**1. Разработка баз данных в СУБД MySQL**

**1.1. Построение EDR-модели**

Существует некоторое хранилище данных, получающее запросы на вывод информации преподавателю и запрос на ввод информации преподавателем. Учебный план, а именно количество, затрачиваемых часов, подтверждается директором. Преподаватель заносит новую информацию. При поступлении в хранилище, необходимая информация фиксируется. Преподаватель ежемесячно готовит отчёты за определённый период времени о выполнении учебного плана, каждой группой в отдельности и их успеваемости.

В процессе проектирования ERD модели БД были выделены следующий сущности:

Group, discipline, material, test, theory, practicalwork, homework.

Описание сущностей:

Group – хранит список групп.

Discipline – хранит дисциплины, которые ведёт преподаватель

Material – хранит материалы, утверждённые учебным планом.

Test – хранит данные о проводимых тестах.

Theory – хранит теоретический материал по разделу.

Practicalwork – хранит практические работы по теории.

Homework – хранит домашние задания.

Описание атрибутов сущностей:

|  |  |  |  |
| --- | --- | --- | --- |
| **Group** | | | |
| Поле | Тип | Атрибуты | Описание |
| idGroup | INT | PK NN AI | Уникальный номер группы |
| Name | VARCHAR(100) |  | Название группы |

|  |  |  |  |
| --- | --- | --- | --- |
| **Discipline** | | | |
| Поле | Тип | Атрибуты | Описание |
| idDiscipline | INT | PK NN AI | Уникальный номер дисциплины |
| Name | VARCHAR(100) |  | Наименование дисциплины |

|  |  |  |  |
| --- | --- | --- | --- |
| **Material** | | | |
| Поле | Тип | Атрибуты | Описание |
| idMaterial | INT | PK NN AI | Уникальный номер материала |
| Section | VARCHAR(100) |  | Раздел |
| NumberOfHours | INT |  | Количество выделяемых часов |
| NumberOfPracties | INT |  | Количество часов практики |
| NumberOfTheories | INT |  | Количество часов теории |
| idGroup | INT | NN | Номер группы |
| idDiscipline | INT | NN | Номер дисциплины |
| DateOfDelivery | DATE |  | Дата сдачи материала |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | | | |
| Поле | Тип | Атрибуты | Описание |
| idTest | INT | PK NN AI | Уникальный номер теста |
| Name | VARCHAR(100) |  | Наименование теста |
| Mark | INT |  | Оценка |
| idMaterial | INT | NN | Номер материала |

|  |  |  |  |
| --- | --- | --- | --- |
| **Theory** | | | |
| Поле | Тип | Атрибуты | Описание |
| idTheory | INT | PK NN AI | Уникальный номер теории |
| Material\_idMaterial | INT | NN | Номер материала |
| Topic | VARCHAR(100) |  | Тема раздела |
| View | VARCHAR(100) |  | Вид раздела |
| Type | VARCHAR(100) |  | Тип раздела |

|  |  |  |  |
| --- | --- | --- | --- |
| **Practicalwork** | | | |
| Поле | Тип | Атрибуты | Описание |
| idPracticalWork | INT | PK NN AI | Уникальный номер технических данных |
| idTheory | INT | NN | Номер теории |
| Title | VARCHAR(45) |  | Название практики |
| Mark | INT |  | Оценка |

|  |  |  |  |
| --- | --- | --- | --- |
| **Homework** | | | |
| Поле | Тип | Атрибуты | Описание |
| idHomework | INT | PK NN AI | Уникальный номер домашнего задания |
| idTheory | INT | NN | Номер теории |
| idPracticalWork | INT | NN | Номер практики |
| Title | VARCHAR(100) |  | Название |
| Execution | INT |  | Выполнение |

Для соединения таблиц использовались следующие типы связей:

 - один ко многим вида: not-**identifying relationship**

**Данный вид связи свидетельствует о том, что данные подчинённой таблицы могут существовать без связи с главной таблицей.**

**После того как были установлены все отношения между сущностями получили следующую модель данных: см.** [Приложение 1](#_Приложение_1)**.**

На основе полученной модели была создана БД средствами мастера. По итогу работы мастера был получен следующий SQL код: см. [Приложение 2](#_Приложение_2._SQL).

**1.2. Заполнение базы данных**

Следующим шагом необходимо заполнить БД:

Таблица group

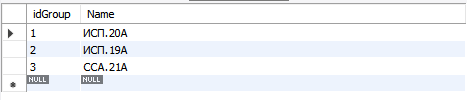


Таблица discipline

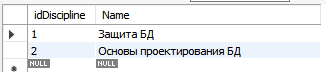


Таблица material

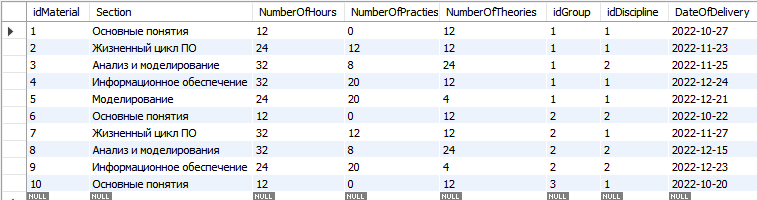


Таблица test

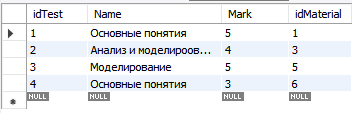


Таблица theory

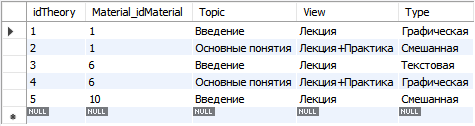


Таблица practicalwork

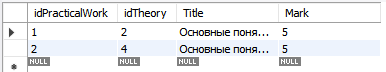
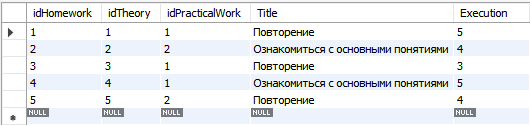


Таблица homework



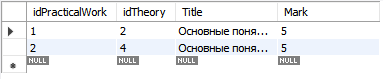
При заполнении таблиц очень важна последовательность: сперва заполняются главные таблицы, затем подчиненные.

