

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего  
образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

## Проектирование баз данных

Методические указания к выполнению лабораторных работ (draft)

Санкт-Петербург

2022

Составитель: Н.В. Путилова

Рецензент:

Методические указания предназначены для студентов специальностей 02.03.03 и 09.03.04, изучающих дисциплину «Проектирование баз данных». В методические указания включены краткие теоретические сведения, необходимые для выполнения лабораторных работ, требования к содержанию отчетов и порядку выполнения работ, а также варианты индивидуальных заданий.

Методические указания подготовлены кафедрой компьютерных технологий и программной инженерии.

## Оглавление

<b>Оглавление .....</b>	<b>3</b>
<b>Лабораторная работа №1 Разработка физической модели базы данных с учетом декларативной ссылочной целостности.....</b>	<b>5</b>
1. Теоретическая часть .....	5
2. Выполнение лабораторной работы .....	10
3. Содержание отчета .....	10
4. Варианты заданий .....	10
<b>Лабораторная работа №2 Создание и модификация базы данных и таблиц базы данных .....</b>	<b>10</b>
1. Теоретическая часть .....	11
2. Выполнение лабораторной работы .....	22
3. Содержание отчета .....	22
4. Варианты заданий .....	23
<b>Лабораторная работа №3 Заполнение таблиц и модификация данных.....</b>	<b>23</b>
1. Теоретическая часть .....	23
2. Выполнение лабораторной работы .....	31
3. Содержание отчета .....	31
4. Варианты заданий .....	32
<b>Лабораторная работа №4 Разработка SQL запросов: виды соединений и шаблоны .....</b>	<b>32</b>
1. Теоретическая часть .....	32
2. Выполнение лабораторной работы .....	38
3. Содержание отчета .....	38
4. Варианты заданий .....	38
<b>Лабораторная работа №5 Разработка SQL запросов: запросы с подзапросами .....</b>	<b>39</b>
1. Теоретическая часть .....	39
2. Выполнение лабораторной работы .....	51
3. Содержание отчета .....	52
4. Варианты заданий .....	52
<b>Лабораторная работа №6 Хранимые процедуры.....</b>	<b>52</b>
1. Теоретическая часть .....	52
2. Теоретическая часть .....	52
Синтаксис хранимых процедур и функций .....	53
3. Выполнение лабораторной работы .....	59

4. Содержание отчета .....	60
5. Варианты заданий .....	60

<b>Лабораторная работа №7 Триггеры. Обеспечение активной целостности данных базы данных.....</b>	<b>Ошибка! Закладка не определена.</b>
--	--

1. Теоретическая часть .....	Ошибка! Закладка не определена.
2. Выполнение лабораторной работы .....	Ошибка! Закладка не определена.
3. Содержание отчета .....	Ошибка! Закладка не определена.
4. Варианты заданий .....	Ошибка! Закладка не определена.

<b>Контрольная работа. Проектирование взаимодействия базы данных и приложения .....</b>	<b>60</b>
---	-----------

1. Теоретическая часть .....	60
2. Выполнение лабораторной работы .....	74
3. Содержание отчета .....	74
4. Варианты заданий .....	75

<b>Приложение 1 Распределение баллов.....</b>	<b>76</b>
---	-----------

Семестр 6.....	76
----------------	----

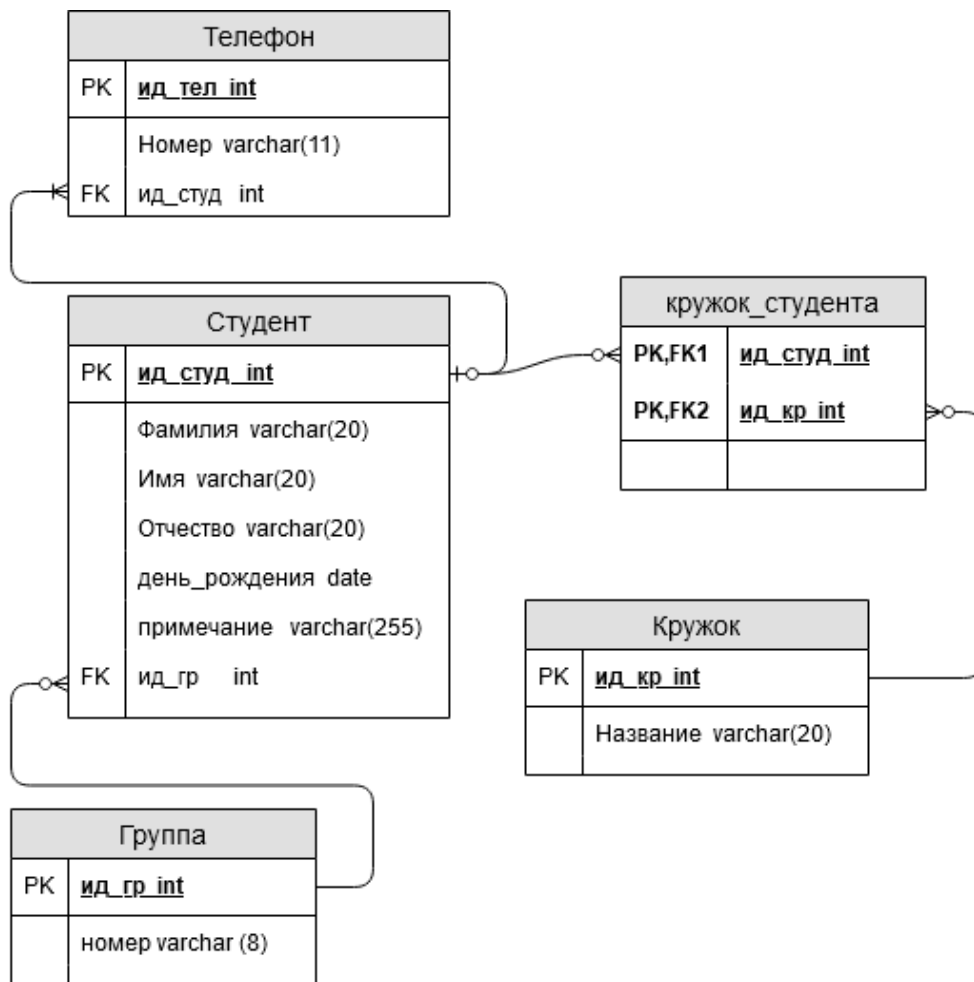
<b>Приложение 2 Варианты заданий лабораторных 1-6 и контрольной работы .....</b>	<b>76</b>
--	-----------

# Лабораторная работа №1 Разработка физической модели базы данных с учетом декларативной ссылочной целостности

## 1. Теоретическая часть

Физические модели данных (PDM) обычно генерируются из логических моделей, но могут быть и воссозданы из действующей базы данных. Средства физического моделирования поддерживают множество стандартных нотаций, а также документирование, генерацию кода и воссоздание структур данных для более чем 70 РСУБД (в том числе новейших версий СУБД Oracle, IBM, Microsoft, Sybase, Teradata, MySQL, HP NeoView и многих других). Поддерживаются все объекты СУБД, необходимые для связывания таблиц и достижения требуемой производительности, такие как индексы, констрейнты, партиции и кластеры, а также новые технологии, такие как Java, XML и веб-сервисы в БД; проектируются средства обеспечения безопасности, применяются передовые методы создания представлений, обеспечивается ряд других возможностей. Физическую модель данных, как правило, можно использовать для оценки объема будущей БД; она может содержать специфическую информацию о выделении дисковой памяти для заданной СУБД.

1. Можно выделить следующие этапы методологии физического проектирования баз данных.
2. 1. Перенос глобальной логической модели данных в среду целевой СУБД.
3. 2. Проектирование основных отношений.
4. 3. Разработка способов получения производных данных.
5. 4. Реализация ограничений предметной области.
6. 5. Проектирование физического представления базы данных.
7. 6. Анализ транзакций.
8. 7. Выбор файловой структуры.
9. 8. Определение индексов.
10. 9. Определение требований к дисковой памяти.
11. 10. Проектирование пользовательских представлений.
12. 11. Разработка механизмов защиты.
13. 12. Обоснование необходимости введения контролируемой избыточности.
14. 13. Текущий контроль и настройка операционной систем
15. Логическая модель расширяет концептуальную путем определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними.
16. Физическая модель данных описывает реализацию объектов логической модели на уровне объектов конкретной базы данных.



Пример Физической модели БД

Логическое представление реляционных баз данных упрощается созданием связей между данными на основе (логической) конструкции, называемой таблицей. Под таблицей понимается двумерная структура, состоящая из строк и столбцов. Пользователь должен понимать, что таблица содержит группу связанных сущностей, т. е. набор сущностей; по этой причине термины набор сущностей и таблица чаще всего означают одно и то же. Таблица также называется отношением (relation), поскольку создатель реляционной модели Э. Ф. Кодд (E. F. Codd) использовал термин "отношение" как синоним слова "таблица".

#### Правила целостности

#### ЦЕЛОСТНОСТЬ НА УРОВНЕ СУЩНОСТИ

**Требование** Все элементы первичного ключа уникальны и никакая часть первичного ключа не может быть пустой (null)

**Назначение** Гарантирует, что каждая сущность (логический объект) будет иметь уникальную идентификацию, а значения внешнего ключа могут должным образом ссылаться на значения первичного ключа

**Пример** Счет не может иметь несколько дублирующихся значений и не может иметь пустое значение (null). Короче говоря, все счета уникально идентифицируются своим номером

#### ЦЕЛОСТНОСТЬ НА УРОВНЕ ССЫЛКИ

**Требования** Внешний ключ может иметь или пустое значение (если только он не является частью первичного ключа данной таблицы), или значение, совпадающее со значением первичного ключа в

связанной таблице. (Каждое непустое значение внешнего ключа должно ссылаться на существующее значение первичного ключа.)

**Назначение** Допускается, что атрибут не имеет соответствующего значения, но атрибут не может принимать недопустимые значения. Выполнение правила целостности на уровне ссылки делает невозможным удаление строки в одной таблице, где первичный ключ имеет обязательное соответствие со значением внешнего ключа в другой таблице

**Пример** Клиенту может быть не назначен (еще) торговый агент, но невозможно назначить клиенту несуществующего агента

Теперь о *внешних ключах*:

- Если сущность С связывает сущности А и В, то она должна включать внешние ключи, соответствующие первичным ключам сущностей А и В.
- Если сущность В обозначает сущность А, то она должна включать внешний ключ, соответствующий первичному ключу сущности А.
- Здесь для обозначения любой из ассоциируемых сущностей (стержней, характеристик, обозначений или даже ассоциаций) используется новый обобщающий термин "Цель" или "Целевая сущность".
- Таким образом, при рассмотрении проблемы выбора способа представления ассоциаций и обозначений в базе данных основной вопрос, на который следует получить ответ: "Каковы внешние ключи?". И далее, для каждого внешнего ключа необходимо решить три вопроса:
- 1. Может ли данный внешний ключ принимать неопределенные значения (NULL-значения)?  
Иначе говоря, может ли существовать некоторый экземпляр сущности данного типа, для которого неизвестна целевая сущность, указываемая внешним ключом? В случае поставок это, вероятно, невозможно – поставка, осуществляемая неизвестным поставщиком, или поставка неизвестного продукта не имеют смысла. Но в случае с сотрудниками такая ситуация однако могла бы иметь смысл – вполне возможно, что какой-либо сотрудник в данный момент не зачислен вообще ни в какой отдел. Заметим, что ответ на данный вопрос не зависит от прихоти проектировщика базы данных, а определяется фактическим образом действий, принятым в той части реального мира, которая должна быть представлена в рассматриваемой базе данных. Подобные замечания имеют отношение и к вопросам, обсуждаемым ниже.
- 2. Что должно случиться при попытке УДАЛЕНИЯ целевой сущности, на которую ссылается внешний ключ? Например, при удалении поставщика, который осуществил по крайней мере одну поставку. Существует три возможности:

**КАСКАДИРУЕТСЯ** Операция удаления "каскадируется" с тем, чтобы удалить также поставки этого поставщика.

**ОГРАНИЧИВАЕТСЯ** Удаляются лишь те поставщики, которые еще не осуществляли поставок. Иначе операция удаления отвергается.

**УСТАНАВЛИВАЕТСЯ** Для всех поставок удаляемого поставщика NULL-значение внешний ключ устанавливается в неопределенное значение, а затем этот поставщик удаляется. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

- 3. Что должно происходить при попытке ОБНОВЛЕНИЯ первичного ключа целевой сущности, на которую ссылается некоторый внешний ключ? Например, может быть предпринята попытка обновить номер такого поставщика, для которого имеется по крайней

мере одна соответствующая поставка. Этот случай для определенности снова рассмотрим подробнее. Имеются те же три возможности, как и при удалении:

- КАСКАДИРУЕТСЯ** Операция обновления "каскадируется" с тем, чтобы обновить также и внешний ключ в поставках этого поставщика.
- ОГРАНИЧИВАЕТСЯ** Обновляются первичные ключи лишь тех поставщиков, которые еще не осуществляли поставок. Иначе операция обновления отвергается.
- УСТАНОВЛИВАЕТСЯ** Для всех поставок такого поставщика NULL-значение внешнего ключ устанавливается в неопределенное значение, а затем обновляется первичный ключ поставщика. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

- Таким образом, для каждого внешнего ключа в проекте проектировщик базы данных должен специфицировать не только поле или комбинацию полей, составляющих этот внешний ключ, и целевую таблицу, которая идентифицируется этим ключом, но также и ответы на указанные выше вопросы (три ограничения, которые относятся к этому внешнему ключу).

Ссылочную целостность при выполнении лабораторной необходимо описать в таблице вида

Дочерняя таблица (с внешним ключом)	Внешний ключ	Родительская таблица	Как поддерживается ссылочная целостность при удалении	Описание ссылочной целостности при удалении	Как поддерживается ссылочная целостность при обновлении	Описание ссылочной целостности при обновлении	Обоснование
Table1	Id_t2	Table2	Каскадируется	При удалении данных из Table2, удалятся все связанные данные из Table1	Каскадируется	При обновлении первичного ключа Table2, обновится внешний ключ из Table1	Логическое обоснование почему так
Table1	Id_t2	Table2	ограничивается	При удалении	ограничивается	При обновлении	Логическое обоснование



				данных из Table2, если есть связанные данные из Table1, удаление будет отменено/запрещено		первичного ключа Table2, если есть связанные данные из Table1, обновление будет отменено/запрещено	почему так
Table1	Id_t2	Table2	устанавливается	При удалении данных из Table2, внешний ключ Table1, ссылающийся на них получит значение null (или значение по умолчанию)	ограничивается	При обновлении первичного ключа Table2, внешний ключ Table1, ссылающийся на них получит значение null (или значение по умолчанию)	Логическое обоснование почему так
student	Id_gr	st_group	каскадируется	При удалении данных из таблицы «st_group», удалятся ссылающиеся	ограничивается	При обновлении первичного ключа «st_group», если есть связанные	Может быть необходимы м удалять группу со всеми студентами, но

				еся на них данные в таблице «student»		данные из «student», обновление будет отменено/ запрещено	необходимос ть менять суррогатный внешний ключ маловероятна
--	--	--	--	--	--	--	--

Описание может быть не в таблице, но должно содержать следующие данные:

- Дочерняя таблица;
- Столбцы, составляющие внешний ключ;
- Родительская таблица;
- Наименование ссылочной целостности при удалении;
- Описание действий по поддержанию ссылочной целостности при удалении;
- Наименование ссылочной целостности при обновлении;
- Описание действий по поддержанию ссылочной целостности при обновлении;
- Обоснование выбора типа поддержки ссылочной целостности.

## 2. Выполнение лабораторной работы

Создать физическую модель базы данных, находящуюся в третьей нормальной форме в соответствии с заданным вариантом. Расписать ссылочную целостность БД в таблице

## 3. Содержание отчета

Содержание отчета:

- Текст задания.
- физическую модель БД.
- таблица с описанием ссылочной целостности.

## 4. Варианты заданий

Варианты заданий приведены в Приложении 2.

## Лабораторная работа №2 Создание и модификация базы данных и таблиц базы данных

## 1. Теоретическая часть

Таблица . Основные типы данных языка SQL, определенные в стандарте ISO(SQL -92)

Тип данных	Объявление	Реализация Microsoft SQL Server	Реализация MySQL
boolean (Логический)	BOOLEAN		BOOL, BOOLEAN, TINYINT(1).
character (Символьный)	CHAR	CHAR[(M)], NCHAR[(M)](unicode)	CHAR [(M)]
	VARCHAR	VARCHAR[(M)], NVARCHAR[(M)](unicode)	VARCHAR[(M)]
bit (Битовый)	BIT, BIT VARYING	BIT	BIT BIT[(M)]
exact numeric (Точные числа)	NUMERIC	NUMERIC[(M[,D])]	DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]
	DECIMAL	DECIMAL[(M[,D])]	DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]
	SMALLINT	SMALLINT	SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
	INTEGER	INT	INT[(M)] [UNSIGNED] [ZEROFILL] INTEGER[(M)] [UNSIGNED] [ZEROFILL]
	TINYINT	TINYINT	TINYINT[(M)] [UNSIGNED] [ZEROFILL] -128 .. 127 или 0..255
	BIGINT	BIGINT	BIGINT (8 байт)
approximate numeric (Округленные числа)	FLOAT	FLOAT[(N)]	FLOAT[(M,D)] [UNSIGNED] [ZEROFILL] REAL[(M,D)] [UNSIGNED] [ZEROFILL](если задан режим REAL AS FLOAT)
	REAL	REAL ( эквивалентно Float(24))	
	DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [ZEROFILL](если не задан режим REAL AS FLOAT)
datetime (Дата/время)	DATE	DATE	DATE
	TIME	TIME	TIME
	TIMESTAMP	DATETIME TIMESTAMP	DATETIME TIMESTAMP
interval	INTERVAL		C другим

(Интервал)	AL		синтаксисом , но используются TIME и TIMESTAMP
LOB (Большой объект)	CHARACTER LARGE OBJECT,	TEXT, VARCHAR(MAX)	TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT
	BINARY LARGE OBJECT	VARBINARY(MAX) VARBINARY(n)	TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
XML (Правильный XML документ)		XML	XML

## Синтаксис оператора **CREATE TABLE**

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]
[table_options] [select_statement]
```

create\_definition:

```
col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
[PRIMARY KEY] [reference_definition]
или PRIMARY KEY (index_col_name,...)
или KEY [index_name] (index_col_name,...)
или INDEX [index_name] (index_col_name,...)
или UNIQUE [INDEX] [index_name] (index_col_name,...)
или FULLTEXT [INDEX] [index_name] (index_col_name,...)
или [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)
[reference_definition]
или CHECK (expr)
```

type:

```
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
или SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
или MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
или INT[(length)] [UNSIGNED] [ZEROFILL]
или INTEGER[(length)] [UNSIGNED] [ZEROFILL]
или BIGINT[(length)] [UNSIGNED] [ZEROFILL]
или REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
или DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
или FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
или DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
или NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
или CHAR(length) [BINARY]
или VARCHAR(length) [BINARY]
или DATE
или TIME
или TIMESTAMP
или DATETIME
или TINYBLOB
или BLOB
или MEDIUMBLOB
или LONGBLOB
или TINYTEXT
или TEXT
или MEDIUMTEXT
```

```

или      LONGTEXT
или      ENUM(value1,value2,value3,...)
или      SET(value1,value2,value3,...)

