Севастопольский государственный университет
Кафедра «Информационные системы»

Управление данными

курс лекций

лектор:
ст. преподаватель кафедры ИС Абрамович А.Ю.

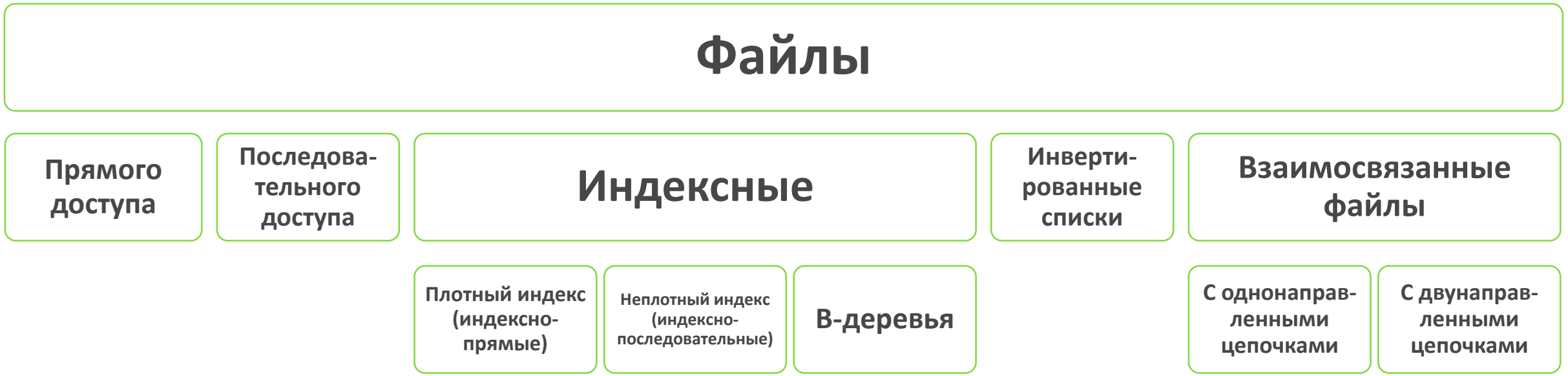
Лекция 15

Физическая организация данных в БД

ФАЙЛОВЫЕ СТРУКТУРЫ

Физические модели баз данных определяют **способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне.**

Классификация файлов и файловых структур, которые используются для хранения информации во внешней памяти.



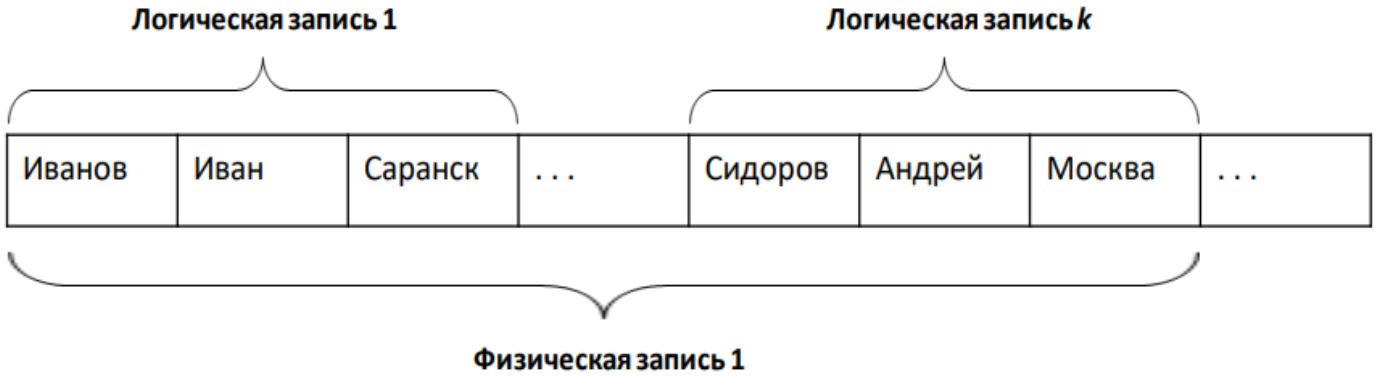
С точки зрения пользователя, **файлом называется поименованная линейная последовательность записей**, расположенных на внешних носителях.

