**Теоретические сведения**

Сущность описывается набором *полей*. Поле имеет *имя и тип данных*. Например, сведения о студенте могут быть представлены так:

last\_name: string;

first\_name: string;

birth\_day: date;

Экземпляр со значениями полей, описывающих конкретную сущностью, называется *записью базы данных*. Пример записи:

last\_name: "Иванов";

first\_name: "Иван";

birth\_day: '01.01.2001';

Диапазон возможных значений, которые может принимать поле, зависят от типа данных поля. Тип поля также называют *доменом*. Например, поле типа string может содержать строки любой длины (в том числе - пустую), поле типа date - корректную дату. От типа данных поля зависит также и набор возможных операций, участвующих в логических выражениях сравнения. Например, числа и даты можно сравнивать на больше/меньше/равно/не равно. Для строк существует такой же набор операций, но, например, сравнение на больше или меньше сравнивает порядок строк в алфавитном порядке.  
В зависимости от типа данных определяется способ записи значений поля. Например, строки всегда записываются в двойных кавычках, даты - в одинарных, а числа - без кавычек.  
Типы данных полей, которые используются в данной работе приводятся в таблице:

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Описание, формат ввода/вывода** | **Операции** |
| **int** | Позволяет хранить целые числа в диапазоне [−2147483648; 2147483647]. Считываются и выводятся как целые десятичные числа, в том числе отрицательные (со знаком минуса в начале). Другие знаки недопустимы, в том числе несколько минусов. Лидирующие нули (например 00056) при вводе - допускаются, при выводе - не выводятся. Если вводимое число выходит за диапазон или нарушен формат - то команда считается некорректной. | Сравнение на равенство (==), на неравенство (!=), больше/меньше (<, <=, >, >=) - как обычные арифметическое сравнение целых чисел. |
| **string** | Позволяет хранить строки любой длины, в том чиле пустые строки без символов. Считываются и выводятся как набор символов, заключенный в кавычки, например: "string1". Внутри кавычек **могут быть пробелы**. Если строка содержит символ кавычка (") или обратный слэш (\), то он предваряется обратным слэшем: как при вводе, так и при выводе, пример с кавычкой: "string with \" quot", пример со слэшем: "string with \\ slash" | Сравнение на равенство (==) - строки в точности идентичны (включая регистр). Сравенение на неравенство (!=) - строки не идентичны. Больше/меньше (<, >) - одна строка находится раньше/позже другой по алфавиту. Больше/меньше или равно (<=, >=) - одна строка находится раньше/позже другой по алфавиту или идентична ей. *Примечание:* сравнивается содержимое строк (то, что стоит внутри кавычек). Обратите внимание, что "" < " " и " " < " a". |
| **decimal** | Десятичное число заданной точности (с фиксированной, не с плавающей запятой). Указывается хранимое количество цифр перед запятой и хранимое после. Команды, пытающиеся ввести числа за этими пределами, не являются корректными. Например decimal(4,2) - это десятичное число с 4 цифрами до запятой и 2 после. Максимальное значение такого числа - 9999.99. Минимальное положительное - 0000.00. Числа могут быть и отрицательными. При вводе может указываться только целая часть, целая часть и точка, точка и дробная часть. Дробная часть может быть указана без всех цифр. Тогда подразумевается что остальные цифры - 0. При вводе целой части допускается указывать лидирующие нули, но количество цифр в целой части, включая их, не должно превышать заданное. Примеры корректного ввода для decimal(3,2): 0; 0.1; 999.; 999.59; 999; .59; .5; -.5; -99; 0001; примеры некорректного ввода: 9999; .599; 0.555; 00001; 00001.00 и т.д.  При выводе число всегда распечатываются все разряды числа после запятой: 999.99; 555.55; 99.00; 0.59; 10.10 | Сравнение на равенство (==), на неравенство (!=), больше/меньше (<, <=, >, >=) - как обычное арифметическое сравнение десятичных чисел. |
| **date** | Календарная дата. Представляет собой 3 числа, записываются через точку. При вводе и выводе записываются в одинарных кавычках. Первое число - день месяца, второе - номер месяца, третье - год. Некорректные даты, такие как '31.02.2017', сигнализируют о некорректной команде. При выводе у месяца и дня выводится лидирующий 0, если цифра меньше 10. При вводе лидирующие нули могут не указываться. Год указывается всегда 4-мя цифрами. Примеры корректного ввода: '1.1.2017', '01.01.2017'. Примеры некорректного ввода: '1.1.17', '35.1.2017', '1/1/17'. | Определены операции сравнения на равенство (==), неравенство (!=), больше/меньше (<, <=, >, >=). |
| **time** | Время суток. Представляет собой 3 числа, записываются через двоеточие. При вводе и выводе записываются в одинарных кавычках. Первое число - часы (от 0 до 23), второе - минуты (0-59), третье - секунды 0-59. Некорректное время, такое как '24:54:06', сигнализирует о некорректной команде. При выводе выводится лидирующий 0, если цифра меньше 10. При вводе лидирующие нули могут не указываться. Примеры корректного ввода: '0:0:0', '00:0:09'. Примеры некорректного ввода: '00:00', '0:0:', ':0:0'. | Определены операции сравнения на равенство (==), неравенство (!=), больше/меньше (<, <=, >, >=). |
