МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский технический университет связи и информатики»
(МТУСИ)

Кафедра
«Сетевые информационные технологии и сервисы»

Лабораторная работа 2

По дисциплине «Интеллектуальные базы данных»

«Создание и связывание таблиц базы данных в среде SQL»

Выполнил студент:
М092301(75) Леонов Н.Н.
Вариант 17
Проверил:
Ст.пр. Фатхулин Т.Д.

Москва 2024

1. Цель работы

В данной лабораторной работе необходимо создать базу данных, таблицы, определить их поля и индексы. Также определить связи между таблицами и ограничениями целостности.

2. Задание

При выполнении лабораторной работы необходимо для заданной предметной области средствами MySQL:

- создать базу данных;
- создать таблицы, определить поля таблиц, индексы;
- определить связи между таблицами и ограничения целостности;
- составить отчет по лабораторной работе.

3. Теоретические сведения

Рассмотрим следующие вопросы:

- создание и выбор базы данных;
- создание таблиц;
- столбцы и типы данных в MySQL;
- создание индексов;
- удаление таблиц, индексов и баз данных;
- изменение структуры таблиц.

Базы данных, таблицы и индексы легко создаются в рамках графического интерфейса MySQL, но мы будем использовать монитор MySQL (клиент командной строки), чтобы лучше понять структуру БД, таблиц и индексов.

Чувствительность к регистру и идентификаторы.

- Имена БД подчиняются тем же правилам зависимости от регистра символов, каким следуют каталоги операционной системы. Имена таблиц следуют тем же правилам, что и имена файлов. Все остальное не зависит от регистра.
- Все идентификаторы, кроме имен псевдонимов, могут содержать до 64 символов. Имена псевдонимов могут иметь до 255 символов.

- Идентификаторы могут содержать любые допустимые символы, но имена баз данных не могут содержать символы /, \ и . , а имена таблиц – символы . и /.
- Зарезервированные слова можно использовать для идентификаторов, если заключить их в кавычки.

Комментарий в SQL. Начинается с двух дефисов (--), за которыми должен следовать пробел. Кроме того, MySQL содержит ряд собственных комментариев. Shell-комментарий # действует аналогично – все, что расположено правее его, является текстом комментария. С-комментарий /* */ является многострочным – комментарий начинается с /* и заканчивается, когда встретится завершение */.

Создание и выбор базы данных. Осуществляется с помощью оператора

CREATE DATABASE имя_базы_данных;

Убедиться в том, что оператор выполнил задачу, можно с помощью оператора

SHOW DATABASES;

Теперь имеется пустая БД, ожидающая создания таблиц. Прежде чем работать с БД, необходимо выбрать эту БД с помощью оператора

USE имя_базы_данных;

Теперь все действия по умолчанию будут применяться именно к этой БД.

Создание таблиц. Используется оператор *CREATE TABLE*, который в общем виде выглядит следующим образом:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS]

имя_таблицы (определение таблицы)

[TYPE=тип_таблицы];

Ключевое слово *TEMPORARY* используется для создания таблиц, которые будут существовать только в текущем сеансе работы с БД и будут автоматически удалены, когда сеанс завершится.

При использовании выражения *IF NOT EXISTS* таблица будет создана только в том случае, если еще нет таблицы с указанным именем.

Создать таблицу с такой же схемой, как у существующей, позволяет команда

CREATE [TEMPORARY] TABLE [IF NOT EXISTS]

имя_таблицы LIKE имя_старой_таблицы;

После имени таблицы в скобках объявляются имена столбцов, их типы и другая информация. В определение столбца можно добавить следующие описания.