Логическая запись 1	Иванов	Иван	Саранск
Логическая запись 2	Петров	Борис	Пенза
Логическая запись 3	Сидоров	Андрей	Москва
. . .	. .	. . .	. . .
Логическая запись *N*	Яковлев	Петр	Саранск

Файл — это **линейная последовательность записей**, то всегда в файле можно определить текущую запись, предшествующую ей и следующую за ней. Всегда существует понятие первой и последней записи файла.

Модель хранения информации последовательного доступа.

Для получения доступа к некоторому элементу требуется «перемотать (пройти)» все предшествующие ему элементы информации.



Файлы **с постоянной длиной записи**, расположенные на устройствах прямого доступа, являются **файлами прямого доступа**.

В этих файлах физический адрес расположения нужной записи может быть **вычислен по номеру записи**.

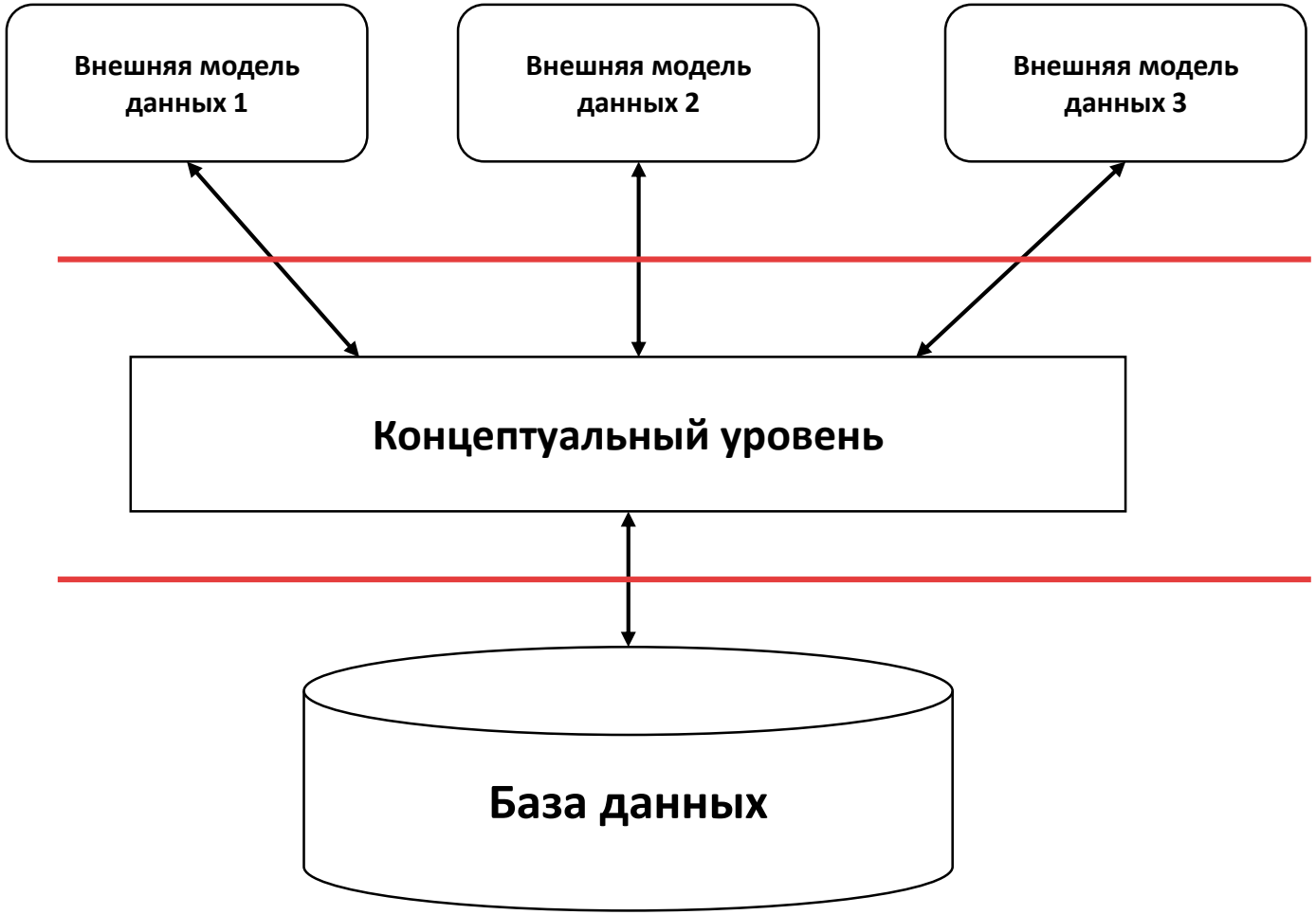
АРХИТЕКТУРА БАЗЫ ДАННЫХ

Трехуровневая система организации БД

Уровень внешних моделей — является самым верхним уровнем или уровнем пользователя. Это совокупность внешних представлений данных, которые обрабатывают приложения и какими их видит пользователь на экране.