| **phone** | Телефонный номер. Представляет собой строку, которая содержит цифры, пробелы скобки и символы +-. Номера вводятся и выводятся в одинарных кавычках. Длина номера - 10 цифр (без кода государства) - 3 из них занимает код оператора связи, остальные 7 - уникальны в пределах оператора. Поддерживаются только номера РФ, остальные являются некорректными. Ввод возможен в следующих форматах: '+7ZZZFFFFFFF', '8ZZZFFFFFFF'. Здесь ZZZ - код оператора, может быть взят в круглые скобки. Другие части в скобки брать - некорректно. Между группами цифр может стоять символ -, но только один. В любом месте номера может находиться неограниченное количество пробелов. Примеры корректных номеров: '+7 999 333-32-23', '8-(999)-333-32-23', '8-(9-9-9)-333-32-23', '8 999333 3223'. Примеры некорректных номеров: '+7+999 333-32-23', '999 333-32-23', '+8 999 333-32-23', '+43 99 333-32-23', '+7 999 333-32-2', '8999 (333-32-23)'. Телефонный номер **выводится всегда в том виде, в котором был введен**. | Опеределны операции сравнения на равенство (==), неравенство (!=), больше, меньше, больше или равно, меньше или равно. Операции проводятся как арифметические операции над числом, составленном из 10 цифр номера (без кода государства). Например, для номера '+7(444)565-45-54' таким числом является 4445654554. |
| **carnum** | Автомобильный номерной знак. Представлен в виде LDDDLLRR, либо LDDDLLRRR, где L - заглавная латинская буква из набора ("ABCEHKMOPTXY"), D - цифра, R - цифра, код региона. Такие поля вводятся и выводятся в кавычках. Если нарушен формат ввода, то команда - некорректна. Примеры корректного ввода: 'X543HB98', 'E666KX178'. Примеры некооректного ввода: 'X543HB8', 'e666KX178', 'E66KX178'. Выводится в том же виде, что и был введен | Определены операции сравнения на равенство (==), неравенство (!=), больше/меньше (<, <=, >, >=). При сравнении двух номеров, на больше-меньше, скажем A и B, следует использовать следующий алгоритм: сначала сравниваются цифры номера, как трехзначное число (DDD). Если числа совпадают, то из трех букв номера составляется строка "LLL" (буквы из номера берутся слева направо) и сравниваются такие строки, в алфавитном порядке. Если совпадают и буквы, то сравнивается код региона, как десятичное число. Примеры истинных сравнений: 'M555KB90' == 'M555KB90', 'M555KB90' > 'O554KB90', 'M555KB90' < 'O555KB90', 'M555KB90' > 'M555KB89'. |
| **enum (перечисление)** | Поле такого типа может принимать одно из значений из заранее указанного набора. Набор возможных значений указывается при описании типа. Например, если задан набор 'pen','book','pencil', то поле может иметь только три различных значения - какое либо из указанных в таком наборе. Вводятся и выводятся как строковые константы, заключенные в одинарные кавычки. Если при вводе задано значение, не входящее в перечислимый тип, то такое значение (и вся команда) не являются корректными. | Сравнение на равенство (a==b) - a принимает указанное значение b. Сравнение на неравенство (a!=b) - a не принимает указанное значение b. Проверка на вхожение во множество (a/in/b) - a принимает одно из указанных в b значений, например a/in/['int','bool','char'] - поле a перечислимого типа равно 'int', 'bool', или 'char'. Проверка на отсуствие вхожения во множество (a/not\_in/b) - a не принимает одно из указанных в b значений, например a/not\_in/['Vasya','Petya','Fedya'] - поле a перечислимого типа не равно 'Vasya','Petya' или 'Fedya'. |
| **set (множество)** | Описывает множество предопределенных констант (enum). Каждая константа из enum может либо входить во множество, либо нет. Множество может быть пустым. Набор констант всегда указан для множества. Например, если задан набор констант 'book', 'pencil, 'notepad', то множество может иметь значение [] - пустое, ['book', 'notepad'] - только два предмета, ['book', 'pencil', 'notepad'] - все предметы. При этом множество не может иметь значение ['book', 'pen'], т.к. 'pen' - неизвестная константа, отсутствующая в наборе. Описываются как при вводе так и при выводе в квадратных скобках, содержащих перечисление через запятую элементов, входящих во множество. Каждый элемент - строковая константа, заключенная в одинарные кавычки. Элементы при вводе перечисляются в произвольном порядке, при **выводе - в алфавитном порядке**. Примеры: [], ['book','pen'] | Сравнение на равенство (a==b) - множества a и b в точности одинаковы. Сравнение на неравенство (a!=b) - множества a и b не одинаковы. Сравнение на вхождение (a/in/b) - множество a входит (является подножеством) множества b Сравнение на вхождение (a/include/b) - множество b входит (является подножеством) множества a Сравнение на меньше/больше (a<b,a>b) - количество элементов в a меньше/больше чем во множестве b Операции <=, >= - не определены, при попытке их использования - команда считается некорректной |