index_col_name:
    col_name [(length)]

reference_definition:
    REFERENCES tbl_name [(index_col_name,...)]
        [MATCH FULL | MATCH PARTIAL]
        [ON DELETE reference_option]
        [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
    TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
или      AUTO_INCREMENT = #
или      AVG_ROW_LENGTH = #
или      CHECKSUM = {0 | 1}
или      COMMENT = "string"
или      MAX_ROWS = #
или      MIN_ROWS = #
или      PACK_KEYS = {0 | 1 | DEFAULT}
или      PASSWORD = "string"
или      DELAY_KEY_WRITE = {0 | 1}
или      ROW_FORMAT= { default | dynamic | fixed | compressed }
или      RAID_TYPE= {1 | STRIPED | RAID0 } RAID_CHUNKS=# RAID_CHUNKSIZE=#
или      UNION = (table_name,[table_name...])
или      INSERT_METHOD= {NO | FIRST | LAST }
или      DATA DIRECTORY="абсолютный путь к каталогу"
или      INDEX DIRECTORY="абсолютный путь к каталогу"

select_statement:
    [IGNORE | REPLACE] SELECT ... (любое корректное выражение SELECT)

```

Оператор `CREATE TABLE` создает таблицу с заданным именем в текущей базе данных.

. Если нет активной текущей базы данных или указанная таблица уже существует, то возникает ошибка выполнения команды.

В версии MySQL 3.22 и более поздних имя таблицы может быть указано как `db_name.tbl_name`. Эта форма записи работает независимо от того, является ли указанная база данных текущей.

В версии MySQL 3.23 при создании таблицы можно использовать ключевое слово `TEMPORARY`. Временная таблица автоматически удаляется по завершении соединения, а ее имя действительно только в течение данного соединения. Это означает, что в двух разных соединениях могут использоваться временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем (существующая таблица скрыта, пока не удалена временная таблица). В версии MySQL 4.0.2 для создания временных таблиц необходимо иметь привилегии `CREATE TEMPORARY TABLES`.

В версии MySQL 3.23 и более поздних можно использовать ключевые слова `IF NOT EXISTS` для того, чтобы не возникала ошибка, если указанная таблица уже существует. **Следует учитывать, что при этом не проверяется идентичность структур этих таблиц.**

Каждая таблица `tbl_name` представлена определенными файлами в директории базы данных. В случае таблиц типа `MyISAM` это следующие файлы:

Файл	Назначение
<code>tbl_name.frm</code>	Файл определения таблицы
<code>tbl_name.MYD</code>	Файл данных
<code>tbl_name.MYI</code>	Файл индексов

Чтобы получить более полную информацию о свойствах различных типов столбцов,

Если не указывается ни `NULL`, ни `NOT NULL`, то столбец интерпретируется так, как будто указано `NULL`.

- Целочисленный столбец может иметь дополнительный атрибут `AUTO_INCREMENT`. При записи величины `NULL` (рекомендуется) или `0` в столбец `AUTO_INCREMENT` данный столбец устанавливается в значение `value+1`, где `value` представляет собой наибольшее для этого столбца значение в таблице на момент записи. Последовательность `AUTO_INCREMENT` начинается с `1`. Если удалить строку, содержащую максимальную величину для столбца `AUTO_INCREMENT`, то в таблицах типа `ISAM` или `BDB` эта величина будет восстановлена, а в таблицах типа `MyISAM` или `InnoDB` - нет. Если удалить все строки в таблице командой `DELETE FROM table_name` (без выражения `WHERE`) в режиме `AUTOCOMMIT`, то для таблиц всех типов последовательность начнется заново. Примечание: в таблице может быть только один столбец `AUTO_INCREMENT`, и он должен быть индексирован. Кроме того, версия `MySQL 3.23` будет правильно работать только с положительными величинами столбца `AUTO_INCREMENT`. В случае внесения отрицательного числа оно интерпретируется как очень большое положительное число. Это делается, чтобы избежать проблем с точностью, когда числа ``заворачиваются" от положительного к отрицательному и, кроме того, для гарантии, что по ошибке не будет получен столбец `AUTO_INCREMENT` со значением `0`. В таблицах `MyISAM` и `BDB` можно указать вторичный столбец `AUTO_INCREMENT` с многостолбцовым ключом.. Последнюю внесенную строку можно найти с помощью следующего запроса (чтобы сделать `MySQL` совместимым с некоторыми `ODBC`-приложениями):
  - `SELECT * FROM tbl_name WHERE auto_col IS NULL`
- Величины `NULL` для столбца типа `TIMESTAMP` обрабатываются иначе, чем для столбцов других типов. В столбце `TIMESTAMP` нельзя хранить литерал `NULL`; при установке данного столбца в `NULL` он будет установлен в текущее значение даты и времени. Поскольку столбцы `TIMESTAMP` ведут себя подобным образом, то атрибуты `NULL` и `NOT NULL` неприменимы в обычном режиме и игнорируются при их задании. С другой стороны, чтобы облегчить клиентам `MySQL` использование столбцов `TIMESTAMP`, сервер сообщает, что таким столбцам могут быть назначены величины `NULL` (что соответствует действительности), хотя реально `TIMESTAMP` никогда не будет содержать величины `NULL`. Это можно увидеть, применив `DESCRIBE tbl_name` для получения описания данной таблицы. Следует учитывать, что установка столбца `TIMESTAMP` в `0` не равнозначна установке его в `NULL`, поскольку `0` для `TIMESTAMP` является допустимой величиной.
- Величина `DEFAULT` должна быть константой, она не может быть функцией или выражением. Если для данного столбца не задается никакой величины `DEFAULT`, то `MySQL` автоматически назначает ее. Если столбец может принимать `NULL` как допустимую величину, то по умолчанию присваивается значение `NULL`. Если столбец объявлен как `NOT NULL`, то значение по умолчанию зависит от типа столбца:

- Для числовых типов, за исключением объявленных с атрибутом `AUTO_INCREMENT`, значение по умолчанию равно 0. Для столбца `AUTO_INCREMENT` значением по умолчанию является следующее значение в последовательности для этого столбца.
- Для типов даты и времени, отличных от `TIMESTAMP`, значение по умолчанию равно соответствующей нулевой величине для данного типа. Для первого столбца `TIMESTAMP` в таблице значение по умолчанию представляет собой текущее значение даты и времени.
- Для строковых типов, кроме `ENUM`, значением по умолчанию является пустая строка. Для `ENUM` значение по умолчанию равно первой перечисляемой величине (если явно не задано другое значение по умолчанию с помощью директивы `DEFAULT`).

Значения по умолчанию должны быть константами. Это означает, например, что нельзя установить для столбца даты в качестве значения по умолчанию величину функции, такой как `NOW()` или `CURRENT_DATE`.

- `KEY` является синонимом для `INDEX`.
- В MySQL ключ `UNIQUE` может иметь только различающиеся значения. При попытке добавить новую строку с ключом, совпадающим с существующей строкой, возникает ошибка выполнения команды.
- `PRIMARY KEY` представляет собой уникальный ключ `KEY` с дополнительным ограничением, что все столбцы с данным ключом должны быть определены как `NOT NULL`. В MySQL этот ключ называется `PRIMARY` (первичный). Таблица может иметь только один первичный ключ `PRIMARY KEY`. Если `PRIMARY KEY` отсутствует в таблицах, а некоторое приложение запрашивает его, то MySQL может превратить в `PRIMARY KEY` первый ключ `UNIQUE`, не имеющий ни одного столбца `NULL`.
- `PRIMARY KEY` может быть многостолбцовым индексом. Однако нельзя создать многостолбцовый индекс, используя в определении столбца атрибут ключа `PRIMARY KEY`. Именно таким образом только один столбец будет отмечен как первичный. Необходимо использовать синтаксис `PRIMARY KEY(index_col_name, ...)`.
- Если ключ `PRIMARY` или `UNIQUE` состоит только из одного столбца и он принадлежит к числовому типу, то на него можно сослаться также как на `_rowid` (новшество версии 3.23.11).
- Если индексу не назначено имя, то ему будет присвоено первое имя в `index_col_name`, возможно, с суффиксами (`_2`, `_3`, ...), делающими это имя уникальным. Имена индексов для таблицы можно увидеть, используя `SHOW INDEX FROM tbl_name.SHOW Syntax`.
- Только таблицы типов `MyISAM`, `InnoDB` и `BDB` поддерживают индексы столбцов, которые могут иметь величины `NULL`. В других случаях, во избежание ошибки, необходимо объявлять такие столбцы как `NOT NULL`.
- С помощью выражения `col_name(length)` можно указать индекс, для которого используется только часть столбца `CHAR` или `VARCHAR`. Это поможет сделать файл индексов намного меньше.
- Индексацию столбцов `BLOB` и `TEXT` поддерживают только таблицы с типом `MyISAM`. Назначая индекс столбцу с типом `BLOB` или `TEXT`, всегда НЕОБХОДИМО указывать длину этого индекса:
 

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```
- При использовании выражений `ORDER BY` или `GROUP BY` со столбцом типа `TEXT` или `BLOB` используются только первые `max_sort_length` байтов.
- В версии MySQL 3.23.23 и более поздних можно создавать также специальные индексы `FULLTEXT`. Они применяются для полнотекстового поиска. Эти индексы поддерживаются только таблицами типа `MyISAM` и они могут быть созданы только из столбцов `VARCHAR` и `TEXT`.

Индексирование всегда выполняется для всего столбца целиком, частичная индексация не поддерживается. Более подробно эта операция описана в разделе

- Выражения `FOREIGN KEY`, `CHECK` и `REFERENCES` фактически ничего не делают. Они введены только из соображений совместимости, чтобы облегчить перенос кода с других SQL-серверов и запускать приложения, создающие таблицы со ссылками
- Для каждого столбца `NULL` требуется один дополнительный бит, при этом величина столбца округляется в большую сторону до ближайшего байта.
- Максимальную длину записи в байтах можно вычислить следующим образом:
  - $\text{длина записи} = 1$
  - $+ (\text{сумма длин столбцов})$
  - $+ (\text{количество столбцов с допустимым NULL} + 7) / 8$
  - $+ (\text{количество столбцов с динамической длиной})$
- Опции `table_options` и `SELECT` реализованы только в версиях MySQL 3.23 и выше. Ниже представлены различные типы таблиц:

Тип таблицы	Описание
BDB или Berkeley_db	Таблицы с поддержкой транзакций и блокировкой страниц. See section <a href="#">7.6 Таблицы BDB или Berkeley_DB</a> .
HEAP	Данные для этой таблицы хранятся только в памяти. See section <a href="#">7.4 Таблицы HEAP</a> .
ISAM	Оригинальный обработчик таблиц. See section <a href="#">7.3 Таблицы ISAM</a> .
InnoDB	Таблицы с поддержкой транзакций и блокировкой строк. See section <a href="#">7.5 Таблицы InnoDB</a> .
MERGE	Набор таблиц MyISAM, используемый как одна таблица. See section <a href="#">7.2 Таблицы MERGE</a> .
MRG_MyISAM	Псевдоним для таблиц MERGE
MyISAM	Новый обработчик, обеспечивающий переносимость таблиц в бинарном виде, который заменяет ISAM. See section <a href="#">7.1 Таблицы MyISAM</a> .

- See section [7 Типы таблиц MySQL](#). Если задается тип таблицы, который не поддерживается данной версией, то MySQL выберет из возможных типов ближайший к указанному. Например, если задается `TYPE=BDB` и данный дистрибутив MySQL не поддерживает таблиц `BDB`, то вместо этого будет создана таблица `myISAM`. Другие табличные опции используются для оптимизации характеристик таблицы. Эти опции в большинстве случаев не требуют специальной установки. Данные опции работают с таблицами всех типов, если не указано иное:

Опция	Описание
AUTO_INCREMENT ENT	Следующая величина <code>AUTO_INCREMENT</code> , которую следует установить для данной таблицы ( <code>myISAM</code> ).
AVG_ROW_LENGTH NGTH	Приближенное значение средней длины строки для данной таблицы. Имеет смысл устанавливать только для обширных таблиц с записями переменной длины.
CHECKSUM	Следует установить в 1, чтобы в MySQL поддерживалась проверка контрольной суммы для всех строк (это делает таблицы немного более медленными при обновлении, но позволяет легче находить поврежденные



	таблицы) (MyISAM).
COMMENT	Комментарий для данной таблицы длиной 60 символов.
MAX_ROWS	Максимальное число строк, которые планируется хранить в данной таблице.
MIN_ROWS	Минимальное число строк, которые планируется хранить в данной таблице.
PACK_KEYS	Следует установить в 1 для получения меньшего индекса. Обычно это замедляет обновление и ускоряет чтение (MyISAM, ISAM). Установка в 0 отключит уплотнение ключей. При установке в DEFAULT (MySQL 4.0) обработчик таблиц будет уплотнять только длинные столбцы CHAR/VARCHAR.
PASSWORD	Шифрует файл '.frm' с помощью пароля. Эта опция не функционирует в стандартной версии MySQL.
DELAY_KEY_WRITE	Установка в 1 задерживает операции обновления таблицы ключей, пока не закроется указанная таблица (MyISAM).
ROW_FORMAT	Определяет, каким образом должны храниться строки. В настоящее время эта опция работает только с таблицами MyISAM, которые поддерживают форматы строк DYNAMIC и FIXED. See section <a href="#">7.1.2 Форматы таблиц MyISAM</a> .

- При использовании таблиц MyISAM MySQL вычисляет выражение `max_rows * avg_row_length`, чтобы определить, насколько велика будет результирующая таблица. Если не задана ни одна из вышеупомянутых опций, то максимальный размер таблицы будет составлять 4Гб (или 2Гб если данная операционная система поддерживает только таблицы величиной до 2Гб). Это делается для того, чтобы, если нет реальной необходимости в больших файлах, ограничить размеры указателей, что позволит сделать индексы меньше и быстрее. Если опция `PACK_KEYS` не используется, то по умолчанию уплотняются только строки, но не числа. При использовании `PACK_KEYS=1` числа тоже будут уплотняться. При уплотнении двоичных числовых ключей MySQL будет использовать сжатие префиксов. Это означает, что выгода от этого будет значительной только в случае большого количества одинаковых чисел. При сжатии префиксов для каждого ключа требуется один дополнительный байт, в котором указано, сколько байтов предыдущего ключа являются такими же, как и для следующего (следует учитывать, что указатель на строку хранится в порядке "старший-байт-в-начале" сразу после ключа - чтобы улучшить компрессию). Это означает, что при наличии нескольких одинаковых ключей в двух строках записи все последующие "аналогичные" ключи будут занимать только по 2 байта (включая указатель строки). Сравним: в обычном случае для хранения последующих ключей требуется `размер_хранения_ключа + размер_указателя` (обычно 4) байтов. С другой стороны, если все ключи абсолютно разные, каждый ключ будет занимать на 1 байт больше, если данный ключ не может иметь величину NULL (в этом случае уплотненный ключ будет храниться в том же байте, который используется для указания, что ключ равен NULL).
- Если после команды `CREATE` указывается команда `SELECT`, то MySQL создаст новые поля для всех элементов в данной команде `SELECT`. Например:
  - `mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,`
  - `PRIMARY KEY (a), KEY(b))`
  - `TYPE=MyISAM SELECT b,c FROM test2;`

Эта команда создаст таблицу MyISAM с тремя столбцами a, b и c. Отметим, что столбцы из команды `SELECT` присоединяются к таблице справа, а не перекрывают ее. Рассмотрим следующий пример:

```
mysql> SELECT * FROM foo;
+----+
| n |
+----+
| 1 |
+----+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM bar;
+-----+-----+
| m      | n |
+-----+-----+
| NULL   | 1 |
+-----+-----+
1 row in set (0.00 sec)
```

Каждая строка в таблице `foo` вносится в таблицу `bar` со своим значением из `foo`, при этом в новые столбцы в таблице `bar` записываются величины, заданные по умолчанию. Команда `CREATE TABLE ... SELECT` не создает автоматически каких-либо индексов. Это сделано преднамеренно, чтобы команда была настолько гибкой, насколько возможно. Чтобы иметь индексы в созданной таблице, необходимо указать их перед данной командой `SELECT`:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Если возникает ошибка при копировании данных в таблицу, то они будут автоматически удалены. Чтобы обеспечить возможность использовать для восстановления таблиц журнал обновлений/двоичный журнал, в MySQL во время выполнения команды `CREATE TABLE ... SELECT` не разрешены параллельные вставки.

- Воспользовавшись опцией `RAID_TYPE`, можно разбить файл данных `MyISAM` на участки с тем, чтобы преодолеть 2Гб/4Гб лимит файловой системы под управлением ОС, не поддерживающих большие файлы. Разбиение не касается файла индексов. Следует учесть, что для файловых систем, которые поддерживают большие файлы, эта опция не рекомендуется! Для получения более высокой скорости ввода-вывода можно разместить RAID-директории на различных физических дисках. `RAID_TYPE` будет работать под любой операционной системой, если конфигурация MySQL выполнена с параметром `--with-raid`. В настоящее время для опции `RAID_TYPE` возможен только параметр `STRIPED` (1 и `RAID0` являются псевдонимами для него). Если указывается `RAID_TYPE=STRIPED` для таблицы `MyISAM`, то `MyISAM` создаст поддиректории `RAID_CHUNKS` с именами ``00'`, ``01'`, ``02'` в директории базы данных. В каждой из этих директорий `MyISAM` создаст файл ``table_name.MYD'`. При записи данных в файл данных обработчик RAID установит соответствие первых `RAID_CHUNKSIZE*1024` байтов первому упомянутому файлу, следующих `RAID_CHUNKSIZE*1024` байтов - следующему файлу и так далее.
- Опция `UNION` применяется, если необходимо использовать совокупность идентичных таблиц как одну таблицу. Она работает только с таблицами `MERGE`. иметь привилегии `SELECT`, `UPDATE` и `DELETE`. Все сопоставляемые таблицы должны принадлежать той же базе данных, что и таблица `MERGE`.
- Для внесения данных в таблицу `MERGE` необходимо указать с помощью `INSERT_METHOD`, в какую таблицу данная строка должна быть внесена.. Эта опция была введена в MySQL 4.0.0.

- В созданной таблице ключ PRIMARY будет помещен первым, за ним все ключи UNIQUE и затем простые ключи. Это помогает оптимизатору MySQL определять приоритеты используемых ключей, а также более быстро определять сдублированные ключи UNIQUE.
- Используя опции DATA DIRECTORY="каталог" или INDEX DIRECTORY="каталог", можно указать, где обработчик таблицы должен помещать свои табличные и индексные файлы. Следует учитывать, что указываемый параметр directory должен представлять собой полный путь к требуемому каталогу (а не относительный путь). Данные опции работают только для таблиц MyISAM в версии MySQL 4.0, если при этом не используется опция --skip-symlink.

## Синтаксис оператора ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]
```

alter\_specification:

```

    ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
или
    ADD [COLUMN] (create_definition, create_definition,...)
или
    ADD INDEX [index_name] (index_col_name,...)
или
    ADD PRIMARY KEY (index_col_name,...)
или
    ADD UNIQUE [index_name] (index_col_name,...)
или
    ADD FULLTEXT [index_name] (index_col_name,...)
или
    ADD [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)
        [reference_definition]
или
    ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
или
    CHANGE [COLUMN] old_col_name create_definition
        [FIRST | AFTER column_name]
или
    MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]
или
    DROP [COLUMN] col_name
или
    DROP PRIMARY KEY
или
    DROP INDEX index_name
или
    DISABLE KEYS
или
    ENABLE KEYS
или
    RENAME [TO] new_tbl_name
или
    ORDER BY col
или
    table_options

```

Оператор ALTER TABLE обеспечивает возможность изменять структуру существующей таблицы. Например, можно добавлять или удалять столбцы, создавать или уничтожать индексы или переименовывать столбцы либо саму таблицу. Можно также изменять комментарий для таблицы и ее тип.

Если оператор ALTER TABLE используется для изменения определения типа столбца, но DESCRIBE tbl\_name показывает, что столбец не изменился, то, возможно, MySQL игнорирует данную модификацию по одной из причин, описанных в разделе. Например, при попытке изменить столбец VARCHAR на CHAR MySQL будет продолжать использовать VARCHAR, если данная таблица содержит другие столбцы с переменной длиной.

Оператор ALTER TABLE во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая переименовывается. Так делается для того, чтобы в новую таблицу автоматически попадали все обновления кроме неудавшихся. Во время выполнения ALTER TABLE исходная таблица доступна для чтения другими клиентами. Операции обновления и записи в этой таблице приостанавливаются, пока не будет готова новая таблица.

Следует отметить, что при использовании любой другой опции для ALTER TABLE кроме RENAME, MySQL всегда будет создавать временную таблицу, даже если данные, строго говоря, и не нуждаются в копировании (например, при изменении имени столбца). Мы планируем исправить это в будущем, однако, поскольку ALTER TABLE выполняется не так часто, мы (разработчики MySQL) не считаем эту задачу первоочередной. Для таблиц myISAM можно увеличить скорость воссоздания индексной части (что является наиболее медленной частью в процессе восстановления таблицы) путем установки переменной myisam\_sort\_buffer\_size достаточно большого значения.

- Для использования оператора ALTER TABLE необходимы привилегии ALTER, INSERT и CREATE для данной таблицы.
- Опция IGNORE является расширением MySQL по отношению к ANSI SQL92. Она управляет работой ALTER TABLE при наличии дубликатов уникальных ключей в новой таблице. Если опция IGNORE не задана, то для данной копии процесс прерывается и происходит откат назад. Если IGNORE указывается, тогда для строк с дубликатами уникальных ключей только первая строка используется, а остальные удаляются.
- Можно запустить несколько выражений ADD, ALTER, DROP и CHANGE в одной команде ALTER TABLE. Это является расширением MySQL по отношению к ANSI SQL92, где допускается только одно выражение из упомянутых в одной команде ALTER TABLE.
- Опции CHANGE col\_name, DROP col\_name и DROP INDEX также являются расширениями MySQL по отношению к ANSI SQL92.
- Опция MODIFY представляет собой расширение Oracle для команды ALTER TABLE.
- Необязательное слово COLUMN представляет собой ``белый шум" и может быть опущено.
- При использовании ALTER TABLE имя\_таблицы RENAME TO новое\_имя без каких-либо других опций MySQL просто переименовывает файлы, соответствующие заданной таблице. В этом случае нет необходимости создавать временную таблицу. .
- В выражении create\_definition для ADD и CHANGE используется тот же синтаксис, что и для CREATE TABLE. Следует учитывать, что этот синтаксис включает имя столбца, а не просто его тип.
- Столбец можно переименовывать, используя выражение CHANGE имя\_столбца create\_definition. Чтобы сделать это, необходимо указать старое и новое имена столбца и его тип в настоящее время. Например, чтобы переименовать столбец INTEGER из a в b, можно сделать следующее:
- ```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

При изменении типа столбца, но не его имени синтаксис выражения CHANGE все равно требует указания обоих имен столбца, даже если они одинаковы. Например:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

Однако начиная с версии MySQL 3.22.16a можно также использовать выражение MODIFY для изменения типа столбца без переименовывания его:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- При использовании CHANGE или MODIFY для того, чтобы уменьшить длину столбца, по части которого построен индекс (например, индекс по первым 10 символам столбца VARCHAR), нельзя сделать столбец короче, чем число проиндексированных символов.
- При изменении типа столбца с использованием CHANGE или MODIFY MySQL пытается преобразовать данные в новый тип как можно корректнее.

- В версии MySQL 3.22 и более поздних можно использовать `FIRST` или `ADD ... AFTER имя_столбца` для добавления столбца на заданную позицию внутри табличной строки. По умолчанию столбец добавляется в конце. Начиная с версии MySQL 4.0.1, можно также использовать ключевые слова `FIRST` и `AFTER` в опциях `CHANGE` или `MODIFY`.
- Опция `ALTER COLUMN` задает для столбца новое значение по умолчанию или удаляет старое. Если старое значение по умолчанию удаляется и данный столбец может принимать значение `NULL`, то новое значение по умолчанию будет `NULL`. Если столбец не может быть `NULL`, то MySQL назначает значение по умолчанию.
- Опция `DROP INDEX` удаляет индекс. Это является расширением MySQL по отношению к ANSI SQL92.
- Если столбцы удаляются из таблицы, то эти столбцы удаляются также и из любого индекса, в который они входят как часть. Если все столбцы, составляющие индекс, удаляются, то данный индекс также удаляется.
- Если таблица содержит только один столбец, то этот столбец не может быть удален. Вместо этого можно удалить данную таблицу, используя команду `DROP TABLE`.
- Опция `DROP PRIMARY KEY` удаляет первичный индекс. Если такого индекса в данной таблице не существует, то удаляется первый индекс `UNIQUE` в этой таблице. (MySQL отмечает первый уникальный ключ `UNIQUE` как первичный ключ `PRIMARY KEY`, если никакой другой первичный ключ `PRIMARY KEY` не был явно указан). При добавлении `UNIQUE INDEX` или `PRIMARY KEY` в таблицу они хранятся перед остальными неуникальными ключами, чтобы можно было определить дублирующиеся ключи как можно раньше.
- Опция `ORDER BY` позволяет создавать новую таблицу со строками, размещенными в заданном порядке. Следует учитывать, что созданная таблица не будет сохранять этот порядок строк после операций вставки и удаления. В некоторых случаях такая возможность может облегчить операцию сортировки в MySQL, если таблица имеет такое расположение столбцов, которое вы хотели бы иметь в дальнейшем. Эта опция в основном полезна, если заранее известен определенный порядок, в котором преимущественно будут запрашиваться строки. Использование данной опции после значительных преобразований таблицы дает возможность получить более высокую производительность.
- При использовании команды `ALTER TABLE` для таблиц `myISAM` все неуникальные индексы создаются в отдельном пакете (подобно `REPAIR`). Благодаря этому команда `ALTER TABLE` при наличии нескольких индексов будет работать быстрее.
- Начиная с MySQL 4.0, вышеуказанная возможность может быть активизирована явным образом. Команда `ALTER TABLE ... DISABLE KEYS` блокирует в MySQL обновление неуникальных индексов для таблиц `myISAM`. После этого можно применить команду `ALTER TABLE ... ENABLE KEYS` для воссоздания недостающих индексов. Так как MySQL делает это с помощью специального алгоритма, который намного быстрее в сравнении со вставкой ключей один за другим, блокировка ключей может дать существенное ускорение на больших массивах вставок.
- Применяя функцию с API `mysql_info()`, можно определить, сколько записей было скопировано, а также (при использовании `IGNORE`) - сколько записей было удалено из-за дублирования значений уникальных ключей.
- Выражения `FOREIGN KEY`, `CHECK` и `REFERENCES` фактически ничего не делают. Они введены только из соображений совместимости, чтобы облегчить перенос кода с других серверов SQL и запуск приложений, создающих таблицы со ссылками.
- Ниже приводятся примеры, показывающие некоторые случаи употребления команды `ALTER TABLE`. Пример начинается с таблицы `t1`, которая создается следующим образом:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

Для того чтобы переименовать таблицу из t1 в t2:

```
mysql> ALTER TABLE t1 RENAME t2;
```

Для того чтобы изменить тип столбца c INTEGER на TINYINT NOT NULL (оставляя имя прежним) и изменить тип столбца b с CHAR(10) на CHAR(20) с переименованием его с b на c:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

Для того чтобы добавить новый столбец TIMESTAMP с именем d:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

Для того чтобы добавить индекс к столбцу d и сделать столбец a первичным ключом:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

Для того чтобы удалить столбец c:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

Для того чтобы добавить новый числовой столбец AUTO\_INCREMENT с именем c:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,  
      ADD INDEX (c);
```

Заметьте, что столбец c индексируется, так как столбцы AUTO\_INCREMENT должны быть индексированы, кроме того, столбец c объявляется как NOT NULL, поскольку индексированные столбцы не могут быть NULL.