- Объявить для любого столбца *NOT NULL* или *NULL* (столбцу запрещено или не запрещено содержать значения *NULL*). По умолчанию – *NULL*.
- Объявить для столбца значение по умолчанию, используя ключевое слово *DEFAULT*, за которым должно следовать значение по умолчанию.
- Использовать ключевое слово *AUTO_INCREMENT*, чтобы генерировать порядковый номер. Автоматически генерируемое значение будет на единицу большим, чем наибольшее значение в таблице. Первая введенная строка будет иметь порядковый номер 1. В таблице можно иметь не более одного столбца *AUTO_INCREMENT*, и он должен индексироваться.
- Объявить столбец первичным ключом таблицы с помощью выражения *PRIMARY KEY*.
- Объявить столбец внешним ключом, используя выражение *FOREIGN KEY*, с ссылкой на соответствующую таблицу с помощью выражения *REFERENCES*.
- Индексировать столбец с помощью слов *INDEX* или *KEY* (синонимы). Такие столбцы не обязательно должны содержать уникальные значения.
- Индексировать столбец с помощью слова *UNIQUE*, которое используется для указания того, что столбец должен содержать уникальные значения.
- Создать полнотекстовые индексы на основе столбцов типа *TEXT, CHAR* или *VARCHAR* с помощью слова *FULLTEXT* (только с таблицами *MyISAM*).

После закрывающей скобки можно указать тип таблицы:

- *MyISAM* – таблицы этого типа являются «родными» для MySQL, работают очень быстро и поддерживают полнотекстовую индексацию;
- *InnoDB* – ACID-совместимый механизм хранения, поддерживающий транзакции, внешние ключи, каскадное удаление и блокировки на уровне строк;
- *BDB (Berkeley DB)* – является механизмом хранения, который обеспечивает поддержку транзакций и блокировки на уровне страниц;
- *MEMORY (HEAP)* – таблицы целиком хранятся в оперативной памяти и никогда не записываются на диск, поэтому работают очень быстро, но ограничены в размерах и не допускают возможности восстановления в случае отказа системы;
- *MERGE* – тип позволяет объединить несколько таблиц *MyISAM* с одной структурой, чтобы к ним можно было направлять запросы как к одной таблице;
- *NDB Cluster* – тип предназначен для организации кластеров MySQL, когда таблицы распределены между несколькими компьютерами, объединенными в сеть;
- *ARCHIVE* – тип введен для хранения большого объема данных в сжатом формате; таблицы поддерживают только два SQL-оператора: *INSERT* и *SELECT,* причем оператор *SELECT* выполняется по методу полного сканирования таблицы;
- *CSV* – формат представляет собой обычный текстовый фал, записи в котором хранятся в строках, а поля разделены точкой с запятой (широко распространен в компьютерном мире, любая программа, поддерживающая CSV-формат, может открыть такой файл);
- *FEDERATED* – тип позволяет хранить данные в таблицах на другой машине сети (при создании таблицы в локальной директории создается только файл определения структуры таблицы, а все данные хранятся на удаленной машине).

MySQL поддерживает следующие типы данных, допустимые для столбцов:

- числовые;
- строковые;
- календарные;
- *NULL* – специальный тип, обозначающий отсутствие информации.

Числовые типы используются для хранения чисел и представляют два подтипа:

- точные числовые типы;
- приближенные числовые типы.

К точным числовым типам (табл. 1) относятся целый тип *INTEGER* и его вариации, а также вещественный тип *DECIMAL* (синонимы *NUMERIC* и *DEC*). Последний используется для представления денежных данных.

Числовые типы могут характеризоваться максимальной длиной *M*. Для типа *DECIMAL* параметр *M* задает число символов для отображения всего числа, а *D* – для его дробной части. Например: *b_price DECIMAL (5, 2)*. Цифра 5 определяет общее число символов под число, а цифра 2 – количество знаков после запятой (интервал величин от –99.99 до 99.99). Можно не использовать параметры вообще, указать только общую длину или указать длину и число десятичных разрядов.

Объявления точных числовых типов можно завершать ключевыми словами *UNSIGNED* и (или) *ZEROFILL*. Ключевое слово *UNSIGNED* указывает, что столбец содержит только положительные числа или нули. Ключевое слово *ZEROFILL* означает, что число будет отображаться с ведущими нулями.