**1.3. Написание запросов по работе с данными**

**1.3.1. Выборка и выборка с условием**

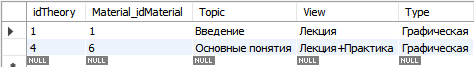
Выбрать данные из одной таблицы.

select \* from practicalwork;



Выбрать данные из одной таблицы по условию WHERE

select \* from theory where Type='Графическая';



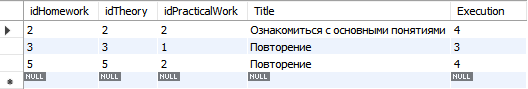
Выбрать данные с применением команды DISTINCT

select distinct View from theory;



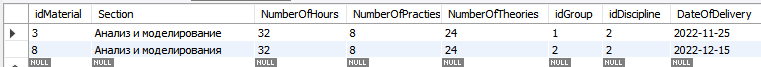
Выбрать данные из таблицы по условию WHERE с применением оператора вхождения в коллекцию – IN

select \* from homework where Execution in (3,4);



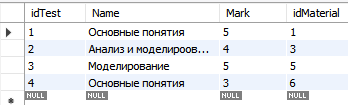
Выбрать данные из таблицы по условию WHERE с использованием оператора Like

select \* from material where Section like "А%";



Выберите данные из таблицы с применением проверки данных на NULL (IS NULL, IS NOT NULL)

select \* from test where Mark is not null;

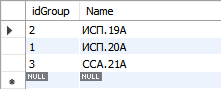


select \* from test where Mark is null;



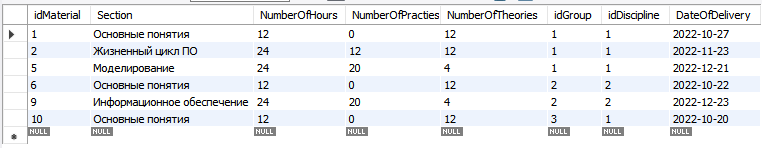
Выберите данные из таблицы с применением сортировки по выбранным полям

select \* from `group` order by `Name`;

****

Выбрать данные с применением оператора BETWEEN

select \* from material where NumberOfHours between 12 and 24;



**1.3.2. Агрегатные функции**

Avg

select round(avg(Mark), 2) as 'Средняя оценка за тесты' from test;



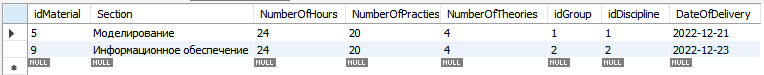
Sum

select sum(NumberOfHours) as 'Суммарное кол-во часов материала первой группы' from material where idGroup = 1;



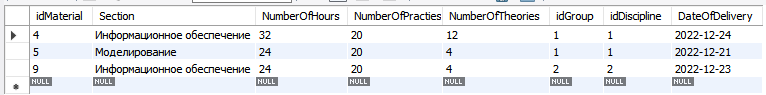
Min

select \* from material where NumberOfTheories = (select min(NumberOfTheories) from material);



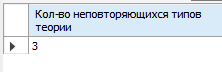
Max

select \* from material where NumberOfPracties = (select max(NumberOfPracties) from material);



Count

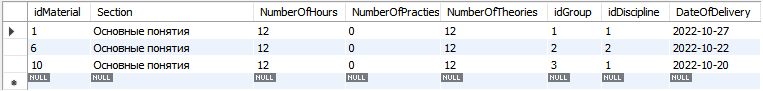
select count(distinct Type) as 'Кол-во неповторяющихся типов теории' from theory;

****

**1.3.3. Группировка**

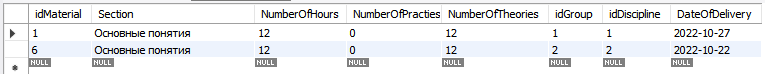
GROUP BY

select \* from material group by idGroup;



GROUP BY с использованием условия HAVING

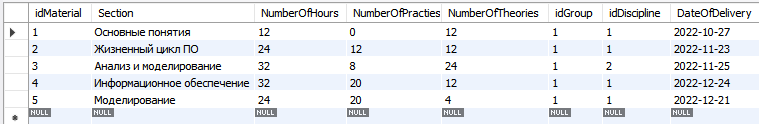
select \* from material group by idGroup having count(\*) > 1;



**1.3.4. Подзапросы**

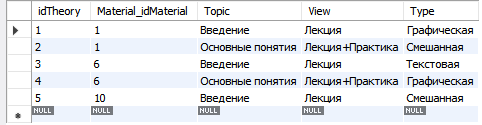
Первый подзапрос

select \* from material where idGroup = (select idGroup from `group` where `Name`='ИСП.20А');



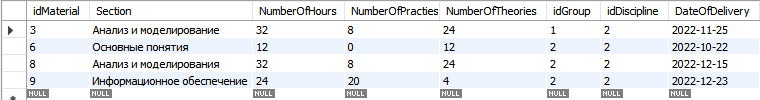
Второй подзапрос

select \* from theory where Material\_idMaterial in (select idMaterial from material where Section='Основные понятия');



