Базы данных

Лекция 02 – Основные концепции. Терминология

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2024

2024.02.13

Оглавление

Основная терминология

Существует много вариантов терминов, используемых при описании данных.

Широко признанным авторитетом в области баз данных, не связанным ни с какой конкретной фирмой-производителем ЭВМ, является CODASYL (COnference on DAta SYstems Languages —Ассоциация по языкам для систем обработки данных).

Байт (Byte)

Байт — наименьшая адресуемая группа битов (обычно состоит из 8 бит[ов]).

Элемент данных (Data Element)

Элемент данных — наименьшая единица поименованных данных. Может состоять из любого количества битов или байтов. Часто элемент данных называют ***полем***.

Элементом данных может быть некоторая величина, например, наименование товара в счет-фактуре или его количество.

Агрегат данных

Агрегат данных — поименованная совокупность элементов данных внутри ***записи***, рассматриваемая как единое целое.

Например, агрегат данных **ДАТА** может состоять из элементов данных **МЕСЯЦ**, **ДЕНЬ**, **ГОД**.

Существуют два типа агрегатов данных – векторы и повторяющиеся группы.

Вектор — одномерная упорядоченная совокупность элементов данных (например, **рабочие дни**).

Повторяющаяся группа — совокупность данных, которые встречаются несколько раз в экземпляре записи, например прием и выдача вкладов в записи счета сберкассы или счет-фактура.

В повторяющуюся группу могут входить отдельные элементы данных, векторы, агрегаты данных или другие повторяющиеся группы.

Допускается вложенность агрегатов.

Запись (Record)

Запись — поименованная совокупность элементов или агрегатов данных.

Запись — агрегатный тип данных, инкапсулирующий набор значений различных типов без их сокрытия.

При чтении из базы данных может быть прочитана логическая запись как целиком, так и частично — в ряде случаев логической записью базы данных является структура данных, в которую входит несколько групп элементов данных (*сегментов*), не все из которых имеет смысл читать одновременно.

Верхний предел для числа возможных экземпляров записи конкретного типа зависит, в основном, от наличия необходимого оборудования для хранения и доступа.

Верхний предел для числа повторяющихся групп внутри записи обычно ограничен небольшим числом.

Сегмент (Segment)

Существует мнение, что нет необходимости различать агрегат данных и запись, поскольку и то и другое является совокупностью элементов данных. В терминологии, используемой фирмой IBM, а также в других источниках и агрегат данных, и запись называют сегментом.

Сегмент состоит из одного или нескольких элементов данных (обычно нескольких) и является основным квантом данных, передаваемым прикладной программе или получаемым от нее под управлением программного обеспечения, работающего с базой данных.

Физический уровень

- уровень блоков;
- уровень файлов.

Таблица

Записи *одинаковой структуры* обычно организуются в двумерные таблицы и представляются в них строками.

Элементы данных записей организованы в виде столбцов.

Таблицы организуются в *базы данных*.

Ниже БД, содержащая 2 таблицы, связанные через поле DEPT_NO.

DEPARTMENT – Отделы

DEPT_NO	DEPT_NAME	BUDGET
D1	Marketing	5 000 000.00
D2	Development	25 000 000.00
D3	Research	10 000 000.00

EMPLOYEES – Служащие

EMP_NO	EMP_NAME	DEPT_NO	SALARY
E1	Ivanov	**D1**	40 000.00
E2	Smirnov	**D2**	45 000.00
E3	Kuznetsov	**D1**	41 000.00
E4	Popov	**D2**	43 000.00
E5	Vasiljev	**D3**	55 000.00
E6	Petrov	**D2**	45 000.00

В разное время и в различных контекстах термин ***запись*** может означать экземпляр записи или тип записи, логическую запись или физическую запись, хранимую запись или виртуальную запись, а возможно, и еще что-нибудь.

В связи с этим в формальной реляционной модели термин запись не используется — вместо него применяется термин *кортеж* (tuple).

Также в реляционной модели не используется термин *поле*, вместо него используется термин *атрибут*.

Кортеж

Термин кортеж в отношении к базам данных приблизительно соответствует понятию строки в таблице (так же, как термин отношение приблизительно соответствует понятию таблицы).

Соответственно, кортеж состоит из атрибутов.

Кортежем вообще называется набор взаимосвязанных величин.

Кортеж, содержащий две величины, называется *парой* (pair).

Кортеж, содержащий N величин, называется N-кортежем.

Таблица состоит из набора кортежей, каждый из которых содержит элементы данных одинакового типа. Она, таким образом, представляет собой двумерную матрицу элементов данных.