Несколько записей образуют базу данных (БД). Набор полей записи (имена и типы) называется *схемой базы данных*. В работе необходимо разработать программу, позволяющую пользователю работать с БД, заданной в вашем варианте задания схемы.  
Под работой пользователя понимается выполнение им операций создания записей в БД, удаления и изменения записей, поиска, а также некоторых других служебных операций.  
Для реализации БД необходимо применить структуру данных *Линейный динамический список* (!НЕ МАССИВ!).  
Каждая запись в БД может быть описана одним элементом динамического списка. Новые записи всегда безусловно добавляются в конец списка. Хранящиеся в БД записи не переставляются местами друг с другом (за исключением ситуаций, когда команда этого явно требует).  
В БД могут быть записи с одинаковыми значениями полей. Общее количество записей в БД неограничено (весь доступный объем памяти программы).

**Постановка задачи**

1. Напишите программу **lab\_db.c**, позволяющую выполнять с БД операции вставки, удаления, обновления и поиска. Команды записаны в файле input.txt, по одной строке на команду. Результаты должны выводиться в output.txt. Программа не должна использовать никаких других "временных" файлов. Описание команд и формата их задания приведно далее. Схема базы данных указана в вашем варианте задания.  
2. Напишите программу **tester\_lab\_db.c**, которая тестирует программу lab\_db в автоматическом режиме.  
3. Опишите в отчете использованные структуры данных, алгоритмы, тесты.  
4. Ответьте на контрольные вопросы.  
  
**Команды к БД**  
Команды к БД записаны в файле *input.txt*. Каждая строка содержит ровно одну команду. Количество команд в файле не ограничивается. Длина команды не ограничивается. Файл может содержать пустые строки, которые должны игнорироваться.  
Команда состоит из ключевого слова и списка аргументов. Аргументы отделены от ключевого слова *одним или несколькими* пробелами.  
  