При добавлении столбца AUTO\_INCREMENT значения этого столбца автоматически заполняются последовательными номерами (при добавлении записей). Первый номер последовательности можно установить путем выполнения команды SET INSERT\_ID=# перед ALTER TABLE или использования табличной опции AUTO\_INCREMENT = #.

## 2. Выполнение лабораторной работы

По аналогии с примерами, приведенными в п. 1 и создать базу данных с разработанной физической моделью. Продемонстрировать умение добавить и удалить столбец командой alter table

## 3. Содержание отчета

Содержание отчета:

- Текст задания.
- физическую модель БД.
- скрипт создания таблиц базы данных на SQL;

— скрипт изменения структуры таблиц базы данных на SQL.

#### 4. Варианты заданий

Варианты заданий приведены в Приложении 2.

### Лабораторная работа №3 Заполнение таблиц и модификация данных

#### 1. Теоретическая часть

В данной работе рассматриваются следующие операторы языка манипулирования данными (DML):

Insert – вставки данных;

Update – изменения данных;

Delete – удаление данных

Merge – слияние данных таблиц (действие 3 предыдущих команд в одной)

#### Синтаксис оператора INSERT

Выделяют 3 различных синтаксиса

1) вставляет строки в соответствии с точно указанными в команде значениями. (форма со списком из нескольких значений поддерживается в версии MySQL 3.22.5 и более поздних)

```
INSERT [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      VALUES (expression,...), (...),...
```

2) вставляет строки, выбранные из другой таблицы или таблиц

```
INSERT      [INTO] tbl_name [(col_name,...)]
      SELECT ...
```

3) синтаксис вставки строк, который поддерживается в версии MySQL 3.22.10 и более поздних

```
INSERT [IGNORE] [INTO] tbl_name
      SET col_name=expression, col_name=expression, ...
```

В каждом из них после слова INSERT может стоять один из 3 модификаторов:

- LOW\_PRIORITY- если таблицу использует другой процесс, вставка будет отложена до её освобождения
- DELAYED использовалось до версии 5.6, аналог LOW\_PRIORITY
- HIGH\_PRIORITY- в противоположность LOW\_PRIORITY – немедленная вставка вне зависимости от занятости таблицы

HIGH\_PRIORITY и LOW\_PRIORITY работают только на таблицах, где блокировка уровня таблицы (MyISAM, MEMORY и MERGE). Следует отметить, что указатель LOW\_PRIORITY обычно не используется с таблицами MyISAM, поскольку при его указании становятся невозможными параллельные вставки.

tbl\_name задает таблицу, в которую должны быть внесены строки. Столбцы, для которых заданы величины в команде, указываются в списке имен столбцов или в части SET:

- Если не указан список столбцов для INSERT ... VALUES или INSERT ... SELECT, то величины для всех столбцов должны быть определены в списке VALUES () или в результате работы SELECT. Если порядок столбцов в таблице неизвестен, для его получения можно использовать DESCRIBE tbl\_name.
- Любой столбец, для которого явно не указано значение, будет установлен в свое значение по умолчанию.
- Выражение expression может относиться к любому столбцу, который ранее был внесен в список значений. Например, можно указать следующее:
- `mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);`

Но нельзя указать:

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

- Если в команде INSERT со строками, имеющими много значений, указывается ключевое слово IGNORE, то все строки, имеющие дублирующиеся ключи PRIMARY или UNIQUE в этой таблице, будут проигнорированы и не будут внесены.

Предупреждения возникают при выполнении любого из следующих условий:

- Внесение NULL в столбец, который был объявлен, как NOT NULL. Данный столбец устанавливается в значение, заданное по умолчанию.
- Установка числового столбца в значение, лежащее за пределами его допустимого диапазона. Данная величина усекается до соответствующей конечной точки этого диапазона.
- Занесение в числовой столбец такой величины, как '10.34 a'. Конечные данные удаляются и вносится только оставшаяся числовая часть. Если величина вовсе не имеет смысла как число, то столбец устанавливается в 0.
- Внесение в столбцы типа CHAR, VARCHAR, TEXT или BLOB строки, превосходящей максимальную длину столбца. Данная величина усекается до максимальной длины столбца.
- Внесение в столбец даты или времени строки, недопустимой для данного типа столбца. Этот столбец устанавливается в нулевую величину, соответствующую данному типу.

## Синтаксис оператора UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2, ...]
[WHERE where_definition]
[LIMIT #]
```

Оператор UPDATE обновляет столбцы в соответствии с их новыми значениями в строках существующей таблицы. В выражении SET указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В выражении WHERE, если оно присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово LOW\_PRIORITY, то выполнение данной команды UPDATE задерживается до тех пор, пока другие клиенты не завершат чтение этой таблицы.

Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.



Если доступ к столбцу из указанного выражения осуществляется по аргументу `tbl_name`, то команда `UPDATE` использует для этого столбца его текущее значение. Например, следующая команда устанавливает столбец `age` в значение, на единицу большее его текущей величины:

```
mysql> UPDATE persondata SET age=age+1;
```

Значения команда `UPDATE` присваивает слева направо. Например, следующая команда дублирует столбец `age`, затем инкрементирует его:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

Если столбец устанавливается в его текущее значение, то MySQL замечает это и не обновляет его.

Команда `UPDATE` возвращает количество фактически измененных строк. В версии MySQL 3.22 и более поздних функция `C API mysql_info()` возвращает количество строк, которые были найдены и обновлены, и количество предупреждений, имевших место при выполнении `UPDATE`.

В версии MySQL 3.23 можно использовать `LIMIT #`, чтобы убедиться, что было изменено только заданное количество строк.

## Синтаксис оператора `DELETE`

```
DELETE [LOW_PRIORITY | QUICK] FROM table_name
      [WHERE where_definition]
      [ORDER BY ...]
      [LIMIT rows]
```

или

```
DELETE [LOW_PRIORITY | QUICK] table_name[.*] [,table_name[.*] ...]
      FROM table-references
      [WHERE where_definition]
```

оили

```
DELETE [LOW_PRIORITY | QUICK]
      FROM table_name[.*], [table_name[.*] ...]
      USING table-references
      [WHERE where_definition]
```

Оператор `DELETE` удаляет из таблицы `table_name` строки, удовлетворяющие заданным в `where_definition` условиям, и возвращает число удаленных записей.

Если оператор `DELETE` запускается без определения `WHERE`, то удаляются все строки. При работе в режиме `AUTOCOMMIT` это будет аналогично использованию оператора `TRUNCATE`. В MySQL 3.23 оператор `DELETE` без определения `WHERE` возвратит ноль как число удаленных записей.

Если действительно необходимо знать число удаленных записей при удалении всех строк, и если допустимы потери в скорости, то можно использовать команду `DELETE` в следующей форме:

```
mysql> DELETE FROM table_name WHERE 1>0;
```

Следует учитывать, что эта форма работает намного медленнее, чем `DELETE FROM table_name` без выражения `WHERE`, поскольку строки удаляются поочередно по одной.

Если указано ключевое слово `LOW_PRIORITY`, выполнение данной команды `DELETE` будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы.

Если задан параметр `QUICK`, то обработчик таблицы при выполнении удаления не будет объединять индексы - в некоторых случаях это может ускорить данную операцию.

В таблицах `MyISAM` удаленные записи сохраняются в связанном списке, а последующие операции `INSERT` повторно используют места, где располагались удаленные записи. Чтобы вернуть неиспользуемое пространство и уменьшить размер файлов, можно применить команду `OPTIMIZE TABLE` или утилиту `myisamchk` для реорганизации таблиц. Команда `OPTIMIZE TABLE` проще, но утилита `myisamchk` работает быстрее.

Первый из числа приведенных в начале данного раздела многотабличный формат команды `DELETE` поддерживается, начиная с `MySQL 4.0.0`. Второй многотабличный формат поддерживается, начиная с `MySQL 4.0.2`.

Идея заключается в том, что удаляются только совпадающие строки из таблиц, перечисленных перед выражениями `FROM` или `USING`. Это позволяет удалять одновременно строки из нескольких таблиц, а также использовать для поиска дополнительные таблицы.

Символы `.` `*` после имен таблиц требуются только для совместимости с `Access`:

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

или

```
DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

В предыдущем случае просто удалены совпадающие строки из таблиц `t1` и `t2`.

Выражение `ORDER BY` и использование нескольких таблиц в команде `DELETE` поддерживается в `MySQL 4.0`.

Если применяется выражение `ORDER BY`, то строки будут удалены в указанном порядке. В действительности это выражение полезно только в сочетании с `LIMIT`. Например:

```
DELETE FROM somelog
      WHERE user = 'jcole'
      ORDER BY timestamp
      LIMIT 1
```

Данный оператор удалит самую старую запись (по `timestamp`), в которой строка соответствует указанной в выражении `WHERE`.

Специфическая для `MySQL` опция `LIMIT` для команды `DELETE` указывает серверу максимальное количество строк, которые следует удалить до возврата управления клиенту. Эта опция может использоваться для гарантии того, что данная команда `DELETE` не потребует слишком много времени для выполнения. Можно просто повторять команду `DELETE` до тех пор, пока количество удаленных строк меньше, чем величина `LIMIT`.

## Слияние данных ( оператор `MERGE`)

Оператор MERGE в стандарте с SQL:2003, улучшен в SQL:2008, удаление введено в оператор с SQL:2011)

Оператор MERGE не поддерживается MySQL, поэтому синтаксис при выполнении работы на mysql привести надо, а запускать запрос -нет

Выполняет операции вставки, обновления или удаления для целевой таблицы на основе результатов соединения с исходной таблицей. Например, можно синхронизировать две таблицы путем вставки, обновления или удаления строк в одной таблице на основании отличий, найденных в другой таблице.

Синтаксис MS SQL

MERGE

```
[ TOP ( expression ) [ PERCENT ] ]  
[ INTO ] <target_table> [ WITH ( <merge_hint> ) ] [ [ AS ] table_alias ]  
USING <table_source>  
ON <merge_search_condition>  
[ WHEN MATCHED [ AND <clause_search_condition> ]  
  THEN <merge_matched> ] [ ...n ]  
[ WHEN NOT MATCHED [ BY TARGET ] [ AND <clause_search_condition> ]  
  THEN <merge_not_matched> ]  
[ WHEN NOT MATCHED BY SOURCE [ AND <clause_search_condition> ]  
  THEN <merge_matched> ] [ ...n ]  
[ <output_clause> ]  
[ OPTION ( <query_hint> [ ,...n ] ) ]  
;
```

Аргументы

TOP ( expression ) [ PERCENT ]

Указывает количество или процент строк, которые подпадают под эту операцию. Аргумент expression может быть либо числом, либо процентной долей строк. Строки, на которые ссылается выражение TOP, не расположены в определенном порядке.

Предложение TOP применяется после соединения всей исходной таблицы и всей целевой таблицы и удаления соединенных строк, которые не рассматриваются как предназначенные для выполнения операций вставки, обновления или удаления. Предложение TOP дополнительно сокращает количество соединенных строк до указанного значения, а затем к оставшимся соединенным строкам применяются операции вставки, обновления или удаления без учета порядка. Иными словами, порядок, в котором строки подвергаются операциям, определенным в предложениях

WHEN, не задан. Например, указание значения TOP (10) затрагивает 10 строк. Из них 7 могут быть обновлены и 3 вставлены или 1 может быть удалена, 5 обновлено и 4 вставлено и т. д.

Инструкция MERGE выполняет полный просмотр исходной и целевой таблиц, поэтому при использовании предложения TOP для изменения большой таблицы путем создания нескольких пакетов производительность ввода-вывода может снизиться. В этом случае необходимо обеспечить, чтобы во всех подряд идущих пакетах осуществлялась обработка новых строк.

target\_table

Таблица или представление, с которыми выполняется сопоставление строк данных из таблицы <table\_source> по условию <clause\_search\_condition>. Параметр target\_table является целевым объектом любых операций вставки, обновления или удаления, указанных предложениями WHEN инструкции MERGE.

USING <table\_source>

Указывается источник данных, который сопоставляется со строками данных в таблице target\_table на основе условия <merge\_search condition>. Результат этого совпадения обуславливает действия, которые выполняются предложениями WHEN инструкции MERGE. Аргумент <table\_source> может быть удаленной таблицей или производной таблицей, которая обращается к удаленным таблицам.

ON <merge\_search\_condition>

Указываются условия, при которых таблица <table\_source> соединяется с таблицей target\_table для сопоставления. Необходимо указать столбцы целевой таблицы, которые сравниваются с соответствующим столбцом исходной таблицы.

WHEN MATCHED THEN <merge\_matched>

Указывается, что все строки target\_table, которые соответствуют строкам, возвращенным <table\_source> ON <merge\_search\_condition>, и удовлетворяют дополнительным условиям поиска, обновляются или удаляются в соответствии с предложением <merge\_matched>.

Инструкция MERGE может иметь не больше двух предложений WHEN MATCHED. Если указаны два предложения, первое предложение должно сопровождаться предложением AND <search\_condition>. Для любой строки второе предложение WHEN MATCHED применяется только в тех случаях, если не применяется первое. Если имеются два предложения WHEN MATCHED, одно должно указывать действие UPDATE, а другое — действие DELETE. Если действие UPDATE указано в предложении <merge\_matched> и более одной строки в <table\_source> соответствует строке в target\_table на основе <merge\_search\_condition>, то SQL Server возвращает ошибку. Инструкцию MERGE нельзя использовать для обновления одной строки более одного раза, а также использовать для обновления и удаления одной и той же строки.

WHEN NOT MATCHED [ BY TARGET ] THEN <merge\_not\_matched>

Указывает, что строка вставлена в таблицу target\_table для каждой строки, возвращенной выражением <table\_source> ON <merge\_search\_condition>, которая не

соответствует строке в таблице `target_table`, но удовлетворяет дополнительному условию поиска (при наличии такового). Значения для вставки указываются с помощью предложения `<merge_not_matched>`. Инструкция `MERGE` может иметь только одно предложение `WHEN NOT MATCHED`.

`WHEN NOT MATCHED BY SOURCE THEN <merge_matched>`

Указывается, что все строки `target_table`, которые не соответствуют строкам, возвращенным `<table_source> ON <merge_search_condition>`, и удовлетворяют дополнительным условиям поиска, обновляются или удаляются в соответствии с предложением `<merge_matched>`.

Инструкция `MERGE` может иметь не более двух предложений `WHEN NOT MATCHED BY SOURCE`. Если указаны два предложения, то первое предложение должно сопровождаться предложением `AND <clause_search_condition>`. Для любой выбранной строки второе предложение `WHEN NOT MATCHED BY SOURCE` применяется только в тех случаях, если не применяется первое. Если имеется два предложения `WHEN NOT MATCHED BY SOURCE`, то одно должно указывать действие `UPDATE`, а другое — действие `DELETE`. В условии `<clause_search_condition>` можно ссылаться только на столбцы целевой таблицы.

`<merge_matched>`

Указывает действие обновления или удаления, которое применяется ко всем строкам `target_table`, не соответствующим строкам, которые возвращаются `<table_source> ON <merge_search_condition>` и удовлетворяют каким-либо дополнительным условиям поиска.

`UPDATE SET <set_clause>`

Указывается список имен столбцов или переменных, которые необходимо обновить в целевой таблице, и значений, которые необходимо использовать для их обновления.

`<merge_not_matched>`

Указываются значения для вставки в целевую таблицу.

`(column_list)`

Список, состоящий из одного или нескольких столбцов целевой таблицы, в которые вставляются данные. Столбцы необходимо указывать в виде однокомпонентного имени, так как в противном случае инструкция `MERGE` возвращает ошибку. Список `column_list` должен быть заключен в круглые скобки, а его элементы должны разделяться запятыми.

`VALUES ( values_list)`

Список с разделителями-запятыми констант, переменных или выражений, которые возвращают значения для вставки в целевую таблицу. Выражения не могут содержать инструкцию `EXECUTE`.

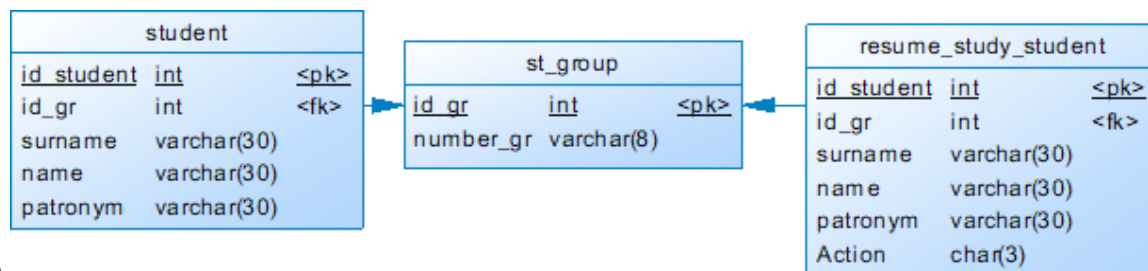
`DEFAULT VALUES`

Заполняет вставленную строку значениями по умолчанию, определенными для каждого столбца.

<search condition>

Указываются условия поиска, используемые для указания <merge\_search\_condition> или <clause\_search\_condition>.

Пример сольем таблицу студентов и восстановившихся студентов (результат в таблице



Student)

```
MERGE student trg -- таблица приемник
USING new_study_student src -- таблица источник
ON trg.id_student = src.id_student -- условие слияния

-- 1. Строка есть в trg (student), но нет сопоставления с
-- строкой из src(new)
WHEN NOT MATCHED BY SOURCE THEN DELETE

-- 2. Есть сопоставление строки trg со строкой из
-- источника src
WHEN MATCHED THEN
UPDATE SET trg.id_gr=src.id_gr, trg.surname= src.surname, trg.name=src.name, trg.patronym
=src.patronym

-- 3. Строка не найдена в trg (student), но есть в src (new)
WHEN NOT MATCHED BY TARGET THEN
INSERT(id_student,id_gr, surname, name, patronym) VALUES (src.id_student, src.id_gr, src.surname,
src.name, src.patronym)
```

Postgres с 15 версии тоже поддерживается MERGE, но без NOT MATCHED BY SOURCE

С синтаксисов приведенным ниже

```

[ WITH with_query [, ...] ]
MERGE INTO target_table_name [ [ AS ] target_alias ]
USING data_source ON join_condition
when_clause [...]

where data_source is:

{ source_table_name | ( source_query ) } [ [ AS ] source_alias ]

and when_clause is:

{ WHEN MATCHED [ AND condition ] THEN { merge_update | merge_delete | DO NOTHING } |
  WHEN NOT MATCHED [ AND condition ] THEN { merge_insert | DO NOTHING } }

and merge_insert is:

INSERT [( column_name [, ...] )]
[ OVERRIDING { SYSTEM | USER } VALUE ]
{ VALUES ( { expression | DEFAULT } [, ...] ) | DEFAULT VALUES }

and merge_update is:

UPDATE SET { column_name = { expression | DEFAULT } |
             ( column_name [, ...] ) = ( { expression | DEFAULT } [, ...] ) } [, ...]

and merge_delete is:

DELETE

```

## 2. Выполнение лабораторной работы

По аналогии с примерами, приведенными в п. 1 Выполнить вставку тестовых данных в таблицы, созданные в ходе выполнения лабораторной работы 2. В строках, вставляемых в таблицы, должны быть данные как удовлетворяющие, так и не удовлетворяющие условиям запросов, приведенных в варианте задания. Необходимо привести свои пример использования оператором *update* и *delete* и *merge*

## 3. Содержание отчета

Содержание отчета:

— Текст задания.

- физическую модель БД.
- наборы данных, содержащихся в таблицах БД;
- Таблицы тестовых данных вида:

| Текст запроса                                      | данные удовлетворяющие условиям запросов | данные не удовлетворяющие условиям запросов |
|----------------------------------------------------|------------------------------------------|---------------------------------------------|
| а. станции в названии которых есть слово «площадь» | Таблица1 (станция)<br>Площадь Ленина     | Таблица1 (станция)<br>Московские ворота     |

- примеры использования insert, update, delete и merge;
- скрипт полного заполнения базы (удобно после использовать для быстрого развертывания БД на другой машине).

#### 4. Варианты заданий

Варианты заданий приведены в Приложении 2.

### Лабораторная работа №4 Разработка SQL запросов: виды соединений и шаблоны

#### 1. Теоретическая часть

##### Оператор выборки в языке SQL

Назначение оператора SELECT состоит в выборке и отображении данных одной или более таблиц базы данных. Это исключительно мощный оператор, способный выполнять действия, эквивалентные операторам реляционной алгебры выборки, проекции и соединения, причем в пределах единственной выполняемой команды. Оператор SELECT является чаще всего используемой командой языка SQL. Общий формат оператора SELECT имеет следующий вид:

```
SELECT [DISTINCT | ALL] .{ *| [columnExpression [AS newName]] [ , ... ] }
FROM TableName [alias] [ , ... ]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]
```

Слово distinct позволяет убрать дублирующиеся строки из результата запроса.

