ПРАКТИКА №2. ЗАПРОСЫ К ДАННЫМ В РСУБД

Вопросы, вносящиеся на рассмотрение

В данной практической работе рассматриваются понятийные аспекты работы с РСУБД, а также способы работы непосредственно с данными, находящимися внутри таблиц.

Реляционные базы данных

Реляционные базы данных (РБД) предназначены для хранения больших массивов структурированной информации определенной структуры, то есть информации, связанной с конкретной предметной областью. Примеры данных для хранения в РБД:

- Материально-учетные данные (бухгалтерия, склад)
- Транзакционные данные (продажи, отзывы, процессы в реальном времени)
- Временные ряды

Реляционные системы управления базами данных (СУБД/РСУБД) – программные реализации для управления информацией, хранящейся в виде структурированных массивов. Они характеризуются полным контролем над системой записи, чтения и хранения данных, а также предоставляют возможность пользователю получать доступ к данным и создавать новые структуры данных (схемы баз данных, базы данных), соблюдая определенный синтаксис языка.

Примеры систем управления базами данных

Ниже приведены примеры реляционных систем управления базами данных, все они используют язык SQL для управления данными. Внутри систем имеются различия, ввиду которых их появилось огромное множество, и каждый инструмент обладает своими преимуществами и недостатками, вследствие чего они решают разные задачи в разных проектах.

- OLTP системы (On-Line Transaction Processing) – PostgreSQL

- OLAP системы (On-Line Analytical Processing) – GreenPlum, Azure Redshift, SnowFlake, ClickHouse.
- Встраиваемые и легковесные системы (Embedded) – SQLite, H2.

OLTP системы, или системы поддержки обработки транзакций предназначены для быстрой обработки операций поступления новых данных, фиксации изменений, обработки ошибок и всех операций, связанных с поддержкой работы базы данных в реальном времени, чтобы справляться с главной задачей – хранением данных.

OLAP системы, или системы поддержки аналитических запросов предназначены для быстрой обработки операций чтения данных. Другими словами, OLAP системы созданы для массивной обработки данных внутри баз данных для поддержки как структуры данных, так и удобства использования с целью получения из данных пользы.

Встраиваемые системы характеризуются переносимостью на разные операционные системы и платформы, легковесностью как с точки зрения функционала, так и с точки зрения конечного размера баз данных, а также простотой реализации. Встраиваемые системы используются в легких задачах с малыми требованиями, где роль больше играет удобство хранения данных, по сравнению с поддержкой скорости обработки изменений.

Какие данные храним

Какими бы ни были различия между деталями реализации, общими остаются модель данных, которые хранят РСУБД, а также основа языка на котором происходит создание, доступ и управление данными – SQL.

Перечисленные выше базы данных хранят структурированные данные – таблицы с записями и полями (строками и столбцами), на пересечении которых должно стоять одно значение, характеризующее объект (запись, строка), отражающее его отдельный показатель или характеристику (поле, столбец). Пример реляционной модели данных и пример структурированных данных показан ниже на рисунке (Рисунок 1).

Дата транзакц...	Магаз...	Чек	Клие...	Товар	Кол-...	Сумма продажи
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006110	1,00	57,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006114	1,00	49,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006115	2,00	100,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006130	1,00	38,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006131	1,00	86,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006134	2,00	155,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006144	1,00	51,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku006148	3,00	125,00
01.04.2017, 00:00	Store01	code00350356	cl100636	sku027910	1,00	11,00
01.04.2017, 00:00	Store01	code00749858	cl100636	sku025594	1,00	549,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku030449	1,00	355,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku013453	1,00	518,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku007369	2,00	387,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku012298	1,00	2 241,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku029092	1,00	258,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku011035	2,00	134,00
01.04.2017, 00:00	Store01	code00174677	cl089338	sku011186	1,00	1 846,00

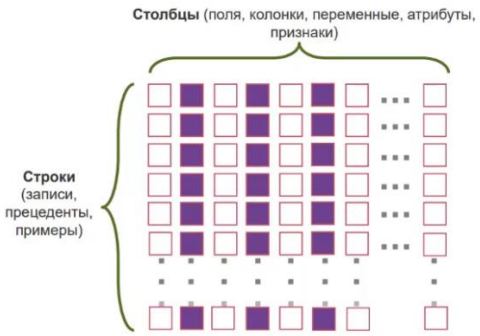

Рисунок 1. Пример структурированных данных и реляционная модель данных

Язык SQL состоит из нескольких групп операторов (языков) и предназначен для

- создания таблиц и их изменения в базе данных (DDL – Data Definition Language) – CREATE, ALTER, DROP, TRUNCATE,
- получения, добавления, изменения, удаления данных (DML – Data Manipulation Language) – SELECT, INSERT, UPDATE, DELETE,
- определения доступа к данным (DCL – Data Control Language) – GRANT REVOKE,
- управления изменениями в базе данных (TCL – Transactions Control Language).

SQLite и почему именно эта БД

SQLite – компактная встраиваемая СУБД. Движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки.

SQLite поддерживает динамическое типизирование данных. Возможные типы значений: INTEGER, REAL, TEXT и BLOB. Также поддерживается специальное значение NULL.

Размеры значений типа TEXT и BLOB не ограничены ничем, кроме константы SQLITE_MAX_LENGTH в исходном коде SQLite, равной миллиарду (10^9).

Каждое значение в любом поле любой записи может быть любого из этих типов, независимо от типа, указанного при объявлении полей таблицы.

Для получения значений из базы есть ряд функций для каждого из типов, и, если тип хранимого значения не соответствует запрашиваемому, оно тоже, по возможности, преобразуется.

Инструменты для работы с БД

Для работы с РСУБД можно пользоваться командной строкой терминала операционной системы, для чего в системе должны быть установлены необходимые модули (драйверы) для подключения к определенной РСУБД, а также в терминале командной строки должны быть пути к этим модулям (к каталогу с исполняемым кодом).

Для подключения к базе данных sqlite в терминале linux достаточно установить его в операционную систему (для систем с пакетным менеджером apt):

```
$ sudo apt-get install sqlite3
```

и далее подключиться к базе данных:

```
$ sqilte3 UserDB.db
```

После подключения к базе данных у вас появляется возможность писать запросы к выбранной базе данных.