Элементы данных обычно обрабатываются группами.

В различных системах программного обеспечения эти группы называются по-разному (сегмент, кортеж).

Общепринятым является термин запись.

Отношение

Отношение — это формальное название таблицы.

Например, можно сказать, что база данных отделов и служащих, представленная выше, содержит два отношения.

В настоящее время в неформальном контексте термины отношение и таблица принято считать синонимами. На практике в подобном контексте термин таблица используется гораздо чаще, чем термин отношение.

Файл

Файл — поименованная совокупность всех экземпляров логических записей заданного типа.

В простом файле в каждой логической записи содержится одинаковое число элементов данных (рисунок 1).

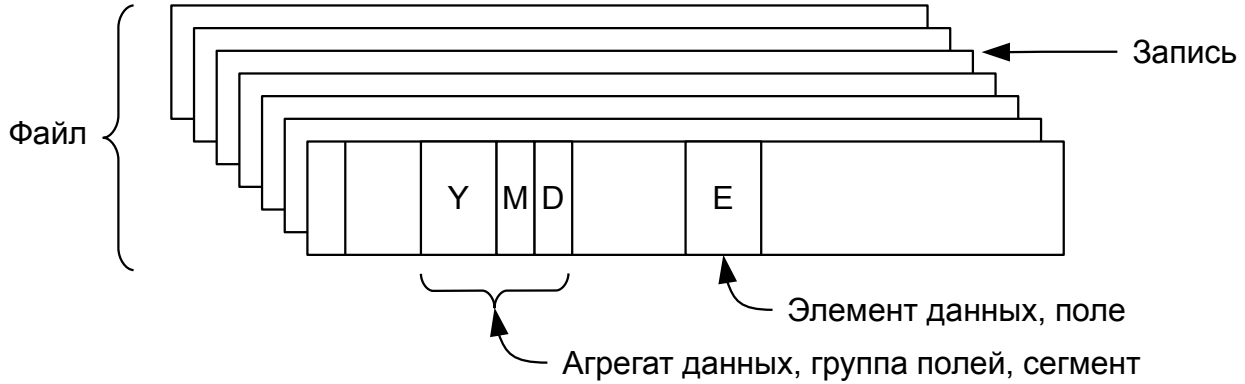

Рисунок 1 — Термины, используемые при описании данных
с точки зрения прикладного программиста

В более сложном файле из-за наличия повторяющихся групп записи могут состоять из различного числа элементов данных.

База данных

База данных — совокупность экземпляров различных типов записей и отношений между записями, агрегатами данных, элементами данных.

Пример – счет-фактура

Некоторая величина в счет-фактуре является *элементом данных*.

Агрегатом данных может быть группа элементов данных в строке счет-фактуры.

Если она повторяется несколько раз, она будет *повторяющейся группой*.

Строка может рассматриваться, как отдельно адресуемая группа данных, в этом случае она будет *сегментом*.

Весь счет-фактура может рассматриваться, как *логическая запись*.

Вся совокупность записей счетов-фактур обычно хранится в одной или нескольких *таблицах*.

Физическая структура хранения логических записей на устройстве хранения зависит от СУБД.

Унифицированная форма № ТОРГ-12
Утверждена постановлением Госкомстата
России от 25.12.98 № 132

		Код
	Форма по ОКУД	0330212
	по ОКПО	
	Вид деятельности по ОКДП	
Грузополучатель	по ОКПО	
Поставщик	по ОКПО	
Плательщик	по ОКПО	
Основание	номер	
	дата	
Транспортная накладная	номер	
	дата	
	Вид операции	

(организация-грузоотправитель, адрес, телефон, факс, банковские реквизиты)

(структурное подразделение)

(организация, адрес, телефон, факс, банковские реквизиты)

(организация, адрес, телефон, факс, банковские реквизиты)

(организация, адрес, телефон, факс, банковские реквизиты)

(договор, заказ-наряд)

Номер документа	Дата составления

ТОВАРНАЯ НАКЛАДНАЯ

Номер по порядку	Товар		Единица измерения		Вид упаковки	Количество		Масса брутто	Количество (масса нетто)	Цена, руб. коп.	Сумма без учета НДС, руб. коп.	НДС		Сумма с учетом НДС, руб. коп.
	наименование, характеристика, сорт, артикул товара	код	наименование	код по ОКЕИ		в одном месте	мест, штук					ставка, %	сумма, руб. коп.	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Итого										X		X		

Приложение № 1
к постановлению Правительства
Российской Федерации
от 26 декабря 2011 г. № 1137

