменение таблиц

Используется предложение ALTER:

ALTER TABLE STUDENTS **ADD COLUMN** STUD_GROUP text;

ALTER TABLE STUDENTS **DROP COLUMN** BIRTH_DATE;

ИТМО ВТ

Типы данных

Каждая колонка имеет свой **тип данных:**

- Ограничивает список возможных значений, которые могут находится в этой колонке
- Определяет «смысл» данных, хранящихся в колонке, чтобы данные могли использоваться при вычислениях.

ИТМО ВТ

Типы данных

Тип данных задается при создании:

```
CREATE TABLE STUDENTS (
    STUD_ID integer,
    STUD_NAME text,
    BIRTH_DATE date
);
```

ИТМО ВТ

Числовые типы

PostgreSQL:

- smallint — целые числа (2 байта);
- integer — целые числа (4 байта);
- bigint — целые числа (8 байт);

ИТМО ВТ

Символьные типы

PostgreSQL:

- varchar(n) — переменной длины с ограничением;
- char(n) — фиксированной длины, остаток заполняется пробелами;
- text — переменной длины без задаваемого ограничения;

где n — положительное число.

```
CREATE TABLE STUDENTS (ST_NAME char(25));
INSERT INTO STUDENTS VALUES ('Valery');
```

ИТМО ВТ

Логический тип

Тип boolean:

- состояния: «true», «false» (третье состояние, «unknown» представляется через SQL-значение NULL);
- «true» может задаваться следующими значениями: **TRUE**, 'true', 't', 'yes', 'y', 'on', '1';
- «false» может задаваться следующими значениями: **FALSE**, 'false', 'f', 'no', 'n', 'off', '0';

ИТМО ВТ

Типы даты/времени

PostgreSQL:

- time — время суток;
- timestamp — дата и время;
- date — дата;
- interval — временной интервал;

TIMESTAMP '2016-11-23 11:13:44+01'

ИТМО ВТ

NULL-значения

- NULL — специальное значение (пустое, несуществующее значение), которое может быть записано в поле таблицы базы данных.

- NULL-значения нужно рассматривать как "отсутствие информации" (**не как** пустые строки, пробелы и тд).

ИТМО ВТ

NULL-значения

- Значения типа NULL **не равны** друг другу
- Столбец, содержащий значение NULL , **игнорируется** при вычислениях агрегатных значений
- В предложениях с DISTINCT, ORDER BY, GROUP BY, значения NULL **не отличаются** друг от друга.

ИТМО ВТ

Значения по умолчанию

```
CREATE TABLE GROUPS (
    GR_ID integer,
    GR_NAME text,
    GR_COUNT integer DEFAULT 0
);
```

Значение по умолчанию может быть **выражением**

ИТМО ВТ

Ограничения целостности

- Типы данных — один из способов ограничивать данные, но его не всегда **достаточно**.

- SQL позволяет определять ограничения для колонок и таблиц.

ИТМО ВТ

CHECK

```
CREATE TABLE STUDENTS (
    ST_ID integer,
    ST_NAME text,
    FAILED_COURSES integer
                        CHECK (FAILED_COURSES >= 0)
);
```

ИТМО ВТ

CHECK

- Позволяет задать для определённой колонки, выражение, которое будет осуществлять проверку, помещаемого в эту колонку значения.
- Выражение должно возвращать логическое значение
- Проверка CHECK проходит, если выражение, указанное в ограничении возвращает значение истина или null

ИТМО ВТ

CHECK

Ограничению можно задать имя (чтобы на него ссылаться):

```
CREATE TABLE STUDENTS (
    ST_ID integer,
    ST_NAME text,
    FAILED_COURSES integer CONSTRAINT fcrs
                           CHECK (FAILED_COURSES >= 0)
);
```

CHECK

Ограничение можно накладывать на таблицу:

```
CREATE TABLE STUDENTS (
    ST_ID integer,
    FAILED_MAX integer,
    FAILED_COURSES integer,
    CONSTRAINT fcrs
        CHECK
            (FAILED_COURSES >= 0 AND
             FAILED_COURSE <= FAILED_MAX)
);
```

ИТМО ВТ

NOT NULL

Колонка не должна содержать значение **null**:

```
CREATE TABLE STUDENTS (
    ST_ID integer NOT NULL,
    FAILED_MAX numeric NOT NULL,
    FAILED_COURSES integer NOT NULL
);
```

Логически эквивалентно созданию ограничения:

```
CHECK (column_name IS NOT NULL)
```