Также имеется возможность использовать специализированные среды администрирования баз данных, основанные на графическом интерфейсе, облегчающем работу с примитивными операциями над базой данных (подключения, создания, доступа, оптимизации работы). Примеры таких сред: DBeaver, DataGrip.

Среда разработки DBeaver CE (Рисунок 2) является свободно распространяется для персонального пользования. Программа позволяет подключаться и управлять драйверами к различным РСУБД, создавать новые базы данных, получать доступ к данным и управлять уровнями доступа.

Для подключения к встраиваемым базам данных для системы необходимо указать только путь к файлу (Рисунок 3), в котором хранятся данные в формате встраиваемой системы.

Для подключения к стандартным РСУБД (Рисунок 4) необходимо указать все параметры строки подключения, среди которых встречаются:

- Хост – ip-адрес и порт, на котором работает сервер приложения
- Имя базы данных
- Имя пользователя
- Пароль
- Способ аутентификации

Все перечисленные выше метаданные передаются по каналу связи в точку назначения и проверяются сервером по указанному адресу и порту для ответного сообщения о результате соединения.

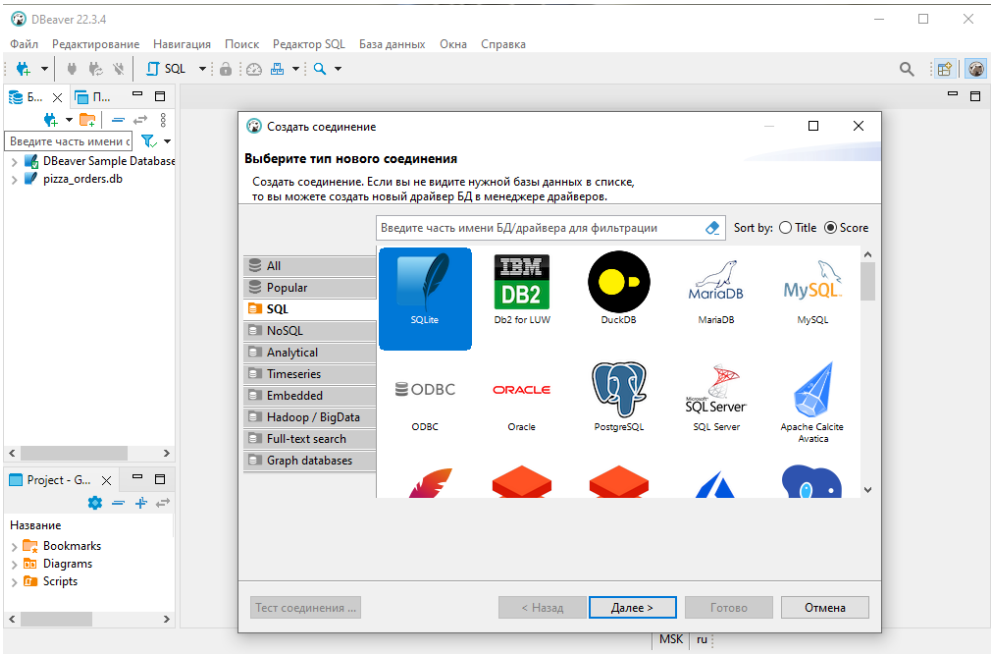

Рисунок 2. Интерфейс программы DBeaver CE и выбор нового соединения

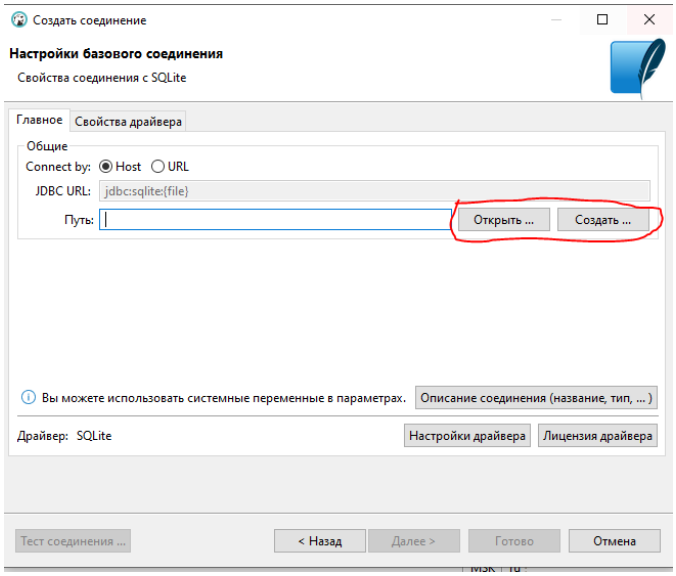

Рисунок 3. Создание соединения к базе данных SQlite

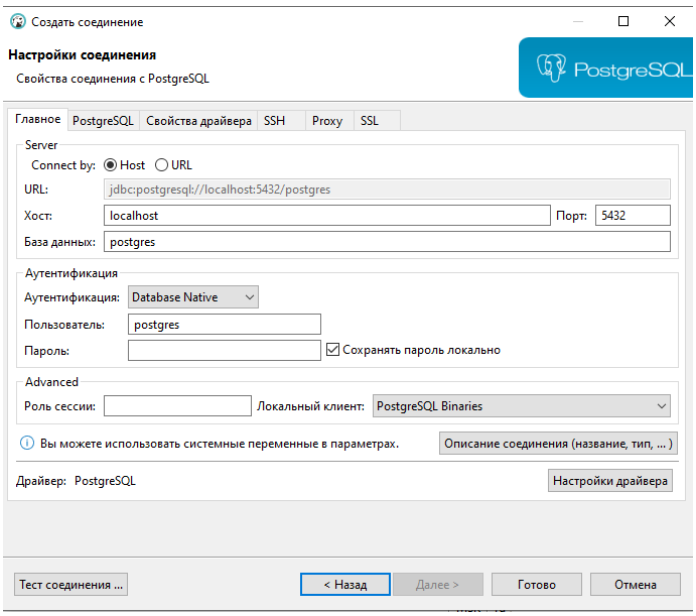

Рисунок 4. Создание подключения для базы данных PostgreSQL

При успешном соединении с базой данных мы можем либо построчно выполнять запросы на доступ к данным, либо писать SQL скрипты для выполнения нескольких запросов на доступ к данным сразу.