Третий подзапрос

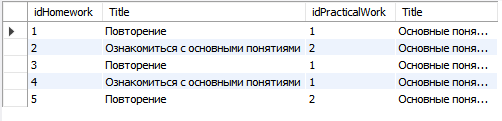
select \* from material where idDiscipline = (select idDiscipline from discipline where `Name`='Основы проектирования БД');



**1.3.5. Соединение таблиц**

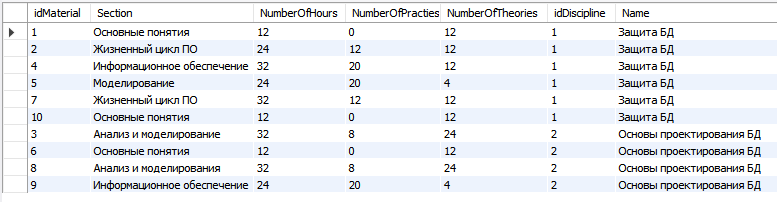
Inner Join

select H.idHomework, H.Title, P.idPracticalWork, P.Title from homework H inner join practicalwork P on P.idPracticalWork = H.idPracticalWork;



Left | Right Join

select M.idMaterial, M.Section, M.NumberOfHours, M.NumberOfPracties, M.NumberOfTheories, D.idDiscipline, D.`Name` from material M right join discipline D on D.idDiscipline = M.idDiscipline;

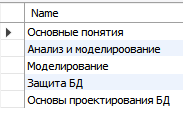


Union

select `Name` from test

union

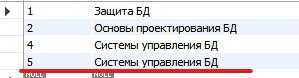
select `Name` from discipline;



**1.3.6. Запросы на добавление, удаление, изменение данных**

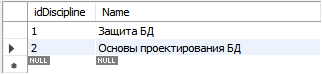
Запрос на добавление

insert into discipline (`Name`) values('Системы управления БД');



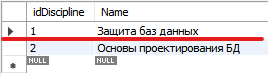
Запрос на удаление

delete from discipline where `Name`='Системы управления БД';



Запрос на обновление

update discipline set `Name`='Защита баз данных' where `Name`='Защита БД';



**1.3.7. Встроенные функции**

Условие CASE

select idHomework, Title, Execution,

case

when Execution = 5

then 'Отлично'

when Execution = 4

then 'Хорошо'

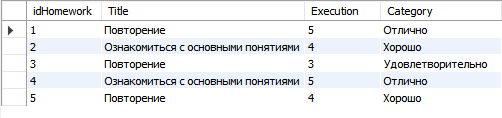
when Execution = 3

then 'Удовлетворительно'

else 'Неудовлетворительно'

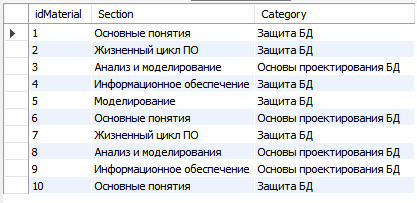
end as 'Category'

from homework;



Условие IF

SELECT \*, IF(Color=3,"Красный","Другой цвет") AS 'Color' FROM auto



**2. Работа со встроенными объектами СУБД MySQL**

**2.1. Работа с процедурами, функциями и представлениями**

**2.1.1. Основные понятия**

Процедура - объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных. Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Функция - представляет собой модуль, который возвращает значение командой RETURN (вместо аргументов OUT или IN OUT). В отличие от вызова процедуры, который представляет собой отдельный оператор, вызов функции всегда является частью исполняемого оператора, то есть включается в выражение или служит в качестве значения по умолчанию, присваиваемого переменной при объявлении.

Представление - это виртуальная таблица, основанная на результирующем наборе инструкции SQL. Представление содержит строки и столбцы, как и настоящая таблица. Поля в представлении - это поля из одной или нескольких реальных таблиц в базе данных. Вы можете добавить в представление инструкции SQL функций, WHERE и JOIN и представить данные так, как если бы они поступали из одной таблицы.

**2.1.2. Практическая часть**

**Процедуры.**

**pr\_1\_AddNewClient**

Описание процедуры.

Данная процедура добавляет в таблицу «Client» нового клиента.

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_1\_AddNewClient`()

BEGIN

insert into client values (11, 'Гайда К.А.', '4322', '543456', 'г. Новгород, ул. 8 марта, д. 15, кв. 19', '+7-(945)-543-6452');

END

Вызов процедуры:

call auto.pr\_1\_AddNewClient();

Результат работы:

SELECT \* FROM auto.client;



**pr\_2\_CountCliet**

Описание процедуры.

Данная процедура выводит количество клиентов из таблицы «Client»

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_2\_CountCliet`()

BEGIN

select count(id\_Client) as 'Количество клиентов' from client;

END

Вызов процедуры:

call auto.pr\_2\_CountCliet();

Результат работы:



**pr\_3\_FindAutoForPrice**

Описание процедуры.

Данная процедура выводит автомобили из таблицы «Auto» где цена начинается от 200000 тысяч и заканчивается 5000000 миллионами

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_3\_FindAutoForPrice`(Начальная\_цена int, Конечная\_цена int)

BEGIN

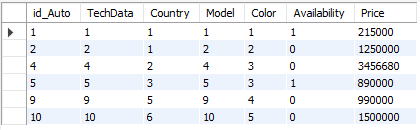
select \* from auto where price between Начальная\_цена and Конечная\_цена;

END

Вызов процедуры:

call auto.pr\_3\_FindAutoForPrice(200000, 5000000);

Результат работы:



**pr\_4\_Color**

Описание процедуры.

