Часть 1 (Предисловие)

Предисловие

SQL – SEQUEL (по другой версии, Structured Query Language) – язык запросов к серверу БД.
SQL интересен, главным образом, как интерфейс между сервером БД и приложением, работающим с БД. Действительно, использование стандартного решения — это решение, которое заведомо в интересах процесса разработки ПО; это и объясняет внедрение, распространение, и широкое использование SQL.
SQL был разработан в 70х годах. Первый стандарт был издан в 86г.
Однако, используя SQL, приходится иметь дело не со стандартом языка , а с его реализацией на конкретном сервере БД. Действительно, разные серверы БД по разному реализуют стандарт языка SQL, и различия могут быть существенны. Однако, изучать SQL в свете поддержки этого языка различными серверами БД было бы затруднительно.
Поэтому, язык SQL будет изложен следующим образом:
сначала будут рассмотрены средства самого языка (они поддерживаются именно в таком виде на сервере MySQL, прочие сервера БД обнаруживают различия даже в этих средствах);
затем будут рассмотрены средства MySQL, дополняющие средства самого языка (действительно,чтобы работать с этим языком, необходимо освоить средства его реализации; в качестве реализации будет рассмотрен сервер MySQL – по вышеописанной причине);

Формальные определения

Для изучения языка SQL необходимо освоить соответствующую терминологию. Далее в этих записях эти термины будут широко использоваться.

БД

это, собственно БД той или иной СУБД; БД, при рассмотрении их в абстракции от конкретной СУБД, интересны только теоретически;

СУБД

это, собственно, система управления базами данных;

Иерархические БД

это БД, в которых связи между данными образуют иерархические структуры;

Сетевые БД

это БД, в которых связи между данными образуют сетевидные структуры;

Прочие БД

это БД, в которых связи между данными образуют прочие структуры; это, например, реляционные БД;

Реляционные БД

это БД, в которых связи между данными образуют какие- либо структуры, рациональные с точки зрения конкретной БД; всвязи с чем, эти БД организуются в виде таблиц, и данные в них связываются при помощи включения в таблицы связующих данных (подробности организации реляционных БД будут рассмотрены далее); описываемые ниже термины относятся к реляционным БД;

Таблица

это, собственно, таблица; она представляет собой секвенцию строк таблицы;

Строка таблицы

это, собственно, строка таблицы; все строки таблицы поделены на столбцы абсолютно одинаково; одна строка таблицы описывает одну сущность (точнее, данные следует хранить так, чтобы одна сток таблицы описывала одну сущность);

Сущность

это, собственно, предмет, описываемый в БД; необходимо помнить, что таблицы разбиваются на столбцы абсолютно одинаково во всех строках, поэтому одна таблица подходит для хранения в ней однотипных сущностей;

Столбец

это, собственно, столбец таблицы;

Ключ

это связующие данные, включаемые в таблицу (для хранения ключей в таблице отводятся целевые столбцы); ключ служит для идентификации сущности; так как сущности хранятся в таблице по строкам, то у каждой строки должен быть ключ, идентифицирующий ее;

Первичный ключ

это ключ, служащий для идентификации сущности (строки таблицы); в интересах идентификации сущности, этот ключ должен однозначно идентифицировать сущность; у сущности может быть только один первичный ключ;

Внешний ключ

это ключ, идентифицирующий сущность из другой таблицы; в интересах идентификации сущности, внешний ключ идентифицирует эту сущность по первичному ключу этой сущности;
у сущности может быть и несколько таких ключей;

Рекурсивный ключ

это ключ, идентифицирующий другую сущность из своей же таблицы;
у сущности может быть и несколько таких ключей;

Составной ключ

это ключ, составленный из нескольких столбцов;
любой ключ может быть составным;

Результирующий набор

это, собственно, результирующий набор данных, запрошенных из целевых столбцов, и возвращенных в результате исполнения сервером БД запроса к БД;

Часть 2 (Введение)

Некоторые общие сведения

Текст запроса на языке SQL составляет собственно код запроса, и комментарии к нему.
Комментарии включаются в текст запроса в стиле C (и на них налагаются те же ограничения).
Собственно код на языке SQL составляют его лексеммы с учетом формального синтаксиса.
Наиболее общие синтаксические правила:
пользовательские лексеммы не должны быть омографичны языковым;
запись языковых лексемм регистронезависима;
запись пользовательских лексемм регистронезависима;
код запроса допускает верстку пробельными символами;

Об именах

Идентификаторы

Идентификаторы — это, как обычно имена. Длина имени — это специфика версии сервера БД. Запись имени может начинаться с литеры, последующие символы могут быть литерами, нумерами, или символами подчеркивания (символами _). Различные версии серверов БД могут вносить в эти правила свою специфику.

О переменных

Переменные — это, как обычно, данные, к которым обращаются по имени.
Поскольку SQL – это язык запросов к БД, то переменными, с которыми обычно ведется работа, являются базы данных и их составляющие. Конечно, есть и не только вышеобозначенные переменные, но и переменные вообще — пользователь может объявить переменную указанного типа и с указанным именем, и использовать ее в дальнейшем. Но, такие переменные реализуются различно на разных серверах БД (подробнее — см. прочие источники). Поэтому, целесообразно вернуться к рассмотрению переменных, с которыми обычно и приходится работать. Переменными, непосредственно хранящими полезную нагрузку, являются столбцы таблицы. Так как таблица (и все хранящиеся в ней

сущности) разбивается на столбцы определенным образом, то типы данных приписываются целым столбцам сразу при объявлении таблицы. Различные серверы БД поддерживают различные наборы типов данных. Наиболее общими для них являются следующие типы данных.

Типы данных

Логический тип

Значения логического типа возвращают логические операторы и операторы сравнения; эти значения — `TRUE`, и `FALSE`; при преобразовании между логическими значениями и числами значение `TRUE` соответствует `1`, `FALSE` соответсвует значению `0`;

Специальный тип NULL

Специальный тип `NULL` служит для хранения специального значения `NULL`; это значение возвращается при считывании пустого столбца, и когда выражение не возвращает определенного значения; операции над значением `NULL` бессмыслены, так как результат будет `NULL`, чтобы результат не был заведомо `NULL` требуется инициализация другим значением; значения `NULL` демонстрируют особенное поведение — они не сравнимы оператором сравнения (причем, даже друг с другом), так что для проверки на `NULL` следует использовать запись `IS NULL`;

Символьные типы

это типы `CHAR`, `VARCHAR`, и типы перечислений `ENUM` и `SET`; они предназначены для хранения секвенций символов в переменной; вместимость полезной нагрузки каждого символьного типа на единицу меньше его емкости; значения символьных типов прописываются в одинарных кавычках; работая с символьными типами необходимо помнить следующее (если тип данных не вмещает вводимую в него секвенцию, то онапросто усекается; кодировка символов может быть любой; эскейп- последовательности — то же, что и в `C`);

CHAR

служит для хранения секвенции символов фиксированной длины; вводимая секвенция символов автоматически дополняется пробелами до фиксированной длины; длина полезной нагрузки может быть ограничена пользователем до указанной им величины — это делается записью `CHAR(SequenceLenght)`, более того, эту запись и следует употреблять — ее отсутствие зачастую принимается серверами БД за человеческую ошибку; емкость типа — 256 byte;

VARCHAR

служит для хранения секвенций символов вариативной длины; вводимая секвенция символов не дополняется пробелами до фиксированной длины; длина полезной нагрузки может быть ограничена пользователем до указанной им величины — это делается записью `VARCHAR(SequenceLenght)`, более того, эту запись и следует употреблять — ее отсутствие зачастую принимается серверами БД за человеческую ошибку;
емкость типа — 256 byte;

ENUM

служит для хранения секвенции символов из списка перечисления (это перечисление хранит секвенцию из одного символа — это один символ из списка перечисления); перечисление `ENUM` задается записью `ENUM(Value0, … );`

SET

служит для хранения секвенции символов из списка перечисления (это перечисление хранит секвенцию из одного или нескольких символов — это один или несколько символов из списка перечисления); перечисление `SET` задается записью `SET(Value0, … );`

Текстовые типы

это типы `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, и `LONGTEXT`; текстовые типы сродни символьным типам, но они предназначены для хранения секвенций символов, по размерам сравнимых с текстовым документом; данные текстовых типов хранятся в таблицах не непосредственно, а по ссылке;

TINYTEXT

емкость типа — 256 byte;

TEXT

емкость типа — 64 Kb;

MEDIUMTEXT

емкость типа — 16 Mb;

LONGTEXT

емкость типа — 4 Gb;

Двоичные типы

это типы TINYBLOB, BLOB, MEDIUMBLOB, и LONGBLOB; двоичные типы сродни текстовым типам, но они предназначены для хранения секвенций байтов; данные двоичных типов хранятся в таблицах не непосредственно, а по ссылке;

TINYBLOB

емкость типа — 256 byte;

BLOB

емкость типа — 64 Kb;

MEDIUMBLOB

емкость типа — 16 Mb;

LONGBLOB

емкость типа — 4 Gb;

Численные типы

это типы BOOL, BIT, TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT, FLOAT, и DOUBLE;
служат для хранения численных значений; все численные типы (целочисленные и fp- типы) могут быть знаковыми, и беззнаковыми (для их объявления используется запись ColType UNSIGNED);
значения всех численных типов прописываются в десятичной системе счисления; значения fp- типов могут прописываться как в обычном виде (2.123), так и в научном (2.123E-789);
для значений fp- типов можно ограничить общее количество разрядов (precesion), и количество разрядов в дробной части (scale), это делается записью в объявлении (ColType(PrecesionValue, ScaleValue)) — в таком случае сохраняемые в этих столбцах значения будут округляться до ближайшего числа (допустимого размера), если не вмещаются в них;

`BIT`

это численный тип; конкретнее, это целочисленный тип; способен хранить один бит данных;
и, как обычно о размере этого типа (в обычном смысле) говорить не целесообразно;

`BOOL`

это численный тип; конкретнее, это целочисленный тип; способен хранить один бит данных;
и, как обычно о размере этого типа (в обычном смысле) говорить не целесообразно;

`TINYINT`

это целочисленный тип; емкость типа — 1 byte;

`SMALLINT`

это целочисленный тип; емкость типа — 2 byte;

`MEDIUMINT`

это целочисленный тип; емкость типа — 3 byte;

`INT`

это целочисленный тип; емкость типа — 4 byte;

`BIGINT`

это целочисленный тип; емкость типа — 8 byte;

`FLOAT`

это fp- тип; емкость типа — 4 byte;

`DOUBLE`

это fp- тип; емкость типа — 8 byte;

Временные типы

это типы `YEAR`, `DATE`, `DATETIME`, `TIME`, `TIMESTAMP`; эти типы служат для хранения временных значений; непосредственные значения временных типов задаются строкой (полезная нагрузка которой представлена в соответствующе формате); работая с временными типами необходимо помнить, что разные серверы БД допускают разные диапазоны значений различных временных типов;

YEAR

служит для хранения года, в формате `YYYY`;

DATE

служит для хранения даты, в формате `YYYY-MM-DD`;

DATETIME

служит для хранения даты и времени, в формате `YYYY-MM-DD HH:MI:SS`;

TIME

служит для хранения времени, в формате `HH:MI:SS`;

TIMESTAMP

служит для хранения временного штампа, в формате `YYYY-MM-DD HH:MI:SS`;

Часть 3 (Операторы)

Операторы

Оператор, как обычно, это запись некой операции. Такая запись, как обычно, завершается точкой с запятой.
Так как SQL – это язык запросов к БД, то операторы в нем принято называть запросами. Запросы подразделяются на блоки. Блок запроса — это просто секвенция лексемм запроса. Подразделение запроса на блоки происходит по ключевым словам — ключевое слово с зависимыми от него словами (языковыми и пользовательскими лексеммами) образует блок.