Схема тестовой таблицы данных

Для ознакомления с SQL запросами к данным приведем тестовую схему базы данных Chinook.db, которая доступна по ссылке: https://github.com/qwerty29544/BigDataEssentials/raw/main/Practice2_SQL_Language/Chinook.db.

В базе данных chinook имеется 11 таблиц.

- В таблице **Employee** хранятся данные о сотрудниках, такие как идентификатор сотрудника, фамилия, отчество и т.д. В нем также есть поле с именем ReportsTo, чтобы указать, кто перед кем отчитывается.
- В таблице **Customer** хранятся данные о клиентах.
- Таблицы **Invoice** & **InvoiceLine**: в этих двух таблицах хранятся данные счета-фактуры. В таблице **Invoice** хранятся данные заголовка счета-фактуры, а в таблице **InvoiceLine** - данные позиций счета-фактуры.

- В таблице **Artist** хранятся данные об исполнителях. Это простая таблица, которая содержит только идентификатор исполнителя и его имя.
- В таблице **Album** хранятся данные о списке треков. Каждый альбом принадлежит одному исполнителю. Однако у одного исполнителя может быть несколько альбомов.
- В таблице **MediaType** хранятся типы носителей, такие как аудиофайлы MPEG audio и AAC.
- В таблице **Genre** хранятся такие типы музыки, как рок, джаз, метал и т.д.
- В таблице **Track** хранятся данные о песнях. Каждый трек принадлежит одному альбому.
- В таблице **Playlist** хранятся данные о плейлистах. Каждый плейлист содержит список треков. Каждый трек может принадлежать нескольким плейлистам. Взаимосвязь между таблицей **Playlist** и таблицей **Track** является отношением "многие ко многим". Таблица **PlaylistTrack** используется для отражения этой взаимосвязи.

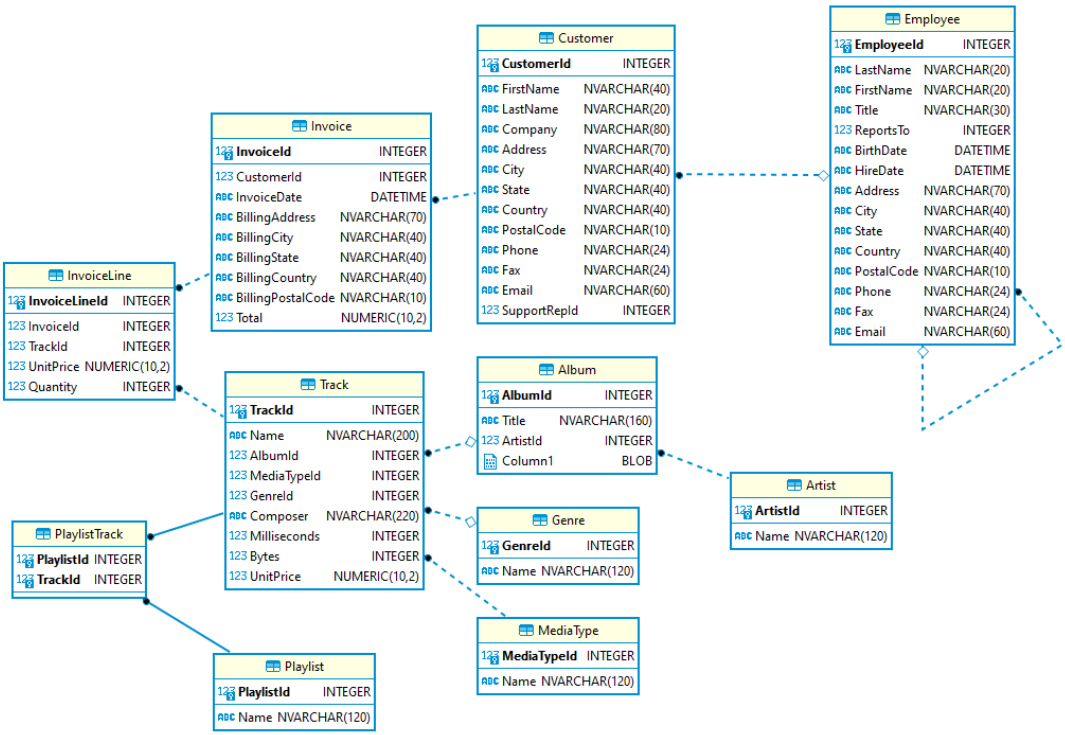

Рисунок 5. Тестовая база данных Chinook.db

Выборка данных из таблиц

Получение записей

Стандартная нотация выборки данных выглядит следующим образом:

```
SELECT *
FROM table;
```

Данный запрос к таблице table позволяет выбрать все записи из таблицы полностью. Символ «*» после ключевого слова SELECT означает выборку всех столбцов из таблицы данных table. Для выборки конкретных столбцов необходимо перечислить названия полей, определенных в таблице:

```
SELECT col1, col2, …
FROM table;
```

Или, если таблиц несколько (совсем скоро понадобится), то непосредственно из какой таблицы были взяты данные:

```
SELECT t.col1, t.col2, …
FROM table t;
SELECT table.col1, table.col2, …
FROM table;
```

Два запроса выше не отличаются, за исключением использования в первом так называемого короткого наименования (alias).

Для запроса с несколькими полями мы имеем возможность создать свои определенные имена с помощью применения сокращений с помощью AS, которые будут отражаться в результирующей таблице:

```
SELECT t.col1 AS name1,
       t.col2 AS name2,
       …
FROM table t;
```

Помимо прочего, можно делать запросы к полям и делать преобразования над полями специальными выражениями:

```
SELECT t.col1 * t.col2 AS prod_cols,
       t.col2 – t.col1 AS diff_cols,
       ABS(t.col4) AS absolute_col4,
       LOWER(t.text_col) as lowered_text
       …
FROM table t;
```

Среди выражений есть группы выражений для вычислений и для агрегаций.