Данная процедура выводит автомобили из таблицы «Auto», и выводит «Чёрный», если цвет автомобиля чёрный, и «Другой цвет», если цвет автомобиля не чёрный.

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_4\_Color`()

BEGIN

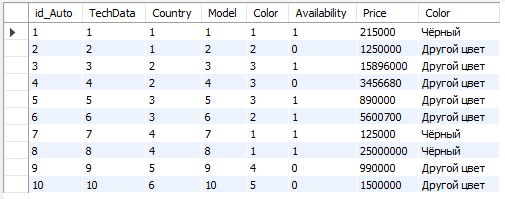
SELECT \*, IF(Color=1,"Чёрный","Другой цвет") AS 'Color' FROM auto;

END

Вызов процедуры:

call auto.pr\_4\_Color();

Результат работы:



**pr\_5\_AutoColor**

Описание процедуры.

Данная функция выводит все красные автомобили из таблицы «Auto»

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_5\_AutoColor`()

BEGIN

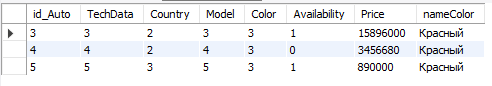
select auto.\*, color.nameColor from auto join color on auto.Color = color.id\_Color where nameColor = 'Красный';

END

Вызов процедуры:

call auto.pr\_5\_AutoColor();

Результат работы:



**Функции**

**fn\_1\_CountClientLikeC**

Описание функции.

Данная функция выводит количество всех клиентов, ФИО которых начинается на букву С из таблицы «Client»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_1\_CountClientLikeC`() RETURNS int

DETERMINISTIC

BEGIN

declare count\_client float;

select count(id\_Client) into count\_client from client where FIO like'С%';

RETURN count\_client;

END

Вызов функции:

select auto.fn\_1\_CountClientLikeC();

Результат работы:



**fn\_2\_MinAutoPrice**

Описание функции.

Данная функция считает наименьшую цену из таблицы «Auto»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_2\_MinAutoPrice`() RETURNS int

DETERMINISTIC

BEGIN

declare min\_price float;

select min(Price) into min\_price from auto;

RETURN min\_price;

END

Вызов функции:

select auto.fn\_2\_MinAutoPrice();

Результат работы:



**fn\_3\_LOOP**

Описание функции.

Данная процедура перебирает курсором фамилии, и коды из таблицы «Client» и где фамилия начинается на «C» меняет адрес на «Москва»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_3\_LOOP`() RETURNS varchar(50) CHARSET utf8

DETERMINISTIC

BEGIN

declare done int default false;

declare id int;

declare cfio varchar(45);

declare info varchar(45);

declare cur1 cursor for select id\_Client, FIO from client;

declare continue handler for not found set done = true;

open cur1;

read\_loop: LOOP

fetch cur1 into id, cfio;

if done then leave read\_loop; end if;

if cfio like 'С%' then update client set Address = 'Москва' where id = id\_Client;

set info = 'Замена выполнена';

end if;

end loop;

close cur1;

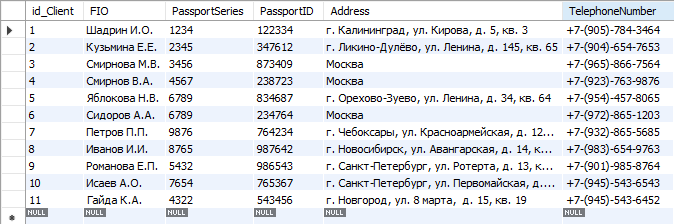
RETURN info;

END

Вызов функции:

call auto.pr\_5\_AutoColor();

Результат работы:



**fn\_4\_DeleteCLient**

Описание функции.

Данная функция удаляет запись по введенному ФИО из таблицы «Client»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_4\_DeleteCLient`(cfio varchar(50)) RETURNS varchar(50) CHARSET utf8

DETERMINISTIC

BEGIN

declare k int;

declare info varchar(100);

set k = 0;

select count(\*) into k from client where FIO = cfio;

if k = 1 then

delete from client where FIO = cfio;

set info = 'Клиент удалён';

else set info = 'Такого клиента нет';

end if;

RETURN info;

END

Вызов функции:

select auto.fn\_4\_DeleteCLient('Гайда К.А.');

Результат работы:

Сотрудник «Гайда К.А.» удалён



**fn\_5\_AddClient**

Описание функции.

Данная функция, пробует добавить новую запись в таблицу и если запись с такими значениями уже есть выдает соответствующее сообщение.

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_5\_AddClient`(clientfio varchar(100), pass\_ser varchar(4), pass\_num varchar(6), address varchar(200), tel\_num varchar(18)) RETURNS varchar(100) CHARSET utf8

DETERMINISTIC

BEGIN

declare k int;

declare info varchar(100);

set k = 0;

select count(\*) into k from client where FIO = clientfio;

if k = 0 then

insert into client(FIO, PassportSeries, PassportID, Address, TelephoneNumber)

values(clientfio, pass\_ser, pass\_num, address, tel\_num);

set info = 'Клиент успешно добавлен';

else set info = 'Клиент с такими данными уже есть уже есть';

end if;

RETURN info;

END

Вызов функции:

select auto.fn\_5\_AddClient('Шадрин И.О.', '4325', '4532', 'Калининград', '89660005205');

Результат работы:

При попытке добавить уже существующего в базу



Добавление нового сотрудника



Представления

**v\_1\_red**

Описание представления.