Составление выражений

Выражения в языке SQL ограничиваются блоками запроса; и, специальных символов для их ограничения не требуется.
Приоритет операторов в выражении, как обычно, можно задать с помощью круглых скобок.

Интерпретирование запроса

Запрос к БД выполняется локально, или через сеть. В любом случае происходит взаимодействи пользователя с сервером БД.
Сервер требует от пользователя входа под своей учетной записью. Пользователь для входа под своей учетной записью сообщает серверу свои username, password, и hostname (в случае входа через сеть).
По проверке этих данных сервер устанавливает соединение (и высылает id соединения пользователю), или же отказывает в нем. Если соединение установлено, то оно существует на протяжении всего сеанса работы пользователя с сервером БД.
При установленном соединении пользователь может вводить запросы к серверу БД. По запросу сервер проверяет, может ли этот запрос быть исполнен. По этой проверке сервер передает этот запрос оптимизатору запросов (или же отбрасывает его). Оптимизатор запросов рассматривает запрос, и генерирует план исполнения запроса. Запрос исполняется сервером по этому плану. По исполнению запроса сервер возвращает пользователю результирующий набор этого запроса.

Классификация операторов

Операторы языка SQL целесообразно классифицировать следующим образом:
арифметические операторы;
логические операторы;
побитовые операторы;
операторы сравнения;
операторы, образующие блоки запроса;

эти категории операторов описываются далее.

Арифметические операторы

Арифметические операторы задаются следующими знаками.

+	-
*	/

Логические операторы

Логические операторы задаются следующими знаками.

NOT	NOT
AND	AND
OR	OR

Для операции XOR не предусмотрено отдельного знака. Эта операция задается эквивалентной последовательностью операций.

Побитовые операторы

Эти операторы — специфика сервера БД (в MySQL – в точности как в C).

Операторы сравнения

Операторы сравнения задаются следующими знаками.

=	РАВНО
!=	НЕ РАВНО
>	БОЛЬШЕ
<	МЕНЬШЕ
>=	БОЛЬШЕ ИЛИ РАВНО
<=	МЕНЬШЕ ИЛИ РАВНО
<>	НЕ РАВНО
LIKE	ColName LIKE Value

IN	ColName IN (Value0, …)
BETWEEN	BETWEEN LowerValue AND UpperValue

Оператор LIKE часто используется для поиска по частичным совпадениям. Частичные совпадения в строках могут быть обозначены в их записи при помощи следующих символов:

_	означает любой символ
%	означает сколько- нибудь любых символов

Логика работы остальных операторов сравнения и так понятна из описания, приведенного в списке этих операторов.

Приоритет операторов

()
унарные + и –
* /
+ –
операторы сравнения
побитовые операторы
логические операторы
операторы, образующие блоки запроса
оператор присваивания

Часть 4 (Блоки)

О классификации

Операторы, образующие блоки запроса, целесообразно подразделять на следующие операторы.

DDL – Data Definition Language

это операторы `CREATE`, `ALTER`, `DROP`;

DML – Data Manipulation Language

это операторы `SELECT`, `INSERT`, `DELETE`, `UPDATE`;

DCL – Data Control Language

это операторы `GRANT`, `REVOKE`, `DENY`;

TCL – Transaction Control Language

это операторы `COMMIT`, `ROLLBACK`, `SAVEPOINT`;

DDL

CREATE

Этот оператор используется для создания БД и их составляющих (для создания собственно БД и их таблиц).
Реляционные БД — это набор таблиц (которые могут быть связаны друг сдругом ключами), таким образом, собственно БД используются как контейнеры таблиц, а полезная нагрузка хранится в таблицах.
собственно БД создаются записью:

```
CREATE DATABASE DBName;
```

после использования этой записи работа будет проистекать с созданной БД. Чтобы приступить к работе с какой- либо определенной БД (из числа уже созданных), прибегают к записи:

```
USE DBName;
```

При создании собственно БД целесообразно прибегать к записи:
```
CREATE DATABASE IF NOT EXISTS DBName;
```

и тогда запрос на создание БД будет признан корректным даже если было прописано имя уже существующей БД.
Чтобы приступить к формированию полезной нагрузки БД, необходимо создать (и затем заполнить) ее таблицы. Таблицы создаются записью:
```
CREATE TABLE TableName (ColName0 ColType0, … );
```

эта запись создаст таблицу, разбитую на столбцы указанным образом. Как видно из этой записи, этот запрос создает не полезную нагрузку таблицы, а ее структуру; поэтому, эти сведения принято называть не данными, а метаданными. Метаданные хранятся в так называемом словаре данных. Метаданные, как и все данные в реляционных БД, хранятся в табличном виде. Метаданные допускают запросы к ним, но они хранятся в БД, специфичных тому или иному серверу БД.
При создании таблиц необходимо определить столбцы, образующие первичный ключ ее строк. Также можно определить и столбцы, образующие внешний ключ. О ключах — в соответствующей главе.
Таблицы могут быть постоянными (хранимыми постоянно), и временными (хранимыми во время действия соединения с сервером БД).
Временные таблицы создаются записью:
```
CREATE TEMPORARY TABLE TableName (ColName0 ColType0, … );
```

При создании таблицы целесообразно прибегать к записи:
```
CREATE TABLE IF NOT EXISTS TableName;
```

и тогда запрос на создание таблицы будет признан корректным, даже если было прописано имя уже существующей таблицы. Разумеется, если создается временная таблица, то следует прописывать `… TEMPORARY TABLE … .`
Таблицу можно создать и по образцу уже существующей таблицы, для этого прибегают к записи:
```
CREATE TABLE TableName LIKE OldTableName;
```

Существующие таблицы могут быть переименованы, следующей записью:
```
RENAME TABLE TableName0 TO NewTableName0,
 … ,
TableNameN TO NewTableNameN;
```

ALTER

Используется для изменения уже существующих таблиц. Форма записи:
```
ALTER TABLE TableName … ;
```

в этой записи … это целевые изменения, то есть это следующее.

```
ADD COLUMN ColName ColType, … ;
```

приведенная выше запись используется для добавления в таблицу столбцов; при добавлении столбцов могут быть задействованы те же средства, что и при применении оператора `CREATE` по отношению к таблицам; лексемму `COLUMN` можно опустить;

```
DROP COLUMN ColName;
```

приведенная выше запись используется для удаления столбцов таблицы; лексемму `COLUMN` можно опустить;

```
ADD CONSTRAINT … ;
```

приведенная выше запись используется для добавления в таблицу ограничений (вместо лексеммы `CONSTRAINT` прописывается целевая лексемма); об ограничениях — в соответствующей главе;

```
DROP CONSTRAINT … ;
```

приведенная выше запись используется для удаления из таблицы ограничений (вместо лексеммы `CONSTRAINT` прописывается целевая лексемма); об ограничениях — в соответствующей главе;

```
MODIFY COLUMN ColName ColType;
```

приведенная выше запись используется для изменения целевого столбца; поддерживается для совместимости с прочими серверами БД, и не является стандартной; при изменении столбца могут быть задействованы те же средства, что и при применении оператора `CREATE` по отношению к таблицам; лексемму `COLUMN` можно опустить;

```
ALTER COLUMN ColName … ;
```

приведенная выше запись используется для изменения целевого столбца; изменения достигаются применением в этой записи `SET DEFAULT Value` или `DROP DEFAULT` вместо `…` ; лексемму `COLUMN` можно опустить;

```
CHANGE COLUMN ColName NewColName ColType … ;
```

приведенная выше запись используется для изменения целевого столбца; поддерживается как средство, специфичное серверу `MySQL`; при изменении столбца могут быть задействованы те же средства, что и при применении

оператора `CREATE` по отношению к таблицам; лексемму `COLUMN` можно опустить;

```
RENAME TO NewTableName;
```

приведенная выше запись используется для переименования таблицы;

DROP

Используется для удаления БД. Удаление БД производится записью:
```
DROP DATABASE DBName;
```

При удалении БД целесообразно прибегать к записи:
```
DROP DATABAS  IF EXISTS DBNName;
```

и тогда запрос будет признан корректным, даже если указано имя не существующей БД.
Чтобы удалить конкретные таблицы из БД используют запись:
```
DROP TABLE TableName0, … , TableNameN;
```

DML

SELECT

Оператор `SELECT` используется для запроса данных. Выполнив этот запрос, сервер БД возвращает запрошенные данные, причем, в виде, характерном для реляционных БД — в табличном виде.
Этот оператор запрашивает столбцы, возвращаемые в результирующий набор. Эти столбцы могут быть столбцами тех или иных таблиц, или же формируемыми при исполнении запроса (то есть, быть значениями выражения, составленного программистом). Столбцы этих двух видов могут быть запрошены в одном запросе — это не возбраняется.
Именами столбцов, возвращаемых в результирующий набор, будут имена запрошенных из таблиц столбцов. Если же запрошены столбцы (формируемые в результате вычисления выражений), то именами таких столбцов будут записи этих выражений. Столбцы этих двух видов могут быть возвращены в результирующий набор и под псевдонимами — для этого программист должен прописать псевдонимы. Псевдоним столбца прописывается просто за именем столбца (или выражением — для столбцов, формируемых в результате вычисления выражений).
Запрос столбцов производится соответствующей записью:
```
SELECT ColName0, … ;
```

Если запрашиваются столбцы (формируемые выражением), то эти выражения просто прописываются в списке запрошенных столбцов (эти выражения целесообразно заключать в круглые скобки).
Из таблицы могут быть запрошены как некоторые столбцы, так и все. Для запроса всех столбцов служит упрощенная запись — имена всех столбцов таблицы заменяются на символ астериск (символ `*`).
В результирующем наборе могут быть значения (сущности), повторяющиеся в строках; для того, чтобы повторы сущностей не учитывались, следует прописать `DISTINCT` непосредственно после ключевого слова `SELECT`. Можно указать, чтобы повторы значений учитывались (хотя они и так учитываются по умолчанию), это делается записью `ALL` непосредственно после ключевого слова `SELECT` (эта запись избыточна).
Блок `SELECT` обычно включает в себя ряд определенных блоков.
Блок `FROM` указывает таблицу, из которой запрашиваются столбцы (хотя, при использовании соединений, столбцы запрашиваются далеко не из единственной таблицы, а из нескольких таблиц; однако, это уже подробности о соединениях — и они будут изложены в соответствующем параграфе этой подглавы). Хотя столбцы могут быть запрошены не только из таблиц, но и сформированы выражениями, блок `FROM` все же следует считать обязательным подблоком блока `SELECT`. Это потому, что отсутствие блока `FROM` является частой причиной ошибок (возникающих по самым разным причинам). Поэтому всегда следует прописывать блок `FROM` (даже если все запрошенные столбцы формируются выражениями — на этот случай есть специфическая запись (`SELECT … FROM DUAL;`), поддерживаемая всеми серверами БД).
В записи блок `FROM` используется так:

```
SELECT ColName0, …
FROM TableName;
```