Здесь параметр *columnExpression* представляет собой имя столбца или выражение из нескольких имен.

*columnExpression* может включать только следующие типы элементов:

- имена столбцов;



- агрегирующие функции;
- константы;
- выражения, включающие комбинации перечисленных выше элементов.

В противоположность списку столбцов \* обозначает возврат всех столбцов всех таблиц запроса.

Параметр *TableName* является именем существующей в базе данных таблицы (или представления), к которой необходимо получить доступ. Необязательный параметр *alias* — это сокращение, псевдоним, устанавливаемое для имени таблицы *TableName*. Обработка элементов оператора SELECT выполняется в следующей последовательности.

- FROM. Определяются имена используемой таблицы или нескольких таблиц, перечисленные через запятую. Также в качестве источника данных в этом разделе могут быть таблицы связанные различными видами соединения, представления и другие запросы (смотри запросы с подзапросами)
- WHERE. Выполняется фильтрация строк объекта в соответствии с заданными условиями.
- GROUP BY. Образуются группы строк, имеющих одно и то же значение в указанном столбце (использовать только с агрегатными функциями).
- HAVING. Фильтруются группы строк объекта в соответствии с указанным условием, относящимся к результату агрегатной функции.
- SELECT. Устанавливается, какие столбцы должны присутствовать в выходных данных.
- ORDER BY. Определяется упорядоченность результатов выполнения оператора.

При этом каждому полю в списке сортировки может быть приписано ASC (по возрастанию — это параметр сортировки поля по умолчанию) или DESC (по убыванию)

Порядок конструкций в операторе SELECT *не может* быть изменен. Только две конструкции оператора — SELECT и FROM — являются обязательными, все остальные конструкции могут быть опущены.

### Условия в конструкции WHERE

(применимы как к оператору SELECT, так и UPDATE или INSERT)

В приведенных выше примерах в результате выполнения операторов SELECT выбирались все строки указанной таблицы. Однако очень часто требуется тем или иным образом ограничить набор строк, помещаемых в результирующую таблицу запроса. Это достигается с помощью указания в запросе конструкции WHERE. Она состоит из ключевого слова WHERE, за которым следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска (или *предикатов*, если пользоваться терминологией ISO).

- Сравнение. Сравниваются результаты вычисления одного выражения с результатами вычисления другого выражения. (<,>,<=,>=)
- Диапазон. Проверяется, попадает ли результат вычисления выражения в заданный диапазон значений. (имя\_поля BETWEEN знач\_1 AND знач\_2)
- Принадлежность к множеству. Проверяется, принадлежит ли результат вычисления выражения к заданному множеству значений. (имя\_поля in (знач\_1,знач\_2..., знач\_n))
- Значение NULL. Проверяется, содержит ли данный столбец NULL (неопределенное значение) (имя\_поля IS NULL).
- Соответствие шаблону. Проверяется, отвечает ли некоторое строковое значение заданному шаблону. (имя\_поля LIKE шаблон)

В

- %. Символ процента представляет любую последовательность из нуля или более символов (поэтому часто именуется также *подстановочным символом*). (В ACCESS вместо него используется \*, но при вызове из программы через ADO или DAO должен быть все равно %)

- \_. Символ подчеркивания представляет любой отдельный символ. . (В ACCESS вместо него используется ?, но при вызове из программы через ADO или DAO должно быть все равно \_)

Все остальные символы в шаблоне представляют сами себя.

Пример

- address LIKE 'Ш%'. Этот шаблон означает, что первый символ значения обязательно должен быть символом Ш, а все остальные символы не представляют интереса и не проверяются.

- address LIKE 'Ш\_ \_ \_'. Этот шаблон означает, что значение должно иметь длину, равную строго четырем символам, причем первым символом обязательно должен быть символ 'Ш'. (пробелы между подчеркиваниями только для визуализации – их нет в шаблоне)

- address LIKE '%ич'. Этот шаблон определяет любую последовательность символов длиной не менее двух символов, причем последними символами обязательно должны быть символы ич. (поиск мужчин по отчеству)

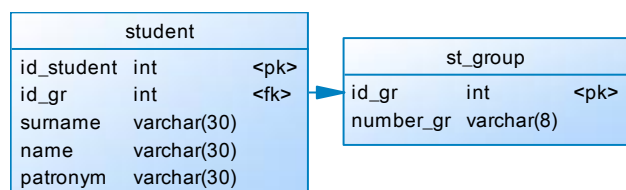
- address LIKE '%слово%'. Этот шаблон означает, что нас интересует любая последовательность символов, включающая подстроку «слово»;

- address NOT LIKE 'Ш%'. Этот шаблон указывает, что требуются любые строки, которые не начинаются с символа Ш.

Если требуемая строка должна включать также служебный символ, обычно применяемый в качестве символа подстановки, то следует определить с помощью конструкции ESCAPE "маскирующий" символ, который указывает, что следующий за ним символ больше не имеет специального значения, и поместить его перед символом подстановки. Например, для проверки значений на соответствие литеральной строке 45% можно воспользоваться таким предикатом: LIKE '15#%' ESCAPE '#'

Пример

Представим себе таблицы группа и студент.



Выберем всех студентов группы Z4432K

```
select id_student, student.id_gr, surname, name, patronym from student, st_group where student.id_gr =st_group.id_gr and number_gr='Z4432K'
```

## Виды соединений в языке SQL

Виды соединений в SQL:

- Декартово произведение CROSS JOIN
- Внутреннее соединение INNER JOIN (часто просто JOIN)
- Внешние соединения:
- Левое соединение LEFT JOIN
- Правое соединение RIGHT JOIN
- Полное (внешнее) соединение OUTER JOIN

Соединение является подмножеством более общей комбинации данных двух таблиц, называемой *декартовым произведением*. Декартово произведение двух таблиц представляет собой другую таблицу, состоящую из всех возможных пар строк, входящих в состав обеих таблиц. Набор столбцов результирующей таблицы представляет собой все столбцы первой таблицы, за которыми следуют все столбцы второй таблицы. Если ввести запрос к двум таблицам без задания конструкции WHERE, результат выполнения запроса в среде SQL будет представлять собой

декартово произведение этих таблиц. Кроме того, стандарт ISO предусматривает специальный формат оператора SELECT, позволяющий вычислить декартово произведение двух таблиц:

```
SELECT [DISTINCT. | ALL] { * | columnList }
FROM tableName1 CROSS JOIN tableName2
```

При выполнении операции соединения данные из двух таблиц комбинируются с образованием пар связанных строк, в которых значения сопоставляемых столбцов являются одинаковыми. Если одно из значений в сопоставляемом столбце одной таблицы не совпадает ни с одним из значений в сопоставляемом столбце другой таблицы, то соответствующая строка удаляется из результирующей таблицы. Именно это правило применялось во всех рассмотренных выше примерах соединения таблиц. Стандартом ISO предусмотрен и другой набор операторов соединений, называемых *внешними соединениями*. Во внешнем соединении в результирующую таблицу помещаются также строки, не удовлетворяющие условию соединения.

Внутреннее соединение используется для связи таблиц по совпадающим значениям столбцов. Внешние соединения нужны в случае, когда значениям в одной таблице не всегда будут иметь соответствующие значения в другой (т.е. внешний ключ – пуст (NULL))

Свойства соединений:

- Внутреннее соединение симметрично, т.е

```
SELECT tableName1.*, tableName2.*, FROM tableName1 INNER JOIN tableName2
```

Эквивалентно

```
SELECT tableName1.*, tableName2.*, FROM tableName2 INNER JOIN tableName1
```

- Полное внешнее соединение симметрично, т.е.

```
SELECT tableName1.*, tableName2.*, FROM tableName1 OUTER JOIN tableName2
```

Эквивалентно

```
SELECT tableName1.*, tableName2.*, FROM tableName2 OUTER JOIN tableName1
```

- Левое и правое соединения симметричны друг по отношению к другу, т.е.

```
SELECT tableName1.*, tableName2.*, FROM tableName1 LEFT JOIN tableName2
```

Эквивалентно

```
SELECT tableName1.*, tableName2.*, FROM tableName2 RIGHT JOIN tableName1
```

Для примеров различных соединений используем таблицы группа и студент

| student    |       |         |         |           | st_group |           |
|------------|-------|---------|---------|-----------|----------|-----------|
| id_student | id_gr | surname | name    | patronym  | id_gr    | number_gr |
| 1          | 1     | Петров  | Петр    | Петрович  | 1        | Z4431K    |
| 2          | 2     | Иванов  | Иван    | Иванович  | 2        | Z5432K    |
| 3          | NULL  | Пупкин  | Василий | Федорович | 3        | B5433     |

Select number\_gr, id\_student, surname, name, patronym from st\_group inner join student on student.id\_gr=st\_group.id\_gr

Результат такого запросы будет таким

st\_group INNER JOIN student

| number_gr | id_student | surname | name | patronym |
|-----------|------------|---------|------|----------|
| Z4431K    | 1          | Петров  | Петр | Петрович |
| Z5432K    | 2          | Иванов  | Иван | Иванович |

Select number\_gr, id\_student, surname, name, patronym from st\_group left join student on student.id\_gr=st\_group.id\_gr

Результат такого запроса будет таким

st\_group LEFT JOIN student

| number_gr | id_student | surname | name | patronym |
|-----------|------------|---------|------|----------|
| Z4431K    | 1          | Петров  | Петр | Петрович |
| Z5432K    | 2          | Иванов  | Иван | Иванович |
| B5433     | NULL       | NULL    | NULL | NULL     |

Select number\_gr, id\_student, surname, name, patronym from st\_group right join student on student.id\_gr=st\_group.id\_gr

Ему эквивалентен запрос Select number\_gr, id\_student, surname, name, patronym from student left join st\_group on student.id\_gr=st\_group.id\_gr

Результат такого запроса будет таким

st\_group RIGHT JOIN student

| number_gr | id_student | surname | name    | patronym  |
|-----------|------------|---------|---------|-----------|
| Z4431K    | 1          | Петров  | Петр    | Петрович  |
| Z5432K    | 2          | Иванов  | Иван    | Иванович  |
| NULL      | 3          | Пупкин  | Василий | Федорович |

Select number\_gr, id\_student, surname, name, patronym from st\_group outer join student on student.id\_gr=st\_group.id\_gr

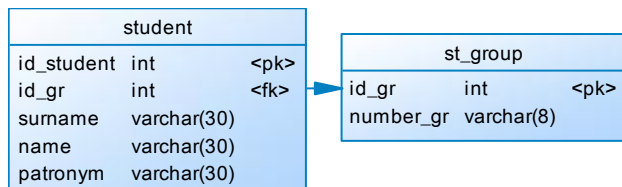
Результат такого запроса будет таким

st\_group OUTER JOIN student

| number_gr | id_student | surname | name    | patronym  |
|-----------|------------|---------|---------|-----------|
| Z4431K    | 1          | Петров  | Петр    | Петрович  |
| Z5432K    | 2          | Иванов  | Иван    | Иванович  |
| NULL      | 3          | Пупкин  | Василий | Федорович |
| B5433     | NULL       | NULL    | NULL    | NULL      |

Это можно использовать для нахождения групп без студентов

**Пример**



Группы без студентов

```
Select number_gr, id_student, surname, name, patronym FROM
st_group LEFT JOIN student ON student.id_gr=st_group.id_gr
WHERE student.id_student is NULL;
```

```
Select number_gr, id_student, surname, name, patronym FROM
student RIGHT JOIN st_group ON student.id_gr=st_group.id_gr
WHERE student.id_student is NULL;
```

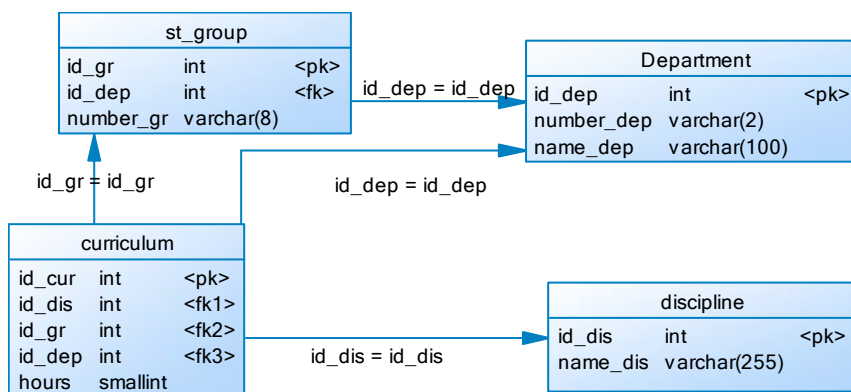
### Запросы с использованием псевдонимов

Необходимость в таких запросах возникает когда одна и та же таблица должна использоваться в разных качествах (кафедра, ведущая дисциплину у группы и кафедра, по которой группа выпускается) или когда нужно чтобы некоторый атрибут принимал 2 значения одновременно (группа, которая изучает и Дифференциальные уравнения и Объектно-ориентированное программирование)

Общий вид псевдонима на таблицу выглядит select ... from таблица as псевдоним. AS иногда можно опустить.

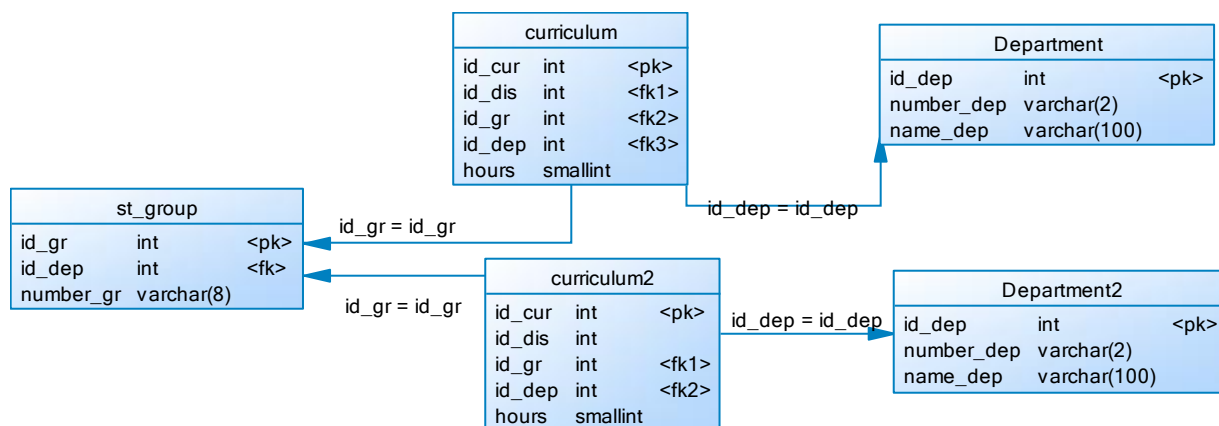
Пример

Схема (кафедра)



Написать запрос: Группы, у которых ведет дисциплины и 43 и 41 кафедра.

Для наглядности



```
Select distinct st_group.id_gr, number_gr from st_group
inner join curriculum on curriculum.id_gr= st_group.id_gr
inner join Department on Department.id_dep=curriculum.id_dep
inner join curriculum as curriculum2 on curriculum2.id_gr= st_group.id_gr
inner join Department as Department2 on Department2.id_dep=curriculum2.id_dep
where Department.number_dep='43' and Department2.number_dep='41'
```

## 2. Выполнение лабораторной работы

По аналогии с примерами, приведенными в п. 1 и 2 реализовать запросы а) .. в), указанные в варианте задания. Все запросы должны не содержать вложенных запросов или агрегатных функций. (Используйте псевдонимы)

Запросы по шаблону должны решаться с использованием только одного like.

## 3. Содержание отчета

Содержание отчета:

- Текст задания.
- физическую модель БД.
- тестовые данные и 3 работы
- текст запросов на SQL;
- наборы данных, возвращаемые запросами.

## 4. Варианты заданий

Варианты заданий приведены в Приложении 2.

## Лабораторная работа №5 Разработка SQL запросов: запросы с подзапросами

### 1. Теоретическая часть

#### Агрегатные функции в операторе выборки языка SQL

Стандарт ISO содержит определение следующих пяти *агрегирующих функций*:

- COUNT — возвращает количество значений в указанном столбце;
- SUM — возвращает сумму значений в указанном столбце;
- AVG — возвращает усредненное значение в указанном столбце;
- MIN — возвращает минимальное значение в указанном столбце;
- MAX — возвращает максимальное значение в указанном столбце.

Все эти функции оперируют со значениями в единственном столбце таблицы и возвращают единственное значение. Функции COUNT, MIN и MAX применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей. За исключением COUNT ( \* ), при вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся непустым значениям столбца. Вариант COUNT ( \* ) является особым случаем использования функции COUNT — его назначение состоит в подсчете всех строк в таблице, независимо от того, содержатся там пустые, повторяющиеся или любые другие значения. Если до применения агрегирующей функции необходимо исключить повторяющиеся значения, следует перед именем столбца в определении функции поместить ключевое слово DISTINCT. Стандарт ISO допускает использование ключевого слова ALL с целью явного указания того, что исключение повторяющихся значений не требуется, хотя это ключевое слово подразумевается по умолчанию, если никакие иные определители не заданы. Ключевое слово DISTINCT не имеет смысла для функций MIN и MAX. Однако его использование может оказывать влияние на результаты выполнения функций SUM и AVG, поэтому следует заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово DISTINCT в каждом запросе может быть указано не более одного раза. Следует отметить, что агрегирующие функции могут использоваться только в списке выборки SELECT и в конструкции HAVING. Во всех других случаях применение этих функций недопустимо. Если список выборки SELECT содержит агрегирующую функцию, а в тексте запроса отсутствует конструкция GROUP BY, обеспечивающая объединение данных в группы, то ни один из элементов списка выборки SELECT не может включать каких-либо ссылок на столбцы, за исключением случая, когда этот столбец используется как параметр агрегирующей функции. \_\_

#### 5.3.4. Группирование результатов (конструкция GROUP BY)

Приведенные выше примеры сводных данных подобны итоговым строкам, обычно размещаемым в конце отчетов. В итогах все детальные данные отчета сжимаются в одну обобщающую строку. Однако очень часто в отчетах требуется формировать и промежуточные итоги. Для этой цели в операторе SELECT может указываться конструкция GROUP BY. Запрос, в котором присутствует конструкция GROUP BY, называется *группирующим запросом*, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная итоговая строка. Столбцы, перечисленные в конструкции GROUP BY, называются *группируемыми столбцами*. Стандарт ISO требует, чтобы конструкции SELECT и GROUP BY были тесно связаны между собой. При использовании в операторе SELECT конструкции GROUP BY каждый элемент списка в списке выборки SELECT должен иметь *единственное значение для всей группы*.

Все имена столбцов, приведенные в списке выборки **SELECT**, должны присутствовать и в конструкции **GROUP BY**, за исключением случаев, когда имя столбца используется только в агрегирующей функции. Противоположное утверждение не всегда справедливо — в конструкции **GROUP BY** могут присутствовать имена столбцов, отсутствующие в списке выборки **SELECT**. Если совместно с конструкцией **GROUP BY** используется конструкция **WHERE**, то она обрабатывается в первую очередь, а группированию подвергаются только те строки, которые удовлетворяют условию поиска. Стандартом **ISO** определено, что при проведении группирования все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значения **NULL** и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

### конструкция **HAVING**

Конструкция **HAVING** предназначена для использования совместно с конструкцией **GROUP BY** для задания ограничений, указываемых с целью отбора тех *групп*, которые будут помещены в результирующую таблицу запроса. Хотя конструкции **HAVING** и **WHERE** имеют сходный синтаксис, их назначение различно.

В основном **HAVING** действует так же, как выражение **WHERE** в операторе **SELECT**. Тем не менее, выражение **WHERE** применяется к столбцам и выражениям для отдельной строки, в то время как оператор **HAVING** применяется к результату действия команды **GROUP BY** (агрегатной функции). Стандарт **ISO** требует, чтобы имена столбцов, применяемые в конструкции **HAVING**, обязательно присутствовали в списке элементов **GROUP BY** или применялись в агрегирующих функциях. На практике условия поиска в конструкции **HAVING** всегда включают, по меньшей мере, одну агрегирующую функцию; в противном случае эти условия поиска должны быть помещены в конструкцию **WHERE** и применены для отбора отдельных строк. (Помните, что агрегирующие функции не могут использоваться в конструкции **WHERE**.)

Конструкция **HAVING** не является необходимой частью языка **SQL** — любой запрос, написанный с использованием конструкции **HAVING**, может быть представлен в ином виде, без ее применения.

Приведем пример использования агрегатных функций и группировки.

Допустим существует таблица **Student\_Uni**

| Student_uni |             |      |
|-------------|-------------|------|
| id_student  | int         | <pk> |
| surname     | varchar(30) |      |
| name        | varchar(30) |      |
| patronym    | varchar(30) |      |
| Form_study  | varchar(1)  |      |
| Faculty     | varchar(1)  |      |
| department  | varchar(2)  |      |



Где From\_study- форма обучения («О», «В» «З»), Faculty- факультет, department- кафедра.

Если мы хотим посчитать всех студентов вуза, необходимо написать запрос

```
SELECT count (id_student) as all_st from Student_Uni
```

, тогда в результате мы получим одно число равное общему количеству студентов.