Группа операторов для вычислений:

- арифметические операции – «+, -, *, /»
- математические операции – acosh(*X*), asin(*X*), asinh(*X*), atan(*X*), atan2(*Y*,*X*), atanh(*X*), ceil(*X*), ceiling(*X*), cos(*X*), cosh(*X*), degrees(*X*), exp(*X*), floor(*X*), ln(*X*), log(*X*), log10(*X*), log(*B*,*X*), log2(*X*), mod(*X*,*Y*),

pi(), pow(*X*,*Y*), power(*X*,*Y*), radians(*X*), sin(*X*), sinh(*X*), sqrt(*X*), tan(*X*), tanh(*X*), trunc(*X*)

- функции даты и времени – date(), time(), datetime(), julianday(), unixepoch(), strftime()
- операции над строками – lower(), trim(), upper(), length(), replace(), substring()

Группа операторов для агрегации:

- avg(X) – среднее значение показателя в группе
- count(*) – количество значений в таблице
- count(X) – количество значений показателя в группе
- group_concat(X,Y) – Функция group_concat() возвращает строку, которая является объединением всех ненулевых значений X. Если параметр Y присутствует, то он используется в качестве разделителя между экземплярами X
- max(X) – максимальное значение показателя в группе
- min(X) – минимальное значение показателя в группе
- sum(X) – сумма значений показателя в группе
- total(X) – сумма значений показателя в группе, считает Null за 0

Приведем пример запроса SELECT для таблицы **InvoiceLine**, подсчитав общую стоимость каждой строчки чека.

```
SELECT il.InvoiceLineId AS LineID,
il.InvoiceId AS ID,
il.TrackId AS TrackID,
il.UnitPrice AS UnitPrice,
il.Quantity AS Quantity,
TOTAL(il.Quantity * ABS(il.UnitPrice)) AS TotalPrice
FROM InvoiceLine il;
```

Первые строчки результата выглядят следующим образом:

LineID	ID	TrackID	UnitPrice	Quantity	TotalPrice
1	1	2	0.99	1	0.99
2	1	4	0.99	1	0.99
3	2	6	0.99	1	0.99
4	2	8	0.99	1	0.99
5	2	10	0.99	1	0.99
6	2	12	0.99	1	0.99
7	3	16	0.99	1	0.99
8	3	20	0.99	1	0.99
9	3	24	0.99	1	0.99
10	3	28	0.99	1	0.99

Также с помощью ключевого слова **DISTINCT** перед полем в запросе позволяет выбрать только уникальные значения поля таблицы. Пример:

```
SELECT DISTINCT col1
FROM table;
```

Запрос с **DISTINCT** выводит уникальные записи в порядке их появления в столбце, а повторяющиеся записи пропускает.

Фильтрация записей в таблице

Фильтрация записей в таблице данных осуществляется дополнительным словом WHERE. После слова WHERE необходимо определить условие для проверки каждой строки. Условие может состоять из сравнений или проверки вхождения интервалу, как и во множестве языков программирования:

```
SELECT col1, col2, …
```

```
FROM table
WHERE  col1 = value1 OR
       col2 != value2 AND
       (col3 < value3 OR
       col4 > value4) AND
       col5 BETWEEN value_lower AND value_upper;
```

Для строк также определено сравнение с подстрокой на проверку вхождения регулярного выражения в каждой из подстрок записей:

```
SELECT col1, col2, char_col…
FROM table
WHERE  char_col LIKE ‘perl_regular_expr’;
```

Приведем пример фильтрации данных для таблицы **Track:**

```
SELECT t.TrackId AS ID,
       t.Name AS Name,
       t.Composer AS Composer,
       t.Milliseconds AS LengthTimeMs,
       t.Bytes AS BytesSize
FROM Track t
WHERE  t.Composer LIKE "%_Malcolm Young_%" AND
       t.Milliseconds BETWEEN 250000 AND 350000 AND
       t.Bytes < 10000000;
```

Приведенный выше запрос выбирает композиции из таблицы данных с определенным перечнем полей, в которых значения лежат в определенных диапазонах. Для поля **Composer** выбрано условие фильтрации, в котором имя «*Malcolm Young*» стоит в середине перечня композиторов, «_» - любой

один символ, «%» - ноль или более символов на месте знака. Результат запроса выглядит следующим образом:

ID	Name	Composer	LengthTimeMs	BytesSize
10	Evil Walks	Angus Young, Malcolm Young, Brian Johnson	263497	8611245
12	Breaking The Rules	Angus Young, Malcolm Young, Brian Johnson	263288	8596840
14	Spellbound	Angus Young, Malcolm Young, Brian Johnson	270863	8817038

Сортировка записей

Сортировка записей осуществляется с помощью ключевого слова ORDER BY в запросе, который употребляется в конце запроса. Слово употребляется с перечислением полей, по которым производится сортировка. Сортировка по нескольким полям осуществляется согласно лексикографическому порядку, т.е. порядок перечисляемых полей имеет значение. Пример:

```
SELECT t.TrackId AS ID,
       t.Name AS Name,
       t.Composer AS Composer,
       t.Milliseconds AS LengthTimeMs,
       t.Bytes AS BytesSize
FROM Track t
WHERE  t.Milliseconds BETWEEN 250000 AND 350000 AND
       t.Bytes < 10000000
ORDER BY BytesSize ASC, LengthTimeMs DESC;
```

Параметры **ASC** (возрастание) и **DESC** (убывание) указывают на порядок сортировки. Результатом запроса выше будет выборка композиций, вес в байтах которых упорядочен по возрастанию, а длина внутри групп одинакового размера отсортирована по убыванию.

В реальных задачах приоритет сортировки ставится перед категориальными полями, так как явно прослеживается необходимость в лексикографическом правиле.

Агрегация данных

Агрегация или группировка данных в запросе SELECT осуществляется с помощью агрегационных функций и ключевого слова GROUP BY, указывающий на группы с одинаковыми значениями категории.

С помощью агрегации мы можем решить простые задачи расчета итоговой суммы чека или подвести итог по среднему значению внутри групп. Пример для агрегации данных по таблице **InvoiceLine**

```
SELECT il.InvoiceId AS InvoiceId,
       ROUND(SUM(il.UnitPrice * il.Quantity), 2) AS TotalSum,
       COUNT(TrackId) AS TracksCount
FROM InvoiceLine il
GROUP BY il.InvoiceId;
```

Предложенный запрос суммирует количество купленных треков, умноженных на цену одного такого внутри групп чеков, которые объединяются посредством схожих значений из **InvoiceId**. Результат суммирования **TotalSum** округляем до второго знака после запятой. А также считаем общее количество треков в чеке **TracksCount**.