Блок `WHERE` указывает условие, по которому запрошенные данные включаются в результирующий набор (это так называемое «условие фильтрации»). Условие фильтрации задается выражением; это выражение составляется на усмотрение программиста (однако, чтобы его применение имело смысл, оно должно зависеть от запрашиваемых столбцов); если это выражение истинно, то значения включаются в результирующий набор. Если же условие фильтрации не задано, то в результирующий набор безусловно возвращаются все значения из указанных в блоке `SELECT` столбцов.
В записи блок `WHERE` используется так:

```
SELECT ColName0, …
FROM TableName
WHERE … ;
```

Блок `ORDER BY` используется для упорядочивания возвращаемых в результирующий набор данных. Так как результирующий набор — это таблица, то его строки предполагаются содержащими сущности; поэтому, упорядочивание происходит по строкам результирующего набора. Блок `ORDER BY` содержит условие упорядочивания. Условием упорядочивания является выражение —

обычно, это запись столбца, по значениям которого и будет происходить упорядочивание. В условии упорядочивания можно указать и список столбцов — тогда упорядочивание будет происходить по их значениям (тогда упорядочивание происходит сначала по первому прописанному в условии столбцу, затем по следующему столбцу — сохраняя при этом упорядоченность по предыдущему столбцу, и так итеративно по следующим столбцам этого списка). Столбцы, по значениям которых происходит упорядочивание, могут быть заданы не только именем, но и порядковым номером — порядковые номера задаются в натуральных числах. Условие упорядочивания это выражение, которое может быть составлено на усмотрение программиста (однако, чтобы его применение имело смысл, это выражение должно зависеть от столбцов таблицы). По умолчанию, упорядочивание происходит по возрастанию (в последующих строках значение возрастает). Можно задать упорядочивание и по убыванию (значения в определенном столбце) — записью `ORDER BY ColName DESC`. Для упорядочивания по возрастанию (значения в определенном столбце) допустима запись `ORDER BY ColName ASC` (эта запись избыточна).
В записи блок `WHERE` используется так:

```
SELECT ColName0, …
FROM TableName
WHERE …
ORDER BY … ;
```

Блок `SELECT` может включать в себя и прочие блоки. Прочие возможности использования блока `SELECT` описываются в следующих параграфах этой подглавы.

Группировка

Запрашиваемые данные могут быть сгруппированы. То есть, если запрос возвращает запрошенные столбцы в нескольких строках, и приэтом они имеют в разных строках одинаковые значения, то результаты запроса могут быть сгруппированы в одну строку. Разумеется, чтобы группировка была возможна, все запрошенные столбцы должны иметь повторяющиеся в разных строках значения (и исключений из этого быть не должно). Когда группировка запрошенных данных не возможна, она просто не будет выполняться (то есть, в результате запроса (использующего группировку) может быть возвращено более одной строки).
Запросы, использующие группировку прописываются так:

```
SELECT …
FROM …
GROUP BY … ;
```

В этой записи столбцы из блока `SELECT` должны повторяться в блоке `GROUP BY`. Действительно, запрашиваемые данные не смогут быть сгруппированы в одну строку при невыполнении этого условия. Конечно, запрашиваемые данные

могут быть возвращены и в нескольких строках, но выполнение этого условия необходимо для принципиальной возможности группировки.
Причем, если необходимо выполнить фильтрацию уже сгруппированных данных, то это делается при помощи соответствующего блока. Когда как блок `WHERE` выполняет фильтрацию до группировки, то блок `HAVING` выполняет фильтрацию после группировки. И тот, и тот блок, как обычно, подразумевает условное выражение (оно, как обычно, составляется на усмотрение программиста).
Запись запроса с этими блоками:

```
SELECT …
FROM …
WHERE …
GROUP BY …
HAVING … ;
```

Агрегатные функции

При применении в запросах группировки, часто применяются и агрегатные функции. Агрегатные функции — это встроенные функции, предназначенные для решения типовых задач, сопутствующих группировке.
Агрегатные функции обладают своей спецификой (следующей из особенностей их применения).
Эти встроенные функции, будучи вызваны, применяются по всем строкам группы строк (то есть по всем строкам, которые могут быть сгруппированы). Поэтому, агрегатные функции доступны после выполнения группировки (не до группировки). Впрочем, агрегатные функции доступны и без использования в запросе блока `GROUP BY` – и тогда группировка будет считаться и так исполненой, и группой строк будет считаться набор всех возвращенных строк (это так называемая «неявная группа»). Если агрегатные функции используются без использования в запросе блока `GROUP BY`, то могут быть запрошены только столбцы, формируемые этими функциями. Это потому, что язык `SQL` запрещает смешивать в запросе столбцы, являющиеся результатами группировки, и не являющиеся результатами группировки. Смешивание таких столбцов невозможно при использования в запросе блока `GROUP BY`, так как такой запрос (будучи синтаксически правильным) подразумевает, что столбцы из блока `SELECT` должны повторяться в блоке `GROUP BY`. Касательно данного случая стоит заметить следующее — те столбцы из блока `SELECT`, которые сформированы агрегатными функциями, не должны повторяться в блоке `GROUP BY`, и в этом плане они составляют исключение.
Касательно реакции агрегатных функций на значения `NULL` стоит заметить следующее. Значения `NULL`, как обычно, не участвуют в вычислениях. Но, они не вызывают ошибок при исполнении запроса. Строки, содержащие значения `NULL` также могут подсчитываться, как и строки с другими значениями.
Набор агрегатных функций может включать специфичные для того или иного сервера БД функции; наиболее общими функциями являются следующие.

`Min()`

используется для возврата минимального значения в группе (точнее, результатах группировки); принимает параметр — целевой столбец таблицы (по нему и будет вычисляться минимальное значение в группе); возвращает минимальное значение в группе; возвращает значение (его тип зависит от типа аргументов);
в области параметров можно указать `DISTINCT` перед записью аргумента (это, как обычно, приводит к исключению повторов значений), или `ALL` (это, как обычно, приводит к учету всех значений);
в качестве параметра можно указать не только целевой столбец, но и выражение (оно составляется на усмотрение программиста; но, чтобы оно имело смысл в рамках логики агрегатной функции, оно должно зависеть от столбцов таблицы, и возвращать единственное значение — действительно, при выполнении этого условия агрегатная функция сможет применяться в рамках своей характерной логики (то есть, по группам строк));

`Max()`

используется для возврата максимального значения в группе (точнее, результатах группировки); принимает параметр — целевой столбец таблицы (по нему и будет вычисляться максимальное значение в группе); возвращает максимальное значение в группе; возвращает значение (его тип зависит от типа аргументов);
в области параметров можно указать `DISTINCT` перед записью аргумента (это, как обычно, приводит к исключению повторов значений), или `ALL` (это, как обычно, приводит к учету всех значений);
в качестве параметра можно указать не только целевой столбец, но и выражение (оно составляется на усмотрение программиста; но, чтобы оно имело смысл в рамках логики агрегатной функции, оно должно зависеть от столбцов таблицы, и возвращать единственное значение — действительно, при выполнении этого условия агрегатная функция сможет применяться в рамках своей характерной логики (то есть, по группам строк));

`Avg()`

используется для возврата среднего по группе (точнее, в результатах группировки) значения; принимает параметр — целевой столбец таблицы (по нему и будет вычисляться среднее значение в группе); возвращает среднее значение в группе; возвращает значение (его тип зависит от типа аргументов);
в области параметров можно указать `DISTINCT` перед записью аргумента (это, как обычно, приводит к исключению повторов значений), или `ALL` (это, как обычно, приводит к учету всех значений);
в качестве параметра можно указать не только целевой столбец, но и выражение (оно составляется на усмотрение программиста; но, чтобы оно имело смысл в рамках логики агрегатной функции, оно должно зависеть от столбцов таблицы, и

возвращать единственное значение — действительно, при выполнении этого условия агрегатная функция сможет применяться в рамках своей характерной логики (то есть, по группам строк));

`Sum()`

используется для возврата суммы значений в группе (точнее, результатах группировки); принимает параметр — целевой столбец таблицы (по нему и будет вычисляться сумма значений в группе); возвращает сумму значений в группе; возвращает значение (его тип зависит от типа аргументов);
в области параметров можно указать `DISTINCT` перед записью аргумента (это, как обычно, приводит к исключению повторов значений), или `ALL` (это, как обычно, приводит к учету всех значений);
в качестве параметра можно указать не только целевой столбец, но и выражение (оно составляется на усмотрение программиста; но, чтобы оно имело смысл в рамках логики агрегатной функции, оно должно зависеть от столбцов таблицы, и возвращать единственное значение — действительно, при выполнении этого условия агрегатная функция сможет применяться в рамках своей характерной логики (то есть, по группам строк));

`Count()`

используется для возврата количества значений в группе (точнее, результатах группировки); принимает параметр — целевой столбец таблицы (по нему и будет вычисляться количество значений (строк) в группе); возвращает количество значений (строк) в группе; возвращает значение (его тип зависит от типа аргументов);
в области параметров можно указать `DISTINCT` перед записью аргумента (это, как обычно, приводит к исключению повторов значений), или `ALL` (это, как обычно, приводит к учету всех значений);
в качестве параметра можно указать не только целевой столбец, но и выражение (оно составляется на усмотрение программиста; но, чтобы оно имело смысл в рамках логики агрегатной функции, оно должно зависеть от столбцов таблицы, и возвращать единственное значение — действительно, при выполнении этого условия агрегатная функция сможет применяться в рамках своей характерной логики (то есть, по группам строк));
в случае этой конкретной функции в области параметров можно указать `*` (действительно, эта функция отвечает за подсчет строк, и поэтому ей не важно, какой целевой столбец ей указать; поэтому для этой функции и предусмотрена специфическая форма записи параметров);

Подзапросы

Подзапросы это запросы, инкапсуллированные другими запросами.

Подзапросы могут оперировать столбцами, использующимися в инкапсуллирующем запросе («связанные подзапросы»), могут и не оперировать («несвязанные подзапросы»). Несвязанные подзапросы завершают свое исполнение заведомо раньше инкапсуллирующего их запроса. Связанные подзапросы завершают свое исполнение отнюдь не заведомо раньше инкапсуллирующего их запроса. Ввиду наличия у сервера БД оптимизатора запросов об исполнении связанных подзапросов нецелесообразно говорить подробнее (по крайней мере, в общем случае).
Подзапросы возвращают данные в том же виде, что и (не вложенные) запросы — в табличном виде. При этом, если они возвращают единственное значение, то это значение (при необходимости) может интерпретироваться как значение элементарных типов данных (а не таблица). Подзапросы, возвращающие единственное значение, принято называть «скалярными»; скалярные подзапросы могут быть как связанными, так и несвязанными, скалярность сама по себе на порядок исполнения подзапроса не влияет.
Подзапросы прописываются в тех составляющих выражения запроса, которые должны возвращать данные. Они могут использоваться в различных запросах (не только `SELECT`). Подзапросы (с официальной точки зрения) могут быть сложными настолько, насколько необходимо. Однако, подзапросы и так усложняют запись запроса (хотя, программист должен понимать, что соглашения об оформлении кода все же должны быть выполнены). Поэтому, чтобы подзапросы не были источниками неисправностей, их запись должна быть выдержана достаточно простой. В особенности, необходимо проследить за следующим: подзапросы не должны запрашивать все столбцы подряд (использовать `*`), и не должны использоваться во всех блоках запроса подряд, и содержать все блоки (характерные для запроса) подряд (реализации языка `SQL` могут накладывать такие ограничения).
Подзапросы прописываются как обычные запросы, заключенные в круглые скобки.