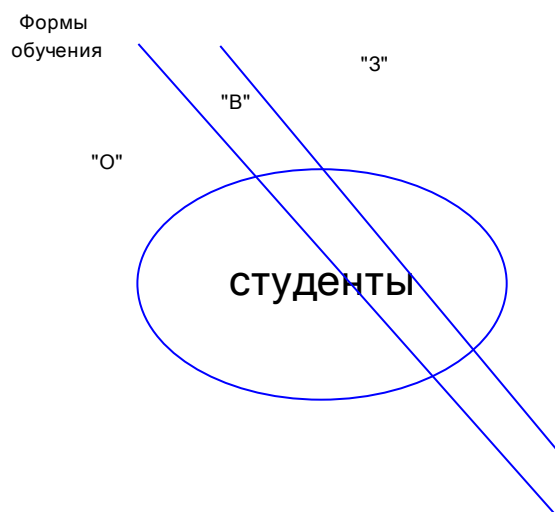
Если мы хотим узнать сколько учится студентов на каждой форме обучения надо написать

```
SELECT count (id_student) as all_st, Form_study from Student_Uni GROUP BY Form_study
```

И получим в результате, что-то вроде

| all<br>_st | Form<br>_study |
|------------|----------------|
| 40         | О              |
| 20         | В              |
| 20         | З              |

На самом деле сгруппировав по форме обучения, мы сделали равенство форм обучения критерием отнесения студента к той или иной группе, в которой потом будет производиться подсчет. Т.е. разбили все множество студентов по форме обучения



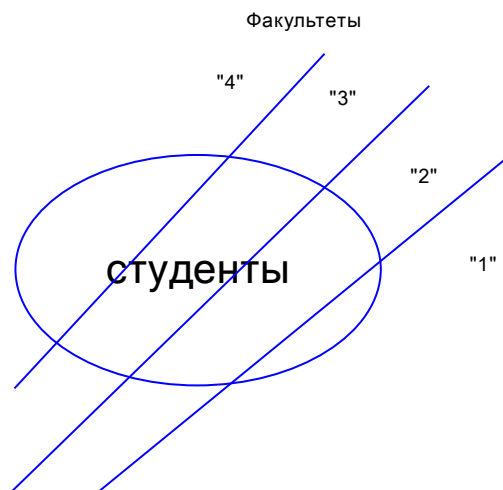
Если мы захотим узнать численность студентов на факультетах, запрос будет выглядеть:

```
SELECT count (id_student) as all_st, Faculty from Student_Uni GROUP BY Faculty
```

Допустим в вузе только 4 факультета, тогда мы получим результат типа

| all<br>_st | Facu<br>lty |
|------------|-------------|
| 10         | 1           |

|    |   |
|----|---|
| 0  |   |
| 10 | 2 |
| 0  |   |
| 20 | 3 |
| 0  |   |
| 22 | 4 |
| 0  |   |

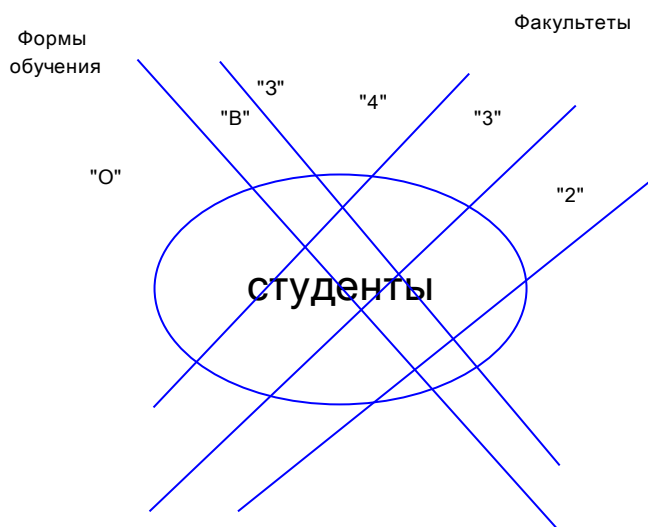


Ячейка уже объединяется по факультету.

А если мы сгруппируем и по факультету и по форме обучения

`SELECT count (id_student) as all_st, Faculty, Form_study from Student_Uni GROUP BY Faculty, Form_study`

То получим ячейку у которой одинаковы оба эти значения



И результат (числа не вписаны- это общий вид)

| All<br>_st | Fac<br>ulty | Form<br>_study |
|------------|-------------|----------------|
|            | 1           | O              |
|            | 1           | B              |
|            | 1           | 3              |
|            | 2           | O              |
|            | 2           | B              |
|            | 2           | 3              |
|            | 3           | O              |
|            | 3           | B              |
|            | 3           | 3              |
|            | 4           | O              |
|            | 4           | B              |
|            | 4           | 3              |

Хотя строк может быть меньше, если не на всех факультетах есть все формы обучения!!!

Каждый раз добавляя в группировку поля мы уменьшаем ячейку, поэтому например запрос типа

```
SELECT count (id_student) as all_st from Student_Uni GROUP BY id_student
```

Выдаст и кучу строк с единицами (размер ячейки подсчета студентов- один студент)

Поэтому обратите внимание !

Если у Вас группировка и агрегатная функция заданы для одного поля, то это ,вероятно, ошибка

### Объединение, пересечение, разность запросов в языке SQL

В языке SQL можно использовать обычные операции над множествами — объединение (union), пересечение (intersection) и разность (difference), — позволяющие комбинировать результаты выполнения двух и более запросов в единую результирующую таблицу.

- *Объединением* двух таблиц А и В называется таблица, содержащая все строки, которые имеются в первой таблице (А), во второй таблице (В) или в обеих этих таблицах одновременно,
- *Пересечением*, двух таблиц называется таблица, содержащая все строки, присутствующие в обеих исходных таблицах одновременно.
- *Разностью* двух таблиц А и В называется таблица, содержащая все строки, которые присутствуют в таблице А, но отсутствуют в таблице В.

Все эти операции над множествами графически представлены на рис. 5.2. На таблицы, которые могут комбинироваться с помощью операций над множествами, накладываются определенные ограничения. Самое важное из них состоит в том, что таблицы должны быть *совместимы, по соединению* — т.е. они должны иметь одну и ту же структуру. Это означает, что таблицы должны иметь одинаковое количество столбцов, причем в соответствующих столбцах должны размещаться данные одного и того же типа и длины. Обязанность убедиться в том, что значения данных соответствующих столбцов принадлежат одному и тому же домену, возлагается на пользователя.

Три операции над множествами, предусмотренные стандартом ISO, носят название UNION, INTERSECT и EXCEPT. В каждом случае формат конструкции с операцией над множествами должен быть следующим:

operator .[ALL] [CORRESPONDING [ BY {columnl [, . . . ]} ] ]

При указании конструкции CORRESPONDING BY операция над множествами выполняется для указанных столбцов. Если задано только ключевое слово CORRESPONDING, а конструкция BY отсутствует, операция над множествами выполняется для столбцов, которые являются общими для обеих таблиц. Если указано ключевое слово ALL, результирующая таблица может содержать повторяющиеся строки.

Одни диалекты языка SQL не поддерживают операций INTERSECT и EXCEPT, а в других вместо ключевого слова EXCEPT используется ключевое слово MINUS.

Пример выберем имена и фамилии преподавателей или студентов с непустым отчеством. (логическое или)

```
(SELECT surname, name
FROM student
WHERE patronym IS NOT NULL)
UNION
(SELECT surname, name
FROM teacher
WHERE patronym IS NOT NULL);
Или
(SELECT *
FROM student
WHERE patronym IS NOT NULL)
UNION CORRESPONDING BY surname, name
```

```
(SELECT *  
FROM teacher  
WHERE patronym IS NOT NULL);
```

Выберем фамилии и имена и отчества преподавателей , которые еще учатся (в магистратуре)(логическое И)(считаем, что полных тезок нет)

```
(SELECT surname, name, patronym  
FROM student )  
INTERSECT  
(SELECT surname, name, patronym  
FROM teacher );
```

Или

```
(SELECT *  
FROM student)  
INTERSECT CORRESPONDING BY surname, name, patronym  
(SELECT *  
FROM teacher);
```

Выберем фамилии и имена и отчества студентов , которые не преподают (в магистратуре) (студенты минус преподаватели)(считаем, что полных тезок нет)

```
(SELECT surname, name, patronym  
FROM student )  
EXCEPT  
(SELECT surname, name, patronym  
FROM teacher );
```

Или

```
(SELECT *  
FROM student)  
EXCEPT CORRESPONDING BY surname, name, patronym  
(SELECT *  
FROM teacher);
```

## Запросы с подзапросами в языке SQL

Здесь мы обсудим использование законченных операторов SELECT, внедренных в тело другого оператора SELECT. *Внешний* (второй) оператор SELECT использует результат выполнения *внутреннего* (первого) оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут находиться в конструкциях WHERE и HAVING внешнего оператора SELECT — в этом случае они получают название *подзапросов*, или *вложенных запросов*. Кроме того, внутренние операторы SELECT могут использоваться в операторах INSERT, UPDATE и DELETE .

Также возможно нахождение подзапроса в конструкции FROM , где он должен получить псевдоним после скобок и с ним можно работать как с любой таблицей.

Существуют три типа подзапросов.

- *Скалярный подзапрос* возвращает значение, выбираемое из пересечения одного столбца с одной строкой, т.е. единственное значение. В принципе скалярный подзапрос может использоваться везде, где требуется указать единственное значение.
- *Строковый подзапрос* возвращает значения нескольких столбцов таблицы, но в виде единственной строки. Строковый подзапрос может использоваться везде, где применяется конструктор строковых значений, — обычно это предикаты.
- *Табличный подзапрос* возвращает значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Табличный подзапрос может использоваться везде, где допускается указывать таблицу, например как операнд предиката IN.

Пример скалярного подзапроса

Вывести группу, в которой учится столько же человек, как и в группе Z4431

```
Select st_group.* from st_group left join student on student.id_gr=st_group.id_gr
```

```
Group by student.id_gr
```

```
Having count (id_student)=
```

```
(select count(*) from st_group gr
```

```
left join student st on st.id_gr=gr.id_gr where number_gr=' Z4431')
```

Left join, т.к в группе может не быть студентов

*Табличный подзапрос*

Вывести всех студентов из групп ,где учится Иванов

```
Select student.* from student where id_gr in
```

(select st\_group.id\_gr from st\_group inner join student on student.id\_gr=st\_group.id\_gr where student.surname='Иванов')

### **К подзапросам применяются следующие правила и ограничения.**

1. В подзапросах не должна использоваться конструкция ORDER BY, хотя она может присутствовать во внешнем операторе SELECT.

2. Список выборки SELECT подзапроса должен состоять из имен отдельных столбцов или составленных из них выражений, за исключением случая, когда в подзапросе используется ключевое слово EXISTS.

3. По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в конструкции FROM подзапроса. Однако разрешается ссылаться и на столбцы таблицы, указанной в конструкции FROM внешнего запроса, для чего используются уточненные имена столбцов (как описано ниже).

4. Если подзапрос является одним из двух операндов, участвующих в операции сравнения, то подзапрос должен указываться в правой части этой операции. Например, приведенный ниже вариант записи запроса является некорректным, поскольку подзапрос размещен в левой части операции сравнения со значением столбца salary.

```
SELECT staffNo, fName, IName, position, salary
FROM Staff
WHERE (SELECT AVG(salary) FROM Staff) < salary;
```

### **Ключевые слова ANY и ALL**

Ключевые слова ANY и ALL могут использоваться с подзапросами, возвращающими один столбец чисел. Если подзапросу будет предшествовать ключевое слово ALL, условие сравнения считается выполненным только в том случае, если оно выполняется для всех значений в результирующем столбце подзапроса. Если тексту подзапроса предшествует ключевое слово ANY, то условие сравнения будет считаться выполненным, если оно удовлетворяется хотя бы для какого-либо (одного или нескольких) значения в результирующем столбце подзапроса. Если в результате выполнения подзапроса будет получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY — невыполненным. Согласно стандарту ISO дополнительно можно использовать ключевое слово SOME, являющееся синонимом ключевого слова ANY.

Пример

*Найдите всех работников, чья зарплата превышает зарплату хотя бы одного работника отделения компании под номером '003'.*

```
SELECT staffNo, fName, IName, position, salary
FROM Staff
WHERE salary > SOME(SELECT salary
FROM Staff
WHERE branchNo = 'B003');
```

Далее, *коррелированный* подзапрос – это особый вид подзапроса (табличного, однострочного или скалярного), а именно такой, в котором есть ссылка на некоторую «внешнюю» таблицу. В следующем примере заключенное в скобки выражение после ключевого слова IN и есть коррелированный подзапрос, потому что включает ссылку на внешнюю таблицу

S (запрос звучит так: «Получить названия поставщиков, которые поставляют деталь P1»).

```
SELECT DISTINCT S.SNAME
FROM S
WHERE 'P1' IN ( SELECT PNO
FROM SP
WHERE SP.SNO = S.SNO )
```

### **Экзистенциальные запросы в языке SQL**

Ключевые слова EXISTS и NOT EXISTS предназначены для использования только совместно с подзапросами или управляющими конструкциями. Результат их обработки представляет собой логическое значение TRUE или FALSE. Для ключевого слова EXISTS результат равен TRUE в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки ключевого слова EXISTS будет значение FALSE. Для ключевого слова NOT EXISTS используются правила обработки, обратные по отношению к ключевому слову EXISTS. Поскольку по ключевым словам EXISTS и NOT EXISTS проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Например, группа, где есть студенты, будет получена запросом

```
Select st_group.* from st_group where
exists (select id_student from student where
student.id_gr=st_group.id_gr)
```

Часто условие NOT EXISTS используется для реализации разности.

Например: Группа без студентов может быть найдена запросом

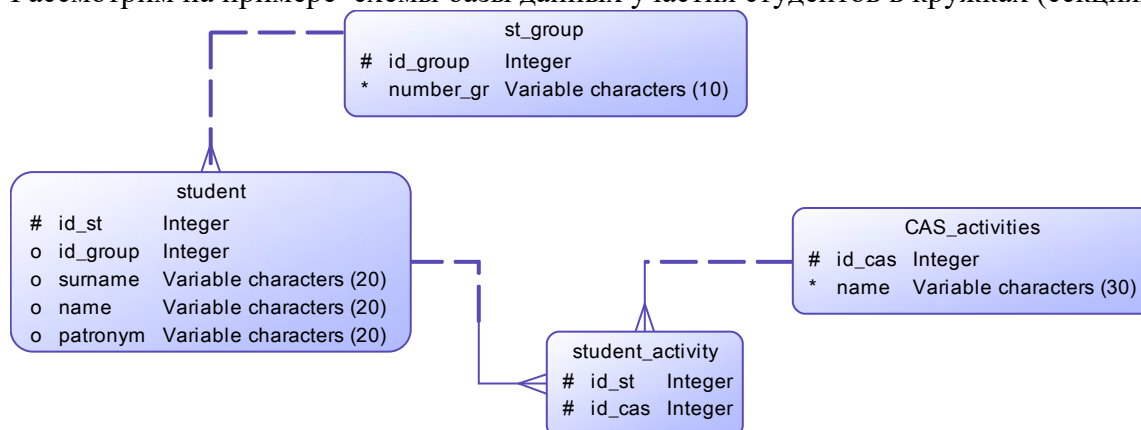
```
Select st_group.* from st_group where
not exists (select id_student from student where
student.id_gr=st_group.id_gr)
```

Обратите внимание, что в подзапросе сквозная видимость



Одно из назначений NOT EXISTS – реализация реляционного деления (так называемые запросы на «все»).

Рассмотрим на примере схемы базы данных участия студентов в кружках (секциях).



На схеме CAS activities(от"Creativity, Action, Service") это секции.

Запрос звучит так:

**Студенты, которые посещают все кружки.**

Для самой простой по смыслу реализации такого запроса посмотрим на него с точки зрения реляционного деления. В данном случае мы должны поделить студентов на кружки, то есть студент- это делимое, а все кружки - делитель.

такой запрос можно разделить на 3 части- запроса:

A  
NOT EXISTS  
(B  
NOT EXISTS (  
C))

При этом каждая часть имеет свое назначение.

Часть А — это делимое , В — делитель, а С отвечают за связку между делимым и делителем (студент и все кружки)

Запросы А и С как правило похожи с точностью до псевдонимов (сквозная видимость- псевдонимы обязательны), однако в С идет дополнительная связка с А по ключу делимого (в данном случае студента) и связка С и В по ключу делителя (в данном случае кружка)

Запрос В задает делитель.

Такой запрос будет выглядеть:

```

Select student.* from student
where not exists
(select * from CAS_activities
where not exists
(Select * from student student_activity as st_a
where student.id_st=st_a.id_st
and st_a.id_cas= CAS_activities.id_cas))
    
```

Если запрос звучит, например

**Студенты, которые посещают все кружки, связанные с математикой**

То меняется здесь только делитель. Вместо «все кружки» он будет «все кружки, связанные с математикой»

Тогда запрос изменится так

```
Select student.* from student
where not exists
(select * from CAS_activities
where name like '%математ%'
and not exists
(Select * from student student_activity as st_a
where student.id_st=st_a.id_st
and st_a.id_cas= CAS_activities.id_cas))
```

## Запросы на разность

Запрос на разность можно реализовать несколькими способами:

с помощью except, not in, not exists, left join.

Рассмотрим на примере запроса «Группы, где есть Иванов, но нет Петрова»

Здесь из групп с Ивановым необходимо вычесть группы, где есть Петров.

Пишем запросы вычитаемое и уменьшаемое.

Уменьшаемое:

Группы, где есть Иванов

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='Иванов'
```

Вычитаемое:

Группы, где есть Петров

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='Петров'
```

Реализация с помощью except

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='Иванов'
except
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='Петров'
```

Реализация с помощью not in

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='Иванов'
and st_group.id_gr not in
```

```
(select st_group.id_gr from st_group inner join student on student.id_gr=st_group.id_gr where student.surname=' Петров')
```

Реализация с помощью not exists

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where student.surname='Иванов'
```

And not exists(

```
select *from st_group as gr2 inner join student on student.id_gr=gr2.id_gr where student.surname=' Петров'
```

And st\_group.id\_gr=gr2.id\_gr

)

Реализация с помощью left join

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr
```

Left join

```
(select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where student.surname=' Петров'
```

```
)q on st_group.id_gr=q.id_gr
```

```
where student.surname='Иванов'
```

```
and q.id_gr is NULL
```

## 2. Выполнение лабораторной работы

По аналогии с примерами, приведенными в п. 1 реализовать запросы г) .. ж), указанные в варианте задания.

Один из запросы на максимум/минимум реализовать и с помощью директивы all(где возможно) и с помощью агрегатных функций

Запрос на «все» (реляционное деление) реализовать с помощью 2 not exists и с помощью агрегатной функции

Запросы на разность реализовать в 3 вариантах: Not in, except (MySQL не поддерживает, поэтому только синтаксис), с использованием левого/правого соединения.

Использовать with можно не более чем в 1 запросе

### **3. Содержание отчета**

Содержание отчета:

- Текст задания.
- физическую модель БД.
- текст запросов на SQL;
- наборы данных, возвращаемые запросами.

### **4. Варианты заданий**

Варианты заданий приведены в Приложении 2.

## **Лабораторная работа №6 Хранимые процедуры**

### **1. Теоретическая часть**

### **2. Теоретическая часть**

Хранимые процедуры представляют собой набор команд SQL, которые могут компилироваться и храниться на сервере. Таким образом, вместо того, чтобы хранить часто используемый запрос, клиенты могут ссылаться на соответствующую хранимую процедуру. Это обеспечивает лучшую производительность, поскольку данный запрос должен анализироваться только однажды и уменьшается трафик между сервером и клиентом. Концептуальный уровень можно также повысить за счет создания на сервере библиотеки функций.

Триггер представляет собой хранимую процедуру, которая активизируется при наступлении определенного события. Например, можно задать хранимую процедуру, которая срабатывает каждый раз при удалении записи из транзакционной таблицы - таким образом, обеспечивается автоматическое удаление соответствующего заказчика из таблицы заказчиков, когда все его транзакции удаляются.

Хранимые программы (процедуры и функции) поддерживаются в MySQL 5.0. Хранимые процедуры – набор SQL-выражений, который может быть сохранен на сервере. Как только это сделано, клиенту уже не нужно повторно передавать запрос, а требуется просто вызвать хранимую программу.

Это может быть полезным тогда, когда:

- многочисленные приложения клиента написаны в разных языках или работают на других платформах, но нужно использовать ту же базу данных операций
- безопасность на 1 месте

Хранимые процедуры и функции (подпрограммы) могут обеспечить лучшую производительность потому, что меньше информации требуется для пересылки между клиентом и сервером. Выбор увеличивает нагрузку на сервер БД, но снижает затраты на стороне клиента. Используйте это, если много клиентских машин (таких как Веб-серверы) обслуживаются одной или несколькими БД.

Хранимые подпрограммы также позволяют вам использовать библиотеки функций, хранимые в БД сервера. Эта возможность представлена для многих современных языков программирования, которые позволяют вызывать их непосредственно (например, используя классы).

MySQL следует в синтаксисе за SQL:2003 для хранимых процедур, который уже используется в IBM's DB2.

Хранимые процедуры требуют наличия таблицы `proc` в базе `mysql`. Эта таблица обычно создается во время установки сервера БД. При создании, модификации, удалении хранимых подпрограмм сервер манипулирует с таблицей `mysql.proc`

Начиная с MySQL 5.0.3 требуются следующие привилегии:

**CREATE ROUTINE** для создания хранимых процедур

**ALTER ROUTINE** необходимы для изменения или удаления процедур. Эта привилегия автоматически назначается создателю процедуры (функции)

**EXECUTE** привилегия потребуется для выполнения подпрограммы. Тем не менее, автоматически назначается создателю процедуры (функции). Также, по умолчанию, **SQL SECURITY** параметр для подпрограммы **DEFINER**, который разрешает пользователям, имеющим доступ к БД вызывать подпрограммы, ассоциированные с этой БД.

### Синтаксис хранимых процедур и функций

Хранимая подпрограмма представляет собой процедуру или функцию. Хранимые подпрограммы создаются с помощью выражений **CREATE PROCEDURE** или **CREATE FUNCTION**. Хранимая подпрограмма вызывается, используя выражение **CALL**, причем только возвращающие значение переменные используются в качестве выходных. Функция может быть вызвана подобно любой другой функции и может возвращать скалярную величину. Хранимые подпрограммы могут вызывать другие хранимые подпрограммы.

Начиная с MySQL 5.0.1, загруженная процедура или функция связана с конкретной базой данных. Это имеет некоторые последствия:

- Когда подпрограмма вызывается, то подразумевается, что надо произвести вызов **USE db\_name** (и отменить использование базы, когда подпрограмма завершилась, и база больше не потребуется)
- Вы можете указывать обычные имена процедур с именем базы данных. Это может быть использовано, чтобы ссылаться на подпрограмму, которая - не в текущей базе данных. Например, для выполнения хранимой процедуры `p` или функции `f` которые связаны с БД `test`, вы можете сказать интерпретатору команд так: `CALL test.p()` или `test.f()`.

- Когда база данных удалена, все загруженные подпрограммы связанные с ней тоже удаляются. В MySQL 5.0.0, загруженные подпрограммы - глобальные и не связанные с базой данных. Они наследуют по умолчанию базу данных из вызывающего оператора. Если `USE db_name` выполнено в пределах подпрограммы, оригинальная текущая БД будет восстановлена после выхода из подпрограммы (Например текущая БД `db_11`, делаем вызов подпрограммы, использующей `db_22`, после выхода из подпрограммы остается текущей `db_11` )

MySQL поддерживает полностью расширения, которые разрешают юзать обычные `SELECT` выражения (без использования курсоров или локальных переменных) внутри хранимых процедур. Результирующий набор, возвращенный от запроса, а просто отправляется напрямую клиенту. Множественный `SELECT` запрос генерирует множество результирующих наборов, поэтому клиент должен использовать библиотеку, поддерживающую множественные результирующие наборы.