Таблица 1

Тип	**Объем памяти**	**Диапазон**
TINYINT (M) *TINYINT UNSIGNED*	1 байт	от -128 до 127 (от -2^7 до 2^7-1) от 0 до 255 (от 0 до 2^8-1)
SMALLINT (M) *SMALLINT UNSIGNED*	2 байта	от -32 768 до 32 767 (от -2^{15} до $2^{15}-1$) от 0 до 65 535 (от 0 до $2^{16}-1$)

MEDIUMINT (M) *MEDIUMINT UNSIGNED*	3 байта	от -8 388 608 до 8 388 607 (от -2^{23} до $2^{23}-1$) от 0 до 16 777 215 (от 0 до $2^{24}-1$)
INT (INTEGER) (M) *INT UNSIGNED*	4 байта	от -2 147 683 648 до 2 147 683 647 (от -2^{31} до $2^{31}-1$) от 0 до 4 294 967 295 (от 0 до $2^{32}-1$)
BIGINT (M) *BIGINT UNSIGNED*	8 байт	(от-2^{63} до$2^{63}-1$) (от 0 до $2^{64}-1$)
BIT (M)	(*M*+7)/8 байт	От 1 до 64 битов, в зависимости от значения *M*
BOOL, BOOLEAN	1 байт	0 (*false*) либо 1 (*true*)
DECIMAL (M, D), *NUMERUC (M, D)*	*M* + 2 байта	Повышенная точность, зависит от параметров *M* и *D*

К приближенным числовым типам (табл. 2) относятся:

- *FLOAT* – представление чисел с плавающей запятой с обычной точностью;
- *DOUBLE* – представление чисел с плавающей запятой с двойной точностью.

Таблица 2

Тип	**Объем памяти**	**Диапазон**
FLOAT (M, D)	4 байта	Минимальное по модулю значение $1.175494351*10^{-39}$ Максимальное по модулю значение $3.402823466*10^{38}$
DOUBLE (M, D), *REAL (M,D),*	8 байт	Минимальное по модулю значение $2.2250738585072014*10^{-308}$

DOUBLE PRECISION (M,D)		Максимальное по модулю значение $1.797693134862315*10^{308}$

Числовые типы с плавающей точкой также могут иметь параметр *UNSIGNED*. Атрибут предотвращает хранение в столбце отрицательных величин, но максимальный интервал величин столбца остается прежним.

Приближенные числовые данные могут задаваться в обычной форме (например, 45.67) и в форме с плавающей точкой (например, 5.456E-02 или 4.674E+04).

Текстовые типы и строки (табл. 3):

- *CHAR* – хранение строк фиксированной длины;
- *VARCHAR* – хранение строк переменной длины;
- *TEXT, BLOB* и их вариации – хранение больших фрагментов текста;
- *ENUM* и *SET* – хранение значений из заданного списка.

Таблица 3

Тип	**Объем памяти**	**Максимальный размер**
CHAR(M)	*M* символов	*M* символов
VARCHAR(M)	*L*+1 символов	*M* символов
TINYBLOB, TINYTEXT	*L*+1 символов	2^8-1 символов
BLOB, TEXT	*L*+2 символов	2^{16}-1 символов
MEDIUMBLOB, MEDIUMTEXT	*L*+3 символов	2^{24}-1 символов
LONGBLOB, LONGTEXT	*L*+4 символов	2^{32}-1 символов
ENUM('value 1', 'value2 ', ...)	1 или 2 байта	65 535 элементов
SET('value 1', 'value2', ...)	1, 2, 3, 4 или 8 байт	64 элемента

Здесь *L* – длина хранимой в ячейке строки, а приплюсованные к *L* байты – накладные расходы для хранения длины строки.

Для строк *VARCHAR* требуется количество символов, равное длине строки плюс 1 байт, тогда как тип *CHAR(M)*, независимо от длины строки, использует для ее хранения все *M* символов. Тип *CHAR* обрабатывается эффективнее переменных типов. Нельзя смешивать в таблице столбцы *CHAR* и *VARCHAR*. Если есть столбец переменной длины, все столбцы типа *CHAR* будут приведены к типу *VARCHAR*.