Результат первых 10-ти строк запроса:

InvoiceId	TotalSum	TracksCount

1	1.98	2
2	3.96	4
3	5.94	6
4	8.91	9
5	13.86	14
6	0.99	1
7	1.98	2
8	1.98	2
9	3.96	4
10	5.94	6

Группировать также можно по нескольким полям. В таком случае совпадения по строкам будут считаться в рамках уникальных пар значений двух полей.

Объединение данных

Данные в РСУБД хранятся в нормализованном виде – в виде разрозненных таблиц, содержащих значения, зависящие только от первичного ключа без дублирований. В таком удобном для хранения виде существует возможность объединять несколько таблиц в одну, заменяя внешние ключи таблиц на их фактические значения, хранящиеся в другой таблице.

Рассмотрим пример. Есть таблица **Album**, в которой хранятся данные об альбомах в виде набора полей: ID альбома, название и ID артиста:

AlbumId	**Title**	**ArtistId**
1	For Those About To Rock We Salute You	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3

6	Jagged Little Pill	4
7	Facelift	5
8	Warner 25 Anos	6
9	Plays Metallica By Four Cellos	7
10	Audioslave	8

В данной таблице у нас имеется ID артиста, за которым не известно кто конкретно исполняет композицию. Нам бы хотелось узнать это, соединив две таблицы, **Album** и **Artist** по уникальному ключу **ArtistId** так, чтобы вместо значений ключей стояли реальные имена артистов.

Это возможно сделать при помощи ключевого слова **JOIN**, которое определяет условие слияния двух таблиц по определенному правилу.

```
SELECT  a.AlbumId,
        a.Title,
        a2.Name
FROM Album a
JOIN Artist a2 ON a2.ArtistId = a.ArtistId;
```

Результатом запроса станет таблица:

AlbumId	**Title**	**ArtistId**
1	For Those About To Rock We Salute You	AC/DC
2	Balls to the Wall	Accept
3	Restless and Wild	Accept
4	Let There Be Rock	AC/DC
5	Big Ones	Aerosmith
6	Jagged Little Pill	Alanis Morissette
7	Facelift	Alice In Chains
8	Warner 25 Anos	Antônio Carlos

		Jobim
9	Plays Metallica By Four Cellos	Apocalyptica
10	Audioslave	Audioslave

Перечень стандартных операторов слияния данных и их принцип работы проиллюстрирован на рисунке (Рисунок 6).

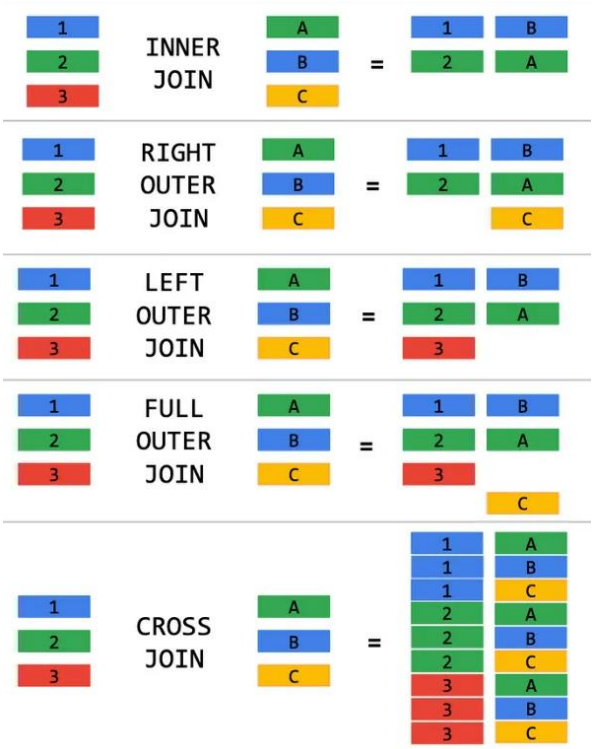

Рисунок 6. SQL объединения таблиц

Среди объединений выделяют следующие:

1. **INNER JOIN** – внутреннее соединение таблиц, результатом которого является объединение полей только по записям с совпадающими значениями.
2. **RIGHT JOIN** и **LEFT JOIN** – правое и левое соединения таблиц, результатом которой будет главная таблица (в случае с RIGHT – правая, в случае с LEFT – левая), дополненная полями по совпадающим значениям, но для индексов главной таблицы, которых нет в присоединяемой, значение присоединяемое значение равно NULL.

3. **FULL JOIN** – полное соединение в результате которого для обоих таблиц по совпадающим значениям данные дополняются, и для отсутствующих значений данные дополняются значениями NULL.
4. **CROSS JOIN** – декартово произведение двух таблиц, в котором результатом является таблица со всеми сочетаниями значений из двух таблиц.

Можно описать все соединения с помощью кругов Эйлера, в которых изображены возможные результаты по правилам алгебры множеств, где множества A и B это таблицы, а элементы множеств это поле первичного и внешнего ключа.

Рисунок 7. SQL Соединения в кругах Эйлера и запросах

В качестве еще одного примера приведем задачу по получению полной информации по таблице **Track**. Обратим внимание на схему данных на рисунке (Рисунок 8):

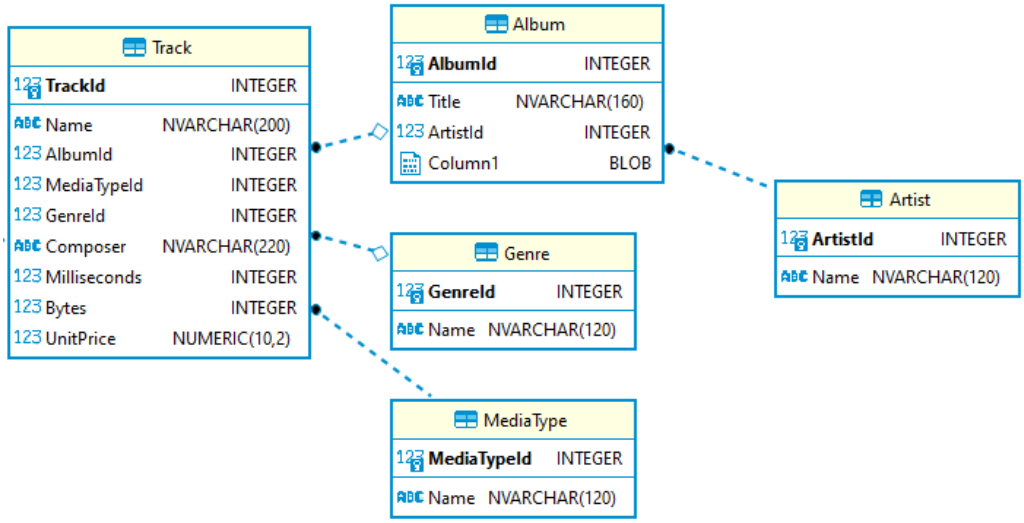

Рисунок 8. Подсхема общей схемы для таблицы **Track** на рисунке (Рисунок 5)