Ограничение уникальности

Данные в колонке или группе колонок являются уникальными по отношению к данным из той же колонки/ группы колонок для других строк в той же таблице:

```
CREATE TABLE STUDENTS (
    ST_ID integer NOT NULL UNIQUE,
    ST_NAME text NOT NULL,
    ST_SURNAME text NOT NULL,
    ST_BIRTH date NOT NULL
);
```

ИТМО ВТ

Ограничение уникальности

```
CREATE TABLE STUDENTS (
    ST_ID integer UNIQUE NOT NULL,
    ST_NAME text NOT NULL,
    ST_SURNAME text NOT NULL,
    ST_BIRTH date NOT NULL,
    UNIQUE (ST_NAME, ST_SURNAME, ST_BIRTH)
);
```

ИТМО ВТ

Первичный ключ

```
CREATE TABLE STUDENTS (
    ST_ID integer PRIMARY KEY,
    ST_NAME text NOT NULL,
    ST_SURNAME text NOT NULL,
    ST_BIRTH date NOT NULL,
    UNIQUE (ST_NAME, ST_SURNAME, ST_BIRTH)
);
```

ИТМО ВТ

Первичный ключ

- Первичный ключ означает, что колонка (группа колонок) используются как уникальный идентификатор строки в таблице
- Ограничение первичного ключа эквивалентно комбинации ограничений уникальности и не-null
- Таблица может иметь не более одного первичного ключа.

ИТМО ВТ

Первичный ключ (PostgreSQL)

- Добавление первичного ключа автоматически создает уникальный b-tree индекс на колонку.

- PostgreSQL позволяет не создавать первичный ключ для таблицы (согласно теории **требуется** наличие первичного ключа для каждой таблицы).

ИТМО ВТ

Внешний ключ

- Позволяет отобразить связи между различными сущностями.
- Поддержка целостности на уровне связей.

ИТМО ВТ

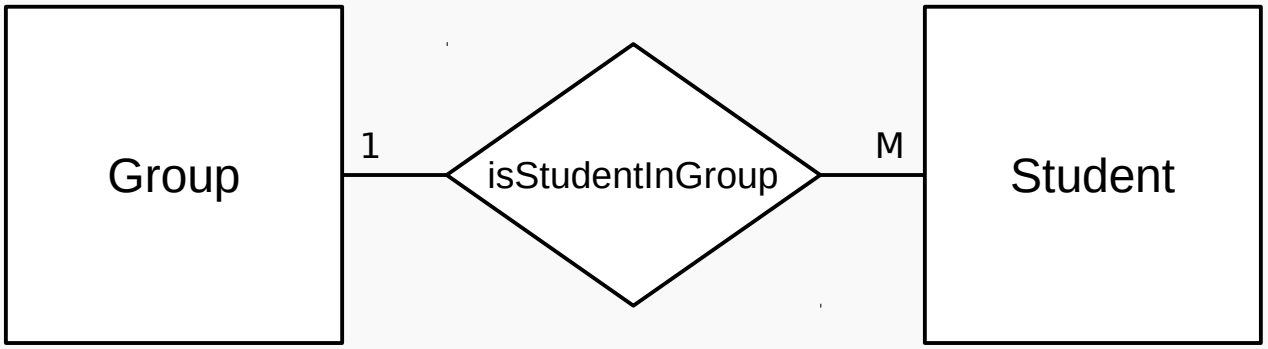
Внешний ключ

Внешний ключ — множество атрибутов сущности для организации связей с экземплярами другой сущности (должен соответствовать *первичному* ключу в первой сущности)

- Если сущность связывает две другие сущности, то она должна включать внешние ключи, соответствующие первичным ключам связываемых сущностей.
- Если одна сущность характеризует другую сущность, то она должна включать внешний ключ, соответствующий первичному ключу первой сущности.

ИТМО ВТ

Внешний ключ




```
CREATE TABLE GROUP (
    GR_ID integer PRIMARY KEY,
    GR_NAME text
);
CREATE TABLE STUDENT (
    ST_ID integer PRIMARY KEY,
    ST_NAME text,
    GR_ID integer REFERENCES GROUP (GR_ID)
);
```

Внешний ключ

```
CREATE TABLE GROUP (
    GR_ID integer PRIMARY KEY,
    GR_NAME text
);
```

GR_ID	GR_NAME
1	3119

```
CREATE TABLE STUDENT (
    ST_ID integer PRIMARY KEY,
    ST_NAME text,
    GR_ID integer REFERENCES GROUP (GR_ID)
);
```

ST_ID	ST_NAME	GR_ID
42	Ivan Ivanov	1

Внешний ключ

```
CREATE TABLE GROUP (
    GR_ID integer PRIMARY KEY,
    GR_NAME text
);
```