Типы *BLOB* и *TEXT* аналогичны и отличаются в деталях. При выполнении операций над столбцами типа *TEXT* учитывается кодировка, а типа *BLOB* – нет. Тип *TEXT* используется для хранения больших объемов текста, тип *BLOB* – для больших двоичных объектов (электронные документы, изображения, звук). Основное отличие *TEXT* от *CHAR* и *VARCHAR* – поддержка полнотекстового поиска.

Строки типов данных *ENUM* и *SET* принимают значения из заданного списка. Значение типа *ENUM* должно содержать точно одно значение из указанного множества, тогда как столбцы *SET* могут содержать любой или все элементы заданного множества одновременно. Для типа *SET* (как и для *ENUM*) при объявлении задается список возможных значений, но ячейка может принимать любое значение из списка, а пустая строка означает, что ни один из элементов списка не выбран.

Типы *ENUM* и *SET* задаются списком строк, но во внутреннем представлении элементы множеств сохраняются в виде чисел. Элементы типа *ENUM* нумеруются последовательно, начиная с 1. Под столбец может отводиться 1 байт (до 256 элементов в списке) или 2 байта (от 257 до 65536 элементов в списке). Элементы типа *SET* обрабатываются как биты, размер типа определяется числом элементов в списке: 1 байт (от 1 до 8 элементов), 2 байта (от 9 до 16 элементов), 3 байта (от 17 до 24 элементов), 4 байта (от 25 до 32 элементов) и 8 байт (от 33 до 64 элементов).

Календарные типы данных (табл. 4):

- *DATE* – для хранения даты (формат *YYYY-MM-DD* для дат вида 2009-10-15 и формат *YY-MM-DD* для дат вида 09-10-15);

- *TIME* – для хранения времени суток (формат *HH:MM:SS*, где *HH* – часы, *MM* – минуты, *SS* – секунды, например, 10:48:56);
- *DATETIME*– для представления и даты, и времени суток;
- *TIMESTAMP* – если в соответствующем столбце строки не указать конкретное значение или *NULL*, там будет записано время, когда соответствующая строка была создана или в последний раз изменена (в формате *DATETIME*);
- *YEAR* – позволяет хранить только год.

Таблица 4

Тип	**Объем памяти**	**Диапазон**
DATE	3 байта	от '1000-01-01' до '9999-12-31'
TIME	3 байта	от '-828:59:59' до '828:59:59'
DATATIME	8 байт	от '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
TIMESTAMP (M)	4 байта	от '1970-01-01 00:00:00' до '2038-12-31 59:59:59'
YEAR(2) *YEAR(4)*	1 байт	формат *YY*, диапазон – от 1970 до 2069 формат *YYYY*, диапазон – от 1901 до 2155

Дни, месяцы, часы, минуты и секунды можно записывать как с ведущим нулем, так и без него. Например, все следующие записи идентичны:

'2009-04-06 02:04:08' '2009-4-06 02:04:08' '2009-4-6 02:04:08'

'2009-4-6 2:04:08' '2009-4-6 2:4:08' '2009-4-6 2:4:8'

В качестве разделителя между годами, месяцами, днями, часами, минутами, секундами может выступать любой символ, отличный от цифры. Так, следующие значения идентичны:

'09-12-31 11:30:45' '09.12.31 11+30+45' '09/12/31 11*30*45'

При указании времени после секунд через точку можно указать микросекунды, т. е. использовать расширенный формат вида *HH:MM:SS.FFFFFF*, например '10:25:14.000001'. Кроме того, можно использовать краткие форматы *HH:MM* и *HH* (вместо пропущенных величин будут подставлены нулевые значения).

Если время задается в недопустимом формате, то в поле записывается нулевое значение. Нулевое значение присваивается полям временного типа по умолчанию, когда им не присваивается инициирующее значение (табл. 5).