Соединения

Соединение — это специфический механизм, позволяющий запрашивать столбцы из разных таблиц, соединяя их в строки (и возвращая в результирующий набор). Возвращаемые запросом столбцы соединяются друг с другом в строки результирующего набора; причем, способ соединения зависит от того, какой вид соединения используется. При использовании соединений необходимо, чтобы соответствующие таблицы были связаны друг с другом (внешний ключ — первичный ключ); но, необходимо помнить, что это высказывание употребляется как единственно верное лишь в дань традиции (по этой же причине оно и было употреблено здесь). Также необходимо, чтобы соответствующим таблицам были приписаны псевдонимы; псевдонимы необходимы даже если используется таблица, возвращенная подзапросом; псевдонимы приписываются записью `TableName AliasName` в записи соединения (и записью `( … ) AliasName` для таблиц, возвращенных подзапросом).

Разные серверы поддерживают разный синтаксис соединений вдобавок к стандартному — это специфики разных реализаций, и поэтому безынтересны.
Описывая разные виды соединений, простоты ради условимся считать, что из каждой таблицы будет запрошен только один столбец (если же в реальных случаях будет запрошен вовсе не один столбец — то это не меняет данной аппрокимативной модели (действительно, секвенция столбцов в одной строке исходной таблицы может быть представлена в виде абстракции — одного единственного столбца)). Можно было бы и не вводить эти условности, но это усложнило бы объяснение логики работы разных видов соединений.
При использовании соединений необходимо помнить, что речь идет о соединении столбцов, запрошенных из разных таблиц, в строках результирующего набора. Сами же таблицы ни коим образом не соединяются. Не стоит обманываться тем, что различные тематические источники говорят о «соединении таблиц» - это лишь просторечие (причем, не уместное).

Различные виды соединений

xJOIN

перекрестное соединение; столбцы соединяются в строки так, что столбцы одной таблицы соединяются со столбцами другой таблицы; причем, соединяются они не в соответственных строках, а каждая с каждой (и количество строк результирующего набора вычисляется как декартово произведение (произведение кол- ва столбцов, запрошенных из одной таблицы на кол- во столбцов из другой)); это соединение задается записью:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 CROSS JOIN TableName1 AliasName1;

/* Comments */
```

INNER JOIN

внутреннее соединение (собственно соединение); столбцы соединяются в строки так, что столбец одной таблицы соединяется со столбцом другой таблицы при исполнении условного выражения (как обычно, условное выражение составляется на усмотрение программиста); это соединение задается записью:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 INNER JOIN TableName1 AliasName1
ON … ;

/* Следует явно задать вид соединения:
 … INNER JOIN …
иначе соединение будет считаться перекрестным,
и тогда условное выражение ON …  будет отброшено,
причем, это касается всех видов соединений;
если условие соединения — это равенство ОДНОИМЕННЫХ столбцов,
```

```
то вместо ON … можно прописать USING (ColName);
причем, эта форма записи соединения употребима во всех
видах соединений, использующих условие соединения */
```

OUTER JOIN

это ряд видов соединений, при которых в результирующий набор включаются не только столбцы, которые сопоставлены (в рамках условного выражения) — в случае несопоставления в соответствующих столбцах результирующего набора будет значение `NULL`; различные виды внешнего соединения реализуют разную логику (они описаны ниже);

LEFT OUTER JOIN

левое внешнее соединение; столбцы соединяются в строки так, что столбец одной таблицы соединяется со столбцом другой таблицы при исполнении условного выражения — а в случае несопоставления (в рамках условного выражения) возвращаются и столбцы из таблицы, левой относительно слова `JOIN`, и им будут сопоставлены значения `NULL`; форма записи:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 LEFT OUTER JOIN TableName1
AliasName1;

/* Comments */
```

RIGHT OUTER JOIN

правое внешнее соединение; столбцы соединяются в строки так, что столбец одной таблицы соединяется со столбцом другой таблицы при исполнении условного выражения — а в случае несопоставления (в рамках условного выражения) возвращаются и столбцы из таблицы, правой относительно слова `JOIN`, и им будут сопоставлены значения `NULL`; форма записи:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 RIGHT OUTER JOIN TableName1
AliasName1;

/* Comments */
```

FULL OUTER JOIN

полное внешнее соединение; столбцы соединяются в строки так, что столбец одной таблицы соединяется со столбцом другой таблицы при исполнении условного выражения — а в случае несопоставления (в рамках условного выражения) возвращаются и столбцы из таблиц, левой и правой относительно слова `JOIN`, и им будут сопоставлены значения `NULL`; форма записи:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 FULL OUTER JOIN TableName1
AliasName1;

/* Этот тип соединения может отсутствовать в MySQL */
```

Об условии соединения и условии фильтрации

Условия соединения и фильтрации — это разные блоки запроса, служащие для разных целей. При использовании соединений условия фильтрации по прежнему доступны, и задействуются следующей записью:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 INNER JOIN TableName1 AliasName1
ON …
WHERE … ;

/* Comments */
```

О соединении ряда таблиц

При соединении столбцов (запрашиваемых из более чем двух таблиц), необходимо помнить, что они соединяются не друг с другом, а со столбцами промежуточного результирующего набора. Промежуточный результирующий набор — это данные, полученные на предыдущем этапе исполнения соединения. Действительно, при соединении столбцов из ряда (более чем двух) таблиц, соединение представляется не тривиальным (происходящим как ряд последовательных операций соединения), и в этом случае приходится говорить о промежуточном результирующем наборе.

Так как соединение столбцов из ряда таблиц происходит как ряд операций соединения, то в этом случае необходимо указать ряд этих операций целевого типа и с целевыми условиями соединения. Запись примет вид:

```
SELECT AliasName0.ColName, …
FROM TableName0 AliasName0 … JOIN TableName1 AliasName1
ON …
… JOIN TableName2 AliasName2
ON …
WHERE … ;

/* Comments */
```

О соединении таблицы самой с собой

Различные тематические источники упомянают о «соединении таблицы самой с собой». Необходимо помнить, что сами таблицы ни коим образом не соединяются. Даже при «соединении таблицы самой с собой». Под этим

выражением подразумевается лишь следующее — неоднократное использование одной таблицы в одном запросе, использующем соединения.
При неоднократном использовании одной таблицы в одном запросе ее столбцы могут соединяться как со столбцами из нее самой (строго говоря, именно этот случай и называется «соединение таблицы самой с собой»), так и со столбцами из других таблиц. Но в любом случае соединение подразумевает ссылочную целостность — задействуемые в соединении таблицы должны быть связаны по ключу (в случае «соединения таблицы самой с собой» в ней должен наличествовать рекурсивный внешний ключ — внешний ключ, указывающий на первичный ключ в пределах своей же таблицы).
Действительно, если логика запроса этого требует, то одна и та же таблица может неоднократно использоваться в запросе, использующем соединения. В таком случае таблица фигурирует в запросе под разными псевдонимами.
Такой запрос выглядит, конечно же, не естественно. Однако, он будет распознан как корректный, если корректно составлен (то есть, если он соответствует правилам синтаксиса, и соглашениям об оформлении кода).

Составные запросы

Составные запросы — это запросы, состоящие из ряда запросов.
Речь идет о запросах `SELECT`.
Составные запросы возвращают данные в установленном виде — в табличном (в виде одной результирующей таблицы). Это обеспечивается тем, что в составных запросах применяются специальные операторы, которые в результате их применения возвращают данные в установленном виде. Эти операторы трактуют свои операнды как множества (возвращенных запросами строк;элементами множеств являются строки), и реализуют операции над множествами. Корректность применения этих операторов обеспечивается тем, что таблицы (по которым они применяются) должны быть разбиты на столбцы одинаковым образом (на одинаковое количество столбцов соответственных типов, имена же этих столбцов могут и различаться). Действительно, при этом данные операторы смогут иметь смысл в применении, и вернуть таблицу определенной структуры. Невыполнение этого условия считается человеческой ошибкой.
Имеются следующие операции над множествами:
«объединение» - результатом объединения множеств является множество, состоящее из элементов, принадлежащих объединяемым множествам (в том числе, из элементов, принадлежащим сразу всем множествам, по которым применена эта операция);
«пересечение» - результатом пересечения множеств является множество, состоящее из элементов, принадлежащих сразу всем множествам, по которым применена эта операция;
«исключение» - результатом исключения из одного множества некого другого множества является множество, состоящее из элементов одного множества за исключением элементов исключаемого множества;

Прочие операции над множествами могут быть получены при помощи этих вышеописанных операций.
Операторы для работы с множествами это бинарные операторы. Они работают с результатами запросов (в составе составных запросов), и прописываются так:

```
SELECT …
Operator
SELECT … ;
```

В составном запросе точка с запятой между составляющими его запросами не ставиться — она ставиться только в конце самого составного запроса.
Составные запросы могут включать в себя и более двух запросов:

```
SELECT …
Operator
SELECT …
Operator
SELECT …
 … ;
```

Результаты составных запросов могут быть упорядочены, но не сгруппированы:

```
SELECT …
Operator
SELECT …
ORDER BY … ;
```

Причем, в блоке `ORDER BY` следует прописать имя столбца из первого запроса. Невыполнение этого требования будет считаться человеческой ошибкой. Действительно, для исполнения данных операторов их операнды — таблицы должны быть разбиты на столбцы одинаковым образом — поэтому, в блоке `ORDER BY` достаточно указать имя столбца из первого запроса.
Операции над множествами реализуются следующими операторами.

`UNION`

этот оператор реализует операцию «объединение»; этот оператор исключает повторы значений — элементов множеств (для того, чтобы в результат включались повторы элементов множеств (то есть, строк), следует прописывать `UNION ALL`); принимает операнды — описано выше; возвращает значение — описано выше (если требуются пояснения — оператор возвращает объединение множеств строк (возвращенных запросами); причем, в том порядке, в котором и прописано в запросе — сначала из одной таблицы, затем из другой (если конечно в записи составного запроса не используется блок `ORDER BY`));

`INTERSECT`

этот оператор реализует операцию «пересечение»; этот оператор исключает повторы значений — элементов множеств (для того, чтобы в результат

включались повторы элементов множеств (то есть, строк), следует прописывать INTERSECT ALL); принимает операнды — описано выше; возвращает значение — описано выше (если требуются пояснения — оператор возвращает пересечение множеств строк (возвращенных запросами); причем, в том порядке, в котором и обнаруживается пересечение этих множеств по ходу исполнения данного оператора (если конечно в записи составного запроса не используется блок ORDER BY); если же эти множества — не пересекающиеся, то результатом будет пустое множество);
этот оператор может не поддерживаться реализацией языка SQL;

EXCEPT

этот оператор реализует операцию «исключение»; этот оператор исключает повторы значений — элементов множеств, и кроме того, и оригиналы повторяющихся значений — действительно, такое поведение соответствует логике операции «исключение» (для того, чтобы оригиналы повторяющихся элементов множеств (то есть, строк) все же включались в результат, следует прописывать EXCEPT ALL); принимает операнды — описано выше; возвращает значение — описано выше (пояснения здесь не требуются);
этот оператор может не поддерживаться реализацией языка SQL;

INSERT

Этот оператор используется для вставки строк в (уже существующую таблицу). Его форма записи:

```
INSERT INTO TableName (ColName0, … )
VALUES (Value0, … );
```

использование этой записи приводит к вставке в целевую таблицу новой строки, в которой в указанных столбцах будут указанные значения.
Альтернативная запись:

```
INSERT INTO TableName (ColName0, … )
SELECT … ;
```

использование этой записи приводит к вставке в целевую таблицу новых строк, в которых в указанных столбцах будут данные, возвращенные блоком SELECT (в данном случае блок SELECT прописывается именно так). В этой форме записи список указанных столбцов (ColName0, …) может быть и опущен, тогда он определяется тем, что возвращает блок SELECT.
Альтернативная запись:

```
INSERT INTO TableName
SET ColName0 = Value0, … ;
```

использование этой записи приводит к вставке в целевую таблицу новой строки, в которой в указанных столбцах будут указанные значения.