**insert field1=value1,field2=value2,field3=value3,...,fieldN=fieldN**  
Вставляет в БД новую запись. Аргументом является один аргумент, состоящий из пар field=value, разделенных запятыми, где field - имя поля записи (например, *lastname*), value - его значение (например, *Ivanov*). Порядок следования аргументов может быть произвольным. Значения задаются в соответствие с типом поля записи.  
Если значение столбца указано в неверном формате или указано некорректно, то такая команда не является корректной. Если указаны не все поля, заданные для записи, - то такая команда также не является корректной. Если в команде указано несуществующее поле, то команда также не является корректной. Если для одного и того же поля значение указано несколько раз - то такая команда также некорректна.  
Созданная новая запись должна быть безусловно создана и вставлена в БД последней, после всех записей. В output.txt в ответ на команду выводится одна строка: *insert*, заем двоеточие, затем - общее количество записей в БД после создания новой записи текущей командой. Например, программа должна вывести строку *insert:2* - если в базе уже была была 1 запись и командой insert только что добавилась еще одна. В таком подтверждающем выводе не должно быть пробелов. Это справедливо также для вывода подтверждающих ответов на все команды.  
  
**select field1,field2,...,fieldN cond1 cond2 cond3 ... condN**  
Выполняет запрос к БД с целью поиска записей, удовлетворяющих ВСЕМ условиям. Первый аргумент содержит имена полей записи через запятую, которые следует вывести в ответ на запрос. Далее следует несколько условий, каждый следующий аргумент является одним условием. Условий может не быть вовсе - тогда следует вывести все записи. Условие указывает - в каком отношении значение поля должно быть с заданным в запросе значением (например, равно, больше). Если ВСЕ условия истинны то запись выводится в ответ на запрос, если какое-либо из условий - ложно, то запись не выводится. В условиях в начале всегда находится имя поля, затем оператор (больше, меньше и т.п.), а затем - введенное константное значение, с которым следует сравнить значения поля.  
Примеры условий:

age<18 - Возраст меньше 18

birthday>'31.12.1980' - Дата рождения с 1981 г.

status/in/['running','ready'] - статус является одним из указанных значений

items/include/['book','pencil'] - набор предметов содержит и 'book' и 'pencil'

items/in/['wheel','gear','lamp'] - набор предметов содержит 'wheel' и/или 'gear' и/или 'lamp' или ничего из этого

items==['wheel','gear] - набор предметов содержит только 'wheel' и 'gear' и ничего более

items!=['wheel','gear'] - набор предметов не должен содержать в точности 'wheel' и 'gear'.

В ответ на запрос программа печатает в выходной файл строку select:N, где N - количество найденных записей. Далее следуют сами записи, по одной на строке. Каждая найденная запись выводится в виде:  
field1=value1 field2=value2 field3=value3 fieldN=valueN  
Где field - имя поля, указанное в команде select, value - его значение. *ВАЖНО*: Поля следует в том порядке, в котором были запрошены в команде select. Пары field=value разделяются ОДНИМ пробелом. Значение поля (value) выводится в соответствие с типом данных поля (формат вывода указан в первой таблице). Одно и то же поле может участвовать в запросе несколько раз (например, на первой и предпоследней позиции) - это является допустимым, в строке вывода на соответствующих местах должно быть распечатано имя и значение поля.  
  
**delete cond1 cond2 cond3 ... condN**  
Удаляет из БД те записи, значения полей в которых удовлетворяют ВСЕМ указанным условиям. Условия следуют через пробел, формат - аналогичен select.  
В ответ на команду программа печатает в файл одну строку: *delete*, двоеточие, количество удаленных записей. Например, если из БД было удалено 7 записей должна быть напечатана строка: *delete:7*. Если условия не указаны - то удаляются все записи.  
  