**CREATE PROCEDURE** – создать хранимую процедуру.

**CREATE FUNCTION** – создать хранимую функцию.

### Синтаксис:

**CREATE PROCEDURE** имя\_процедуры ([параметр\_процедуры[, ...]])  
[характеристика ...] тело\_подпрограммы

**CREATE FUNCTION** имя\_функции ([параметр\_функции[, ...]])

**RETURNS** тип

[характеристика ...] тело\_подпрограммы

параметр\_процедуры:

[ **IN** | **OUT** | **INOUT** ] имя\_параметра тип

параметр\_функции:

имя\_параметра тип

тип:

Любой тип данных MySQL

характеристика:

`LANGUAGE SQL`

| `[NOT] DETERMINISTIC`

| `{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }`

| `SQL SECURITY { DEFINER | INVOKER }`

| `COMMENT 'string'`

тело\_подпрограммы:

Правильное SQL выражение.

Рассмотрим все на практике.

Сначала создадим хранимую процедуру следующим запросом:

```
CREATE PROCEDURE `my_proc`(OUT t INTEGER(11))
    NOT DETERMINISTIC
    SQL SECURITY INVOKER
    COMMENT ''
BEGIN
```

```
select val1+val2 into 't' from `my` LIMIT 0,1;
END;
```

Применение выражения `LIMIT` в этом запросе сделано из соображений того, что не любой клиент способен принять многострочный результирующий набор.

После этого вызовем ее:

```
CALL my_proc(@a);
SELECT @a;
```

Для отделения внутреннего запроса от внешнего всегда используют разделитель отличный от обычно (для задания используют команду **DELIMITER** <строка/символ>)

Вот еще один пример с учетом всех требований.

```
delimiter //
CREATE PROCEDURE simpleproc (OUT param1 INT)
BEGIN
SELECT COUNT(*) INTO param1 FROM t;
END;
//

delimiter ;

CALL simpleproc(@a);
SELECT @a;
```

```
+-----+
|  @a  |
+-----+
|   3   |
+-----+
1 row in set (0.00 sec)
```

Готово.

### Получение значения первичного ключа для дальнейшей вставки

Если это неавтоинкрементный ключ, то

```
SELECT ifnull(max(id)+1,0) INTO new_id FROM my_table;
INSERT INTO my_table(id,col1, col2,...) VALUES (new_id,'val1', 'val2'...);
```

Представим `mysql` `INSERT` в одной из таблиц, а таблица имеет столбец `item_id`, который установлен в `autoincrement` и `primary key`.

```
INSERT INTO table_name (col1, col2,...) VALUES ('val1', 'val2'...);
set new_id=(SELECT LAST_INSERT_ID());
```

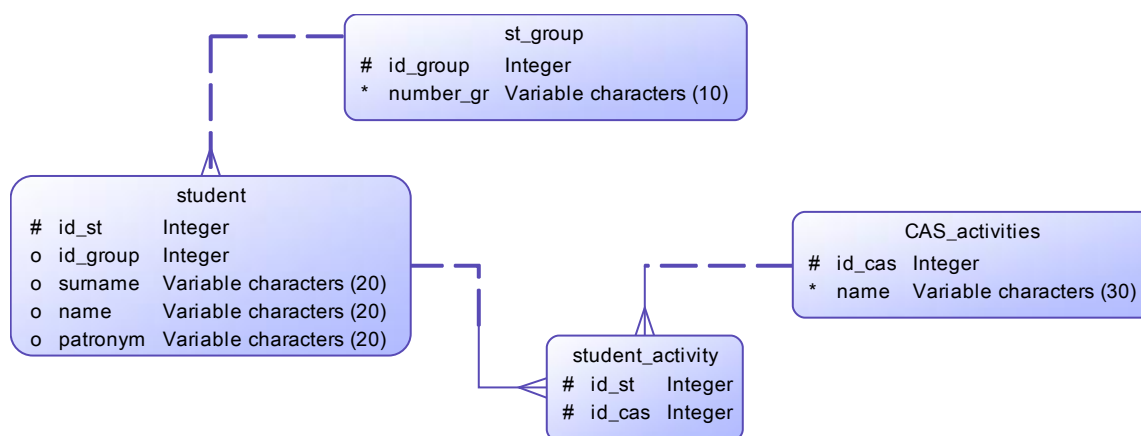
Это вернет вам значение `PRIMARY KEY` последней введенной строки:

Идентификатор, который был сгенерирован, поддерживается на сервере для каждого подключения. Это означает, что значение, возвращаемое функцией для данного клиента, является первым значением `AUTO_INCREMENT`, сгенерированным для самого последнего оператора, влияющим на столбец `AUTO_INCREMENT` этого клиента.

Вам также нужно иметь в виду, что это будет работать, только если `Last_INSERT_ID()` будет запущен после вашего запроса `Insert`. Это запрос возвращает идентификатор, вставленный в схему. Вы не можете получить конкретный последний вставленный идентификатор таблицы.

Наиболее часто

Примеры выполнены для базы данных со схемой



### Процедура вставки с пополнением справочника.

Вставляется информация о студенте, если указанный номер группы отсутствует в БД, запись добавляется в таблицу с перечнем групп( справочник групп). Для **не** автоинкрементной таблицы группа



```

delimiter //
CREATE PROCEDURE ins_stud (gr_num varchar(8),name_ varchar(15),
surname_ varchar(20),patronym_ varchar(25))
BEGIN
declare id_gr_new int;
declare id_st_new int;
if exists(select * from st_group where num_gr=gr_num)
then select id_gr into id_gr_new from st_group where num_gr=gr_num;
else begin
set id_gr_new=( select ifnull(max(id_gr)+1,0) from st_group );
INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
end;
end if;
set id_st_new=( select ifnull(max(id_st)+1,0) from student);
insert into student (id_st, surname, name,patronym,id_gr)
VALUES (id_st_new,surname_,name_,patronym_,id_gr_new);
END;//
delimiter ;

```

### Процедура удаления с очисткой справочника

Удаляется информация о студенте, если в его группе нет больше студентов, запись удаляется из таблицы с перечнем групп.

```

delimiter //
create procedure del_student_clear_gr(id_st_del int)
BEGIN
declare id_gr_del int;
select id_gr into id_gr_del from student where id_st=id_st_del;
delete from student where id_st=id_st_del;
if not exists(select * from student where id_gr=id_gr_del)
then delete from st_group where id_gr=id_gr_del;
end if;
END;//
delimiter ;

```

### Процедура каскадного удаления.

Перед удалением записи о группе удаляются записи обо всех студентах этой группы и их кружках

```

delimiter //
create procedure del_group_cascade ( id_gr_del int)
BEGIN
delete from student_activity where id_st in
      (select id_st from student where id_gr= id_gr_del);
delete from student where id_gr= id_gr_del;
  delete from st_group where id_gr=id_gr_del;
END;//
delimiter ;

```

### Процедура вычисления и возврат значения агрегатной функции

Возврат количества групп

```

delimiter ;
use uni;
delimiter //
create procedure count_groups (out cnt_gr int)
BEGIN
select ifnull(count(id_gr),0) into cnt_gr from st_group;
END;//
delimiter ;

```

---

```

call count_groups( @cnt);
select @cnt;

```

```

delimiter //
create function count_groups1() returns int DETERMINISTIC
BEGIN
declare cnt_gr int DEFAULT 0;
set cnt_gr=(select ifnull(count(id_gr),0) from st_group);
return cnt_gr;
END;//
delimiter ;

```

---

```

select count_groups1();

```

### Формирование статистики во временной таблице

```

delimiter //
create procedure cas_statistics ()
BEGIN
create temporary table if not exists cas_stat
(
id_stat int auto_increment primary key,
id_st int,
count_cas int,
count_cas_avg double default 0,
diff_cnt_avg double default 0
);
insert into cas_stat (id_st,count_cas)
select student.id_st, count(id_cas) as count_cas from student
left join student_activity on student.id_st= student_activity.id_st group by student.id_st;

update cas_stat set count_cas_avg=
(select avg(count_cas) from
(select student.id_st, count(id_cas) as count_cas from student
left join student_activity on student.id_st= student_activity.id_st group by student.id_st)q);

update cas_stat set diff_cnt_avg=count_cas-count_cas_avg;

select * from cas_stat;
select avg(diff_cnt_avg*diff_cnt_avg) from cas_stat;

drop table cas_stat;
END;//
delimiter ;

```

### 3. Выполнение лабораторной работы

По аналогии с примерами, приведенными в п. 1 , создать в БД ХП, реализующие:

- вставку с пополнением справочников (вставляется информация о студенте, если указанный номер группы отсутствует в БД, запись добавляется в таблицу с перечнем групп) (получаем ссылку на внешний ключ по значению данного из родительской таблицы);
- удаление с очисткой справочников – удаление всех зависимых данных(удаляется информация о студенте, если в его группе нет больше студентов, запись удаляется из таблицы с перечнем групп);
- каскадное удаление (перед удалением записи о группе удаляются записи обо всех студентах этой группы);
- вычисление и возврат значения агрегатной функции (т.к. агрегатная функция дает единственный результат);
- формирование статистики во временной таблице (например, для рассматриваемой БД — для каждого факультета: количество групп, количество обучающихся студентов, количество изучаемых дисциплин, количество сданных дисциплин, средний балл по факультету).

## 4. Содержание отчета

Содержание отчета:

- физическую модель БД.
- назначение, тексты ХП и их вызовов;
- наборы данных(скриншоты), возвращаемые ХП.

## 5. Варианты заданий

Варианты заданий приведены в Приложении 2.

# Контрольная работа. Проектирование взаимодействия базы данных и приложения

## 1. Теоретическая часть

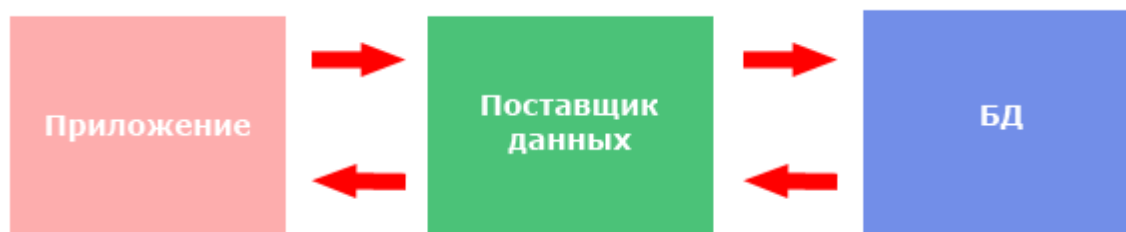
Наиболее часто встречающиеся способы доступа к БД из приложения:

- скриптовые языки со встроенной поддержкой работы с БД (php, например)
- ADO.Net объекты
- ODBC/JDBC объекты
- ORM (Object Relational mapping) привязка классов .Net к таблицам БД и от типов данных CLR в типы SQL.

Рассмотрим пару примеров

### 1.1 ADO.NET

Для работы с базой данных нам потребуется **поставщик данных** (data provider), он обеспечивает подключение к БД, позволяет выполнять команды и получать результаты. По сути это обычный файл (.dll), внутри которого содержатся типы, которые настроены на взаимодействие с какой-то одной конкретной СУБД: MySQL, Oracle, Microsoft SQL Server и так далее.



В Microsoft ADO.NET основное количество поставщиков данных содержится в сборке Sytem.Data.dll, но в этой сборке нет поставщика данных для работы с MySQL.

Скачать его можно с сайта MySQL <https://dev.mysql.com/downloads/connector/>

Так же для подключения к базе данных потребуется узнать ip адрес сервера

Затем настроим **подключение к базе данных**

1. Создадим и заполним объект `MySqlConnectionStringBuilder`, который будет хранить следующие значения: имя сервера, где лежит база данных, имя пользователя и пароль для подключения к БД, а так же имя базы данных.

```
MySqlConnectionStringBuilder mysqlCSB;  
mysqlCSB = new MySqlConnectionStringBuilder();  
mysqlCSB.Server = "ip адрес сервера";  
mysqlCSB.Database = "имя БД";  
mysqlCSB.UserID = "имя пользователя";  
mysqlCSB.Password = "пароль";
```

2. Создадим строку запроса, в ней мы выбираем все комментарии за сегодняшний день.

```
string queryString = @"SELECT comment_author,  
comment_date,  
comment_content  
FROM wp_comments  
WHERE comment_date >= CURDATE() ";
```

Создадим объект `DataTable`, который будет возвращать наш метод и принимать `datagridView`.

```
DataTable dt = new DataTable();
```

4. Создадим объект подключения, используя класс `MySqlConnection`.

```
using(MySqlConnection con = new MySqlConnection())  
{  
}
```

4.1 Настроим созданный объект, передав в свойство `ConnectionString` наш созданный ранее объект типа `MySqlConnectionStringBuilder`.

```
con.ConnectionString =  
mysqlCSB.ConnectionString;
```

5. Открываем соединение с базой данных

```
con.Open();
```

6. Создаем объект команды, в конструктор передаем строку запроса и объект подключения

```
MySqlCommand com = new MySqlCommand(queryString, con);
```

7. Выполним метод `ExecuteReader`, который позволит получить объект чтения данных

**MySqlDataReader**

```
using(MySqlDataReader dr = com.ExecuteReader())  
{  
    //есть записи?  
    if (dr.HasRows)  
    {  
        //заполняем объект DataTable  
        dt.Load(dr);  
    }  
}
```

8 Осталось поместить полученные данные в dataGridView.

```
private void button1_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = GetComments();
}
```

А можно не привязывать dataGridView (шаги 7 и 8), а сделать ручную обработку

```
MySqlDataReader reader = command.ExecuteReader();
while (reader.Read())
{
    Console.WriteLine("\t{0}\t{1}\t{2}",
        reader[0], reader[1], reader[2]);
}
reader.Close();
```

Пример кода с отображением в listView

```
private void Load_b2_Click(object sender, EventArgs e)
{
    MySqlConnectionStringBuilder mysqlCSB;
    mysqlCSB = new MySqlConnectionStringBuilder();
    mysqlCSB.Server = "127.0.0.1";
    mysqlCSB.Database = "uni";
    mysqlCSB.UserID = "root"; mysqlCSB.Password = "MySQL";

    string queryString = @"SELECT * from student where id_gr=@id_g";
    using (MySqlConnection con = new MySqlConnection())
    {
        con.ConnectionString = mysqlCSB.ConnectionString;
        MySqlCommand com = new MySqlCommand(queryString, con);
        com.Parameters.AddWithValue("id_g", 1);

        try
        {
            con.Open();
            using (MySqlDataReader dr = com.ExecuteReader())
            {
                listView1.Columns.Clear();
                if (dr.HasRows) {
                    for (int i=0;i<dr.FieldCount;i++)
                    {
                        listView1.Columns.Add(dr.GetName(i));
                    }
                }
                while (dr.Read())
                {
                    ListViewItem item = listView1.Items.Add(dr[0].ToString());
                    for (int i = 1; i < dr.FieldCount; i++)
                    {
                        item.SubItems.Add(dr[i].ToString());
                    }
                }
            }
        }
        catch (Exception ex) {MessageBox.Show(ex.Message);}
        con.Close();
    }
}
```

Пример кода с подключением источника данных к Data Aware компоненту

```
namespace ado_net_try
{
    Ссылка: 3
    public partial class Form1 : Form
    {
        Ссылка: 1
        public Form1()
        {
            InitializeComponent();
        }

        Ссылка: 1
        private void Load_b_Click(object sender, EventArgs e)
        {
            MySqlConnectionStringBuilder mysqlCSB;
            mysqlCSB = new MySqlConnectionStringBuilder();
            mysqlCSB.Server = "127.0.0.1";
            mysqlCSB.Database = "uni";
            mysqlCSB.UserID = "root"; mysqlCSB.Password = "MySQL";
            DataTable dt = new DataTable();

            string queryString = @"SELECT * from student";
            using (MySqlConnection con = new MySqlConnection())
            {
                con.ConnectionString = mysqlCSB.ConnectionString;
                MySqlCommand com = new MySqlCommand(queryString, con);
                try
                {
                    con.Open();
                    using (MySqlDataReader dr = com.ExecuteReader())
                    {
                        if (dr.HasRows) { dt.Load(dr); }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
                con.Close();
                dataGridView1.DataSource = dt;
            }
        }
    }
}
```

## 1.2 Hibernate

Nhibernate это решение для Объектно Реалиционного мапинга для платформы .NET. Этот фреймворк позволяет делать мапинг Объектно ориентированных моделей к традиционным БД. Его основное преимущество маппирование классов .Net к таблицам БД и от типов данных CLR в типы SQL.

Предположим, что у нас есть база данных с таблицей Employee (работник)

Столбец ID должен быть первичным ключом, и должен быть автоинкрементом.

| INBLRWIT2653\... dbo.Employee                                                     |             |           |                                     |
|-----------------------------------------------------------------------------------|-------------|-----------|-------------------------------------|
|                                                                                   | Column Name | Data Type | Allow Nulls                         |
|  | ID          | int       | <input type="checkbox"/>            |
|                                                                                   | Name        | nchar(30) | <input checked="" type="checkbox"/> |
|                                                                                   |             |           | <input type="checkbox"/>            |

**Primary Key**

Теперь запускаем VisualStudio и создаем новое приложение WindowsFormApplication, проект назовем NhibernateBasics. Добави новый класс, назовите его Employee.cs и вставим следующий код.

```
namespace NHibernateBasics
{
    public class Employee
    {
        public virtual int ID { get; set; }
        public virtual string Name { get; set; }
    }
}
```

Nhibernate не нужно специальных интерфейсов для бизнес классов (объекты предметной области). Эти объекты не зависят от механизма загрузки и сохранения данных. Однако требуется чтобы свойства класса были описаны как виртуальные, чтобы они могли создавать Proxy. Из-за отсутствия специфического кода для гибернации, кто то должен взять на себя ответственность за трансляцию из БД в бизнес-объекты и обратно. Эта трансляция может быть выполнена с помощью связывания через XML файл или связывания атрибутов на классы и свойства.

Используем маппинг файл чтобы не захламлять класс бизнес объекта. Добавьте новый XML файл в проект. XML файл будет использоваться как маппинг файл. Имя файла должно быть Employee.hbm.xml. Файл класса .cs и маппинг файл .hbm.xml должны лежать в одной папке и должно быть одинаково названы.

Код xml файла:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2"
    namespace="NHibernateBasics" assembly="NHibernateBasics">
    <class name="Employee" table="Employee">
        <id name="ID" column="ID">
            <generator class="identity"/>
        </id>
        <property name="Name" column="Name" />
    </class>
</hibernate-mapping>
```

В свойствах XML файла в проекте необходимо выставить свойство Build Action = Embedded Resource.



Добавьте новый конфигурационный файл приложения (app.config).

Скопируйте следующий код.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="hibernate-configuration"
      type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
  </configSections>
  <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    <session-factory>
      <property name="connection.provider">
        NHibernate.Connection.DriverConnectionProvider</property>
      <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
      <property name="query.substitutions">hqlFunction=SQLFUNC</property>
      <property name="connection.driver_class">
        NHibernate.Driver.SqlClientDriver</property>
      <property name="connection.connection_string">
        Data Source=(Local)\SQL2008;Initial Catalog=NHibernateBasics;
        Integrated Security=True</property>
      <property name="show_sql">true</property>
      <mapping assembly="NHibernateBasics" />
    </session-factory>
  </hibernate-configuration>
</configuration>
```

Настройте свойство Connection-string чтобы подключиться к вашей БД. Выставьте свойство CATALOG=, значением NHibernateBasics. Выставьте свойство mapping assembly=, опять же значением NHibernateBasics.

NHibernte гарантирует нам что две ссылки на объект будут указывать на один и тот же объект если ссылки установлены в одной сессии. Если мы сохраним объекты в одной сессии и загрузим в другой, то два объекта будут разными.