GR_ID	GR_NAME
1	3119

```
CREATE TABLE STUDENT (
    ST_ID integer PRIMARY KEY,
    ST_NAME text,
    GR_ID integer REFERENCES GROUP
);
```

ST_ID	ST_NAME	GR_ID
42	Ivan Ivanov	1

*(сокр. Запись) — будет использовано значение первичного ключа из таблицы GROUP

Внешний ключ (?)

```
CREATE TABLE GROUP (
    GR_ID integer PRIMARY KEY,
    GR_NAME text
);
```

GR_ID	GR_NAME
1	3119

```
CREATE TABLE STUDENT (
    ST_ID integer PRIMARY KEY,
    ST_NAME text,
    GR_ID integer REFERENCES GROUP
);
```

ST_ID	ST_NAME	GR_ID
42	Ivan Ivanov	1

Что произойдет при удалении группы?

```
DELETE FROM GROUP WHERE GR_ID = 1;
```

ИТМО ВТ

Варианты

1. Ничего не делать.
2. Удалить каскадно.
3. Установить в null.
4. Установить значение по умолчанию.

Внешний ключ

```
CREATE TABLE GROUP (
    GR_ID integer PRIMARY KEY,
    GR_NAME text
);
```

GR_ID	GR_NAME
1	3119

```
CREATE TABLE STUDENT (
    ST_ID integer PRIMARY KEY,
    ST_NAME text,
    GR_ID integer REFERENCES GROUP ON DELETE CASCADE
);
```

ST_ID	ST_NAME	GR_ID
42	Ivan Ivanov	1

*другие варианты: ON DELETE RESTRICT,
ON DELETE SET DEFAULT, ON DELETE SET NULL
**аналогично ON UPDATE ...

5. Отображение ER-диаграмм в БД

ИТМО ВТ

Отображение

- *Сущность* — таблица
- *Экземпляр сущности* — строка таблицы
- *Атрибут* — столбец в таблице
- *Связь* — внешний ключ, таблица

ИТМО ВТ

Связь «один-к-*»

- Реализуется путем добавления в таблицу внешнего ключа.
- Внешний ключ обычно добавляется в сущность-ассоциацию или в сущность-характеристику.
- Для моделирования характера связи на внешний ключ вводятся доп. ограничения (например, «один-к-одному» - UNIQUE).

ИТМО ВТ

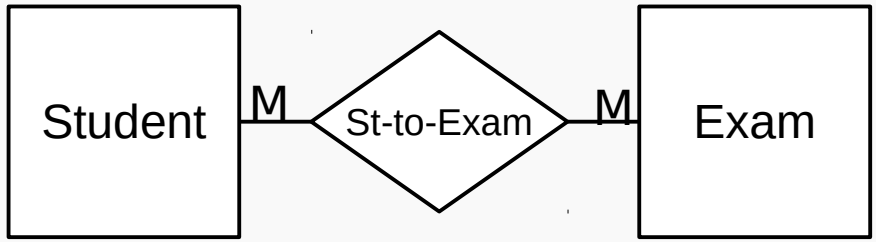
Связь «многие-ко-многим»

- Для реализации связи вида «многие-ко-многим» создается вспомогательная таблица:

Связь «многие-ко-многим»

```
CREATE TABLE STUDENT (
    ST_ID integer PRIMARY KEY,
    ST_NAME text
);

CREATE TABLE EXAM (
    EX_ID integer PRIMARY KEY,
    EX_NAME text
);
```




```
CREATE TABLE STUD_TO_EXAM (
    ST_ID integer REFERENCES STUDENTS,
    EX_ID integer REFERENCES EXAMS,
    PRIMARY KEY (ST_ID, EX_ID)
);
```

ИТМО ВТ

Литература

При подготовке презентации использовались материалы из:

- Введение в реляционные базы данных / В. В. Кириллов, Г. Ю. Громов, Издательство: BHV, 2009 г.
- Документация PostgreSQL.

  https://www.postgresql.org/about/licence/

ИТМО ВТ

Лекция 3: создание Базы Данных: ограничения, представления

1. Работа с таблицами

ИТМО ВТ

Проектирование базы данных

1. Пользовательское представление о предметной области.
2. Инфологическая модель.
3. **Физическая реализация.**

Проектирование «сверху-вниз» (top-down approach)

ИТМО ВТ

Временные таблицы

Время существования ограничено сессией/транзакцией.

```
CREATE TEMPORARY TABLE SESSION_STUDENT (
        StudentID int,
        Name text,
        Surname text
);
```

ИТМО ВТ

Создание таблицы с LIKE

Предположим, что есть таблица:

```
CREATE TABLE APPLICANT (
        StudentID int,
        Name text,
        Surname text
);
```

Можно создать таблицу на основе уже созданной таблицы.