Как мы видим в таблице **Track** присутствуют внешние ключи на таблицы **Album**, **Genre**, **MediaType.** Если мы попробуем сделать выборку только по таблице **Track** мы получим значения в них в виде ключей:

```
SELECT t.TrackId, t.Name,
       t.AlbumId, t.MediaTypeId,
       t.GenreId, t.Composer
FROM Track t
LIMIT 10;
```

Track ID	Name	Album Id	Media TypeID	Genre Id	Composer
1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson
2	Balls to the Wall	2	2	1	
3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman
4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman
5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel

6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian Johnson
7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian Johnson
8	Inject The Venom	1	1	1	Angus Young, Malcolm Young, Brian Johnson
9	Snowballed	1	1	1	Angus Young, Malcolm Young, Brian Johnson
10	Evil Walks	1	1	1	Angus Young, Malcolm Young, Brian Johnson

Чтобы дополнить информацию внешними таблицами воспользуемся левым соединением данных внешних таблиц на таблицу **Track.**

```
SELECT t.Name AS TrackName,
          a.Title AS AlbumTitle,
          a.Name AS ArtistName,
          g.Name AS GenreName,
          mt.Name AS MTName,
          t.Composer AS Composer
FROM Track t
LEFT JOIN (SELECT a.AlbumId,
                  a.Title,
                  a2.Name
           FROM Album a
           LEFT JOIN Artist a2) a ON a.AlbumId = t.AlbumId
LEFT JOIN Genre g ON g.GenreId = t.GenreId
LEFT JOIN MediaType mt ON mt.MediaTypeId = t.MediaTypeId
ORDER BY ArtistName
LIMIT 10;
```

В запросе, сформированном выше, видим первое левое соединение на таблицу, запрос к которой формируется «на месте». Этот прием называется **вложенный запрос**. Вложенный запрос может находиться в любом месте, где должна находиться таблица в запросе SELECT. Таким образом мы

соединяем с главной таблице (**Track**) таблицу (**Album**), к которой мы присоединяем ещё и одну внешнюю таблицу (**Artist**). Для полученной вложенной таблицы специально даем сокращенное наименование, чтобы к ней проще обращаться.

Далее по запросу идет серия однотипных соединений, которые присоединяют к главной внешние таблицы. После этого в SELECT можно написать перечень столбцов, которые нам нужны в полученном запросе.

После ключевых слов JOIN далее могут идти и другие обработки, такие как агрегация, или фильтрация. Также в стандартном запросе SELECT всегда можно ограничить вывод количества записей с помощью слова LIMIT в конце запроса (**ограничение вывода**). Результат запроса:

TrackName	**AlbumName**	**ArtistName**	**GenreName**	**MTname**	**Composer**
For Those About To Rock (We Salute You)	For Those About To Rock We Salute You	A Cor Do Som	Rock	MPEG audio file	Angus Young, Malcolm Young, Brian Johnson
Balls to the Wall	Balls to the Wall	A Cor Do Som	Rock	Protected AAC audio file	
Fast As a Shark	Restless and Wild	A Cor Do Som	Rock	Protected AAC audio file	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman
Restless and Wild	Restless and Wild	A Cor Do Som	Rock	Protected AAC audio file	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman
Princess of the Dawn	Restless and Wild	A Cor Do Som	Rock	Protected AAC audio file	Deaffy & R.A. Smith-Diesel
Put The Finger On You	For Those About To Rock We Salute You	A Cor Do Som	Rock	MPEG audio file	Angus Young, Malcolm Young, Brian Johnson
Let's Get It Up	For Those About To Rock We Salute You	A Cor Do Som	Rock	MPEG audio file	Angus Young, Malcolm Young, Brian Johnson
Inject The Venom	For Those About To Rock We Salute You	A Cor Do Som	Rock	MPEG audio file	Angus Young, Malcolm Young, Brian Johnson
Snowballed	For Those About To Rock We Salute You	A Cor Do Som	Rock	MPEG audio file	Angus Young, Malcolm Young, Brian Johnson
Evil Walks	For Those About To Rock We Salute You	A Cor Do Som	Rock	MPEG audio file	Angus Young, Malcolm Young, Brian Johnson

Запрос выше можно упростить с помощью написания временных запросов, а также использования специальной конструкции для соединения таблиц по общим столбцам.

```
WITH
TempAlbum AS (SELECT a.AlbumId,
                     a.Title,
                     a2.Name
              FROM Album a
              LEFT JOIN Artist a2)
SELECT t.Name AS TrackName,
       a.Title AS AlbumTitle,
       a.Name AS ArtistName,
       g.Name AS GenreName,
       mt.Name AS MTName,
       t.Composer AS Composer
FROM Track t
LEFT JOIN TempAlbum a USING(AlbumId)
LEFT JOIN Genre g USING(GenreId)
LEFT JOIN MediaType mt USING(MediaTypeId)
ORDER BY ArtistName
LIMIT 10;
```

Выше представлен запрос, эквивалентный ранее показанному. В запросе использован механизм временных запросов в рамках одного запроса SELECT. Временный запрос работает на основе ключевого слова WITH следующим образом:

```
WITH
Temp1 AS (SELECT …),
Temp2 AS (SELECT …),
…,
TempK AS (SELECT …)
```

```
SELECT *
FROM Table t
JOIN …;
```

В запросе выше для выборки данных теперь доступны и временные таблицы Temp.

Для соединения таблиц запрос с использованием функции **USING()**

```
SELECT *
FROM Table1 t1
JOIN Table2 t2 USING(Key);
```

полностью эквивалентен записи запроса

```
SELECT *
FROM Table1 t1
JOIN Table2 t2 ON t1.Key = t2.Key;
```

и работает только для столбцов с одинаковым именем, как показано выше.