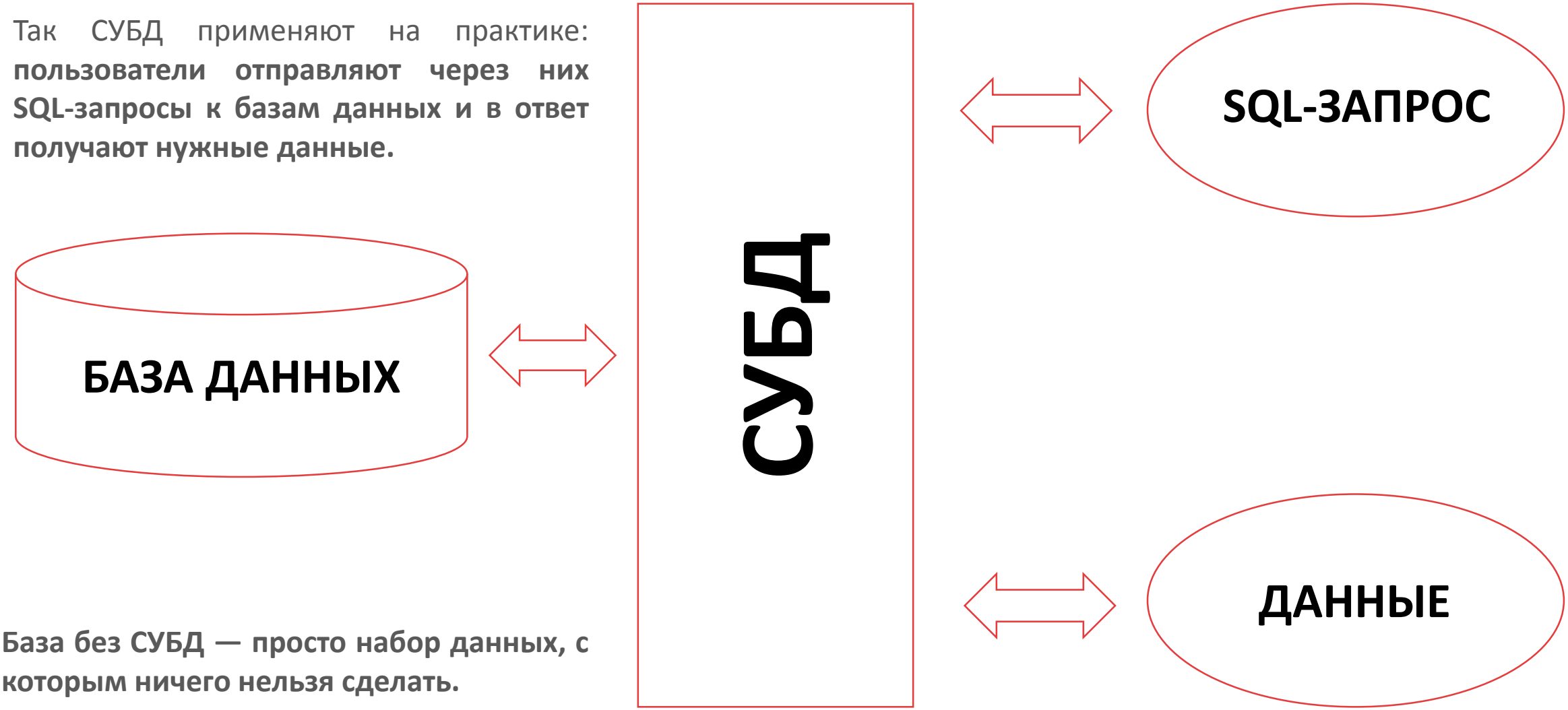
Концептуальный уровень — центральное управляющее звено, здесь база данных представлена в наиболее общем виде, объединяет данные. Фактически отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась БД.