ИТМО ВТ

Создание таблицы с LIKE

```
CREATE TABLE STUDENT LIKE APPLICANT (
        GroupID int
);
```

Копируются атрибуты с типами данных, NOT NULL ограничения.

Для настройки копируемых частей:

INCLUDING DEFAULTS, INCLUDING CONSTRAINTS, ...

ИТМО ВТ

Последовательности

Используются для генерации значений на основе заданной логики:

CREATE [options] SEQUENCE INDEX_NAME

[INCREMENT [BY] incValue]

[MINVALUE minValue | NO MINVALUE]

[MAXVALUE maxValue | NO MAXVALUE]

[START [WITH] startValue]

[CACHE cache] [[NO] CYCLE]

...

Последовательности

```
CREATE SEQUENCE studentIDSeq
        START WITH 1
        INCREMENT BY 2
        MAXVALUE 10000
        NOCACHE;
```

Функции для управления: currval, nextval, setval, lastval

```
SELECT currval('studentIDSeq');
```

ИТМО ВТ

Последовательности

```
INSERT INTO STUDENT(StudentID, Name, Surname)
VALUES (
        nextval('studentIDSeq'),
        'Ivan',
        'Ivanov'
);
```

ИТМО ВТ

Последовательности

```
CREATE TABLE STUDENT(
        StudentID int DEFAULT nextval('studentIDSeq')
                                PRIMARY KEY,
        Name text,
        Surname text
);
```

ИТМО ВТ

serial

Тип для генерации значений на основе последовательности.

3 вида:

- smallserial — на основе smallint;
- serial — на основе int;
- bigserial — на основе bigint;

ИТМО ВТ

serial

```
CREATE TABLE STUDENT(
        StudentID serial,
        Name text,
        Surname text
);
```

На самом деле создается последовательность, связанная с таблицей STUDENT.

ИТМО ВТ

Виртуальные таблицы

Виртуальные — представления, неименованные таблицы (таблицы, которые не существуют постоянно в базе данных).

Можно построить на основе **запроса**.

2. Запросы

ИТМО ВТ

Запрос

Части запроса:

- **WHERE** — для определения, какие строки должны быть выбраны или включены в GROUP BY.
- **GROUP BY** — для объединения строк с общими значениями в элементы меньшего набора строк.
- **HAVING** — для определения, какие строки после GROUP BY должны быть выбраны.
- **ORDER BY** — сортировка результирующего набора данных.

SELECT поля FROM таблицы WHERE условия…

ИТМО ВТ

Логические операторы

- AND
- OR
- NOT

Состояния: true, false, NULL

AND и OR коммутативны: от перемены мест операндов результат не меняется

ИТМО ВТ

Операторы сравнения

>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
=	Равно
<> или !=	Не равно

- Операторы бинарные
- Возвращают значения типа boolean
- NULL, когда любой из операндов NULL

ИТМО ВТ

Предикаты сравнения

a BETWEEN *x* AND *y* /* между */

a NOT BETWEEN *x* AND *y* /* не между */

a IS DISTINCT FROM *b* /* как <>, кроме NULL */

a IS NOT DISTINCT FROM *b* /*как =, кроме NULL */

выражение IS NULL /* является NULL */

выражение IS NOT NULL /* не является NULL */

ИТМО ВТ

Математические операторы

Оп.	Описание	Пример
+	Сложение	58 + 61
-	Вычитание	23 - 9
*	Умножение	12 * 4
/	Деление (при целочисленном делении остаток отбрасывается)	10 / 5
%	Остаток от деления	5 % 2
^	Возведение в степень	7.0 ^ 2.0
|/	Квадратный корень	|/ 64
||/	Кубический корень	||/ 64
!	Факториал	4 !
!!	Факториал (префиксная форма)	!! 4

ИТМО ВТ

Поиск по шаблону: LIKE

- *строка* LIKE *шаблон*
- *строка* NOT LIKE *шаблон*

Выражение LIKE возвращает true, если строка соответствует заданному шаблону.

```
'abc' LIKE '_b_'    /* true */
'abc' LIKE 'c'      /* false */
```

ILIKE — регистро-независимый поиск

Поиск по шаблону: SIMILAR TO

- *строка* SIMILAR TO *шаблон*
- *строка* NOT SIMILAR TO *шаблон*

Выражение SIMILAR TO возвращает true, если строка соответствует заданному шаблону.