DELETE

Этот оператор используется для удаления данных из таблицы. Форма записи:

```
DELETE FROM TableName
WHERE … ;
```

использование этой формы записи приводит к удалению из указанной таблицы сущностей, при выполнении условия, заданного блоком `WHERE` (этот блок содержит условное выражение; его контент — на откуп программиста). Блок `WHERE` может быть опущен — тогда удаление будет безусловным (и удалятся все строки из таблицы).

UPDATE

Этот оператор используется для обновления данных (уже существующей) таблицы. Форма записи:

```
UPDATE TableName
SET ColName = Value, …
WHERE … ;
```

использование этой формы записи приводит к обновлению данных указанной таблицы (инициализации данных в указанных столбцах указанными значениями), при выполнении условия, заданного блоком `WHERE` (этот блок содержит условное выражение; его контент — на откуп программиста). Блок `WHERE` может быть опущен — тогда обновление будет безусловным (и обновятся все строки из таблицы).

DCL

GRANT

Этот оператор используется для наделения пользователей полномочиями. Форма записи:

```
GRANT ActionName0 SubjectName0, …
ON HighterLevelSubjectName
TO UserName0 IDENTIFIED BY 'TheContentOfPassword0'
 … , UserNameN IDENTIFIED BY 'TheContentOfPassWordN';
```

Применение этой записи приведет к наделению указанных пользователей указанными правами. В этой записи `ActionName` может быть одним из следующего (`ALL` – all privileges, `SELECT` – `SELECT`, `CREATE` – `CREATE TABLE`, `CREATE TEMPORARY TABLES` – `CREATE TEMPORARY TABLE`, `INDEX` –

CREATE INDEX/ DROOP INDEX, ALTER – ALTER TABLE, UPDATE – UPDATE, INSERT – INSERT, DELETE – DELETE, DROP – DROP TABLE, USAGE – without privileges); SubJectName может быть следующим (ColName0, …); HighterLevelSubjectName может быть одним из следующего (* - *, *.* - *.*, DBName.* - DBName.*, TableName – TableName); UserName – имя пользователя; 'TheContentOfPassword … ' - строка с паролем пользователя. О прочих возможностях — см. другие источники.

REVOKE

Этот оператор используется для отзыва полномочий пользователей.
Форма записи:

```
REVOKE ActionName SubjectName
ON HighterLevelSubjectName
FROM UserName0, … ;
```

Применение этой записи приведет к отзыву указанных прав у указанных пользователей. Об обозначениях в этой записи — см. GRANT.

DENY

Этот оператор проблематично поддерживается (отсутствует) на сервере MySQL.

TCL

Эти операторы будут рассмотрены в следующих подглавах.

О транзакциях

Транзакция — это секвенция операций, исполняемая как атомарная операция. По исполнении транзакции она фиксируется, по неисполнении — откатывается. По завершении транзакции (успешном, или неуспешном) все используемые в ней разделяемые ресурсы высвобождаются.
Разные серверы БД поддерживают разные механизмы управления транзакциями. Есть следующие механизмы:
все транзакции формируются и управляются сервером неявно, пользователю не доступны какие- либо явные механизмы (при этом сеанс работы пользователя с сервером БД соответствует одной транзакции);
транзакции формируются пользователем явно, и пользователю явно доступны механизмы управления транзакциями (если же транзакции не формируются явно, то они автоматически формируются сервером — один запрос соответствует

одной транзакции); когда транзакции автоматически формируются сервером, это принято называть «работой в режиме автоматической фиксации» (строго говоря, это понятие характерно только данному механизму управления транзакциями);

Серверы БД обычно позволяют отключать режим автоматической фиксации. При отключении автоматической фиксации управление транзакциями по прежнему доступно в ручном режиме (но, если при этом управления в ручном режиме не происходит, то сеанс работы с сервером БД считается одной транзакцией).
Если сервер позволяет управлять режимом автоматической фиксации, то это задается специфичными серверу способами (действительно, управление такой функциональностью сервера это то, что имеет значение для сервера БД, а не языка SQL вообще).
На сервере MySQL управление режимом автоматической фиксации задается записью:

```
SET AUTOCOMMIT= …
```

Значение 0 выключает этот режим, значение 1 — включает.
Если же сервер позволяет управлять транзакциями явно, то это делается при помощи стандартных лексемм языка SQL (хотя, серверы БД могут отклоняться от стандарта).
На сервере MySQL явное управление транзакциями осуществляется стандартными лексеммами языка SQL.
При ручном упрвлении транзакциями начало транзакции задается записью:

```
START TRANSACTION
```

Завершение же транзакции задается ее фиксацией, или же откатом. Фиксация задается командой COMMIT, откат — командой ROLLBACK. Необходимо помнить, что даже при ручном управлении транзакциями транзакция может быть завершена и до того, как ее завершение задано явно. Это может произойти всвязи с соответсвующим событием (либо пользователем явное задано начало новой транзакции, либо исполнение оператора управления схемой, либо завершение транзакции со стороны сервера по той или иной причине).
Запись транзакции имеет вид:

```
START TRANSACTION;
 …
COMMIT;
```

Точки сохранения транзакций

При ручном управлении транзакциями можно задать точки сохранения. Точки сохранения позволяют производить откат транзакции до целевой точки (но, само по себе задание точки сохранения к сохранению не приводит). Точки сохранения задаются на целевом участке кода транзакции, это делается при помощи записи:

```
SAVEPOINT SavepointName;
```

Откат к целевой точке сохранения задается следующей записью в коде транзакции:

```
ROLLBACK TO SAVEPOINT SavepointName;
```

Запись использования точек сохранения в коде транзакции имеет вид:

```
START TRANSACTION;
 …
SAVEPOINT SavepointName;
 …
ROLLBACK TO SAVEPOINT SavepointName;
 …
COMMIT;
```

При задании точек сохранения откат возможен как к целевой точке сохранения, так и всей транзакции вообще — это делается, как обычно, следующей записью в коде транзакции:

```
ROLLBACK;
```

Доступные пользователю возможности управления транзакциями зависят от используемого механизма хранения.

О механизмах синхронизаци

Серверы БД предназначены для одновременной работы с большим количеством пользователей. Когда пользователи разделяют использование одного ресурса, то требуется синхронизация доступа к нему.
Поддерживаются следующие механизмы синхронизации.

1

при обращении к данным по записи требуется блокировка по записи; при обращении к данным по чтению требуется блокировка по чтению; обращение по записи возможно только от одного пользователя в один момент времени; обращение по чтению, как обычно, допустимо от нескольких пользователей в один момент времени;

2

при обращении к данным по записи, как обычно, требуется блокировка по записи; при обращении к данным по чтению блокировка не требуется;

Блокировка может применяться со следующей гранулярностью: блокировка таблицы, и блокировка строки таблицы.
Доступность тех или иных механизмов синхронизации и гранулярностей блокировки зависит от используемого сервера БД и механизма хранения.

Вообще, от программиста не требуется явно запрашивать блокировку разделяемых ресурсов — это и так выполняется не явно, самим сервером БД. Что касается явных запросов блокировки — то они производятся при помощи соответствующих средств (операторов или же ввстроенных функций). Все эти средства — специфика конкретного сервера БД. Обычно, эти средства проблематичны в их реализации (эти средства, прежде всего, источник багов). Даже в случае корректной реализации на сервере эти средства являются источниками неисправности — практика показывает, что блокировки должны запрашиваться самим сервером (если же блокировки запрашиваются пользователем, то это приводит к неоптимальной работе — настолько неоптимальной, что зачастую приводит к краху сервера БД).

О механизмах хранения

Механизмы хранения — это собственно механизмы хранения данных в БД.
Механизмы хранения определяют, какие возможности работы с данными в БД доступны пользователю. Эти механизмы можно определять при использовании соответствующих операторов.
Разные серверы БД поддерживают разные наборы механизмов хранения.
На сервере `MySQL` доступны следующие механизмы хранения.

`ISAM`

транзакции не поддерживаются; гранулярность блокировки — таблица; этот механизм хранения устарел, и может не поддерживаться на сервере `MySQL`;

`MyISAM`

транзакции не поддерживаются; гранулярность блокировки — таблица;

`InnoDB`

транзакции поддерживаются; гранулярность блокировки — строка таблицы;

`Merge`

транзакции не поддерживаются; гранулярность блокировки — таблица;

Серверы БД обычно поддерживают и прочие механизмы хранения.

Часть 5 (Функции)

Встроенные функции

Речь идет о встроенных функциях, так как пользовательские функции не поддерживаются. Различные встроенные функции предназначены для решения различных видов задач; встроенные функции целесообразно подразделять на следующие: функции для работы со строками, для работы с числами, для работы с датами, и прочие функции.
Встроенные функции реализованы весьма специфично на различных серверах БД (на сервере MySQL они реализованы нижеописанным образом).

Функции для работы со строками

Это следующие функции.

`CONCAT()`

используется для конкатенации строк; принимает параметры — ряд строк (так же могут быть приняты и параметры временных типов — они будут автоматически преобразованы в строки); осуществляет конкатенацию принятых в параметрах строк; возвращает строку;

`CONCAT_WS()`

используется для конкатенации строк; принимает параметры — ряд строк (так же могут быть приняты и параметры временных типов — они будут автоматически преобразованы в строки), причем первая из них трактуется как разделитель (который будет употребляться при конкатенации); осуществляет конкатенацию принятых в параметрах строк; возвращает строку;

`LENGHT()`

используется для возврата длины строки; принимает параметр — строку; возвращает длину указанной строки; возвращает число;

`TRIM()`

используется для устрижения строки; принимает параметр — строку; устригает указанную строку (удаляет пробелы по ее концам); возвращает строку;

имеет дополнение к параметру — оно опционально, и прописывается перед принимаемым параметром (запись `LEADING FROM ' … '` в области параметров укажет, что пробелы должны удаляться в начале указанной строки, запись `TRAILING FROM ' … '` в области параметров укажет, что пробелы должны удаляться в конце указанной строки, запись `BOTH FROM ' … '` в области параметров укажет, что пробелы должны удаляться в начале и конце указанной строки (и поэтому избыточна));
строка устригается в специфическом смысле — в действительности просто возвращаются символы, которые остались в результате устрижения (действительно, устригать непосредственное значение не имеет смысла, даже в языке `SQL`);

`LTRIM()`

используется для устрижения строки; принимает параметр — строку; устригает принятую строку (удаляет пробелы слева (с начала)); возвращает строку;
строка устригается в специфическом смысле — в действительности просто возвращаются символы, которые остались в результате устрижения (действительно, устригать непосредственное значение не имеет смысла, даже в языке `SQL`);

`RTRIM()`

используется для устрижения строки; принимает параметр — строку; устригает принятую строку (удаляет пробелы справа (с конца)); возвращает строку;
строка устригается в специфическом смысле — в действительности просто возвращаются символы, которые остались в результате устрижения (действительно, устригать непосредственное значение не имеет смысла, даже в языке `SQL`);