Физический уровень — собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.



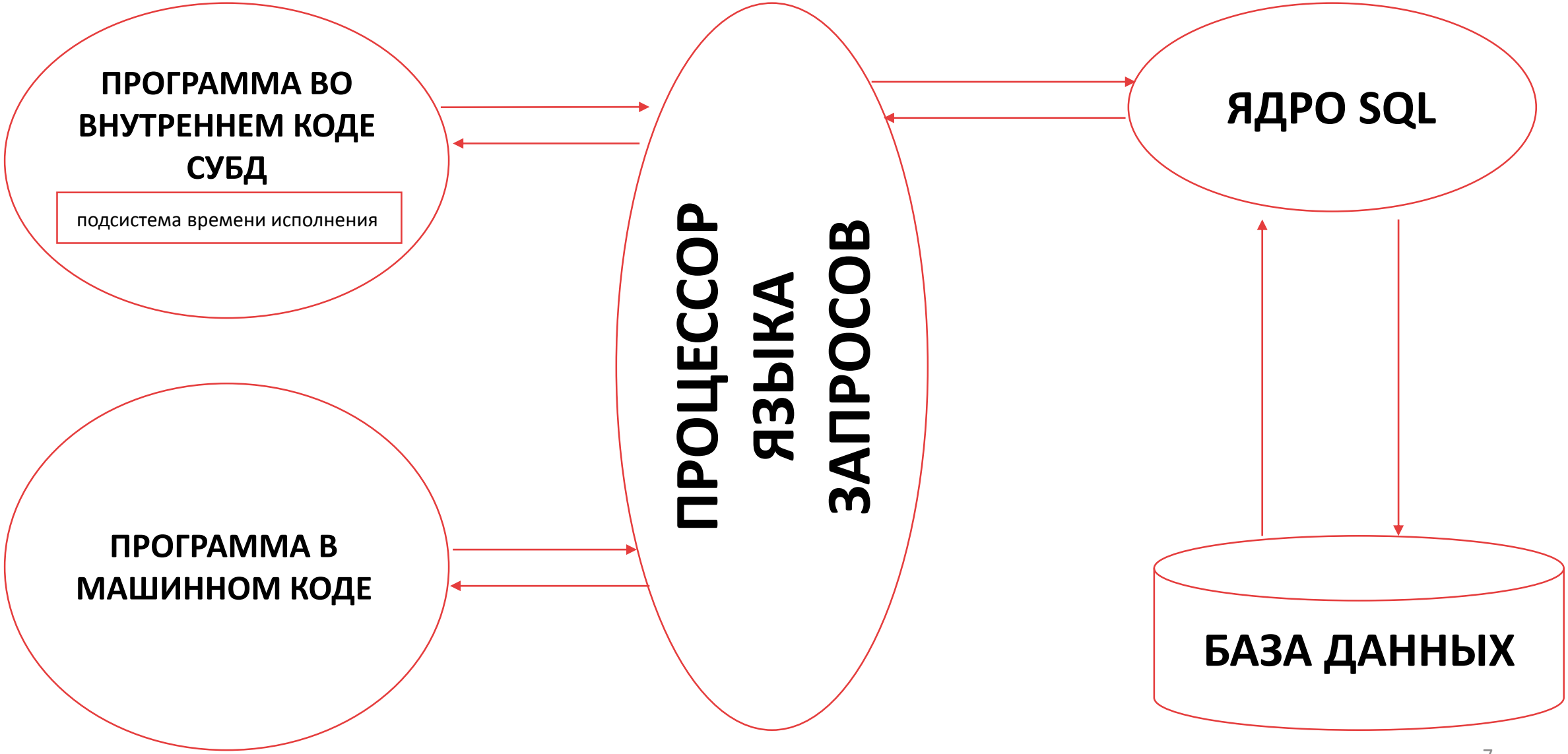
МЕХАНИЗМЫ СРЕДЫ ХРАНЕНИЯ И АРХИТЕКТУРА СУБД

Так СУБД применяют на практике: **пользователи отправляют через них SQL-запросы к базам данных и в ответ получают нужные данные.**



База без СУБД — просто набор данных, с которым ничего нельзя сделать.