Создание представлений таблиц из запросов

На основе созданных запросов можно создавать виртуальные отношения (таблицы), которые могут храниться в памяти в виде запросов. Такие таблицы называются представлениями (VIEW). Представления создаются командой

```
CREATE VIEW view_table AS (SELECT …)
```

Представления являются очень удобным способом хранения структурированных данных в виде OLAP-кубов, полных объединенных таблиц для аналитики, содержащих полную информацию о каждой записи без внешних ключей. OLAP-кубы характеризуются наличием большого количества измерений, которые удобно хранить в колоночном формате.

ЗАДАЧА ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

Постановка задания

Практическое задание, предложенное для выполнения, ставит целью проверку умения написания студентом запросов к существующей базе данных с целью получить информацию из таблиц.

Файл базы данных можно получить по ссылке https://github.com/qwerty29544/BigDataEssentials/raw/main/Practice1_LinuxCommands/Data/pizza_orders.db.

Оценка практического задания проходит в трехступенчатом формате. Для начала, вы должны написать перечень запросов к базе данных исходя из условий, перечисленных в пункте «Задачи на выборку данных». Затем вам необходимо продемонстрировать преподавателю, по его усмотрению, список запросов и их результатов с целью валидировать правильность их выполнения. В завершении, преподаватель имеет право задать вам несколько вопросов по запросам к базе данных или из предложенного в конце перечня вопросов.

По итогам проверки преподаватель фиксирует количество баллов, полученных вами на протяжении выполнения и сдачи вами самостоятельного задания. На усмотрение преподавателя, вам может быть выставлен:

1. полный балл за успешную сдачу работы по всем пунктам;
2. неполный балл за неточный ответ или неверно исполненный запрос;
3. дополнительный балл(ы) за дополнительный вопрос или задание преподавателя.

Результат прохождения вами самостоятельной работы учитывается по завершении курса.

Схема базы данных

На представленном ниже рисунке (Рисунок 9) представлена схема существующей, предложенной вам для рассмотрения базы данных

транзакций в пиццерии за 2015 отчетный год. В схеме базы данных представлены таблицы, их типы связей, атрибуты и их типы данных, а также первичные ключи. По схожим названиям атрибутов также вполне возможно установить и связь по внешним ключам между связанными таблицами.

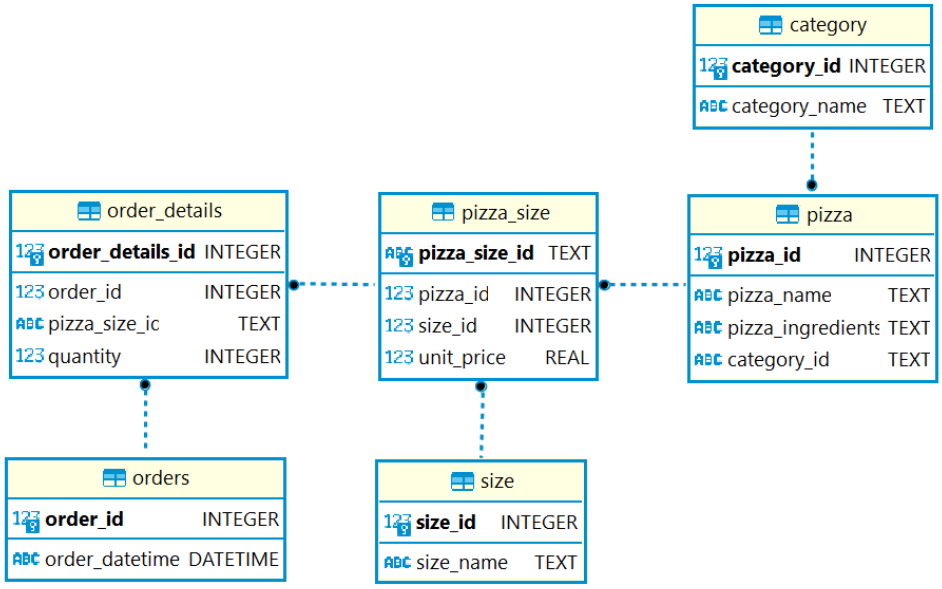

Рисунок 9. Схема базы данных транзакций пиццерии

Ниже приведена расшифровка предложенной выше схемы базы данных.

Таблица category

Описание. Таблица перечня категорий пицц. В данной таблице содержатся названия категорий пицц, которые представлены в ресторане. На 2015 год были следующие категории: (“Classic”, “Veggie”, “Supreme”, “Chicken”). Атрибуты таблицы могут расширяться в сторону увеличения описания каждой категории, следовательно они представлены отдельной таблицей. Данная таблица является *справочной*.

Атрибуты:

1. category_id – первичный ключ, представленный целым числом, идентифицирующий каждую категорию;
2. category_name – атрибут, представленный текстовым типом данных, несущий информацию о названии категории.

Запрос на создание таблицы.

```
CREATE TABLE category (
	category_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	category_name TEXT NOT NULL);
```

Таблица pizza

Описание. Таблица содержит информацию о видах пицц, представленных в ресторане, их отношение к одной из категорий и перечень ингредиентов, из которых она состоит. Данная таблица является *справочной*.

Атрибуты:

1. pizza_id – первичный ключ, представленный целочисленным типом данных, идентифицирующий каждый вид пиццы;
2. pizza_name – название видов пиццы, представленное текстовым типом данных;
3. pizza_ingredients – перечень ингредиентов каждого вида пиццы, представленный текстовым типом данных;
4. category_id – внешний ключ на таблицу **category**, представленный целочисленным типом данных, представляет информацию о категории каждого вида пиццы.

Запрос на создание таблицы.

```
CREATE TABLE pizza (
	pizza_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
	pizza_name TEXT NOT NULL,
	pizza_ingredients TEXT NOT NULL,
	category_id TEXT NOT NULL,
	FOREIGN KEY(category_id) REFERENCES category(category_id));
```

Таблица size

Описание. Таблица содержит информацию о существующих размерах пицц, представленных в ресторане. Атрибуты таблицы могут расширяться в сторону увеличения описания каждого размера, следовательно они представлены отдельной таблицей. Таблица является *справочной*.