Данное представление берет автомобили из таблицы «Auto», идет в таблицу «Color» и находит красные автомобили

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `auto`.`v\_1\_red` AS

SELECT

`auto`.`auto`.`id\_Auto` AS `id\_Auto`,

`auto`.`auto`.`TechData` AS `TechData`,

`auto`.`auto`.`Country` AS `Country`,

`auto`.`auto`.`Model` AS `Model`,

`auto`.`auto`.`Color` AS `Color`,

`auto`.`auto`.`Availability` AS `Availability`,

`auto`.`auto`.`Price` AS `Price`

FROM

(`auto`.`auto`

JOIN `auto`.`color` ON ((`auto`.`auto`.`Color` = `auto`.`color`.`id\_Color`)))

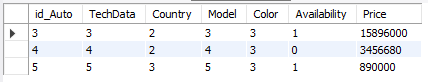
WHERE

(`auto`.`color`.`nameColor` = 'Красный')

Вызов представления:

SELECT \* FROM auto.v\_1\_red;

Результат работы:



**v\_2\_minprice**

Описание представления.

Данное представление выводит минимальную цену автомобиля из таблицы «Auto»

Код представления:

CREATE

ALGORITHM=UNDEFINED

DEFINER=`root`@`localhost`

SQL SECURITY DEFINER

VIEW `auto`.`v\_2\_minprice` AS

select min(`auto`.`auto`.`Price`) AS `Минимальная цена`

from `auto`.`auto`

Вызов представления:

SELECT \* FROM auto.v\_2\_minprice;

Результат работы:



**v\_3\_clientpurchase**

Описание представления.

Данное представление объединяет 2 таблицы «Purchase» и «Client».

Выводит дату покупки и ФИО клиента

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `auto`.`v\_3\_clientpurchase` AS

SELECT

`auto`.`client`.`FIO` AS `FIO`,

`auto`.`purchase`.`PDate` AS Дата покупки

FROM

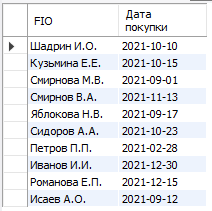
(auto.purchase

JOIN auto.client ON ((auto.`purchase`.idClient = auto.client.id\_Client)))

Вызов представления:

SELECT \* FROM auto.v\_3\_clientpurchase;

Результат работы:



**v\_4\_price**

Описание представления.

Данное представление выводит автомобили с ценой выше 500000 из таблицы «Auto»

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `auto`.`v\_4\_price` AS

SELECT

`auto`.`auto`.`id\_Auto` AS `id\_Auto`,

`auto`.`auto`.`TechData` AS `TechData`,

`auto`.`auto`.`Country` AS `Country`,

`auto`.`auto`.`Model` AS `Model`,

`auto`.`auto`.`Color` AS `Color`,

`auto`.`auto`.`Availability` AS `Availability`,

`auto`.`auto`.`Price` AS `Price`

FROM

`auto`.`auto`

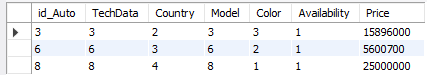
WHERE

(`auto`.`auto`.`Price` > 5000000)

Вызов представления:

SELECT \* FROM auto.v\_4\_price;

Результат работы:



**v\_5\_date**

Описание представления.

Данное представление выводит покупки из таблицы «Purchase», где дата покупки между 01.10.2021 и 31.12.2021

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `auto`.`v\_5\_date` AS

SELECT

`auto`.`purchase`.`id\_Purchase` AS `id\_Purchase`,

`auto`.`purchase`.`idAuto` AS `idAuto`,

`auto`.`purchase`.`idClient` AS `idClient`,

`auto`.`purchase`.`PDate` AS `PDate`,

`auto`.`purchase`.`Delivery` AS `Delivery`,

`auto`.`purchase`.`TypeOfPayment` AS `TypeOfPayment`

FROM

`auto`.`purchase`

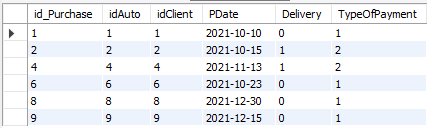
WHERE

(`auto`.`purchase`.`PDate` BETWEEN '2021-10-01' AND '2021-12-31')

Вызов представления:

SELECT \* FROM auto.v\_5\_date;

Результат работы:



**2.2. Пользователи и его права**

Пользователь базы данных — это идентификатор имени входа при подключении к базе данных. Имя пользователя базы данных может совпадать с именем входа, но это не является обязательным требованием.

Права — это возможность выполнения конкретного типа SQL-оператора или доступа к объекту базы данных, принадлежащему другому пользователю.

**2.2.1. Практическая часть. Работа с пользователями и правами через написание команд**

Создайте нового пользователя со своим именем латиницей.

CREATE USER kseniya@localhost IDENTIFIED BY '1111';

Присвойте пользователю все привилегии на вашу БД выбрав в ней одну любую таблицу.

grant all privileges on information.material to 'kseniya'@'localhost';

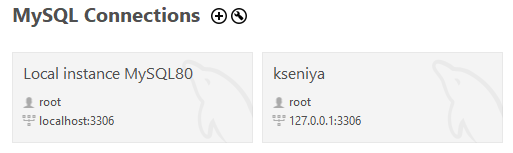
flush privileges;

Выведите таблицу mysql.user и убедитесь, что созданный вами пользователь в ней есть.