Структура СУБД: когда кто-то отправляет запрос к базе данных, он проходит через специальное ПО, процессор языка запросов и ядро, а потом тот же путь проходят результаты в виде данных



МЕХАНИЗМЫ СРЕДЫ ХРАНЕНИЯ И АРХИТЕКТУРА СУБД

С точки зрения пользователя работа с данными происходит **на уровне записей концептуального уровня** и заключается в добавлении, поиске, изменении и удалении записей. При этом механизмы среды хранения **выполняют следующее**:

1. **При запоминании новой записи:**
   - определение места размещения новой записи в пространстве памяти;
   - выделение необходимого ресурса памяти; ? запоминание этой записи (сохранение в памяти);
   - формирование связей с другими записями (конкретный механизм зависит от модели данных).
2. **При поиске записи:**
   - поиск места размещения записи в пространстве памяти по заданным значениям атрибутов;
   - выборка записи для обработки в оперативную память (в буфер данных).
3. **При изменении атрибутов записи:**
   - поиск записи и считывание её;
   - изменение значений атрибута (атрибутов) записи;
   - сохранение записи на диск.
4. **При удалении записи:**
   - удаление записи с освобождением памяти (физическое удаление) или без освобождения (логическое удаление);
   - разрушение связей с другими записями (конкретный механизм зависит от модели данных).

МЕХАНИЗМЫ СРЕДЫ ХРАНЕНИЯ И АРХИТЕКТУРА СУБД

В случае **логического удаления** запись **помечается как удаленная**, но фактически она остаётся на прежнем месте. **При физическом удалении** записи ранее **занятый участок освобождается и становится доступным** для повторного использования.

Все операции **на физическом уровне** **выполняются по запросам механизмов концептуального уровня СУБД**. **На физическом уровне никаких операций** непосредственного обновления пользовательских данных или преобразований представления хранимых данных **не происходит**, это задача более высоких архитектурных уровней. **Управление памятью выполняется** операционной системой **по запросам СУБД или непосредственно самой СУБД**.

В трехуровневой модели архитектуры СУБД декларируется независимость архитектурных уровней. Но для достижения более высокой производительности на уровне организации среды хранения часто приходится учитывать специфику концептуальной модели. Аналогично организация файловой системы не может не оказывать влияния на среду хранения.

СТРУКТУРА ХРАНИМЫХ ДАННЫХ

Единицей хранения данных в БД является **хранимая запись.** Хранимые записи одного типа состоят из фиксированной совокупности полей и могут иметь **формат фиксированной или переменной длины.**

Записи **переменной длины возникают**, если допускается **использование повторяющихся групп полей (агрегатов) с переменным числом повторов или полей переменной длины**. Работа с хранимыми записями переменной длины существенно **усложняет управление пространством памяти.**

Хранимая запись состоит из двух частей:

1. **Служебная часть.** Используется для идентификации записи, задания её типа, хранения признака логического удаления, для кодирования значений элементов записи, для установления структурных ассоциаций между записями и прочее. Никакие пользовательские программы не имеют доступа к служебной части хранимой записи.
2. **Информационная часть.** Содержит значения элементов данных.

Поля хранимой записи могут иметь ***фиксированную или переменную длину***.

Хранение полей переменной длины осуществляется одним ***из двух способов:*** размещение полей через разделитель или хранение размера значения поля. Наличие полей переменной длины **позволяет не хранить незначащие символы и снижает затраты памяти на хранение данных**; но при этом **увеличивается время на извлечение записи**.

УПРАВЛЕНИЕ ПРОСТРАНСТВОМ ПАМЯТИ И РАЗМЕЩЕНИЕМ ДАННЫХ

Ресурсам пространства памяти соответствуют **объекты внешней памяти ЭВМ**, управляемые средствами операционной системы или СУБД.

Для обеспечения естественной структуризации хранимых данных, более эффективного управления ресурсами и/или для технологического удобства **всё пространство памяти БД обычно разделяется на части (области, сегменты и др.).** В каждой области памяти, как правило, **хранятся данные одного объекта БД (одной таблицы)**. Сведения о месте расположения данных таблицы (ссылка на область хранения) СУБД **хранит в словаре-справочнике данных (ССД).** Области **разбиваются на пронумерованные страницы (блоки)** фиксированного размера. В большинстве систем обработку данных на **уровне страниц ведёт операционная система (ОС),** а обработку записей **внутри страницы обеспечивает только СУБД.**

Система автоматически управляет свободным пространством памяти на страницах. Как правило, это обеспечивается одним из двух способов:

- ведение списков свободных участков;
- динамическая реорганизация страниц.