Атрибуты:

1. size_id
2. size_name

Запрос на создание таблицы:

```
CREATE TABLE size (
      size_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
      size_name TEXT NOT NULL);
```

Таблица pizza_size

Описание. Таблица содержит информацию о меню ресторана, которое представлено пиццами различных размеров. Для каждой позиции в меню имеется стоимость одного экземпляра. Данная таблица является *справочной* таблицей и отражает отношение «многие ко многим» для таблиц **pizza** и **size**.

Атрибуты:

1. pizza_size_id – первичный ключ, представленный текстовым типом данных, уникально идентифицирующий каждую отдельную позицию в меню;
2. pizza_id – внешний ключ для таблицы **pizza**;
3. size_id – внешний ключ для таблицы **size**;
4. unit_price – атрибут, представленный целочисленным типом данных, показывающий цену одного экземпляра позиции в меню.

Запрос на создание таблицы:

```
CREATE TABLE pizza_size (
```

```
    pizza_size_id TEXT NOT NULL PRIMARY KEY,
    pizza_id INTEGER NOT NULL,
    size_id INTEGER NOT NULL,
    unit_price REAL NOT NULL,
    FOREIGN KEY(pizza_id) REFERENCES pizza(pizza_id),
    FOREIGN KEY(size_id) REFERENCES size(size_id));
```

Таблица orders

Описание. Содержит общую информацию о перечне заказов, а также о дате и времени сделанных заказов. Данная таблица является таблицей *фактов*.

Атрибуты:

1. order_id – первичный ключ для номера заказа/чека в ресторане;
2. order_datetime – дата и время для заказа, представленное в формате timestamp.

Запрос на создание таблицы:

```
CREATE TABLE orders (
    order_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    order_datetime DATETIME NOT NULL);
```

Таблица order_details

Описание. Таблица содержит в себе детализацию всех сделанных заказов в пиццерии за все время наблюдения. Для каждому отдельному заказу соответствуют несколько строк детализации заказа. В данной таблице содержатся строчки всех чеков (заказов) с позициями меню (пицца и размер), а также количество таких купленных позиций в одном чеке. Данная таблица является таблицей *фактов* и является промежуточной для связи «многие ко многим» для таблиц **orders** и **pizza_size**.

Атрибуты:

1. order_details_id – первичный ключ для номера позиции в чеке покупки, уникальным образом идентифицирует строку в таблице деталей транзакции;
2. order_id – внешний ключ для таблицы **orders**;
3. pizza_size_id – внешний ключ для таблицы **pizza_size**;
4. quantity – атрибут, представленный целочисленным типом данных, показывающий количество купленных пицц одного вида и размера в пределах одного чека (**order_id**);

Запрос на создание таблицы:

```
CREATE TABLE order_details (
        order_details_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        order_id INTEGER NOT NULL,
        pizza_size_id TEXT NOT NULL,
        quantity INTEGER NOT NULL,
        FOREIGN KEY(order_id) REFERENCES orders(order_id),
        FOREIGN KEY(pizza_size_id) REFERENCES pizza_size(pizza_size_id));
```

Задачи на выборку данных

1. Найдите информацию по поводу ключевого слова HAVING запроса SELECT. Объясните, для чего оно нужно, приведите пример запроса с HAVING. (1 балл).
2. Получите список всех имен таблиц и их запросов из таблицы sqlite_schema. (1 балл)
3. Получите перечень названий пицц и их ингредиентов. (1 балл)
4. Покажите какие пиццы являются или вегетарианскими (“Veggie”) или куриными (“Chicken”). Также укажите перечень ингредиентов. (1 балл)

5. Какие пиццы содержат в себе моцареллу (“Mozzarella Cheese”). Также среди атрибутов укажите ингредиенты и названия категорий. (1 балл)
6. Выведите полный список номеров заказов, которые были сделаны в промежутке с начала апреля 2015 года включительно по август 2015 года не включительно с 13:00 по 17:00. (1 балл)
7. Выведите номера заказов в порядке убывания количества купленных пицц. Ограничьте вывод 10-ю заказами. (1 балл)
8. Получите дату и время 10-ти заказов, в которых было куплено больше всего пицц. Упорядочьте записи по убыванию количества купленных позиций, по возрастанию даты и затем времени в лексикографическом порядке. (1 балла)
9. Получите полное меню ресторана и цены на каждую позицию. В запросе должны быть отражены название, размер, стоимость, категория пиццы и ингредиенты, из которых она изготовлена. (1 балл)
10. Выведите количество раз, когда каждая позиция меню (пицца и размер) была куплена не в единственном экземпляре (quantity != 1) за весь промежуток времени. (1 балл)
11. Получите полную таблицу транзакций и детализацию покупок за все время наблюдения (соединение всех таблиц в одну). Отдельными столбцами выведите дату покупки и время покупки, а также полную стоимость позиции, исходя из расчета на количество купленных товаров (total_price). (2 балла)
12. На полученную таблицу из задания 11 создайте представление с помощью команды CREATE VIEW "новое_имя_представления" AS ("ваш запрос"). (1 балл)
13. Подсчитать полный доход от разных категорий пицц за весь период наблюдения. Отсортируйте результат в порядке убывания

дохода. Округлите результат подсчета дохода до второго знака после запятой. (1 балл)

14. Пиццы какого размера продавались больше всего за 3-ий и за 4-ый квартал 2015 года? (1 балл)

Вопросы на защиту

1. Объясните разницу между OLTP, OLAP и встраиваемыми системами управления данными.
2. Как осуществляется доступ к файловым СУБД?
3. Как в СУБД осуществляется доступ к записям таблиц?
4. Каким образом можно отфильтровать записи в таблице данных?
5. Каким образом можно сортировать записи таблицы данных по определенному полю в запросе на записи таблицы?
6. Как вводится ограничение на количество записей, выводимых в запросе?
7. Каким образом можно ввести правило на лексикографическую сортировку таблицы по нескольким атрибутам? Как можно настроить порядок лексикографического правила сортировки?
8. Как осуществляется операция агрегации данных в пределах одной таблицы в РСУБД?
9. Каким образом можно объединить атрибуты нескольких таблиц в реляционной базе данных?
10. Каким образом можно использовать вложенные запросы внутри запросов к таблицам базы данных?
11. Что такое временные таблицы и как они помогают при сложных запросах к базе данных?
12. Что такое представление таблиц в РСУБД и каким образом его можно создать?