Поддерживает больше видов регулярных выражений, чем LIKE

ИТМО ВТ

Регулярные выражения POSIX

- ~ /* Проверяет соответствие регулярному выражению с учётом регистра */
- ~* /* Проверяет соответствие регулярному выражению без учёта регистра */
- !~ /* Проверяет несоответствие регулярному выражению с учётом регистра */
- !~* /* Проверяет несоответствие регулярному выражению без учёта регистра */

ИТМО ВТ

Регулярные выражения POSIX

- Предоставляют более мощные средства поиска по шаблонам, чем операторы LIKE и SIMILAR TO

- Примеры:

```
'abcd' ~ 'abcd'        /* true */
'abcd' ~ '(b|d)'       /* true */
'abcd' ~ '^(b|c)'      /* false */
```

Условные выражения: CASE

if/then/else в других языках программирования:

CASE WHEN *условие* THEN *результат*

[WHEN ...]

[ELSE результат]

END

ИТМО ВТ

Условные выражения

1. CASE можно использовать везде, где допускаются выражения.

2. Каждое условие представляет собой выражение, возвращающее boolean.

3. Если не выполняются условия WHEN, значением CASE становится результат в ELSE.

4. Если в 3. предложение ELSE отсутствует, результатом будет NULL.

Условные выражения

```
SELECT Result FROM EXAM;
Result
------
 76
 95

SELECT Result,
       CASE WHEN Result>=91 THEN 'Отл.'
            WHEN Result>=75 THEN 'Хор.'
            WHEN Result>=60 THEN 'Удовл.'
            ELSE 'other result'
       END AS Description
    FROM EXAM;

 Result| Description
-------+-------------
   76  | Хор.
   95  | Отл.
```

Преобразования типов

- CAST (*expression* AS *type*) /* SQL-синтаксис */
- *expression*::*type* /* PostgreSQL */
- *typename* (*expression*) /* через функцию */

Пример:

```
SELECT CAST(42 AS float8);
/* через функцию float8(int4) */
```

```
CREATE CAST (source_type AS target_type)
    WITH FUNCTION function_name (argument_type [, ...])
```

ИТМО ВТ

IN

- *выражение* IN (*значение* [, …])

Результат «true», если значение *выражения* равняется одному из значений выражений в правой части:

(выражение = значение1) OR (выражение = значение2)

OR ...

3. Запросы с использованием нескольких таблиц

Запросы с использованием нескольких таблиц

Вывести список всех студентов из России:

2 таблицы:

STUDENT(StudentID, Name, Surname, City)

CITIES(CityName, Country)

Соединение с помощью фразы WHERE

Вывести список всех студентов из России:

```
SELECT Name, City
FROM STUDENT, CITIES
WHERE CITIES.CityName = STUDENT.City AND
        CITIES.Country = 'Россия';
```

ИТМО ВТ

Запросы с использованием нескольких таблиц

1. СУБД формирует строки декартова произведения таблиц, перечисленных во фразе FROM.
2. СУБД проверяет, удовлетворяют ли данные в сформированной строке условиям из фразы WHERE.
3. Если данные в строке удовлетворяют условию из фразы WHERE, то СУБД включает в ответ на запрос те поля строки, которые соответствуют столбцам, перечисленным во фразе SELECT.

Декартово произведение таблиц

lt:

id	name
1	aaa
2	bbb
3	ccc

rt:

id	value
1	xxx
3	yyy
7	zzz

```
=> SELECT * FROM lt,rt;
```

id	name	id	value
1	aaa	1	xxx
1	aaa	3	yyy
1	aaa	7	zzz
2	bbb	1	xxx
2	bbb	3	yyy
2	bbb	7	zzz
3	ccc	1	xxx
3	ccc	3	yyy
3	ccc	7	zzz

ИТМО ВТ

Декартово произведение таблиц

- Декартово произведение n таблиц — это таблица, содержащая все возможные строки s, такие, что s является сцеплением какой-либо строки из первой таблицы, строки из второй таблицы, ... и строки из n-й таблицы.
- Количество строк декартова произведения таблиц равно произведению количества строк соединяемых таблиц.

4. Соединения (Joins)

ИТМО ВТ

Соединения

Начиная с SQL:1992 и для соединений таблиц принято использовать фразу JOIN.