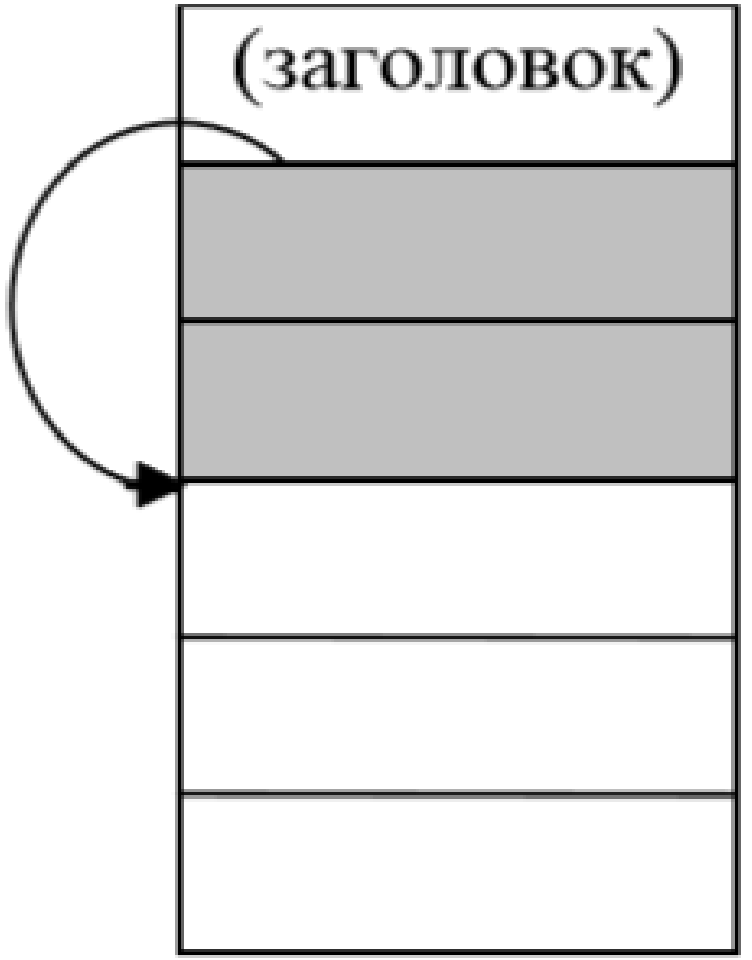
При **динамической реорганизации** страниц **записи БД плотно размещаются вслед за заголовком страницы, а после них расположен свободный участок**

При **удалении записи** оставшиеся записи переписываются подряд в начало страницы и изменяется смещение начала свободного участка. **При увеличении размера существующей** записи она записывается по прежнему адресу, а вслед идущие записи сдвигаются.

Достоинство такого подхода – отсутствие фрагментации.

Недостатки:

- адрес записи может быть определён с точностью до адреса страницы, т.к. внутри страницы запись может перемещаться;
- поиск места размещения новой записи может занять много времени.