Таблица 5

Тип	**Нулевое значение**
DATE	'0000-00-00'
TIME	'00:00:00'
DATATIME	'0000-00-00 00:00:00'
TIMESTAMP	00000000000000
YEAR	0000

Формат *TIMESTAMP* совпадает с *DATETIME*, но во внутреннем представлении дата хранится как число секунд, прошедших с полуночи 1 января 1970 г. (такое исчисление принято в операционной системе UNIX, а дата 01.01.1970 считается началом эпохи UNIX и днем рождения операционной системы).

Если в таблице несколько столбцов *TIMESTAMP*, при модификации записи текущее время будет записываться только в один из них (первый). Можно явно указать столбец, которому необходимо назначать текущую дату при создании или изменении записи. Чтобы поля принимали текущую дату при создании записи, следует после определения столбца добавить *DEFAULT CURRENT_TIMESTAMP*. Если текущее время должно выставляться при модификации записи, при использовании оператора *UPDATE* следует добавить *ON UPDATE CURRENT_TIMESTAMP*.

Тип данных *NULL* используется, когда информации недостаточно и для части данных нельзя определить, какое значение они примут. Для указания

того, что поле может принимать неопределенное значение, в определении столбца после типа данных следует указать ключевое слово *NULL*. Если поле не должно принимать значение *NULL*, следует указать ключевое слово *NOT NULL*.

Рекомендации по выбору типа данных.

- Обработка числовых данных происходит быстрее строковых. Так как типы *ENUM* и *SET* имеют внутреннее числовое представление, им следует отдавать предпочтение перед другими видами строковых данных, если это возможно.
- Производительность можно увеличить за счет представления строк в виде чисел. Пример – преобразование IP-адреса из строки в *BIGINT*. Это позволит уменьшить размер таблицы и значительно увеличить скорость при сортировке и выборке данных, но потребует дополнительных преобразований.
- Базы данных хранятся на жестком диске, и чем меньше места они занимают, тем быстрее происходит поиск и извлечение. Если есть возможность, следует выбирать типы данных, занимающие меньше места.
- Типы фиксированной длины обрабатываются быстрее типов переменной длины, т. к. в последнем случае при частых удалениях и модификациях таблицы происходит ее фрагментация.
- Если применение столбцов с данными переменной длины неизбежно, для дефрагментации таблицы следует применять команду *OPTIMIZE TABLE*.

Обеспечение ссылочной целостности. Задается конструкцией:

FOREIGN KEY [name_key] (col1, ...) REFERENCES tbl (tbl_col, ...)
[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}] [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]

Конструкция позволяет задать внешний ключ с необязательным именем *name_key* на столбцах, которые задаются в круглых скобках (один или несколько). Ключевое слово *REFERENCES* указывает таблицу *tbl*, на которую ссылается внешний ключ, в круглых скобках указываются имена столбцов.

Необязательные конструкции *ON DELETE* и *ON UPDATE* позволяют задать поведение СУБД при удалении и обновлении строк из таблицы-предка. Параметры, следующие за этими ключевыми словами, имеют следующие значения:

- *CASCADE* – при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, записи со ссылками на это значение в таблице-потомке удаляются или обновляются автоматически;
- *SET NULL* – при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, в таблице-потомке значения внешнего ключа, ссылающегося на таблицу-предка, устанавливаются в *NULL*;
- *NO ACTION* – при удалении или обновлении записей, содержащих первичный ключ, с таблицей-потомком никаких действий не производится;
- *RESTRICT* – если в таблице-потомке имеются записи, ссылающиеся на первичный ключ таблицы-предка, при удалении или обновлении записей с таким первичным ключом возвращается ошибка;
- *SET DEFAULT* – согласно стандарту *SQL*, при удалении или обновлении первичного ключа в таблице-потомке для ссылающихся на него записей в поле внешнего ключа должно устанавливаться значение по умолчанию (в MySQL это ключевое слово зарезервировано, но не обрабатывается).