**update field1=value1,field2=value2,...,field1=fieldN cond1 cond2 cond3 ... condN**  
Обновляет указанные поля в БД на указанные значение только у тех записей, которые подходят под условия cond1 cond2 cond3 ... condN. Формат задания условий - как в командах select и delete. Если условия не указаны - то обновляются все записи. В записях обновляются только указанные в команде поля, остальные поля остаются без изменения. Должно быть указано как минимум одно поле для обновления, иначе команда считается некорректной.  
После выполнения команды команда печатает в выходной файл строку: *update:N*, где N - количество обновленных записей.  
  
**uniq field1,field2,...,fieldN**  
Удаляет из БД записи, значения всех указанных в команде полей в которых совпадает со значением полей в других записях (удаляет дубликаты). Если найдено несколько записей, у которых значения полей field1 field2 ... fieldN сопадают одновременно, то должна остаться только одна запись с такими значениями полей. Оставлять в БД следует ту запись, которая ближе к концу БД. Записи, которые ближе к началу, должны быть удалены. Если не указано ни одного поля, то команда считается некорректной. Если одно и то же поле указано несколько раз, то команда считается некорректной.  
В ответ на команду программа печатает одну строку: *uniq*, двоеточие, количество удаленных записей. Например, если из БД было удалено 3 записи должна быть напечатана строка: *uniq:3*  
  
**sort field1=order,field2=order,...,fieldN=order**  
Выполняет сортировку: переставляет записи в БД таким образом, чтобы записи со значением поля field1 меньшим/большим, чем у других записей стояли в БД раньше, чем остальные. Если значения полей field1 у двух записей совпадает, то раньше/позже должна встать запись в зависимости от значения поля field2 и т.д. до поля fieldN. Если значения и fieldN совпадают, то записи не переставляются местами, а остаются в том же порядке. Переставлять их местами - ошибочно (т.е. выполняется *устойчивая сортировка*). В команде sort обязательно должно быть указано как минимум одно поле, иначе - команда считается некорректной. Если одно и то же поле указано несколько раз, то команда считается некорректной. Значением order является одно из ключевых слов - *asc* или *desc*. Они записываются без каких-либо кавычек. Первое слово означает сортировку по возрастанию, второе - по убыванию.  
Сортировка не может быть применена к полям с типом enum или set. Если команда требует этого, то она должна считаться некорректной.  
Особое внимание следует уделить ОПТИМАЛЬНОСТИ алгоритма сортировки: рекомендуется использовать широко применяемые в БД алгоритмы, имеющие сложность O(nlogn) - среднюю или постоянную (QuickSort, MergeSort). В противном случае - сортировка значительного объема записей будет занимать значительное время, что неприемлимо. Если вы применяете не устойчивый алгоритм сортировки, то следует реализовать функцию сравнения элементов так, чтобы результат был устойчивым. После выполнения сортировки записей БД программа записывает в выходной файл строку: *sort:N*, где N - общее количество записей в БД.  
При дальнейших запросах select к БД, выполненных после sort, записи будут выводиться уже в другом порядке, т.к. порядок хранимых записей изменился сортировкой.   
  
Если команда **нераспознана**, либо же распознана, но она - некорректна (например, были указаны не все аргументы, либо же встретились иные синтаксические ошибки, ошибки при задании значений аргументов), в выходной файл должна выводиться строка *incorrect*, затем двоеточие, затем - первые 20 символов такой ошибочной команды (или меньше - если строка команды была короче), заключенные в одинарные кавычки: *incorrect:'первые 20 символов ошибочной команды'*

**Контроль управления дин. памятью**

В ходе работы программа должна подсчитывать количество вызовов функций управления динамической памятью (malloc, calloc, realloc, free и т.д.), а по завершению работы программа должна формировать файл *memstat.txt*, содержащий собранную статистику в виде 'имя\_функции:количество\_вызовов', по одной записи на строке, например:  
**memstat.txt**

malloc:55

relloc:77

calloc:5

free:60

*Примечание*: Вызов функции realloc с первым аргументом равным NULL должен считаться как вызов malloc.  
Если количество free не совпадает с alloc, то имеет место либо **утечка памяти**, либо ошибка **двойного освобождения памяти**.