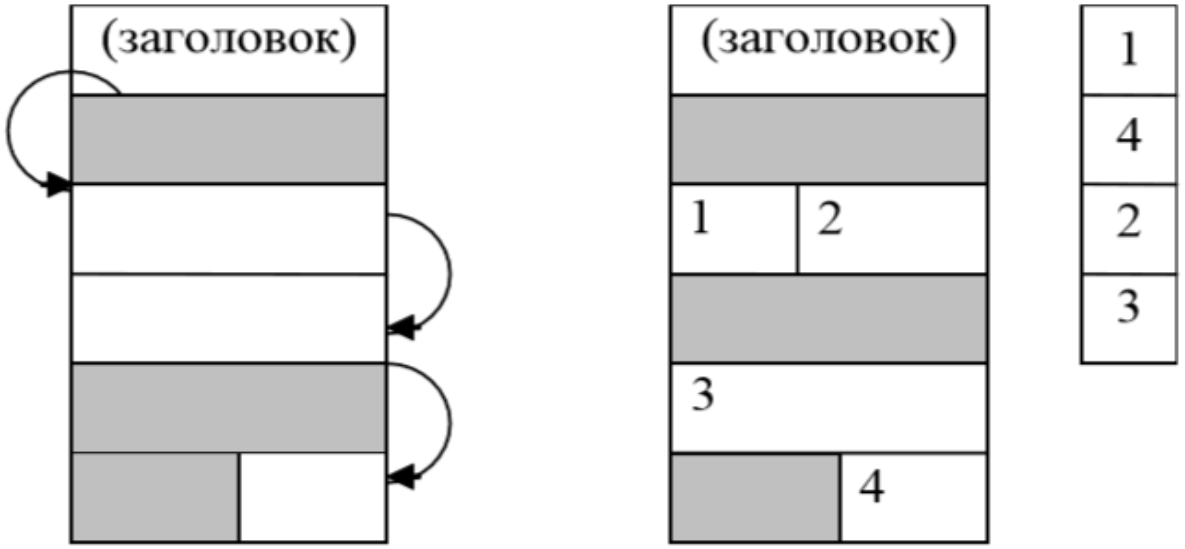
Динамическая реорганизация страниц

Некоторые СУБД управляют памятью по-другому - они **ведут список свободных участков**:

- ссылка **на первый свободный участок на странице хранится в заголовке страницы**, и **каждый свободный участок хранит ссылку на следующий** (или признак конца списка. Каждый **освобождаемый участок включается в список свободных участков** на странице;
- списки **свободных участков реализуются в виде отдельных структур**. Эти структуры также **хранятся на отдельных инвентарных страницах**. Каждая инвентарная страница относится к области (или группе страниц) памяти и **содержит информацию о свободных участках в этой области**. Список ведётся как **стек, очередь или упорядоченный список**.

Ведение списков свободных участков **не приводит к перемещению записи**, и адрес записи можно **определить с точностью до смещения на странице**. Это ускоряет поиск данных.

Основным недостатком, возникающим при использовании списков свободных участков, является **фрагментация пространства памяти**, т.е. появление разрозненных незаполненных участков памяти.



ВИДЫ АДРЕСАЦИИ ХРАНИМЫХ ЗАПИСЕЙ

В общем случае адреса записей БД нигде не хранятся.

В **РСУБД** для ускорения поиска данных **применяются индексы** – специальные структуры, устанавливающие соответствие значений ключевых полей записи и «адреса» этой записи (КБД). Таким образом, **вид адресации хранимых записей оказывает влияние на производительность**, а также на переносимость БД с одного носителя на другой.

ПРЯМАЯ АДРЕСАЦИЯ предусматривает указание непосредственного местоположения записи в пространстве памяти.

КОСВЕННАЯ АДРЕСАЦИЯ заключается в том, что в качестве КБД выступает не сам «адрес записи», а адрес места хранения «адреса записи».

Общий принцип **ОТНОСИТЕЛЬНОЙ АДРЕСАЦИИ** заключается в том, что адрес отсчитывается от начала той области памяти, которую занимают данные объекта БД.

Некоторые СУБД, использующие относительную адресацию, при необходимости перемещения отдельной записи **оставляют КБД прежним**. Физически запись хранится на новом месте, а по старому адресу хранится новый адрес записи. Это позволяет не менять КБД и не перестраивать индексы, но приводит к увеличению времени доступа к записи (2 физических чтения вместо одного).

СПОСОБЫ РАЗМЕЩЕНИЯ ДАННЫХ

При создании новой записи во многих случаях **существенно размещение этой записи в памяти**, т.к. это **оказывает огромное влияние на время выборки**. Простейшая стратегия размещения данных заключается в том, что **новая запись размещается на первом свободном участке (если ведется учёт свободного пространства) или вслед за последней из ранее размещённых записей**.

Среди более сложных методов размещения данных – **хеширование и кластеризация**.

Хеширование заключается в том, что **специально подобранная хеш-функция преобразует значение ключа записи в адрес блока (страницы) памяти, в котором эта запись будет размещаться.** Под ключом записи здесь подразумевается поле или набор полей, позволяющие классифицировать запись.

Кластеризация – это **способ хранения в одной области памяти таблиц, связанных внешними ключами (одна родительская таблица, одна или несколько подчинённых таблиц).** Для размещения записей используется значение внешнего ключа таким образом, чтобы все данные, имеющие одинаковое значение внешнего ключа, размещались в одном блоке данных.