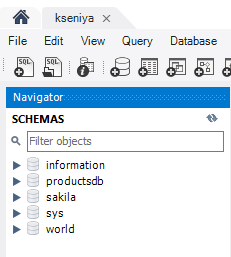
select \* from mysql.user



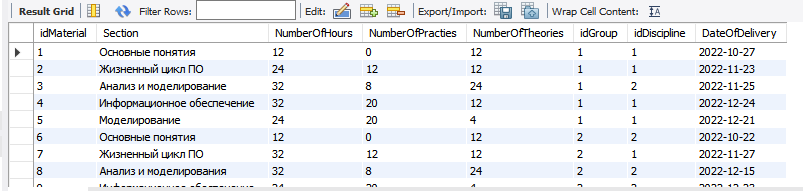
Создайте новое подключение для созданного пользователя.



Подключитесь используя данное подключение.



Посмотрите, что вам доступна именно та БД и та таблица, которой вы назначили привилегии для вашего пользователя.



Удалите данного пользователя.

drop user 'kseniya'@'localhost';

Создайте заново его-же, но уже с привилегиями: SELECT, UPDATE, INSERT.

CREATE USER kseniya@localhost IDENTIFIED BY '1111';

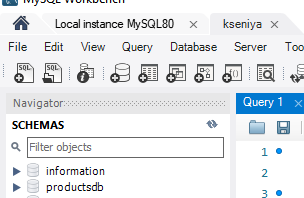
grant select, update, insert on information.material to 'kseniya'@'localhost';

Убедитесь, что никакие другие действия с таблицей не разрешены.

Измените пароль своему пользователю на «1111»

ALTER USER 'maria'@'localhost' IDENTIFIED BY '1111';

Перезайдите в СУБД с новым паролем.

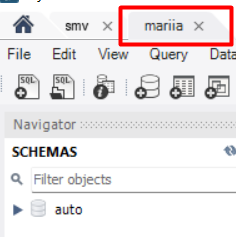


Удалите пароль своему пользователю.

update mysql.user set authentication\_string=null where User='maria';

flush privileges;

Перезайдите заново в СУБД (*уже не вводя пароль*)



Добавьте пароль своему пользователю «1234»

alter user 'maria'@'localhost' identified with caching\_sha2\_password by '1234';

**2.3. Индексы**

**Выбор индексов в MySQL**

Выполняем запрос без индекса.

explain select \* from auto where Color = 3;



Создаем индекс.

CREATE INDEX nameColor ON auto(Color);

explain select \* from auto where Color = 3;



Колонка rows показывает число записей, которые пришлось прочитать базе данных для выполнения этого запроса.

**Вывод:** запрос после создания индекса «nameColor» работает быстрее.

**Сортировка.**

Выполним сортировку по фамилиям.

explain select \* from client order by FIO;



Создадим индекс.

CREATE INDEX fio ON client(FIO);

explain select \* from client order by FIO;



Вывод: создание индекса для поля «FIO» особо не повлияла на скорость выполнения запроса.

**Уникальные индексы**

Создаю уникальный индекс для поля «Address»

EXPLAIN SELECT \* FROM client WHERE Address = 'г. Калининград, ул. Кирова, д. 5, кв. 3';

CREATE UNIQUE INDEX adres ON client(Address);

EXPLAIN SELECT \* FROM client WHERE Address = 'г. Калининград, ул. Кирова, д. 5, кв. 3';

**Вывод:** запрос выполняется быстрее с уникальным индексом.

**Составные индексы**

Запрос на выборку данных по адресу и номеру паспорта.

SELECT \* FROM client WHERE Address like '%Калининград%' AND PassportID = '873409';

EXPLAIN SELECT \* FROM client WHERE Address like '%Калининград%' AND PassportID = '873409' 

Создаю индекс «adres\_pasportid»

CREATE INDEX adres\_pasportid ON client(Address, PassportID);

EXPLAIN SELECT \* FROM client WHERE Address like '%Калининград%' AND PassportID = '873409'; 

Удаляю индекс adres\_pasportid.

drop index `pasportid\_adres` on client;

Создаю индекс «pasportid\_adres».

CREATE INDEX pasportid\_adres ON client(PassportID, Address);

EXPLAIN SELECT \* FROM client WHERE Address like '%Калининград%' AND PassportID = '873409'; 

**Вывод:** в первом случае (без индексов) будет проверено все 10 записей таблицы. После создания индекса «adres\_pasportid» будет проверено 10 строк, а при индексе «pasportid\_adres» уже первая строка окажется нужной. Т.к. при упорядочивании по полю Address будете рассмотрены все варианты для Address like '%Калининград%', а уже потом применено условие PassportID = '873409'. При индексе «pasportid\_adres» условие PassportID = '873409' будет найдено сразу же.

**2.4. Исключения**

**2.4.1. Практическая часть**

Напишите процедуру «create\_table\_auto», которая создает таблицу «auto», обработав ошибку номер 1050 (таблица уже существует)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `create\_table\_auto`()

BEGIN

declare exit handler for 1050

select 'Таблица auto уже существует' as msg;

create table auto(

id int primary key,

modelauto varchar(45));

END

Вызываем процедуру:



Напишите процедуру «found\_table», которая выбирает данные из указанной таблицы. Если такой таблицы нет, то выводится сообщение об ошибке 1146 (Таблица не существует)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `found\_table`(in tbl\_name varchar(45))

BEGIN

declare exit handler for 1146

select concat('Таблицы "',tbl\_name,'" не существует!') as msg;

set @qr = concat('select \* from ', tbl\_name);

prepare result from @qr;

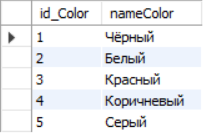
execute result;

END

Вызываем процедуру с существующей таблицей:

call auto.found\_table('Color');