**Автоматизация тестирования**

Помимо программы lab\_db.c необходимо разработать утилиту tester\_lab\_db.c, тестирующую скомпилированную программу lab\_db.exe.  
Программа должа проводить ряд тестов, каждый из тестов выполняется по следующему алгоритму:

* - Создаются входной файл: input.txt
* - Запускается программа lab\_db.exe с помощью системного вызова system(), ожидается ее завершение.
* - Считываются созданные программой lab\_db выходные файлы.
* - Считанные данные анализируются: проверяются на соответствие ожидаемым.
* - В случае соответствия в стандартный поток выводится 'test ZZ:ok', иначе - 'test ZZ:fail', где ZZ - номер теста. По одной строке на каждый тест.

Количество тестов, которыми должна проверяться программа, указывается вашим преподавателем.

**Содержание отчета**

* 1. Цели работы.
* 2. Описание задачи.
* 3. Подходы к решению. Описание одного или нескольких способов решения данной задачи, аргументация выбранного способа.
* 4. Структуры данных и алгоритмы. В разделе опишите использованные вами структуры данных и алгоритмы программы.
* 5. Тестирование. Описание тестирующей программы, описание тестовых ситуаций, создаваемых тестирующей программой tester\_lab\_db.c (входных данных) и ожидаемых от lab\_db.c выходных данных.
* 6. Выводы. Описание полученных вами в ходе выполнения работы знаний, умений и навыков. Описание трудностей, которые были встречены в работе, совершенных ошибок и способов их устранения.
* 7. Тексты программ.

**Варианты**

|  |  |
| --- | --- |
| Басаргин Клим Анатольевич | 1 |
| Бурмакина Дарья Дмитриевна | 2 |
| Волохова Алина Александровна | 3 |
| Данилина Марина Андреевна | 4 |
| Двойнина Александра Станиславна | 5 |
| Дюпина Александра Сергеевна | 6 |
| Замыслов Илья Владимирович | 7 |
| Кашкаров Олег Евгеньевич(С) | 8 |
| Коротков Савелий Александрович | 9 |
| Котляр Андрей Вячеславович | 10 |
| Кузьмин Андрей Анатольевич | 11 |
| Логинов Захар Григорьевич | 12 |
| Лягун Владислав Александрович | 13 |
| Малова Анастасия Сергеевна | 12 |
| Михайлов Дмитрий Владимирович | 11 |
| Пигарева Анастасия Сергеевна | 10 |
| Рудь Александр Александрович | 9 |
| Смирнов Дмитрий Анатольевич | 8 |
| Таиров Александр Олегович | 7 |
| Хмелёв Павел Петрович | 6 |
| Черепанов Александр Иванович | 5 |
| Юсупова Диана Рустамовна | 4 |

**Вариант 1**

БД запущенных программ в ОС.

pid:int; // PID процесса

name:string; // Командная строка

priority:int; // Приоритет

kern\_tm:time; // Время последнего переключения в kernelmode

file\_tm:time; // Время последней работы с файлом

cpu\_usage:decimal(3,2) // Средняя загрузка процессора

status:enum of ['running','ready','paused','blocked','dying','sleeping'] // Текущий статус

**Вариант 2**

БД результатов автоматического тестирования студенческих л/р.

last\_nm:string; // Фамилия студента

first\_nm:string; // Имя студента

curse\_id:int; // Номер курса

lab\_id:int; // Номер л/р

start\_tm:time; // Время начала тестирования

end\_tm:time; // Время окончания тестирования

result:set of ['test01','test02',...,'test99'] // Множество: пройденные тестов

**Вариант 3**

БД поступлений товаров на склад

comes:date; // Дата поступления

sender:string; // Грузоотправитель

name:string; // Наименование товара

weight:int; // Масса товара

count:int; // Поступило в количестве

images:set of ['fragile','toxic','perishable','acrid','inflammable','frozen'] // Значки на коробке

worker:string; // Принял работник (фамилия)

**Вариант 4**

БД сети метеорологических станций, содержит данные на некоторую дату.