Посмотрим код операций:

Загрузка:

```
using (mySession.BeginTransaction())
{
    // Создаем критерию и загружаем данные
    ICriteria criteria = mySession.CreateCriteria<employee>();
    IList<employee> list = criteria.List<employee>();
    for (int i = 0; i < myFinalObjects.Length; i++)
    {
        myFinalObjects[i] = list[i];
        MessageBox.Show("ID: " + myFinalObjects[i].ID + "
                        Name: " + myFinalObjects[i].Name);
    }
    mySession.Transaction.Commit();
}
```

Отображение:

```
using (mySession.BeginTransaction())
{
    ICriteria criteria = mySession.CreateCriteria<employee>();
    IList<employee> list = criteria.List<employee>();
    StringBuilder messageString = new StringBuilder();
    // Load and display the data
    foreach (Employee employee in list)
```

```

    {
        messageString.AppendLine("ID: " + employee.ID + " Name: " + employee.Name);
    }
    MessageBox.Show(messageString.ToString());
}

```

**Сохранение:**

```

using (mySession.BeginTransaction())
{
    // Insert two employees in Database
    mySession.Save(myInitialObjects[0]);
    mySession.Save(myInitialObjects[1]);
    mySession.Transaction.Commit();
}

```

**Удаление:**

```

using (mySession.BeginTransaction())
{
    // Delete one object from Database
    mySession.Delete(myInitialObjects[0]);
    mySession.Transaction.Commit();
}

```

### 1.3 Entity Framework

Способы взаимодействия Entity Framework с БД:

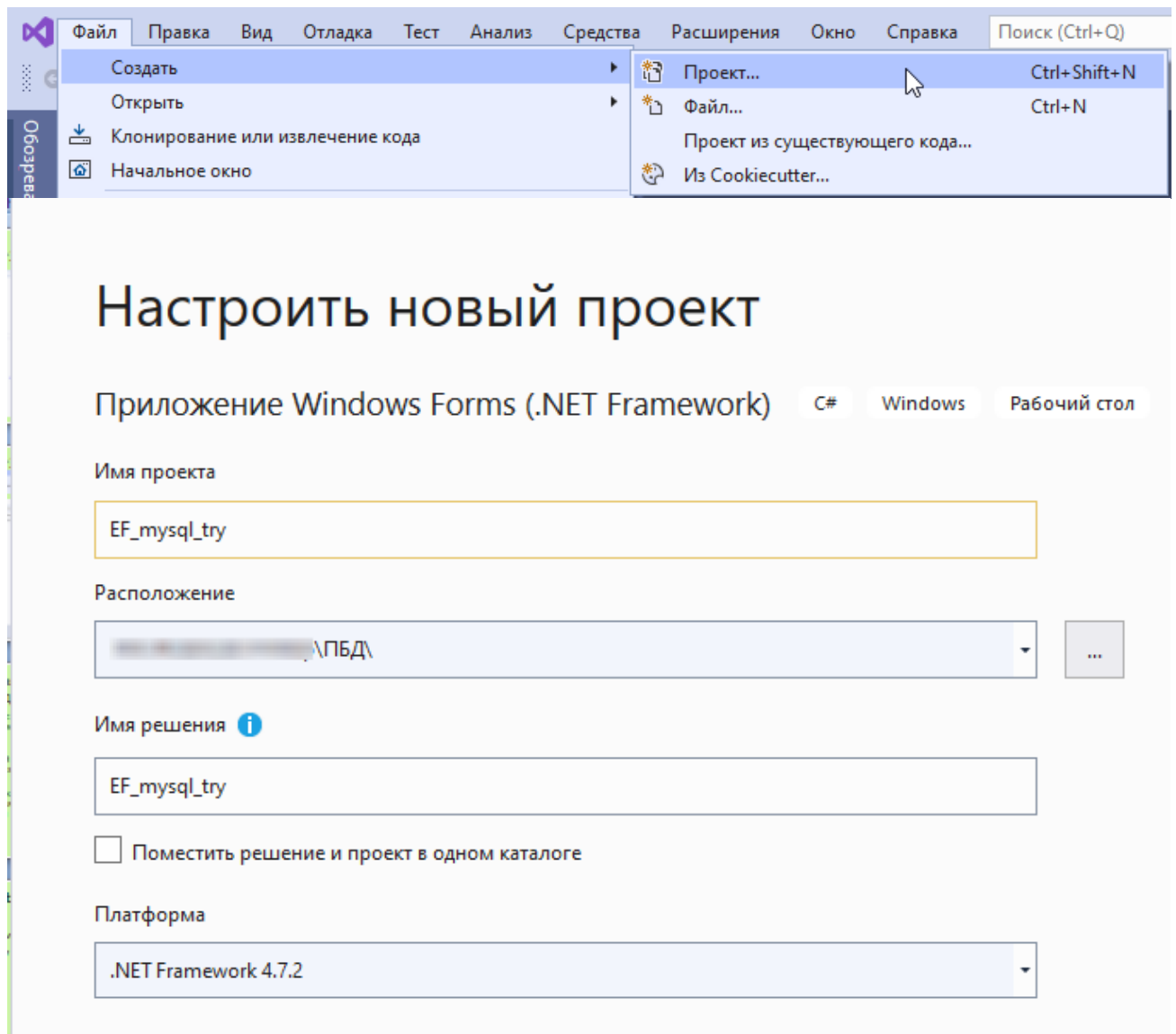
- **Database first:** Entity Framework создает набор классов, которые отражают модель конкретной базы данных
- **Model first:** сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.
- **Code first:** разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы

Для нас актуален **Database first**

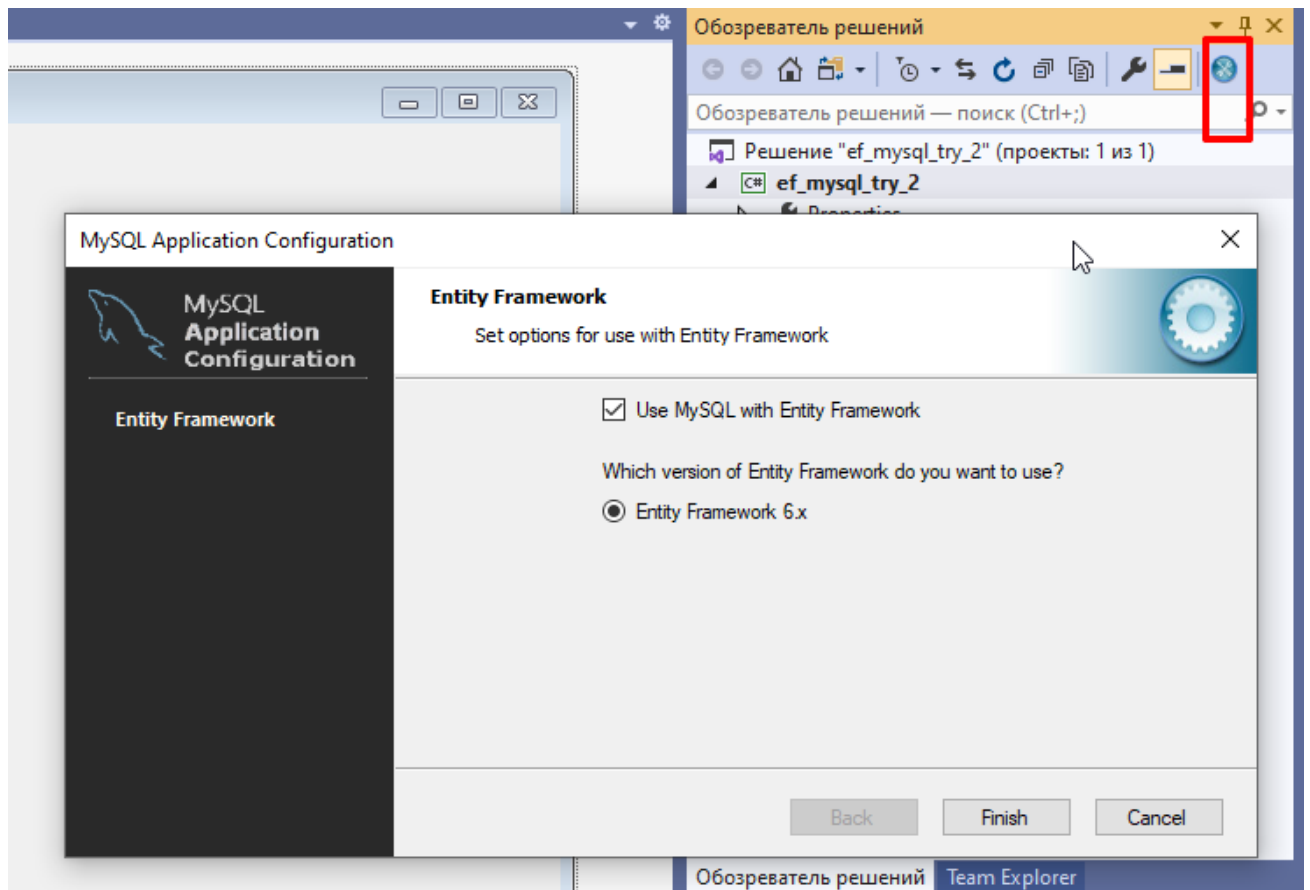
**Пошаговое создание в Visual Studio:**

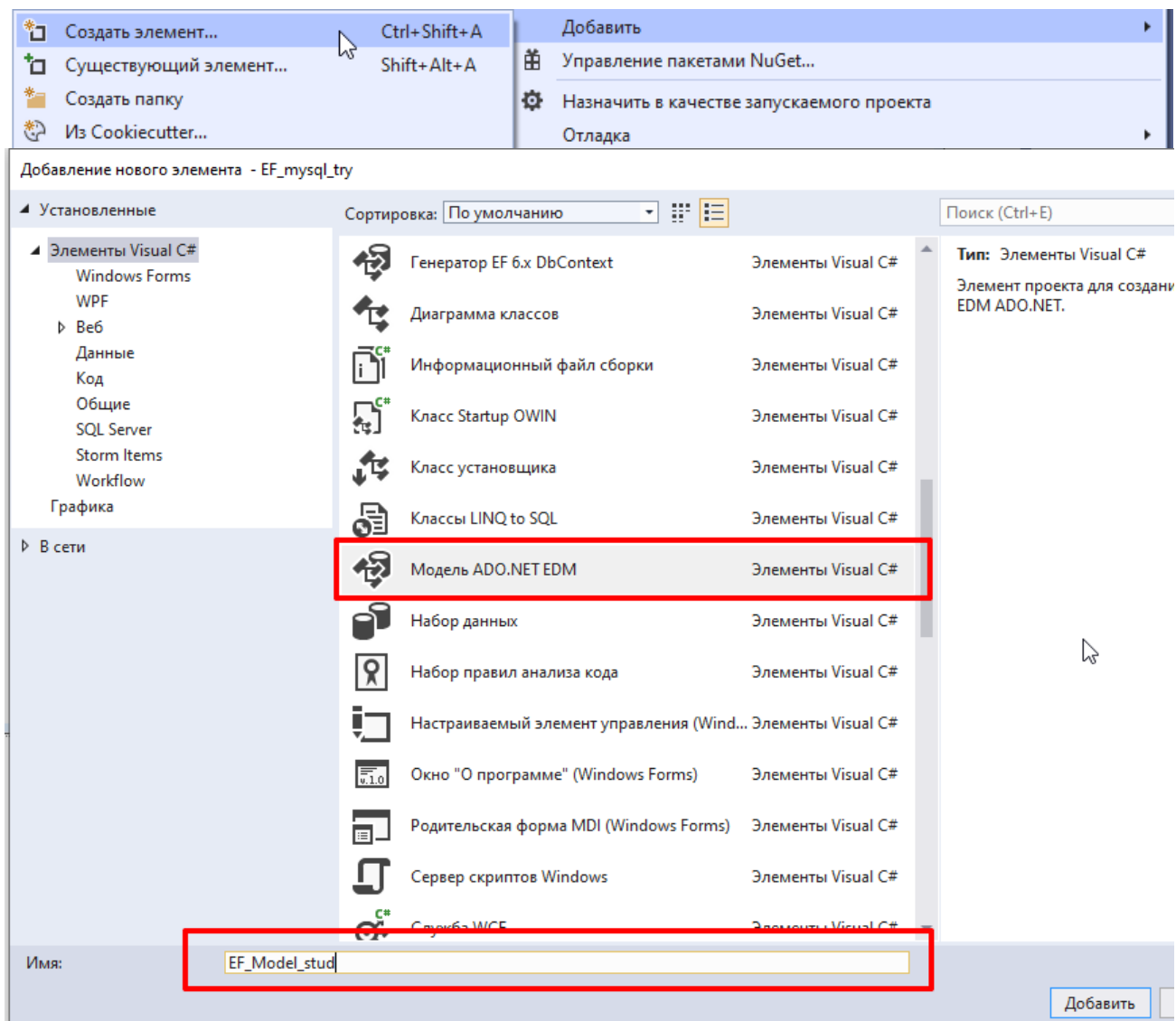
1. Создать проект с поддержкой .Net Framework
2. Добавить элемент Модель ADO.NET EDM (Entity Framework Data Model)
3. Выбрать необходимые таблицы на модели
4. Отобразить элементы класса на форме

1. Создать проект с поддержкой .Net Framework




2. Добавить элемент Модель Ado.Net EDM (Entity Framework Data Model)  
Предварительно в свойствах проекта включить поддержку 6 Entity Framework






Мастер моделей EDM

✕




Выбор содержимого модели


Что должна содержать модель?




Конструктор  
EF из базы  
данных



Пустая  
модель  
конструктора  
EF



Пустая  
модель Code  
First



Code First из  
базы данных

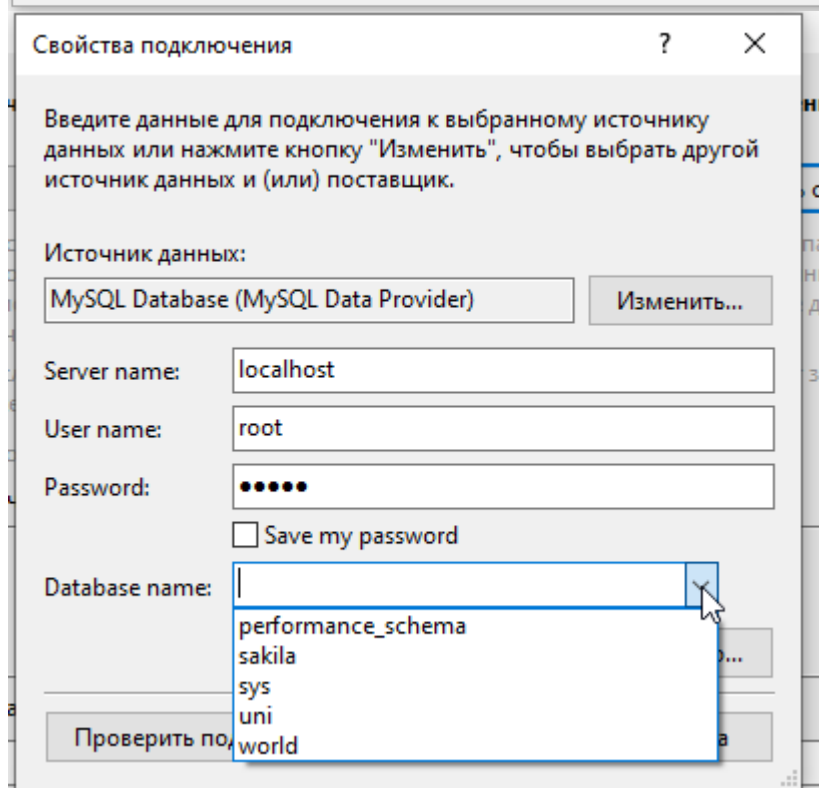
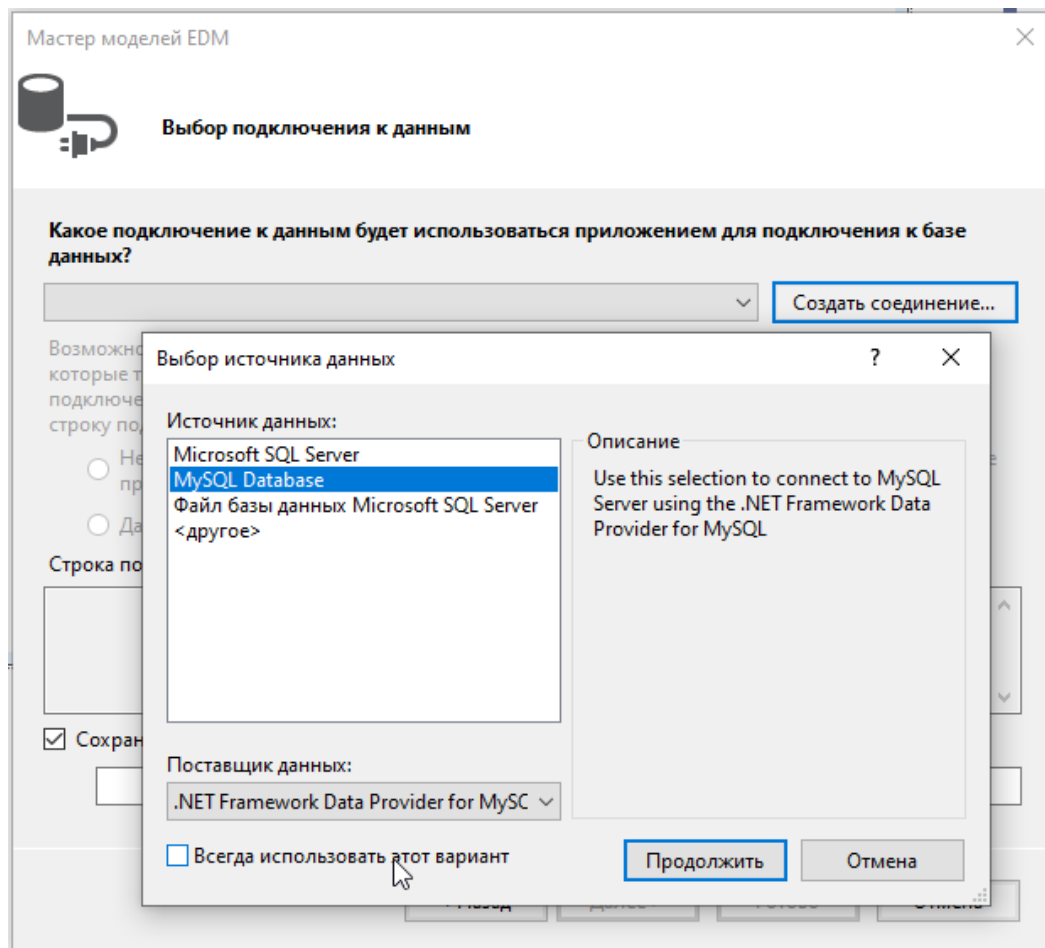
Создает модель в конструкторе EF на основе существующей базы данных. Можно выбрать подключение к базе данных, параметры для модели и объекты базы данных для включения в модель. Классы, с которыми будет взаимодействовать ваше приложение, формируются из этой модели.

< Назад

Далее >

Готово

Отмена



3. Выбрать необходимые таблицы на модели

Мастер моделей EDM

Выберите параметры и объекты базы данных

Какие объекты базы данных нужно включить в модель?

☒ Таблицы

☒ uni

☐ cas\_activities

☐ st\_group

☒ student

☐ student\_activity

☐ trigger\_test

☐ Представления

☐ Хранимые процедуры и функции

☒ Формировать имена объектов во множественном или единственном числе

☒ Включить столбцы внешних ключей в модель

☐ Импортировать выбранные хранимые процедуры и функции в модель сущностей

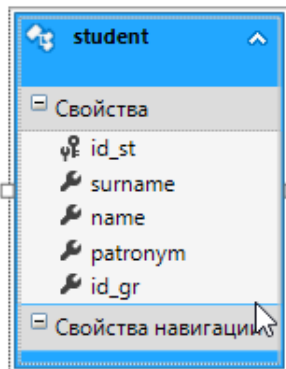
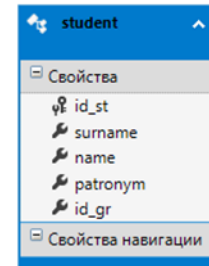
Пространство имен модели:

uniModel

< Назад Далее > Готово Отмена



ef\_Model\_stud.edmx [Diagram1] - Form1.cs [Консоль]



System.Data.DataSetExtensions

System.Deployment

System.Drawing

System.Drawing.Design

System.Management

System.Net.Http

Обозреватель решений Team Explorer

Свойства

uniModel.student EntityType

Диаграмма

Цвет заливки 0; 122; 204

Общие

Базовый тип (Нет)

Документация

Имя student

Имя набора сущностей students

Создание кода

Абстрактный Значение false

Доступ Public

Имя

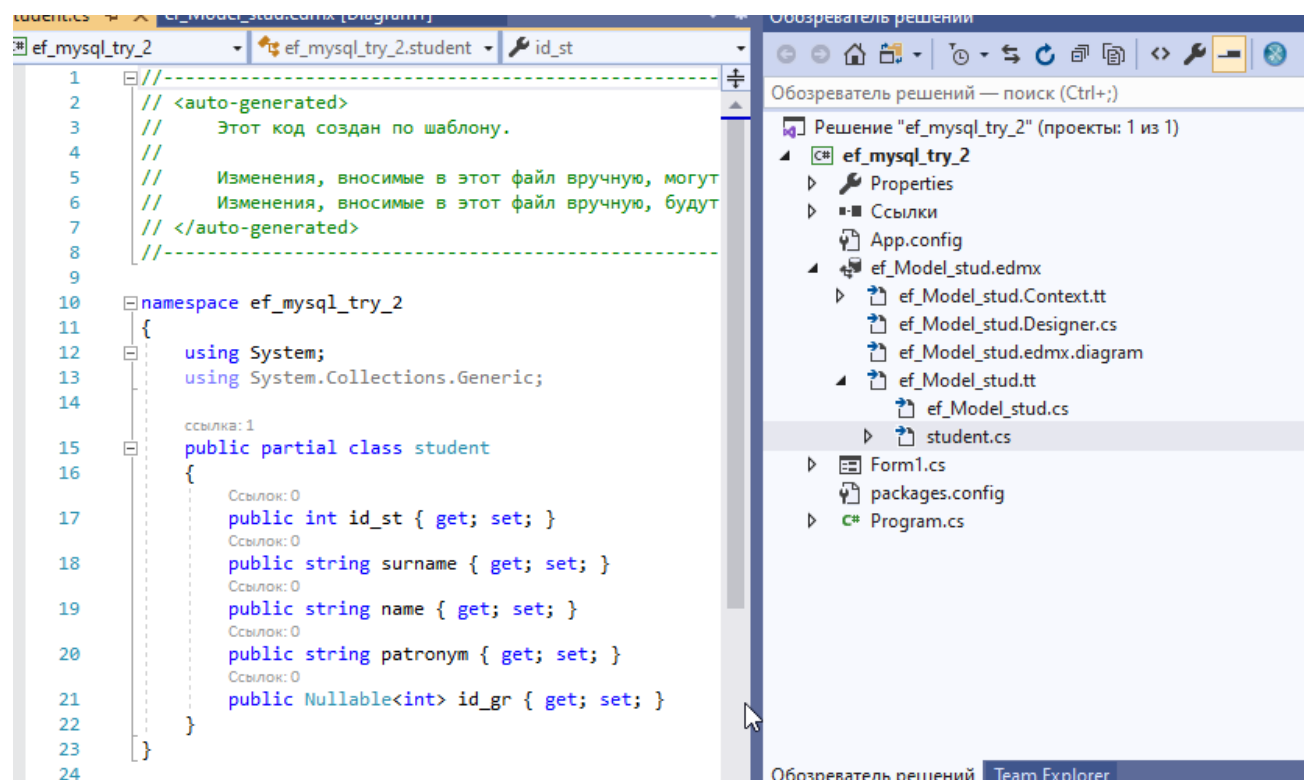
Имя сущности.