- T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 ON boolean_expression
- T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING (join column list)
- T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2

Соединения (JOIN ON)

- В предложении с ON указываются выражения логического типа.
- Пара строк из таблиц 1 и 2 соответствуют друг другу, если выражение ON возвращает для них true.

```
SELECT Surname
FROM STUDENT
JOIN CITIES ON (CityName = City);
```

Позволяет сформировать **эквисоединение таблиц**

Соединения (JOIN USING)

- USING — сокращённая запись условия, когда с обеих сторон соединения столбцы имеют одинаковые имена. Например, запись соединения таблиц LT и RT с USING (at1, at2) формирует условие:

  ON LT.at1 = RT.at1 AND LT.at2 = RT.at2

- При выводе JOIN USING исключаются избыточные столбцы.

```
SELECT Surname
FROM STUDENT
JOIN STUDENT_OLYMPIAD USING (StudentID);
```

ИТМО ВТ

Соединения (NATURAL JOIN)

NATURAL — упрощённая форма USING: образует список USING из всех имён столбцов, существующих в обеих входных таблицах.

Если столбцов с одинаковыми именами **нет**, NATURAL работает как CROSS JOIN.

```
SELECT Surname
FROM STUDENT
NATURAL JOIN STUDENT_OLYMPIAD;
```

ИТМО ВТ

Виды соединений

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- CROSS JOIN

INNER JOIN

Для каждой строки из исходной таблицы LT итоговая таблица включает строку для каждой строки из таблицы RT, если строка удовлетворяет условию соединения.

INNER JOIN

lt:

id	name
1	aaa
2	bbb
3	ccc

rt:

id	value
1	xxx
3	yyy
7	zzz

```
=> SELECT lt.id, name, value
   FROM lt INNER JOIN rt ON lt.id = rt.id;
 lt.id| name | value
------+------+-------
    1 |  aaa | xxx
    3 |  ccc | yyy
```

LEFT OUTER JOIN

1. Сначала выполняется внутреннее соединение (INNER JOIN).
2. В результат добавляются все строки из таблицы LT, которым не соответствуют никакие строки из таблицы RT; вместо значений столбцов RT вставляются NULL.

В результирующей таблице всегда будет минимум одна строка для каждой строки из LT.

LEFT OUTER JOIN

lt:

id	name
1	aaa
2	bbb
3	ccc

rt:

id	value
1	xxx
3	yyy
7	zzz

```
=> SELECT lt.id, name, value FROM lt
   LEFT JOIN rt ON lt.id = rt.id;
```

lt.id	name	value
1	aaa	xxx
2	bbb	
3	ccc	yyy

ИТМО ВТ

RIGHT OUTER JOIN

1. Сначала выполняется внутреннее соединение (INNER JOIN).
2. В результат добавляются все строки из таблицы RT, которым не соответствуют никакие строки из таблицы LT; вместо значений столбцов LT вставляются NULL.

Это соединение является обратным к левому (LEFT JOIN): в результирующей таблице всегда будет минимум одна строка для каждой строки из таблицы RT.

RIGHT OUTER JOIN

lt:

```
 id  | name
-----+------
   1 | aaa
   2 | bbb
   3 | ccc
```

rt:

```
 id  | value
-----+-------
   1 | xxx
   3 | yyy
   7 | zzz
```

```
=> SELECT name, rt.id, value FROM lt
   RIGHT JOIN rt ON lt.id = rt.id;
 name | rt.id | value
------+-------+-------
 aaa  |   1   | xxx
 ccc  |   3   | yyy
      |   7   | zzz
```

FULL OUTER JOIN

1. Сначала выполняется внутреннее соединение.
2. В результат добавляются все строки из таблицы LT, которым не соответствуют никакие строки из таблицы RT; вместо значений столбцов RT вставляются NULL.
3. В результат добавляются все строки из таблицы RT, которым не соответствуют никакие строки из таблицы LT; вместо значений столбцов LT вставляются NULL.

FULL OUTER JOIN

lt:

id	name
1	aaa
2	bbb
3	ccc

rt:

id	value
1	xxx
3	yyy
7	zzz

```
=> SELECT * FROM lt
FULL JOIN rt ON lt.id = rt.id;
 id  | name | id  | value
-----+------+-----+-------
   1 | aaa  |   1 | xxx
   2 | bbb  |     |
   3 | ccc  |   3 | yyy
     |      |   7 | zzz
```

CROSS JOIN

Осуществляет полное перекрестное соединение двух таблиц.

Каждая строка первой таблицы соединяется со всеми строками второй таблицы, что создает результирующий набор — **декартово произведение таблиц**.

CROSS JOIN

lt:

id	name
1	aaa
2	bbb
3	ccc

rt:

id	value
1	xxx
3	yyy
7	zzz

```
=> SELECT * FROM lt CROSS JOIN rt;
```

id	name	id	value
1	aaa	1	xxx
1	aaa	3	yyy
1	aaa	7	zzz
2	bbb	1	xxx
2	bbb	3	yyy
2	bbb	7	zzz
3	ccc	1	xxx
3	ccc	3	yyy
3	ccc	7	zzz

5. Вложенные подзапросы

ИТМО ВТ

Вложенный подзапрос

- Вложенный подзапрос (subquery) — предложение SELECT, которое заключено в круглые скобки и вложено в WHERE/HAVING часть другого SQL-предложения.
- Может содержать в своей WHERE (HAVING) - части другой вложенный подзапрос.
- Во внешнем и вложенном подзапросе может использоваться одна и та же таблица.