Создание индексов. Индексы играют большую роль в БД, т. к. это основной способ ускорения их работы. Записи в таблице располагаются хаотически. Чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит много времени. Идея индексов состоит в том, чтобы создать для столбца копию, которая постоянно будет поддерживаться в отсортированном состоянии. Это позволяет быстро осуществлять поиск по такому столбцу.

Все необходимые индексы формируются при создании таблицы. Индексированы будут все столбцы, объявленные как *PRIMARY KEY, KEY, UNIQUE* или *INDEX*. Индекс также можно добавить с помощью оператора *CREATE INDEX*. Перед выполнением оператор преобразуется в оператор

ALTER TABLE. Например, создание индекса с именем *name* на основе поля *u_name* из таблицы *users*:

CREATE INDEX name ON users (u_name);

Перед ключевым словом *INDEX* может присутствовать *UNIQUE*, требующее уникальности ограничения.

Корректность таблиц в БД можно проверить с помощью оператора

SHOW TABLES;

Более подробную информацию о структуре таблицы дает команда

DESCRIBE имя_таблицы;

Переименование БД. Специального оператора переименования БД нет, но можно переименовать каталог БД в системном каталоге (*...\DATA*).

Удаление БД. Удалить всю БД вместе с ее содержимым можно командой:

DROP DATABASE [IF EXISTS] имя_базы_данных;

Удаление таблиц и индексов. Удалить таблицу можно с помощью оператора:

DROP TABLE [IF EXISTS] имя_таблицы;

Удалить индекс можно с помощью оператора:

DROP INDEX имя_индекса ON имя_таблицы;

Изменение структуры таблиц. Изменить структуру существующей таблицы можно с помощью оператора *ALTER TABLE*. Например, можно создать индекс *name* для таблицы *users* следующим образом:

ALTER TABLE users ADD INDEX name (u_name);

Оператор *ALTER TABLE* является исключительно гибким, поэтому он имеет огромное множество дополнительных ключевых слов.

4. Выполнение лабораторной работы

Создадим новую базу данных db_cost_accounting с помощью команды CREATE BATABASE.

Далее создадим четыре таблицы в данной БД с полями и параметрами согласно схеме, представленной на рисунке 1.

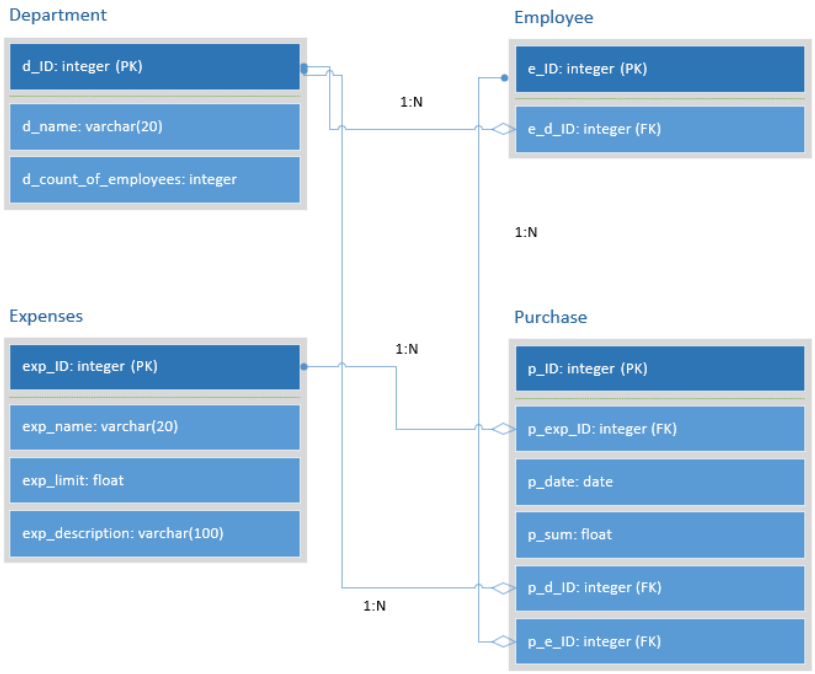


Рисунок 1 – Физическая модель БД

Для создания таблицы используем команду CREATE TABLE. На рисунках … представлено создание необходимых таблиц