`LOCATE()`

используется для поиска подстроки в строке; принимает параметры — строку (это искомая подстрока), строку (это строка, в которой происходит поиск), и число (это позиция, с которой стартует поиск, этот параметр опционален); возвращает позицию вхождения подстроки в строку; возвращает число (0 если вхождений нет, и позицию вхождения — если есть вхождение (позиции в строке нумеруются в натуральных числах));

`LEFT()`

используется для устрижения строки; принимает параметры — строку (это устригаемая строка), и число (число символов, на которые будет устрижение);

устригает указанную строку слева (с начала) на указанное количество символов; возвращает строку;
строка устригается в специфическом смысле — в действительности просто возвращаются символы, на которые произошло устрижение (действительно, устригать непосредственное значение не имеет смысла, даже в языке SQL);

RIGHT()

используется для устрижения строки; принимает параметры — строку (это устригаемая строка), и число (число символов, на которые будет устрижение); устригает указанную строку справа (с конца) на указанное количество символов; возвращает строку;
строка устригается в специфическом смысле — в действительности просто возвращаются символы, на которые произошло устрижение (действительно, устригать непосредственное значение не имеет смысла, даже в SQL);

SUBSTRING()

используется для возврата подстроки из строки; принимает параметры — строку (из которой возврщается подстрока), число (позиция, с которой возвращается подстрока), и число (число возвращаемых символов, этот параметр опционален); возвращает подстроку из указанной строки; возвращает строку;

SUBSTRING_INDEX()

используется для возврата подстроки из строки; принимает параметры — строку (из которой возвращается подстрока), строку (трактуется как разделитель), и число (трактуется как позиция разделителя); возвращает подстроку из указанной строки, причем подстрока возвращается до указанной позиции указанного разделителя (позиция разделителя может быть указана натуральным числом — тогда она отсчитывается слева, или же отрицательным числом — тогда она отсчитывается справа);

REPLACE()

используется для замены части строки; принимает параметры — строку (это заменяемая подстрока), строку (это указанная строка, в ней и будет заменяться часть), и строку (это заменяющая подстрока); заменяет часть указанной строки описанным образом; возвращает строку (это указанная строка, измененная);

INSERT()

используется для замены части строки; принимает параметры — строку (это указанная строка, в ней и будет заменяться часть), число (это стартовая позиция

замены в указанной строке), число (это количество заменяемых символов), и строка (из нее и будут браться символы для замены); заменяет часть указанной строки описанным образом; возвращает строку (это указанная строка, измененная);

`REVERSE()`

используется для реверса строки; принимает параметр — строку (она и реверсируется); реверсирует порядок символов в указанной строке; возвращает строку (это указанная строка, измененная);

`LOWER()`

используется для смены регистра строки; принимает параметр — строку (ее регистр и будет сменяться); сменяет регнистр указанной строки на нижний; возвращает строку (это указанная строка, измененная);

`UPPER()`

используется для смены регистра строки; принимает параметр — строку (ее регистр и будет сменяться); сменяет регнистр указанной строки на верхний; возвращает строку (это указанная строка, измененная);

`SPACE()`

используется для возврата строки из пробелов; принимает параметр — число (это число пробелов, из которых и будет создана строка); создает строку из указанного числа пробелов; возвращает строку (это созданная строка);

`REPEAT()`

используется для возврата строки, полученной путем повторений; принимает параметры — строку (это повторяемая строка), и число (это число ее повторов); возвращает строку, полученную описанным образом; возвращает строку;

`LPAD()`

используется для возврата строки, полученной путем дополнений; принимает параметры — строку (это дополняемая строка), число (это число символов, на которое будет дополняться дополняемая строка), и строка (ее символами и будет выполняться дополнение); возвращает строку, полученную описанным образом; возвращает строку;
дополнение происходит слева дополняемой строки;

`RPAD()`

используется для возврата строки, полученной путем дополнений; принимает параметры — строку (это дополняемая строка), число (это число символов, на которое будет дополняться дополняемая строка), и строка (ее символами и будет выполняться дополнение); возвращает строку, полученную описанным образом; возвращает строку;
дополнение происходит справа дополняемой строки;

Функции для работы с числами

Это следующие функции.

`SIGN()`

используется для определения знака числа; принимает параметр — число; определяет знак указанного числа, и возвращает соответствующее значение; возвращает число (`-1` для отрицательных, `0` для нуля, `1` для положительных);

`ABS()`

используется для возврата значения числа; принимает параметр — число; возвращает значение указанного числа; возвращает число;

`LEAST()`

используется для определения наименьшего числа; принимает параметры — ряд чисел; определяет наименьшее число из указанных; возвращает число;

`GREATEST()`

используется для определения наибольшего числа; принимает параметры — ряд чисел; определяет наибольшее число из указанных;
возвращает число;

`MOD()`

используется для возврата остатка от деления; принимает параметры — число (делимое), и число (делитель); возвращает остаток от деления;
возвращает число;

`ROUND()`

используется для округления; принимает параметры — число (округляемое), и число (количество знаков дробной части, этот параметр может быть опущен); округляет указанное число до ближайшего целого, если не передан второй параметр (логика определения ближайшего целого зависит от сервера БД (и даже его версии)); возвращает число;

`CEILING()`

используется для округления; принимает параметр — число; округляет указанное число до целого в меньшую сторону (до меньшего по модулю числа); возвращает число (результат — безусловно в типе `BIGINT`);

`FLOOR()`

используется для округления числа; принимает параметр — число; округляет указанное число до целого в большую сторону (до большего по модулю числа); возвращает число (результат — безусловно в типе `BIGINT`);

`TRUNCATE()`

используется для округления числа; принимает параметры — число (округляемое), и число (количество знаков дробной части); округляет указанное число до указанной точности путем отбрасывания; возвращает число;

`SQRT()`

используется для возврата квадратного корня; принимает параметр — число (из него и извлекается корень); вычисляет корень квадратный (положительный корень квадратный); возвращает число;

`POW()`

используется для вычисления икс в степени игрек; принимает параметры — число (икс), и число (игрек); вычисляет икс в степени игрек; возвращает число; синоним — функция `POWER()`;

`EXP()`

используется для вычисления е в степени икс; принимает число (икс); вычисляет е в степени икс; возвращает число;

`LOG()`

используется для вычисления логарифма натурального икс; принимает параметр (икс); вычисляет логарифм натуральный икс; возвращает число;

`LOG10()`

используется для вычисления логарифма десятичного икс; принимает число (икс); вычисляет логарифм десятичный икс; возвращает число;

`RAND()`

используется для возврата случайного числа; принимает параметр — число (целое число, оно ограничивает результат снизу, этот параметр может быть опущен); возвращает случайное число; возвращает число (fp- число);

`PI()`

используется для возврата числа пи; параметров не принимает; возвращает число пи; возвращает число;

`DEGREES()`

используется для преобразования радианов к градусам; принимает аргумент — число (трактуется как радианы); преобразует полученное значение к градусам; возвращает число;

`RADIANS()`

используется для преобразования градусов к радианам; принимает аргумент — число (трактуется как градусы); преобразует полученное значение к радианам; возвращает число;

`SIN()`

используется для возврата синуса; принимает параметр — число (трактуется как радианы); возвращает синус от аргумента; возвращает число;

`COS()`

используется для возврата косинуса; принимает параметр — число (трактуется как радианы); возвращает косинус от аргумента; возвращает число;

`TAN()`

используется для возврата тангенса; принимает параметр — число (трактуется как радианы); возвращает тангенс от аргумента; возвращает число;

`COT()`

используется для возврата котангенса; принимает параметр — число (трактуется как радианы); возвращает котангенс от аргумента; возвращает число;

`ASIN()`

используется для возврата арксинуса; принимает параметр — число (трактуется как значение синуса); возвращает арксинус от аргумента; возвращает число;

`ACOS()`

используется для воврата арккосинуса; принимает параметр — число (трактуется как значение косинуса); возвращает арккосинус от аргумента; возвращает число;

`ATAN()`

используется для возврата арктангенса; принимает параметр — число (трактуется как значение тангенса); возвращает арктангенс от аргумента; возвращает число;
может принимать параметры и иначе — число (икс), и число (игрек);

`ATAN2()`

используется для возврата арктангенса; принимает параметры — число (икс), и число (игрек); возвращает арктангенс от аргумента; возвращает число;
использует знаки аргументов для определения квадранта результата;

Функции для работы с датами

Это следующие функции.

`NOW()`

используется для возврата значения типа `DATETIME`, текущего; параметров не принимает; возвращает значение типа `DATETIME`, текущее; возвращает значение типа `DATETIME`;

значение возвращается, строго говоря, в виде строки;

SYSDATE()

используется для возврата значения типа DATETIME, текущего; параметров не принимает; возвращает значение типа DATETIME, текущее; возвращает значение типа DATETIME;
значение возвращается, строго говоря, в виде строки;

CURRENT_TIMESTAMP()

используется для возврата значения типа TIMESTAMP, текущего; параметров не принимает; возвращает значение типа TIMESTAMP, текущее; возвращает значение типа TIMESTAMP;
значение возвращается, строго говоря, в виде строки;

CURDATE()

используется для возврата значения типа DATE, текущего; параметров не принимает; возвращает значение типа DATE, текущее; синоним CURRENT_DATE();
значение возвращается, строго говоря, в виде строки;

CURTIME()

используется для возврата значения типа TIME, текущего; параметров не принимает; возвращает значение типа TIME, текущее; синоним CURRENT_TIME();
значение возвращается, строго говоря, в виде строки;

UTC_DATE

используется для возврата значения типа DATE (по GMT), текущего; параметров не принимает; возвращает значение типа DATE (по GMT), текущее; возвращает значение типа DATE;
значение возвращается, строго говоря, в виде строки;

UTC_TIME

используется для возврата значения типа TIME (по GMT), текущего; параметров не принимает; возвращает значение типа TIME (по GMT), текущее; возвращает значение типа TIME;

значение возвращается, строго говоря, в виде строки;

`DAYOFMONTH()`

используется для возврата значения месяца; принимает параметр — значение типа `DATE`; возвращает значение месяца указанной даты; возвращает число (в натуральных числах); синоним `MONTH()`;

`DAYOFWEEK()`

используется для возврата значения дня недели; принимает параметр — значение типа `DATE`; возвращает значение дня недели указанной даты; возвращает число (в натуральных числах); синоним `WEEK()`;

`DAYOFYEAR()`

используется для возврата значения года; принимает параметр — значение типа `DATE`; возвращает значение года указанной даты; возвращает число; синоним `YEAR()`;

`QURTER()`

используется для возврата квартала года; принимает параметр — значение типа `DATE`; возвращает квартал года (в натуральных числах) указанной даты; возвращает число;

`LAST_DAY()`

используется для возврата последнего дня; принимает параметр — значение типа DATE; возвращает значение последнего дня (последнего дня месяца указанного аргумента) в составе возвращаемого значения; возвращает строку;

`DAYNAME()`

используется для возврата имени дня; принимает параметр — значение типа `DATE`; возвращает значение имени дня; возвращает строку;

`MONTHNAME()`

используется для возврата имени месяца; принимает параметр — значение типа `DATE`; возвращает значение имени месяца; возвращает строку;

HOUR()

используется для возврата значения часа; принимает параметр — значение типа TIME; возвращает значение часа указанного аргумента; возвращает строку;

MINUTE()

используется для возврата значения минуты; принимает параметр — значение типа TIME; возвращает значение мин. указанного аргумента; возвращает строку;

SECUNDE()

используется для возврата значения секунды; принимает параметр — значение типа TIME; возвращает значение сек. указанного аргумента; возвращает строку;

EXTRACT()