debug\ef\_mysql\_try\_2.exe

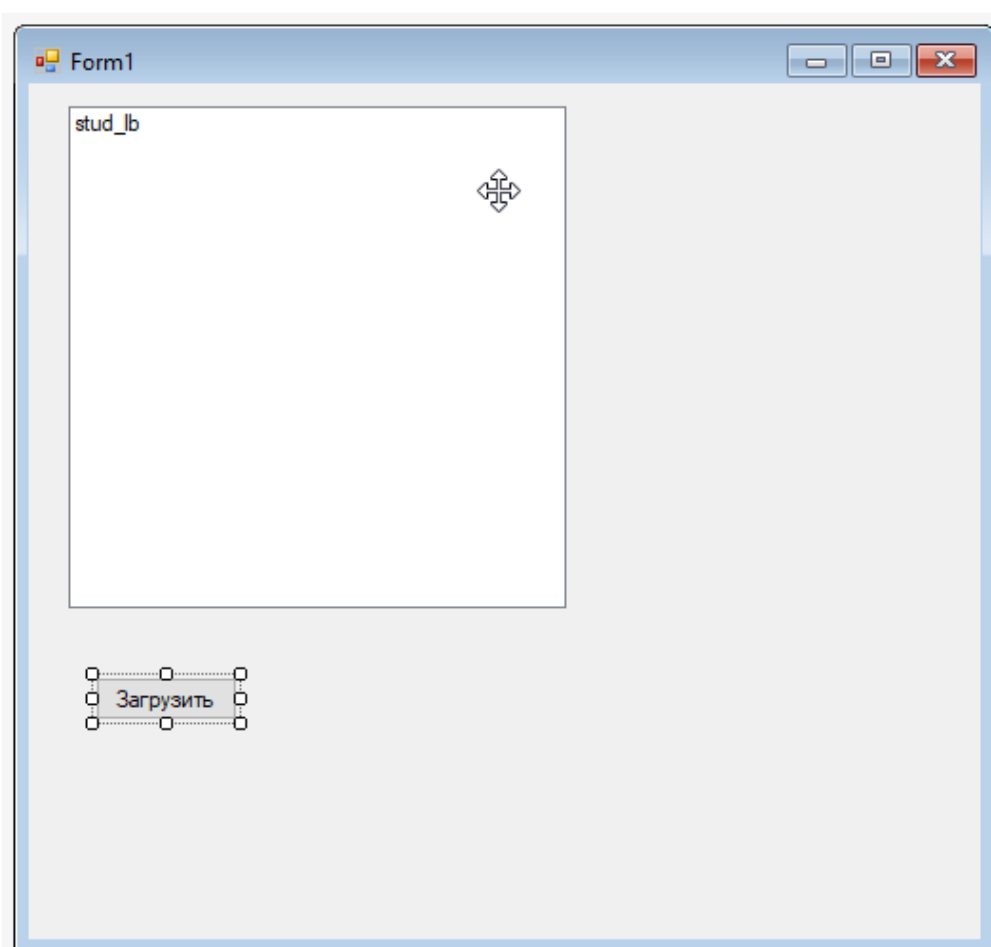
====

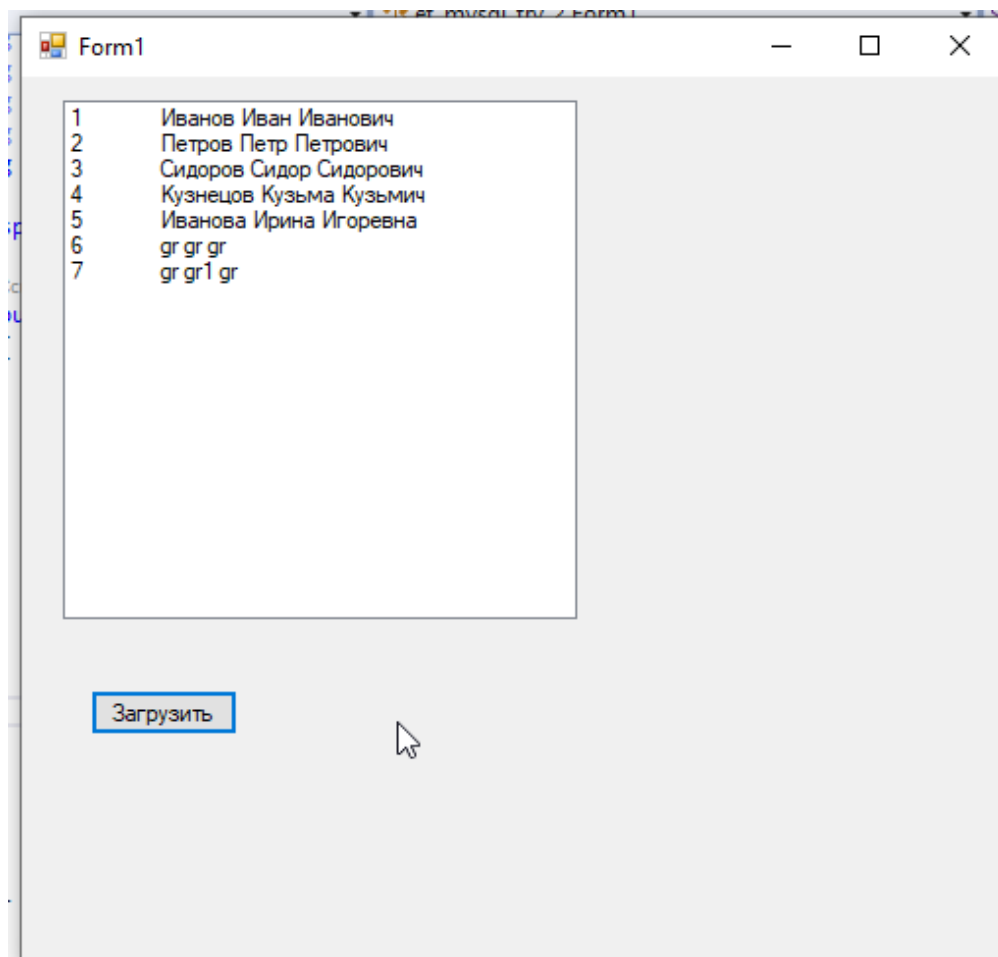
'False'





4. Отобразить элементы класса на форме





## 2. Выполнение лабораторной работы

Создать приложение –форму с 2 способами подсоединения приложения к базе данных и: с двумя способами подсоединения базы данных:

1)Соединение через компоненты ADO.NET или его аналога с помощью строки связи. При этом отображаться должны 2 таблицы/элемента данных (не обязательно в форме таблицы, может быть просто текст): одна через DataGridView ,с подключением источника данных, а другая через чтение результатов запроса SqlCommand в цикле.

2) С использованием технологии ORM (Object-Relational Mapping) с отображение третьей таблицы /элемента данных

## 3. Содержание отчета

Содержание отчета:

- Текст задания.
- физическую модель БД.

- код приложения с подключением через ADO.net
- код приложения, включая классы, с подключением с использованием технологии ORM
- код xml файла для привязки классов к объектам
- Скриншоты приложения

#### **4. Варианты заданий**

Варианты заданий приведены в Приложении 2.

## Приложение 1 Распределение баллов

### Семестр 6

№	Наименование лабораторной	Количество баллов
1.	Разработка физической модели базы данных с учетом декларативной ссылочной целостности	5
2.	Создание и модификация базы данных и таблиц базы данных	4
3.	Заполнение таблиц и модификация данных	5
4.	Разработка SQL запросов: виды соединений и шаблоны	7
5.	Разработка SQL запросов: запросы с подзапросами	7
6.	Хранимые процедуры	7
7.	Контрольная работа. Проектирование взаимодействия базы данных и приложения	20
8.	Работа на практических занятиях	5
	Итого	60

## Приложение 2 Варианты заданий лабораторных 1-7 и контрольной работы

1. Программа для рисования графов: название вершины, координаты левой верхней точки отображаемой вершины, и её размеры, автор графа, пользователи, которым можно просматривать или редактировать граф
  - а. Вершины, название/текст которых содержит слово «начало», но не начинается с него
  - б. Вершины, у которых нет исходящих ребер
  - в. Графы, в которых есть пара вершин, связанных ребрами в обе стороны
  - г. Граф, ширина которых превышает 500 пикселей. Ширина графа в пикселях (от максимальной сумма координаты по горизонтали с шириной отнять минимальную левую координату)
  - д. авторы графов, графы которых может просматривать наибольшее число пользователей
  - е. Вершины, для которых есть исходящие ребра, ведущие ко всем остальным вершинам её графа
  - ж. Вершина, у которой нет входящих ребер от вершины со словом «выход»
2. Садоводство: участки, владельцы с учетом совместной собственности, линии/номер участка, площадь стоимость постройки, тип построек ,взносы в фонд садоводства
  - а. номера участков владельцев с отчеством, заканчивающимся на «ич»
  - б. участки, на которых зарегистрировано более 1 типа постройки

- в. тип построек, которые отсутствуют на участках
  - г. Владелец (владельцы) участка максимальной площади
  - д. Владельцы участков с максимальным числом типов построек
  - е. Владельцы, оплатившие все типы взносов
  - ж. Участки, на которых нет бань, но есть туалеты
3. ТСЖ: квартира, владелец, количество комнат, площадь, жилая площадь, этаж, показания счетчиков
- а. квартиры владельцев, отчества которых заканчиваются на «на»
  - б. владельцев, у которых есть квартиры с разным количеством комнат
  - в. этажи, на которых нет квартир
  - г. владельцы квартир с максимальной жилой площадью
  - д. этаж, на котором меньше всего квартир находится в собственности
  - е. этаж квартиры, в которых стоят все типы счетчиков
  - ж. владелец, у которого нет двухкомнатных квартир, но есть трехкомнатные
4. парк: деревья, породы, дата высадки, дата обрезки, расположение, аллеи
- а. аллеи, на которых встречаются дубы (дуб в названии породы дерева)
  - б. деревья, стоящие на перекрестке аллей (1 дерево на 2 аллеях)
  - в. породы, не высаженные в парке
  - г. дерево, которое было посажено позже всех
  - д. порода, деревья которой обрезали меньше всего
  - е. порода дерева, встречающаяся на всех аллеях заданного парка
  - ж. аллея, на которой растут липы, но нет клёнов
5. расписание экзаменов/зачетов: даты экзаменов, дисциплина, преподаватель, группа, аудитория
- а. аудитории, которых проходят экзамены по дисциплинам, начинающимся со слова «систем», но это не единственное слово названия
  - б. дата, когда в одной аудитории проходит несколько экзаменов
  - в. дисциплины, по которым не бывает экзаменов
  - г. Преподаватели, принимающие самые первые экзамены
  - д. группа, у которой меньше всего экзаменов
  - е. дисциплина, по которой есть экзамены у всех групп
  - ж. дата, в которую не проходят экзамены у группы Z0431
6. поликлиника: прием, пациент, процедура/прием, врач, стоимость
- а. пациенты, приходившие на любые процедуры, связанные с лазером (лазер в любом месте названия)
  - б. врач, проводивший одному пациенту и прием и процедуру в одном кабинете

- в. процедуры, которые никому не делали
- г. процедуры, которые проводили Иванову Ивану Ивановичу, с наименьшей стоимостью среди всех процедур
- д. кабинет, в котором проводилось больше всего приемов
- е. врач, проводивший все процедуры
- ж. врач, не принимавший Иванова Петра, но проводивший ему процедуру.

7. личный кабинет: дисциплина, работа, тип работы, студент, преподаватель, дата сдачи , баллы

- а. преподаватели, за которыми закреплены дисциплины, начинающиеся со слова «систем»
- б. дисциплина, по которой есть несколько типов работ
- в. задание, по которому не прикреплено ни одной работы
- г. студент, сдавший лабораторную позже всех
- д. группа, прикрепившая в этом году наибольшее число работ
- е. дисциплина, по которой есть все типы работ (КР, ЛР, практические)
- ж. преподаватель, которому не прикрепляли отчетов по курсовым работам

8. костюмерная театра: роль, спектакль, название костюма, деталь костюма, размер, автор модели, дата разработки

- а. роли, в которых используются костюмы, имеющие в названии слово «принц»
- б. костюм, в котором есть и плащ и штаны
- в. автор, у которого пока нет костюмов
- г. спектакль, к которой разрабатывался самый старый из костюмов
- д. автор, разработавший костюм с наибольшим числом деталей
- е. костюм, в котором есть все типы деталей
- ж. автор, не разрабатывавший костюмы к «Валькирии», но разрабатывавший к «Мастеру и Маргарите»

9. охраняемые парковки: адрес парковки, машина, владелец, место, рег. номер машины, дата и время заезда, дата и время выезда

- а. все парковки, расположенные на улицах, в названии которых есть слово «Большая»
- б. владелец машины, у которого несколько машин разных марок
- в. машины, которые не парковались на парковках
- г. парковка, открывающаяся раньше всех
- д. владелец машины, останавливавшийся на минимальном числе парковок
- е. машина, которая стояла на всех парковках Адмиралтейского района
- ж. владелец, не парковавшийся на Вознесенском проспекте, но парковавшийся на Московском

10. спортзал: абонементы, виды спорта, тренеры, дата покупки абонемента, тип абонемента

- а. виды спорта, на которые есть абонементы со словом «утренний» в названии типа)
  - б. тренер, который ведет занятия по нескольким различным видам спорта
  - в. вид спорта, по которому нет тренеров
  - г. тренеры, ведущие занятия по виду спорта числом занятий в предлагаемом абонементе меньше среднего
  - д. вид спорта, по которому занятия ведет максимальное количество тренеров
  - е. дата, когда не покупали абонементы на дзюдо
  - ж. совокупный абонемент, на все виды латинских танцев.
11. База данных по теме «сбор в поездку» должна иметь структуру, позволяющую реализовать следующие запросы:
- а. Найти какие поездки за текущий год имели в названии слово «поход», но не заканчивались на него
  - б. Найти категорию без предметов
  - в. Найти в какую поездку брались и очки, и купальные принадлежности
  - г. Найти пользователя, у которого в текущем квартале больше 2 поездок
  - д. Какие вещи берутся в поездки всех типов, куда ездил заданный пользователь
  - е. Какие категории вещей не берутся в поездку типа «конференция»
  - ж. Найти тип(ы) поездок с наибольшим количеством поездок.
12. База данных сериалов.
- а. Сериал, в названии которого есть слово доктор, но оно не первое
  - б. Персонаж, которого в разных эпизодах 1 сезона играют разные актеры
  - в. Эпизод, в котором не заявлено ни одного режиссера.
  - г. Сезон и сериал этого сезона, с максимальным количеством эпизодов
  - д. Актер, играющий в самом старом сериале
  - е. Эпизод, в котором играют все актеры сериала
  - ж. Актер, который играл в комедийных сериалах, но не в исторических
13. вакансии: название вакансии, организация работодатель, адрес работодателя, диапазон зарплаты, требования к образованию, Обязанности, график работы, требования обязательные, желательные, дата выставления вакансии
- а. вакансии, имеющие в названии «база данных», но не заканчивающиеся на него
  - б. требование, не присутствующее ни в одной вакансии
  - в. работодатели в Москве, выставившие вакансии и программиста Java и системного администратора
  - г. вакансия с зарплатой выше среднего
  - д. вакансии с максимальным количеством необязательных требований
  - е. требование, присутствующее во всех вакансиях на программиста (любые вакансии со словом программист)
  - ж. вакансии, которые есть в Санкт-Петербурге или Москве, но которых нет в Пскове

14. калькулятор бюджета физического лица: категория дохода (продажа, зарплата), категория расхода (еда, счета за КУ, здоровье ...), статьи дохода и расхода, дата расхода/дохода. Категория- более общее понятие чем статья. Например категория- еда, а статьи в ней мясо, рыба, вкусное к чаю, а конкретный расход «печенье «журабье» к чаю 11.09»
- а. все расходы категорий, которые относятся к медицине (содержат часть слова «мед», но не являются мёдом)
  - б. месяц, в котором были статьи дохода от работы и от аренды (сдачи в аренду)
  - в. категория доходов, по которой нет статей
  - г. категория, по которой были наибольшие расходы в прошлом году
  - д. категория, по которой не было расходов в январе, но были в феврале
  - е. категория расхода, по которой траты были во всех месяцах 2022 года
  - ж. месяц, в котором были расходы минимального количества статей
15. Книга контактов: люди, метки (друзья, коллеги, семья), дни рождения, телефон, электронная почта, примечания к человеку или контакту, праздники, которые празднуют люди
- а. друзья, у которых номер телефона начинается на 8-921
  - б. Люди (контакты) без электронной почты
  - в. люди, которые празднуют и Новый год и Рождество
  - г. самые молодые люди среди контактов
  - д. месяц, когда есть праздники у коллег, но нет у друзей
  - е. люди, к которым относится максимальное меток
  - ж. месяц, в котором есть дни рождения у людей в совокупности, относящихся ко всем меткам
16. вузы для абитуриента: город, вуз, факультеты, направления, направленности, ЕГЭ которые нужно сдать, дата начала/конца приемной кампании.  
(Направление -09.03.04 «Программная инженерия», Направленность — его конкретизация «Разработка программно-информационных систем», именно направленность закреплена за кафедрой и соответственно факультетом)
- а. направленности, в которых есть слово «разработка», но оно не последнее
  - б. факультет, не принимающий ни на одну направленность
  - в. направление, на которое надо сдавать математику и физику
  - г. вуз, принимающий на количество направлений больше среднего
  - д. город, в котором есть все направления
  - е. вуз, с самым поздним концом приемной кампании
  - ж. направление, на которое не надо сдавать ЕГЭ по информатике, но надо по физике
17. школьные экскурсии: тип (развлекательная/образовательная), дисциплины к которым имеет отношение образовательная экскурсия, стоимость с человека, список участников, ответственный за проведение учитель
- а. экскурсии на заводы (слово «завод» в любом месте названия)
  - б. экскурсии, относящиеся к биологии и истории



- в. учащиеся, которые не ездили за экскурсии
  - г. учащиеся, которые не ездили в музей истории религии, но ездил в Кунсткамеру
  - д. учитель, отвечавший за на наибольшее число экскурсий
  - е. учителя, отвечающие за самые дешевые экскурсии
  - ж. учащийся, который был на всех экскурсиях
18. собачий питомник: собака, владелец, порода, родители, медали с выставок, дата рождения
- а. породы, названия которых содержат слово «шнауцер», но не заканчивается на него
  - б. собаки без медалей
  - в. собаки, у родителей которых один и тот же владелец
  - г. владелец, у которого есть собаки всех пород на букву «к»
  - д. владелец, у которого есть овчарки, но нет колли
  - е. собака, имеющая больше среднего медалей
  - ж. владельцы самых молодых собак
19. служба доставки: адрес доставки, контактное лицо, стоимость посылки, диапазон желаемого времени доставки. время доставки фактическое, вес посылки, отметка о доставке, фирма отправитель
- а. все посылки, отправляемые на улицу, в названии которой есть окончание «гаг», но это не единственные буквы в нем
  - б. улица, на которую никогда не доставляли посылки
  - в. контактное лицо, получавшее посылки на улице и на проспекте
  - г. фирма, отправившая меньше всего посылок
  - д. посылка с самым большим весом
  - е. контактное лицо, получавшее посылки от всех фирм
  - ж. контактное лицо, никогда не получавшее посылок в мае, и получавшее посылки в прошлом году
20. туристический путеводитель: город, достопримечательность, адрес, тип достопримечательности (памятник, архитектурный комплекс, природный комплекс), дата создания
- а. достопримечательности, в которых есть слово «Кремль», но с него название не начинается
  - б. город без улиц
  - в. улица, на которой есть и памятники, и архитектурные комплексы
  - г. город, в котором нет памятников, но есть музеи
  - д. улица, на которой больше всего музеев
  - е. улица с самыми молодыми достопримечательностями
  - ж. тип достопримечательностей, который есть во всех городах где есть достопримечательности
21. метрополитен: линия, станция, время закрытия, время открытия, адрес выходов, время проезда между станциями, дата открытия станции

- а. станции, в названии которых есть окончание «ая», но оно на него не заканчивается
- б. станция без выходов
- в. пересадочные станции с линии 2 на линию 3
- г. линия с временем проезда меньше среднего
- д. линия, на которой собраны все станции, на букву «а»
- е. станция с самым поздним закрытием
- ж. линия, на которой нет перегона меньше 2 минут

22. Медпункт организации: сотрудник организации, данные флюорографии (результат, дата), данные прививок, жалобы сотрудников (прием в медпункте)

- а. все сотрудники, мужского пола (отчество, оканчивается на «ич»)
- б. сотрудники, никогда не делавшие флюорографию
- в. сотрудники организации, делавшие за последний год прививки от гриппа и ковид-19
- г. сотрудник, последним сделавший флюорографию в прошлом году
- д. сотрудник, у которого сделаны все типы прививок
- е. сотрудник с самым большим количеством жалоб
- ж. сотрудник, не делавший прививки от кори, но делавший от энцефалита

23. Инвентаризация: ответственное лицо, оборудование, категории имущества (компьютерная техника, канцелярские товары, мебель, освещение, спец. оборудование и т.д.), помещения, дата поставки, дата списания

- а. любые помещения, где стоит специализированные тестовые стенды (оборудование, содержащее слова «стенд» , но не начинается с него.
- б. Мат. ответственное лицо, за которым пока не числится никакого имущества
- в. помещение, в котором есть и компьютерная техника и специальное оборудование
- г. имущество, списанное последним в этом году
- д. категория оборудования, которая стоит во всех помещениях
- е. мат. ответственное лицо, ответственное за наименьшее количество имущества
- ж. мат ответственное лицо, ответственное за мебель, но не за компьютерную технику

24. Календарь корпоративных мероприятий: время дата, событие, статус события для участника (обязательное, рекомендуемое, нейтральное), должности и подразделения участников, комнаты проведения

- а. Все мероприятия, прошлого года, в названии которых есть слово «встреча», но на него с названия не заканчивается
- б. Событие, которому пока не назначили участников
- в. Комната, в которой проходят 2 мероприятия разных организаторов в один день
- г. Участники самых поздно заканчивающихся мероприятий
- д. Событие, с самым большим количеством не пришедших, но приглашенных на него участников.(участники на мероприятие могут быть приглашены и при этом могут прийти/не прийти).

- е. Мероприятие, в котором обязательно присутствие всех сотрудников отдела кадров
  - ж. Сотрудник, не посещавший мероприятий в июле прошлого года, но посещавший в мае этого.
25. Распределение обязанностей: обязанность, название, частота выполнения, категория обязанности (домашняя, оплата счетов, посещение мероприятий, общественная)
- а. Люди, выполняющие обязанности, имеющие в названии слово «поливать», но не заканчивающиеся на него
  - б. Человек без обязанностей
  - в. Человек, у которого есть обязанности с различной частотой выполнения (например ежедневные и ежемесячные)
  - г. Ежедневная обязанность, которая выполняется первой по дате (дню)
  - д. Частота выполнения обязанностей, обязанности из которой выполняют все, кто вообще имеет обязанности
  - е. Категория, в которой обязанностей больше среднего
  - ж. Человек, который не выполняет обязанности из категории «оплата счетов», но выполняет из категории «уборка»
26. Багтрекинг: проекты, баги, тестировщики, разработчики (тестировщики не исправляют)
- а. Баги, в названии которых содержится слово «память», но оно не последнее
  - б. Баги, найденные одним тестировщиком и исправляемые разными разработчиками
  - в. Проект без багов
  - г. Критический баг, который добавили последним
  - д. Тестер, с самым большим количеством добавленных багов
  - е. Разработчик, у которого нет неисправленных багов
  - ж. Тестировщик, работавший со всеми проектами.
27. Домашняя аптечка. Группа лекарств, срок(дата) годности.
- а. Найти лекарство, которое содержит текст «септ», но не в начале слова
  - б. Найти лекарство, у которого есть аналоги по действующему веществу как в форме таблеток, так и в форме спрея.
  - в. Найти группу лекарств, в которой нет ни одного лекарства
  - г. Найти лекарство с самым близким, но не истекшим сроком годности.
  - д. Найти группы лекарств, с количеством лекарств больше среднего.
  - е. Найти действующее вещество, для которого есть лекарства во всех формах.
  - ж. Найти группу лекарств, в которой нет таблеток с не истекшим сроком годности, но есть лекарства в других формах с не истекшим сроком годности.