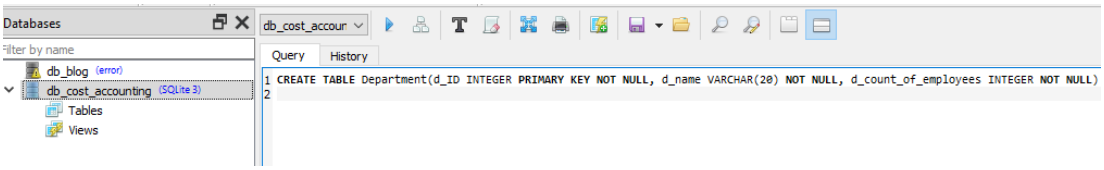


Рисунок 2 – Создание таблицы Department

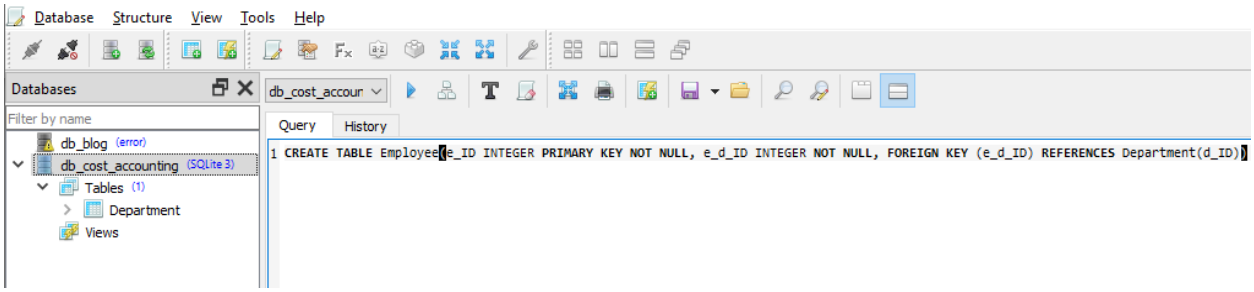


Рисунок 3 – Создание таблицы Employee

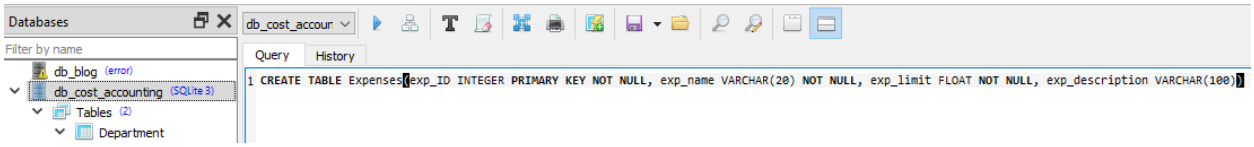
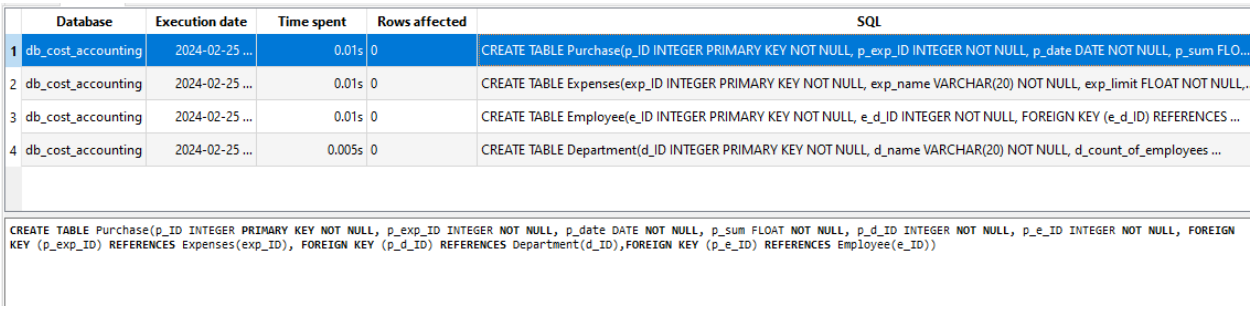