используется для извлечения составляющей из значения DATETIME; принимает параметры — область параметров имеет вид (ColType FROM " … "), где ColType это тип составляещей значения DATETIME; возвращает указанную составляющую из указанного значения DATETIME; возвращает строку;

DATE_ADD()

используется для сложения дат; принимает параметры — область параметров имеет вид (« … », INTERVAL Value ColType), где « … » – указанная дата, INTERVAL – собственно ключевое слово, Value – количество значений указанного типа (которые будут складываться с указанной датой), ColType – тип складываемых значений; складывает даты указанным способом; возвращает значение DATE или DATETIME;
строго говоря, возвращает строку;

DATE_SUB()

идентичен методу DATE_ADD(), за исключением того,что выполняет вычитание;

DATE_DIFF()

идентичен методу DATE_SUB(), за исключением того, что область параметров имеет вид (« … », « … »), и вычитание выполняется между указанными значениями даты;

TO_DAYS()

используется для преобразования даты ко дням; принимает значение DATE; преобразует принятый параметр ко дням; возвращает число (это число — значение указанной даты в количестве дней от начала счетчика (от 0));

TIME_TO_SEC()

используется для преобразования времени к секундам; принимает значение TIME; преобразует принятый параметр к секундам; возвращает число (это число — указанное значение TIME в количестве секунд от начала счетчика (от 0));

DATE_FORMAT()

используется для форматриования значения типа DATE;
принимает параметры — строку (трактуется как значение типа DATE), и строку (тракуется как строка, содержащая целевой формат);
возвращает указанное значение, форматированное указанным образом; возвращает строку;
строка с форматом задается на усмотрение программиста, для ее составления используются специальные обозначения (%Y – обозначает год YYYY, %y – год YY, %M – месяц January … December, %m – месяц 01 … 12, %b – месяц Jan … Dec, %c – месяц 1 … 12, %D – день 1st … 31st, %d – день 00 … 31, %e – день 0 … 31, %W – день недели Sunday … Saturday, %a – день недели Sun … Sat, %H – часы 00 … 23, %h – часы 01 … 12, %k – часы 0 … 23, %l – часы 1 … 12, %i – минуты 00 … 59, %S – секунды 00 … 59, %p – PM/AM, %T – время в 24ч формате HH:MM:SS, %r –время в 12ч формате HH:MM:SS AM/AP), если ежду этими обозначениями употреблен какой- либо символ — он будет использоваться как разделитель;

TIME_FORMAT()

используется для форматриования значения типа TIME;
принимает параметры — строку (трактуется как значение типа TIME), и строку (тракуется как строка, содержащая целевой формат);
возвращает указанное значение, форматированное указанным образом;
возвращает строку;
строка с форматом задается аналогично тому, как в функции DATE_FORMAT();

Прочие функции

Это следующие функции.

Функции приведения типа

CAST()

используется для приведения типа к целевому; принимает параметры — область параметров имеет вид (`Value AS ColType`); выполняет приведение типа к целевому (целевым может быть тип: `DATE`, `DATETIME`, `TIME`, `INTEGER`); возвращает значение (целевого типа);

CONVERT()

используется для приведения типа к целевому; принимает параметры — область параметров имеет вид (`Value, ColType`); выполняет приведение типа к целевому (целевым может быть тип: `DATE`, `DATETIME`, `TIME`, `INTEGER`); возвращает значение (целевого типа);

Остальные прочие функции

VERSION()

используется для получения версии сервера БД; параметров не принимает; возвращает версию сервера БД; возвращает строку;

DATABASE()

используется для получения имени текущей БД; параметров не принимает; возвращает имя текущей БД; возвращает строку;

USER()

используется для получения username; параметров не принимает; возвращает username; возвращает строку;
синоним `SYSTEM_USER()`, `SESSION_USER()`;

CONNECTION _ID()

используется для получения id соединения; параметров не принимает; возвращает id текущего соединения; возвращает строку;

PASSWORD()

используется для шифрования строки; принимает параметр — строку; выполняет шифрование (по своей логике) принятой строки; возвращает строку (содержащую результат шифрования);
эта функция применяется сервером БД в собственных нуждах; ее значение для программиста оспоримо;

ENCODE()

используется для шифрования строки; принимает параметры — строку (шифруемая), и строку (пароль); выполняет шифрование (по своей логике) принятой строки; возвращает BLOB (результат);
длина аргумента — это длина символьных типов; длина результата — это длина шифруемой строки;

DECODE()

используется для расшифрования результата вызова ENCODE(); принимает параметры — результат вызова ENCODE(), и строку (пароль); выполняет расшифрование первого аргумента; возвращает строку (результат);
длина аргумента — это длина символьных типов; длина результата — это длина шифруемой строки;

MD5()

используется для получения хэша; принимает строку; выполняет хэширование принятой строки по алгоритму MD5; возвращает строку (результат);
длина аргумента — это длина символьных типов; длина результата — 16 byte (это 128 bit);

SHA()
используется для получения хэша; принимает параметр — строку; выполняет хэширование принятой строки по алгоритму SHA; возвращает строку (результат);
длина аргумента — это длина символьных типов; длина результата — 20 byte (это 160 bit); синоним SHA1();

AES_ENCRYPT()

используется для шифрования строки; принимает параметры — строку (шифруемая), и строку (пароль); выполняет шифрование первого аргумента по алгоритму AES; возвращает строку (результат);

если функция принимает в каком- либо аргументе значение NULL, то она возвращает NULL; если функция в процессе своей работы обнаруживает проблемы в вычислениях (не может дополнить аргументы до нужной длины, etc), то она возвращает NULL;
этот алгоритм использует шифрование по 128- битному ключу;

AES_DECRYPT()

используется для шифрования строки; принимает параметры — строку (шифруемая), и строку (пароль); выполняет расшифрование первого аргумента, зашифрованного по алгоритму AES; возвращает строку (результат);

DES_ENCRYPT()

используется для шифрования строки; принимает параметры — строку (шифруемая), и строку (пароль); выполняет шифрование первого аргумента по алгоритму DES; возвращает строку (результат);

DES_DECRYPT()

используется для шифрования строки; принимает параметры — строку (шифруемая), и строку (пароль); выполняет расшифрование первого аргумента, зашифрованного по алгоритму DES; возвращает строку (результат);

Часть 6 (Условная логика)

Условная логика

Условная логика — это средства, предназначенные для условного исполнения тех или иных составляющих запроса.
Условная логика реализуется в следующей конструкции:

```
CASE
 WHEN WhenExpression0
 THAN ThanExpression0
  …
 WHEN WhenExpressionN
 THAN ThanExpressionN
  ELSE ElseExpression
END
```

Конструкция возвращает в код запроса то, что возвращает та ее ветвь, которая исполнилась. Логика работы этой конструкции сродни логике работы оператора `switch` в ЯПВУ `C` (и действие блока `ELSE` сродни действию метки `default`). Блок `ELSE` опционален; все ветви этой конструкции должны возвращать значения одного типа.
Прочие средства условной логики реализованы во встроенных функциях, и поэтому ограниченно интересны.

Часть 7 (Ограничения)

Ограничения

Ограничения — это собственно ограничения, накладываемые на данные, хранимые в таблице (в ее целевых столбцах). Ограничения служат для регулирования использования данных (в плане использования тем или иным способом, или хранения тех или иных значений).
Есть следующие ограничения.

Ограничения первичного ключа

Ограничения первичного ключа применяются для того, чтобы определить столбцы, образующие первичный ключ таблицы.
Запись этого ограничения (при создании таблицы):

```
CREATE TABLE TableName ( … ,
CONSTRAINT  PkConstraintName PRIMARY KEY (ColName) );
```

в этой записи `PkConstraintName` это имя ограничения первичного ключа (оно опционально; со словом `CONSTRAINT` всегда может использоваться опциональное имя ограничения; если имя ограничения используется, то оно следует непосредственно после слова `CONSTRAINT`); `ColName` – это столбец, его образующий. Первичный ключ не может быть `NULL`. Первичныйключ должен быть уникален. Для того, чтобы значения первичного ключа были безусловно уникальными, эту задачу следует возложить на сервер БД. Это делается соответствующей записью при объявлении соответствующего столбца (записью ограничения уникальности).
Запись ограничения первичного ключа после создания таблицы:

```
ALTER TABLE TableName
ADD CONSTRAINT  PkConstraintName PRIMARY KEY (ColName);
```

Чтобы затем убрать это ограничение, используется этот же запрос, только вместо `ADD` используется `DROP`.

Ограничения внешнего ключа

Ограничения внешнего ключа применяются для того, чтобы определить столбцы, образующие внешний ключ таблицы.
Запись этого ограничения (при создании таблицы):

```
CREATE TABLE TableName ( … ,
CONSTRAINT  FkConstraintName FOREIGN KEY (ColName)
REFERENCES ForeignTableName (ForeignColName) );
```

в этой записи FkConstraintName это имя ограничения внешнего ключа (оно опционально); ColName – это столбец, его образующий; ForeignTableName (ForeignColName) — это сведения об указываемой таблиц и ее первичном ключе. По факту использования этого ограничения столбец (используемый как внешний ключ) сможет принимать только значения первичного ключа (на который он указывает).
Запись этого ограничения после создания таблицы:

```
ALTER TABLE TableName
ADD CONSTRAINT FkConstraintName FOREIGN KEY (ColName)
REFERENCES ForeignTableName (ForeignColName);
```

Чтобы затем убрать это ограничение, используется этот же запрос, только вместо ADD используется DROP.
При определении внешнего ключа целесообразно использовать блоки ON DELETE, и ON UPDATE.
Эти блоки задаются для исполнения инкапсуллируемого ими выражения при наступлении соответствующих событий в использовании БД — при изменении или удалении указываемых строк в указанной таблице.
Каждый из этих блоков опционален, но при использовании они могут сочетаться, и тогда прописываются так:

```
REFERENCES ForeignTableName (ForeignColName)
ON DELETE ExpressionOnDelete
ON UPDATE ExpressionOnUpdate
```

причем, выражением (инкапсуллируемым каким- либо из этих блоков) должно быть одно из следующих предопределенных выражений.