Результат:



Вызываем процедуру с несуществующей таблицей:

call auto.found\_table('meow');

Результат:



Напишите процедуру «select\_column\_client», которая выводит данные указанного столбца таблицы «имя вашей таблицы», а в случае если такого столбца нет, выводит ошибку. (код ошибки 1054)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `select\_column\_client`(in namecol varchar(45))

BEGIN

declare exit handler for 1054

select concat('Не существующий столбец "',namecol,'"') as msg;

set @qr = concat('select ',namecol,' from client');

prepare result from @qr;

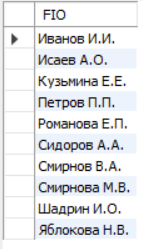
execute result;

END

Вызываем процедуру с существующим столбцом:

call auto.select\_column\_client('FIO');

Результат:



Вызываем процедуру с несуществующим столбцом:

call auto.select\_column\_client('age');

Результат:



Напишите процедуру «create\_database», которая создает БД, а в случае если такая БД есть, выводит ошибку. (код ошибки 1007)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `create\_database`(in namedb varchar(45))

BEGIN

declare exit handler for 1007

select concat('База данных "',namedb,'" уже существует!') as msg;

set @qr = concat('create database ', namedb);

prepare result from @qr;

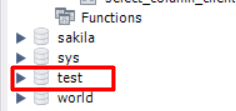
execute result;

END

Вызываем процедуру с несуществующей БД:

call auto.create\_database('test');

Результат:



Вызываем процедуру с существующей БД:

call auto.create\_database('auto');

Результат:



Напишите процедуру «drop\_database», которая удаляет БД, а в случае если такой БД нет, выводит ошибку. (код ошибки 1008)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `drop\_database`(in namedb varchar(45))

BEGIN

declare exit handler for 1008

select concat('Базы данных "',namedb,'" не существует!') as msg;

set @qr = concat('drop database ', namedb);

prepare result from @qr;

execute result;

END

Вызываем процедуру с несуществующей БД:

call auto.drop\_database('meow');

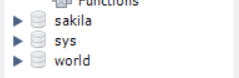
Результат:



Вызываем процедуру с существующей БД:

call auto.drop\_database('test');

Результат:



**2.5. Триггеры**

**2.5.1 Практическая часть**

Создаем таблицу log.

Скрипт:

CREATE TABLE `log` (

`id` int unsigned NOT NULL AUTO\_INCREMENT,

`msg` varchar(255) NOT NULL,

`time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`row\_id` int NOT NULL,

`table\_name` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM

Создаем триггер «insert\_color»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `insert\_color` AFTER INSERT ON `color` FOR EACH ROW BEGIN

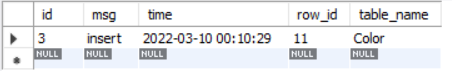
INSERT INTO log Set msg = 'insert', row\_id = NEW.id\_Color, table\_name = 'Color';

END

Запрос:

INSERT INTO `auto`.`color` (`nameColor`) VALUES ('Бежевый');

Результат:



Создаем триггер «insert\_country»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `insert\_country` AFTER INSERT ON `country` FOR EACH ROW BEGIN

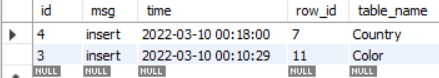
INSERT INTO log Set msg = 'insert', row\_id = NEW.id\_Country, table\_name = 'Country';

END

Запрос:

INSERT INTO `auto`.`country` (`Country`) VALUES ('Россия');

Результат:



Создаем триггер «update\_color»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `update\_color` AFTER UPDATE ON `color` FOR EACH ROW BEGIN

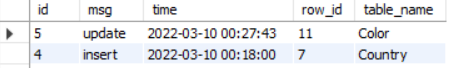
INSERT INTO log Set msg = 'update', row\_id = NEW.id\_Color, table\_name = 'Color';

END

Запрос:

UPDATE `auto`.`color` SET `nameColor` = 'Фиолетовый' WHERE (`id\_Color` = '11');

Результат:



Создаем триггер «update \_country»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `update\_country` AFTER UPDATE ON `country` FOR EACH ROW BEGIN

INSERT INTO log Set msg = 'update', row\_id = NEW.id\_Country, table\_name = 'Country';

END

Запрос:

UPDATE `auto`.`country` SET `Country` = 'Финляндия' WHERE (`id\_Country` = '7');

Результат:



Создаем триггер «delete\_color»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `delete\_color` AFTER DELETE ON `color` FOR EACH ROW BEGIN

INSERT INTO log Set msg = 'delete', row\_id = OLD.id\_Color, table\_name = 'Color';

END

Запрос:

DELETE FROM `auto`.`color` WHERE (`id\_Color` = '11');

Результат:



Создаем триггер «delete \_country»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `delete\_country` AFTER DELETE ON `country` FOR EACH ROW BEGIN

INSERT INTO log Set msg = 'delete', row\_id = OLD.id\_Country, table\_name = 'Country';

END

Запрос:

DELETE FROM `auto`.`color` WHERE (`id\_Color` = '7');

Результат:



**Приложения**

Приложение 1. **EDR - модель**

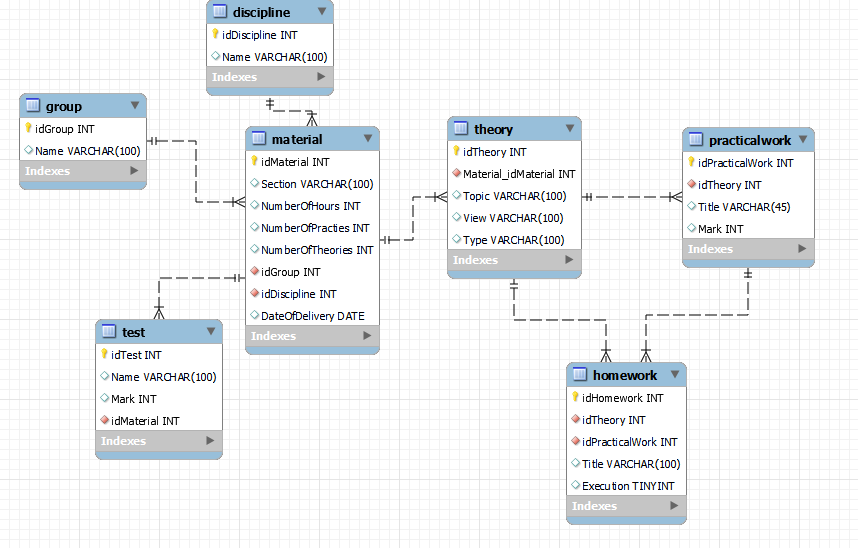


Рис. 1 Модель БД «Information»

Приложение 2. **SQL код создания БД «Information»**

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema information

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema information

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `information` DEFAULT CHARACTER SET utf8mb3 ;

USE `information` ;

-- -----------------------------------------------------

-- Table `information`.`discipline`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`discipline` (

`idDiscipline` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(100) NULL DEFAULT NULL,

PRIMARY KEY (`idDiscipline`))

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `information`.`group`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`group` (

`idGroup` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(100) NULL DEFAULT NULL,

PRIMARY KEY (`idGroup`))

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `information`.`material`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`material` (

`idMaterial` INT NOT NULL,

`Section` VARCHAR(100) NULL DEFAULT NULL,

`NumberOfHours` INT NULL DEFAULT NULL,

`NumberOfPracties` INT NULL DEFAULT NULL,

`NumberOfTheories` INT NULL DEFAULT NULL,

`idGroup` INT NOT NULL,

`idDiscipline` INT NOT NULL,

`DateOfDelivery` DATE NULL DEFAULT NULL,

PRIMARY KEY (`idMaterial`),

INDEX `fk\_Material\_Group\_idx` (`idGroup` ASC) VISIBLE,

INDEX `fk\_Material\_Discipline1\_idx` (`idDiscipline` ASC) VISIBLE,

CONSTRAINT `fk\_Material\_Discipline1`

FOREIGN KEY (`idDiscipline`)

REFERENCES `information`.`discipline` (`idDiscipline`),

CONSTRAINT `fk\_Material\_Group`

FOREIGN KEY (`idGroup`)

REFERENCES `information`.`group` (`idGroup`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `information`.`theory`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`theory` (

`idTheory` INT NOT NULL AUTO\_INCREMENT,

`Material\_idMaterial` INT NOT NULL,

`Topic` VARCHAR(100) NULL DEFAULT NULL,

`View` VARCHAR(100) NULL DEFAULT NULL,

`Type` VARCHAR(100) NULL DEFAULT NULL,

PRIMARY KEY (`idTheory`),

INDEX `fk\_Theory\_Material1\_idx` (`Material\_idMaterial` ASC) VISIBLE,

CONSTRAINT `fk\_Theory\_Material1`

FOREIGN KEY (`Material\_idMaterial`)

REFERENCES `information`.`material` (`idMaterial`))

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `information`.`practicalwork`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`practicalwork` (

`idPracticalWork` INT NOT NULL AUTO\_INCREMENT,

`idTheory` INT NOT NULL,

`Title` VARCHAR(45) NULL DEFAULT NULL,

`Mark` INT NULL DEFAULT NULL,

PRIMARY KEY (`idPracticalWork`),

INDEX `fk\_PracticalWork\_Theory1\_idx` (`idTheory` ASC) VISIBLE,

CONSTRAINT `fk\_PracticalWork\_Theory1`

FOREIGN KEY (`idTheory`)

REFERENCES `information`.`theory` (`idTheory`))

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `information`.`homework`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`homework` (

`idHomework` INT NOT NULL AUTO\_INCREMENT,

`idTheory` INT NOT NULL,

`idPracticalWork` INT NOT NULL,

`Title` VARCHAR(100) NULL DEFAULT NULL,

`Execution` TINYINT NULL DEFAULT NULL,

PRIMARY KEY (`idHomework`),

INDEX `fk\_Homework\_Theory1\_idx` (`idTheory` ASC) VISIBLE,

INDEX `fk\_Homework\_PracticalWork1\_idx` (`idPracticalWork` ASC) VISIBLE,

CONSTRAINT `fk\_Homework\_PracticalWork1`

FOREIGN KEY (`idPracticalWork`)

REFERENCES `information`.`practicalwork` (`idPracticalWork`),

CONSTRAINT `fk\_Homework\_Theory1`

FOREIGN KEY (`idTheory`)

REFERENCES `information`.`theory` (`idTheory`))

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb3;

-- -----------------------------------------------------

-- Table `information`.`test`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `information`.`test` (

`idTest` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(100) NULL DEFAULT NULL,

`Mark` INT NULL DEFAULT NULL,

`idMaterial` INT NOT NULL,

PRIMARY KEY (`idTest`),

INDEX `fk\_Test\_Material1\_idx` (`idMaterial` ASC) VISIBLE,

CONSTRAINT `fk\_Test\_Material1`

FOREIGN KEY (`idMaterial`)

REFERENCES `information`.`material` (`idMaterial`))

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8mb3;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;