Способы доступа к данным

ПОСЛЕДОВАТЕЛЬНАЯ ОБРАБОТКА ОБЛАСТИ БД

- Последовательная обработка предполагает, что система последовательно просматривает страницы, пропускает пустые участки и выдаёт записи в физической последовательности их хранения.

ДОСТУП ПО КЛЮЧУ БАЗЫ ДАННЫХ (КБД)

- КБД определяет местоположение записи в памяти ЭВМ. Зная его, система может извлечь нужную запись за одно обращение к памяти.

Доступ по ключу (в частности, первичному)

- Если система обеспечивает доступ по ключу, то этот ключ также может использоваться при запоминании записи (для определения места размещения записи в памяти).

Индексирование данных

Индекс – это структура, которая **определяет соответствие значения ключа записи (атрибута или группы атрибутов) и местоположения этой записи – КБД.** Каждый индекс связан с определённой таблицей, но является **внешним по отношению к таблице** и обычно хранится отдельно от неё.

Индекс

Значение атрибута	*КБД*
Белова	FA:00
Волков	F6:1E
Волкова	F6:00
Осипов	FA:2B
Поспелов	F6:31
Фридман	FA:1D

Пространство памяти

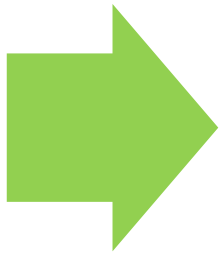
F6:00	Волкова	…
F6:1E	Волков	…
F6:31	Поспелов	…
	…	
FA:00	Белова	…
FA:1D	Фридман	…
FA:2B	Осипов	…

Индексирование **используется для ускорения доступа к записям по значению ключа** и не влияет на размещение данных этой таблицы. Ускорение поиска данных через индекс **обеспечивается за счёт**:

- упорядочивания значений индексируемого атрибута. Это позволяет просматривать в среднем половину индекса при линейном поиске;
- индекс занимает меньше страниц памяти, чем сама таблица, поэтому система тратит меньше времени на чтение индекса, чем на чтение таблицы.

Обращение к записи таблицы через индексы осуществляется в два этапа:

СУБД считывает индекс в оперативную память (ОП) и находит в нём требуемое значение атрибута и соответствующий адрес записи (КБД).



По этому адресу происходит обращение к внешнему запоминающему устройству.

Индекс называется **первичным**, если каждому значению индекса соответствует уникальное значение ключа. Индекс по ключу, допускающему дубликаты значений, называется **вторичным**. **Большинство СУБД автоматически строят индекс по первичному ключу и по уникальным столбцам.**

Для каждой таблицы можно одновременно иметь несколько первичных и вторичных индексов, что также относится к достоинствам индексирования.

Хеширование

Принцип хеширования заключается в том, что для определения адреса записи в области хранения к значению ключевого поля этой записи применяется так называемая **хеш-функция h(K)**. Она преобразует **значение ключа K в адрес участка памяти** (это называется свёрткой ключа). **Новая запись будет размещаться по тому адресу, который выдаст хеш-функция для ключа этой записи.** При поиске записи по значению ключа K хеш-функция выдаст адрес, указывающий на начало того участка памяти, в котором надо искать эту запись.

Хеш-функция h(K) должна обладать двумя основными свойствами:

- выдавать такие значения адресов, чтобы обеспечить равномерное распределение записей в памяти, в частности, для близких значений ключа значения адресов должны сильно отличаться, чтобы избегать перекосов в размещении данных:

$$K1 \approx K2 \rightarrow h(K1) >> h(K2) \vee h(K2) >> h(K1),$$

- для разных значений ключа выдавать разные адреса:

$$K1 \neq K2 \rightarrow h(K1) \neq h(K2).$$

Недостаток методов подбора хеш-функций заключается в том, что **количество данных и распределение значений ключа должны быть известны заранее.** Также методы хеширования неудобны тем, что записи обычно **неупорядочены по значению ключа**, что приводит к дополнительным затратам, например, при выполнении сортировки.

К преимуществам хеширования относится то, что **ускоряется доступ к данным по значению ключа**. Обращение к данным происходит за одну операцию ввода/вывода, т.к. значение ключа с помощью хеш-функции непосредственно преобразуется в адрес соответствующей записи (или адрес блока памяти, в котором хранится эта запись). При этом не нужно создавать никаких дополнительных структур (типа индекса) и тратить память на их хранение.