Рисунок 3 – Создание таблицы Expenses



	Database	Execution date	Time spent	Rows affected	SQL
1	db_cost_accounting	2024-02-25 ...	0.01s	0	CREATE TABLE Purchase(p_ID INTEGER PRIMARY KEY NOT NULL, p_exp_ID INTEGER NOT NULL, p_date DATE NOT NULL, p_sum FLO...
2	db_cost_accounting	2024-02-25 ...	0.01s	0	CREATE TABLE Expenses(exp_ID INTEGER PRIMARY KEY NOT NULL, exp_name VARCHAR(20) NOT NULL, exp_limit FLOAT NOT NULL,...
3	db_cost_accounting	2024-02-25 ...	0.01s	0	CREATE TABLE Employee(e_ID INTEGER PRIMARY KEY NOT NULL, e_d_ID INTEGER NOT NULL, FOREIGN KEY (e_d_ID) REFERENCES ...
4	db_cost_accounting	2024-02-25 ...	0.005s	0	CREATE TABLE Department(d_ID INTEGER PRIMARY KEY NOT NULL, d_name VARCHAR(20) NOT NULL, d_count_of_employees ...

```
CREATE TABLE Purchase(p_ID INTEGER PRIMARY KEY NOT NULL, p_exp_ID INTEGER NOT NULL, p_date DATE NOT NULL, p_sum FLOAT NOT NULL, p_d_ID INTEGER NOT NULL, p_e_ID INTEGER NOT NULL, FOREIGN
KEY (p_exp_ID) REFERENCES Expenses(exp_ID), FOREIGN KEY (p_d_ID) REFERENCES Department(d_ID),FOREIGN KEY (p_e_ID) REFERENCES Employee(e_ID))
```

Рисунок 4 – Создание таблицы Purchase

Итоговый вид базы данных представлен на рисунке 5.

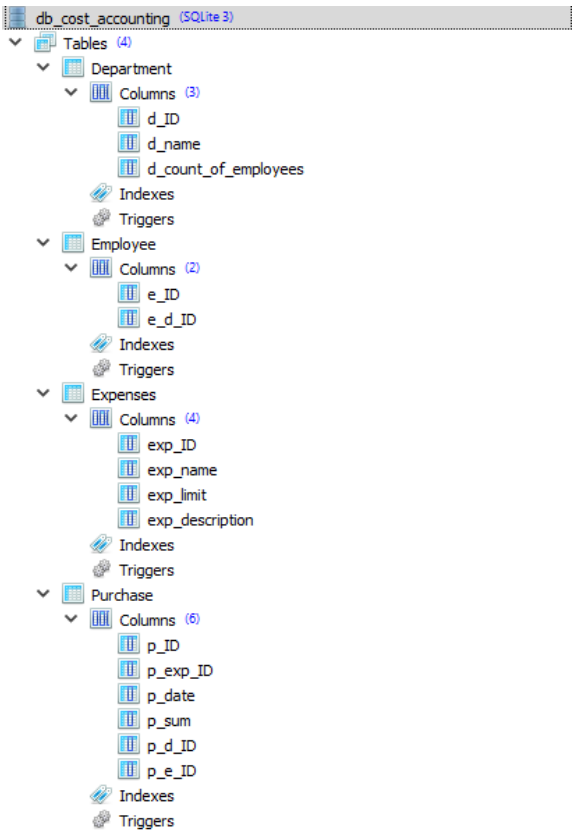


Рисунок 5 – Итоговый вид базы данных

5. Выводы

В ходе лабораторной работы была создана новая база данных “Cost Accounting” в MySQL. Была проведена работа по созданию таблиц с соответствующими полями, типами и индексами. Были определены связи между таблицами и ограничения целостности. Также была произведена проверка созданной базы данных и соответствующих ей таблиц.