Простые вложенные подзапросы

ИТМО ВТ

- Простые вложенные подзапросы применяются для получения множества значений, которые проверяются в основном запросе:

```
SELECT Surname
FROM STUDENT
WHERE CityName IN (
        SELECT City FROM CITIES
        WHERE CITIES.Country = 'Россия'
);
```

Простые вложенные подзапросы

ИТМО ВТ

- Обрабатываются "снизу вверх": первым обрабатывается вложенный подзапрос самого нижнего уровня.

```
SELECT Surname
FROM STUDENT
WHERE CityName IN (
        SELECT City FROM CITIES
        WHERE CITIES.Country = 'Россия'
);
```

Получаем список городов в России, используем их для выполнения внешнего запроса

Простые вложенные подзапросы

- Обрабатываются "снизу вверх": первым обрабатывается вложенный подзапрос самого нижнего уровня.

```
SELECT Surname
FROM STUDENT
WHERE CityName IN (
        'Москва', 'Санкт-Петербург', ...
);
```

Простые вложенные подзапросы

ИТМО ВТ

То же можно получить с помощью соединения:

```
SELECT Surname
FROM STUDENT
WHERE CityName IN (
SELECT City FROM CITIES
WHERE
    CITIES.Country = 'Россия'
);
```

```
SELECT Surname
FROM STUDENT
JOIN CITIES ON CityName = City
WHERE CITIES.Country = 'Россия';
```

Коррелированные вложенные подзапросы

Вложенный подзапрос не может быть выполнен до обработки внешнего запроса:

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        SELECT 1 FROM STUDENT_OLYMPIAD
        WHERE StID = STUDENT.StudentID
);
```

Коррелированные вложенные подзапросы

- Вложенный подзапрос зависит от значения **STUDENT.StudentID.**
- Это значение изменяется по мере того, как СУБД проверяет строки таблицы **STUDENT**.

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        SELECT 1 FROM STUDENT_OLYMPIAD
        WHERE StID = STUDENT.StudentID
);
```

Коррелированные вложенные подзапросы

1. СУБД проверяет первую строку таблицы STUDENT (StudentID = 1). Значение STUDENT.StudentID в этот момент = 1. СУБД обрабатывает внутренний запрос, получая в результате 1, если такой олимпиадник есть:

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        SELECT 1 FROM STUDENT_OLYMPIAD WHERE
        StID = STUDENT.StudentID
);
```

Коррелированные вложенные подзапросы

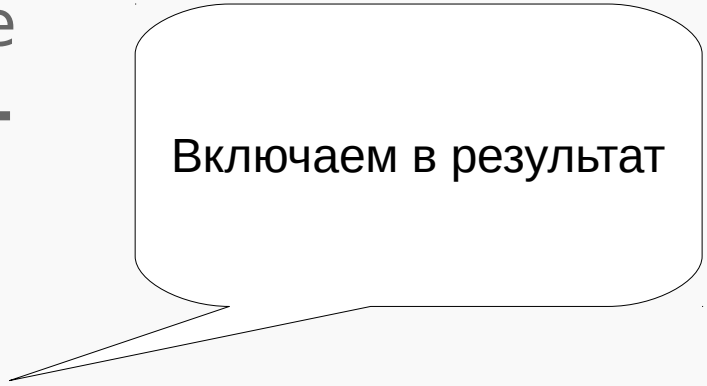
1. СУБД проверяет первую строку таблицы STUDENT (StudentID = 1). Значение STUDENT.StudentID в этот момент = 1. СУБД обрабатывает внутренний запрос, получая в результате 1, если такой олимпиадник есть:

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        SELECT 1 FROM STUDENT_OLYMPIAD WHERE
        StID = 1
);
```

Коррелированные вложенные подзапросы

1. СУБД проверяет первую строку таблицы STUDENT (StudentID = 1). Значение STUDENT.StudentID в этот момент = 1. СУБД обрабатывает внутренний запрос, получая в результате 1, если такой олимпиадник есть:

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        1
);
```



Коррелированные вложенные подзапросы

2. Система будет повторять обработку для каждой строки из таблицы STUDENT, пока не будут рассмотрены все строки таблицы STUDENT.

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        SELECT 1 FROM STUDENT_OLYMPIAD
        WHERE StID = 2
);
```

Коррелированные вложенные подзапросы

- Обрабатываются СУБД в обратном порядке.
- Сначала выбирается первая строка таблицы, сформированной основным запросом. Из этой строки выбираются значения тех столбцов, которые используются во вложенном подзапросе.
- Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат.