RESTRICT

это выражение, строго говоря, не исполняется (в привычном смысле), а просто пресекает изменение\ удаление строк в указываемой таблице (при наличии указывающих строк в своей таблице);
это выражение работает именно так, как описывается;

NO ACTION

синоним RESTRICT;

SET DEFAULT

это выражение, строго говоря, не исполняется (в привычном смысле), а просто устанавливает для внешнего ключа значение по умолчанию (при удалении связанной строки из указываемой таблицы);

SET NULL

это выражение, строго говоря, не исполняется (в привычном смысле), а просто устанавливает для внешнего ключа значение NULL если оно допустимо (при изменении\ удалении связанной строки из главной таблицы);

CASCADE

это выражение, строго говоря, не исполняется (в привычном смысле), а просто приводит к удалению\ изменению связанных строк в своей таблице (при изменении\ удалении в указываемой);

Ограничения индекса

Ограничения индекса применяются для того, чтобы определить столбцы, образующие индекс.
Таблицы пополняются новыми данными в последовательном порядке. Сортировки при этом не происходит. Поэтому поиск данных может занимать существенное время. В целях сокращения этого времени и применяются индексы. Индексы, как и прочие данные, хранятся в табличном виде (организация их хранения остается на откуп сервера БД). Они хранятся в отсортированном порядке и ускоряют поиск данных. Индексы в реляционных БД организуются (в общих чертах) так же, как и индексы в книгах. Индексы представляют собой таблицы, записи (сущности) которых содержат индексируемые данные (данные столбца, по которому создается индекс), и данные, описывающие размещение сущностей (номера строк).
Ограничения индекса задаются записью:

```
ALTER TABLE TableName
ADD INDEX IndexName (ColName);
```

Удаление индекса производится записью:

```
ALTER TABLE TableName
DROP INDEX IndexName;
```

Создание (как и удаление) индекса (на сервере MySQL) производится именно такой записью (записью ALTER TABLE …). Хотя прочие серверы БД предусматривают и прочие записи.
Вышеприведенная запись создаст индекс для указанной таблицы по указанному столбцу (ColName). У таблицы может быть и несколько индексов. Оптимизатор запросов пользуется индексами на свое усмотрение.
Кроме индексов, созданных программистом, сервер БД автоматически создает индекс по первичному ключу таблицы.
Так называемый уникальный индекс позволяет не только индексировать таблицу по целевому столбцу, но и наложить на целевой столбец ограничение — он

сможет хранить только уникальные не дублирующиеся значения. Уникальный индекс задается несколько специфичной записью:

```
ALTER TABLE TableName
ADD UNIQUE IndexName (ColName);
```

И удаляется записью:

```
ALTER TABLE TableName
DROP UNIQUE IndexName;
```

Индексы могут быть и составными — и допускать индексирование по ряду столбцов. Составные индексы допускают индексирование как по всем целевым столбцам, так и по старшей составляющей списка столбцов — но не по младшей (под старшей составляющей списка столбцов имеются ввиду столбцы, прописанные первее). Составные индексы демонстрируют такое поведение потому, что индексы и используются как индексы (по аналогии с индексами в книгах: в телефонном справочнике можно искать по всему индексу (фамилия и имя), можно и только по старшей составляющей индекса (фамилии), но нельзя искать только по младшей составляющей индекса (имя)). По этой причине разрешается создавать и несколько составных индексов (по одному и тому же набору столбцов, прописанных в разном порядке) — это позволит пользоваться преимуществами индексации при поиске по этим столбцам (в разном порядке использования этих столбцов в поиске).

О реализации индексов

индексы по умолчанию реализуются на основе сбалансированного дерева; они реализованы следующим образом — в качестве узлов (того или иного уровня иерархии) выступают целевые столбцы (того или иного уровня старшинства), а в качестве листьев — сведения о размещении сущностей. Разумеется, это аппроксимация (реализация же усложняется тем, что формирование узлов того или иного уровня иерархии остается на откуп серверу — он создает узлы того или иного уровня иерархии исходя из своей логики (исходя из того, насколько просто будет производить поиск) — это и называется балансированием (если быть точнее, сервер создает большее количество уровней иерархии, и это ускоряет поиск));

индексы могут быть реализованы и на основе битовых карт; такие индексы реализуются следующим образом: для индексируемого столбца составляется список значений (которые он может принимать) и для каждого из этих значений выставляется флаг (обозначающий наличие или отсутствие этого значения в той или иной строке таблицы); таким образом и формируется битовая карта, в результате она имеет вид такой таблицы:

```
                Строка0 …       СтрокаN
Значение0
 …
ЗначениеN
```

контентом этой таблицы являются флаги; как и было оговорено выше, для составления битовой карты составляется список значений (которые

предполагается хранить в индексируемом столбце) — при большой длине списка значений использование этого типа индексов не даст большого прироста производительности и не целессообразно;
возможность применения индекса того или иного типа зависит от механизма хранения и сервера БД; сервер MySQL поддерживает индексы на основе сбалансированного дерева (они доступны при использовании механизма хранения InnoDB);

Ограничения значения

Это следующие ограничения.

Ограничения значения по умолчанию

Ограничения значения по умолчанию позволяют определить значение по умолчанию для целевого столбца. Ограничения значения по умолчанию задаются записью:

```
ColNameColType DEFAULT Value
```

указанное значение будет использоваться как значение по умолчанию для данного столбца. Значение по умолчанию используется для инициализации данных столбца, если при вставке данных ему не прописано никакого значения.

Ограничения значения NULL

Ограничения значения `NULL` позволяют запретить значение `NULL` для целевого столбца. Ограничения значения `NULL` задаются записью:

```
ColName ColType NOT NULL
```

если же необходимо указать, что значение `NULL` допустимо для целевого столбца, то это можно сделать записью:

```
ColName ColType NULL
```

эта запись избыточна, так как по умолчанию значения `NULL` и так допустимы.

Ограничения проверочные

Ограничения проверочные позволяют инициализировать данные целевого столбца по факту проверки на допустимость значения инициализации. Если значение инициализации будет не допустимым, то инициализации не будет. Ограничения проверочные задаются записью:

```
ColName Coltype CHECK (ColName IN (Value0, … ))
```

эта запись приведет к проверке при инициализации целевого столбца на допустимость значения инициализации (на соответствие одному из допустимых

значений инициализации). Условное выражение, указанное в этой записи (`CHECK ( … )`), так или иначе должно использовать операторы сравнения, чтобы быть синтаксически корректным.

Ограничения уникальности (первичного ключа)

Ограничения уникальности значения позволяют делегировать серверу БД обеспечение уникальности значений в целевом столбце. Это делается соответствующей записью при объявлении соответствующего столбца (записью `ColName ColType AUTO_INCREMENT`).

По факту использования этой записи значения в целевом столбце будут безусловно уникальными. Сервер БД будет формировать уникальные значения автоматически, инкрементом предыдущего значения. И при добавлении новых строк в эту таблицу значение целевого столбца может быть задано `NULL` – сервер БД сам инициализирует его уникальным значением. По умолчанию эти значения стартуют с единицы.

Ограничения уникальности широко применяются для первичных ключей. При применении этого ограничения становится допустимым инициализировать первичный ключ значением `NULL`, и это невозбраняемо практикуется.

Часть 8 (Конвенция)

Об оформлении кода

Соглашения об оформлении кода на SQL подразумевают характерное.

О составлении имен

Здесь все как обычно. Разве что, имеет место некоторая специфика. Пользовательские имена должны прописываться в нижнем регистре. Если пользовательское имя — это сложная лексемма (с точки зрения естественного языка), то в его записи составляющие его слова следует прописывать через символ прочерка.
Прочие аспекты именования должны уточняться для конкретных случаев применения БД.

О составлении запросов

В разных случаях оказываются уместны более- менее длинные, или же более-менее короткие запросы. Длина запроса — это то, что зависит от конкретного случая применения БД.
Запросы к БД редко когда бывают совсем тривиальными.
Более- менее длинные запросы следует разбивать по строкам. Разбиение по строкам следует проводить по блокам запроса.
Блоки запроса, являющиеся тривиальными (не включающими в себя другие блоки), можно прописывать подряд в одной строке.
Блоки запроса, не являющиеся тривиальными, следует прописывать в отдельных строках. Исключение из этого случая — когда блок запроса включает в себя подзапрос или соединение — тогда блок может быть прописан в одной строке (если ее длина не будет слишком большой).
Блоки запроса прописываются так: прописывается ключевое слово, обозначающее блок, и через пробел прописываются прочие составляющие блока. Блоки, идущие в разных строках, прописываются без отступа — вместо этого они выравниваются друг по другу, причем справа. И не важно, какого уровня вложенности эти блоки — это правило действует для блоков любого уровня вложенности. Исключение из этого правила — когда в запросе используется подзапрос или соединение (притом, прописываясь с новой строки). В таком случае подзапрос или соединение прописывается с отступом от объемлющего его блока (отступ выполняется слева; величина отступа должна быть достаточной, чтобы обеспечить выход из «коридора», образованного выравниванием объемлющего блока справа).

Об употреблении пробельных символов

Употребление пробельных символов было рассмотрено выше. Кроме того, пробелы употребляются: см. JavaIII.

Прочее об употреблении пробельных символов. Для разбиения запроса на блоки используются (вышеописанным образом) переносы строк. Но пустые строки для этого не используются.

Часть 9 (Специфика MySQL)

Специфика MySQL

Выражение SELECT

Блок LIMIT

Используется для ограничения количества возвращаемых строк; этот блок всегда последний в записи запроса; прописывается так:

```
SELECT …
 …
LIMIT Value;
```

в результате в результирующий набор будет возвращено `Value` первых строк, предварительно отобранных для результирультирующего набора.
Альтернативная запись:

```
SELECT …
 …
LIMIT Value0, Value1;
```

приводит к следующему — будет возвращено `Value1` строк, начиная со строки `Value0` (нумерация с нуля).

Блок INTO OUTFILE

Используеся для вывода результирующего набора в файл; форма записи:

```
SELECT …
INTO OUTFILE « … »
FROM …
 … ;
```

в этой записи выходной файл (« … ») задается записью вида «`C:\\DirName\\ … \\FileName.ExtName`»;
при выводе в файл используется следующее форматирование — строки разделяются друг с другом переводом строк, а в пределах строк разделителем является табуляция; разделители строк и в пределах строк можно задать явно:

```
SELECT …
INTO OUTFILE « … »
FIELDS TERMINATED BY « … »
LINES TERMINATED BY « … »
FROM …
```

```
… ;
```

команда записи в файл не перезаписывает существующий файл, поэтому каждый раз требуется новый файл.

Блоки INSERT, UPDATE, DELETE

При вставке строк в таблицу возможна следующая ситуация — ключи вставляемой строки совпадают с и так присутствующими значениями ключей; эта ситуация может быть урегулирована соответствующими средствами:

```
INSERT …
ON DUPLICATE KEY UPDATE ColName … ;
```

при использовании этой записи происходит не вставка строки, а операция `UPDATE` в отношении указанных столбцов (при условии совпадения ключей).

Блок `UPDATE` допускает использование блоков `LIMIT` (этот блок, как обычно, ограничивает количество сущностей, с которыми оперирует запрос; в данном случае допустима только форма записи с одним операндом) и `ORDER BY` (этот блок, как обычно, производит упорядочивание сущностей в таблице, с которой оперирует запрос).

Блок `DELETE` тоже допускает использование блоков `LIMIT` (этот блок, как обычно, ограничивает количество сущностей, с которыми оперирует запрос; в данном случае допустима только форма записи с одним операндом) и `ORDER BY` (этот блок, как обычно, производит упорядочивание сущностей в таблице, с которой оперирует запрос).

Блоки `UPDATE` и `DELETE` может быть целесообразно применить сразу по ряду таблиц — и для этого предусмотрены соответствующие средства. Эти средства — многоблочные блоки `UPDATE` и `DELETE`.

Форма записи многоблочного `DELETE`:

```
DELETE TableName0, …
FROM …
WHERE … ;
```

в этой записи список таблиц указывает, откуда производить удаление; блок `FROM`, как обычно, содержит выражение, зависимое от таблиц; блок `WHERE`, как обычно, содержит условное выражение; эти блоки по содержанию сходны с одноименными блоками в составе `SELECT`, но трактуются по несколько иной логике — а конкретно, они определяют, какие строки следует удалить. Как и в составе `SELECT`, тут могут прописываться и соединения — но, необходимо помнить: назначение этих блоков в многоблочном `DELETE` состоит в том, чтобы определять, какие строки следует удалить.

Что касается многоблочного `UPDATE` – дело во многом аналогично. Его запись:

```
UPDATE TableName0 … JOIN TableName1 … ;
```

в этой записи для задействования ряда таблиц используется запись соединений.

Синтаксис многоблочного `UPDATE` похож на синтаксис одноблочного, за тем исключением, что задействуется ряд таблиц.