geo\_id:int; // Идентификатор станции

geo\_pos:string; // Размещение станции, адрес, координаты

mea\_date:date; // Дата наблюдений

level:int; // Высота на уровнем моря (м)

sunrise:time; // Время восхода солнца

weather:enum of ['fair','rain','cloudy','snow'] // Общее состояние погоды во время восхода

sundown:time; // Заход солнца

**Вариант 5**

БД сайта службы доставки.

section:string; // Раздел

code:int; // Артикул

name:string; // Название

ingredients:string; // Состав

price:decimal(50,2); // Цена

images:set of ['hot','super\_hot','vegetarian','meat','new','slim','thick','stock']; // Значки

likes:int; // "Лайков" от пользователей

**Вариант 6**

БД проданных блюд клиентам ресторана.

check:int; // Уникальный идентификатор чека

name:string; // Блюдо

price:decimal(50,2); // Цена

count:int; // Количество

kind:enum of ['salad','starter','first','main','drink','beer','snack']; // Вид блюда

req\_tm:time; // Время заказа

put\_tm:time; // Время подачи

**Вариант 7**

БД эксплуатируемых в автопарке а/м.

car\_vendor:string; // Марка а/м

car\_model:string; // Модель а/м

car\_year:int; // Год производства

car\_id:carnum; // Номерной знак а/м

devices:set of ['radio','tachograph','gps','glonass','gsm','cam']; // Укомплектованность а/м техническими модулями

carrying:int; // Грузоподъемность (кг)

axles:int; // Ведущих осей

**Вариант 8**

БД периодического контроля состояния узлов и агрегатов эксплуатируемых в автопарке а/м.

unit\_id:int; // Идентификатор агрегата

unit\_model:string; // Модель агрегата

car\_id:carnum; // Номерной знак а/м

chk\_date:date; // Дата контроля

status:enum of ['well','wearlow','wearhigh','broken','notcheck']; // Состояние узла

mechanic:string; // Фамилия механика

driver:string; // Фамилия водителя

**Вариант 9**

БД мобильного оператора о подключенных услугах.

last\_name:string; // Фамилия владельца

first\_name:string; // Имя владельца

middle\_name:string; // Отчество владельца

number:phone; // Номер телефона

services:set of ['calls','sms','gprs','internet','s1','s2','s3',...,'s10'];

// Подключенные услуги

discount\_id:int; // Код скидочной программы

bonus\_id:int; // Код бонусной программы

**Вариант 10**

БД мобильного оператора об остатке денег на счету

last\_name:string; // Фамилия владельца

first\_name:string; // Имя владельца

middle\_name:string; // Отчество владельца

number:phone; // Номер телефона

money:decimal(50,2); // Денег на счету

min\_money:decimal(50,2); // Порог отключения

status: enum of ['normal','only\_incoming\_calls','no\_calls','disabled'];

// Текущий статус абонента

**Вариант 11**

БД пропусков въезда на территорию

first\_name:string; // Фамилия

last\_name:string; // Имя

call:phone; // Моб. телефон

car:carnum; // Номер а/м

status: enum of ['active','disabled','vacation','lost','broken'];

// Статус пропуска

id:int; // Идентификатор пропуска

used:int; // Сколько раз использован

**Вариант 12**

БД въездов на территорию

first\_name:string; // Фамилия водителя

last\_name:string; // Имя водителя

car\_id:carnum; // Номер а/м

car\_type: enum of ['pass','truck','bus','truck-train','excavator','tractor']; // Тип т/с

tm:time; // Время въезда

id:int; // Идентификатор пропуска

enter\_id:int; // Номер въезда

**Вариант 13**

БД страховой компании о страховых случаях клиентов

first\_name:string; // Фамилия водителя

last\_name:string; // Имя водителя

middle\_name:string; // Отчество водителя

car\_id:carnum; // Гос. номер а/м

occ\_date:date; // Дата просшествия

occ\_damage: set of ['door','wing','engine','top','bonnet','rear-door','bumper','glass']; // Повреждения а/м

id:int; // Присвоенный уникальный номер