6. Операторы для работы со вложенными подзапросами

ИТМО ВТ

IN

- *выражение* IN (*подзапрос*)

- *Подзапрос* должен возвращать ровно один столбец.
- Результат *выражения* сравнивается с каждым значением, возвращённым *подзапросом*.
- Результатом выражения IN будет «true», если значение *выражения* соответствует хотя бы одному значению, которое вернул подзапрос; «false» в противном случае *(включая случай, когда подзапрос не возвращает строк)*.

ИТМО ВТ

EXISTS

- EXISTS (*подзапрос*)

- EXISTS принимает оператор SELECT (*подзапрос)*
- Если *подзапрос* возвращает хотя бы одну строку, то результатом EXISTS будет «true», а если не возвращает ни одной — «false».

ИТМО ВТ

EXISTS

- Вывести фамилии студентов, попавших на комиссию?

STUDENT (StudentID, Surname, Name, GroupID)

EXAM (ExamID, StID, Result)

ИТМО ВТ

EXISTS

- EXISTS (*подзапрос*)

```
SELECT Surname
FROM STUDENT
WHERE EXISTS (
        SELECT 1 FROM EXAM WHERE
          STUDENT.StudentID = EXAM.StID
          AND EXAM.Result < 60
);
```

ИТМО ВТ

ANY/SOME

выражение оператор ANY (*подзапрос*)

выражение оператор SOME (*подзапрос*)

- *Подзапрос* должен возвращать ровно один столбец
- Значение *выражения* сравнивается со значением в каждой строке результата *подзапроса* с помощью заданного *оператора*, который должен возвращать логическое значение.
- Результатом ANY будет «true», если хотя бы для одной строки условие истинно, и «false» в противном случае (в том числе, если подзапрос не возвращает строк).

ИТМО ВТ

ANY, пример

```
SELECT *
    FROM STUDENT
        WHERE StudentId = ANY
            ( SELECT StID FROM EXAMS );
```

ИТМО ВТ

ALL

выражение оператор ALL (*подзапрос*)

- *Подзапрос* должен возвращать ровно один столбец
- Значение *выражения* сравнивается со значением в каждой строке результата *подзапроса* с помощью *оператора*, который должен возвращать логическое значение.
- Результатом ALL будет «true», если условие истинно для всех строк (и когда *подзапрос* не возвращает строк), и «false», если находятся строки, для которых оно ложно.
- Результат будет NULL, если сравнение не возвращает false ни для одной из строк, но как минимум для одной результат сравнения NULL.

7. Представления

ИТМО ВТ

Создание таблиц

Таблицы:

- *Базовые* — в действительности существующие, хранящиеся в физической памяти машины.

- ***Виртуальные*** — представления, неименованные таблицы (таблицы, которые не существуют постоянно в базе данных).

ИТМО ВТ

Представление

Представление — именованный запрос.

```
CREATE VIEW VIEW_NAME
[ ( ColumnName [, ...] ) ]
AS подзапрос
...
```

ИТМО ВТ

Представление

```
CREATE VIEW PICTStudents AS
    SELECT * FROM STUDENT
    WHERE GroupID IN (
        SELECT GroupID FROM GROUP
         WHERE GroupName LIKE 'P3%'
    );
```

ИТМО ВТ

Представление

```
CREATE VIEW PICTStudents2 AS
    (PICTId, pSurname) AS
    SELECT StudentID, Surname FROM STUDENT
    WHERE GroupID IN (
        SELECT GroupID FROM GROUP
         WHERE GroupName LIKE 'P3%'
    );


SELECT PICTid FROM PICTStudents2;
```

Материализованные представления

```
CREATE MATERIALIZED VIEW PICTStudents3 AS
    (PICTId, pSurname) AS
    SELECT StudentID, Surname FROM STUDENT
    WHERE GroupID IN (
        SELECT GroupID FROM GROUP
         WHERE GroupName LIKE 'P3%'
    );
```

- Результат запроса сохраняется в базе данных.
- Для обновления данных:

```
REFRESH MATERIALIZED VIEW PICTStudents3;
```

ИТМО ВТ

Литература

При подготовке презентации использовались материалы из:

- Введение в реляционные базы данных / В. В. Кириллов, Г. Ю. Громов, Издательство: BHV, 2009 г.
- Документация PostgreSQL.

  https://www.postgresql.org/about/licence/