(в ред. Постановления Правительства РФ от 19.08.2017 № 981)

СЧЕТ-ФАКТУРА № _____ от "____" _____ (1)
ИСПРАВЛЕНИЕ № _____ от "____" _____ (1а)

Продавец _____ (2)
Адрес _____ (2а)
ИНН/КПП продавца _____ (2б)
Грузоотправитель и его адрес _____ (3)
Грузополучатель и его адрес _____ (4)
К платежно-расчетному документу № _____ от _____ (5)
Покупатель _____ (6)
Адрес _____ (6а)
ИНН/КПП покупателя _____ (6б)
Валюта: наименование, код _____ (7)
Идентификатор государственного контракта, договора (соглашения) (при наличии) _____ (8)

Наименование товара (описание выполненных работ, оказанных услуг), имущественного права	Код вида товара	Единица измерения		Коли-чество (объем)	Цена (тариф) за единицу измерения	Стоимость товаров (работ, услуг), имущественных прав без налога - всего	В том числе сумма акциза	Налоговая ставка	Сумма налога, предъяв-ляемая покупателю	Стоимость товаров (работ, услуг), имущественных прав с налогом - всего	Страна происхождения товара		Регистра-ционный номер таможенной декларации
		код	условное обозначе-ние (нацио-нальное)								цифровой код	краткое наименование	
1	1а	2	2а	3	4	5	6	7	8	9	10	10а	11
Всего к оплате							X						

Руководитель организации
или иное уполномоченное лицо _____ _____
(подпись) (ф.и.о.)

Главный бухгалтер
или иное уполномоченное лицо _____ _____
(подпись) (ф.и.о.)

Индивидуальный предприниматель
или иное уполномоченное лицо _____ _____ _____
(подпись) (ф.и.о.) (реквизиты свидетельства о государственной регистрации индивидуального предпринимателя)

Виртуальные данные

Слово *виртуальный*, относящееся к техническим средствам или данным, указывает на то, что некоторый элемент в запросе представляется прикладному программисту существующим, тогда как фактически в представленном виде он отсутствует.

Программист может обращаться за виртуальными данными, которые им предполагаются существующими, но не существуют фактически, по крайней мере в данной форме.

Каждый раз, когда программист обращается за этими данными, они генерируются определенным образом, что возможно обеспечивает более компактную и удобную форму их хранения.

Прозрачные средства и данные

Виртуальное только представляется существующим, прозрачное представляется несуществующим, но на самом деле существует.

Многие данные и механизмы, используемые при их хранении и передаче, могут быть скрыты от программиста (view).

База данных

Базу данных можно определить как совокупность взаимосвязанных хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Данные хранятся таким образом, чтобы они были независимы от программ, использующих эти данные.

Для добавления новых или модификации существующих данных, а также для поиска данных в базе данных применяются общие методы (способы). Данные при этом структурируются таким образом, чтобы была обеспечена возможность дальнейшего наращивания приложений.

Говорят, что система содержит ***совокупность баз данных***, если эти базы данных структурно полностью самостоятельны.

База данных может разрабатываться для:

- пакетной обработки данных;
- обработки в реальном времени (жесткие ограничения на время обработки запроса);
- оперативной обработки (в этом случае обработка каждой транзакции завершается к определенному моменту времени, но при этом на время обработки не накладывается жестких ограничений, существующих в системах реального времени).

Во многих базах данных предусмотрена совокупность вышеуказанных методов обработки, причем все три вида обработки могут исполняться параллельно.

OLTP (Online Transaction Processing) — обработка транзакций в реальном времени. Способ организации БД, при котором система работает с небольшими по размерам транзакциями, но идущими большим потоком, и при этом клиенту требуется от системы минимальное время отклика.

Избыточность

База данных представляет возможность в значительной степени избавиться от избыточности.

Базу данных иногда определяют как *неизбыточную совокупность элементов данных*, однако в действительности избыточность в определенной степени допускают во многих базах данных с целью уменьшения времени доступа к данным и/или упрощения способов адресации.

Иногда некоторые записи дублируются для того, чтобы обеспечить возможность быстрого восстановления данных при их случайной потере.

Чтобы база данных была неизбыточной и удовлетворяла другим требованиям, приходится идти на компромисс.

В этом случае говорят об *управляемой*, или *минимальной*, избыточности или о том, что хорошо разработанная база данных свободна от *излишней* избыточности.

Неуправляемая избыточность имеет несколько недостатков:

1) хранение нескольких копий данных приводит к дополнительным затратам;

2) что особенно серьезно, приходится выполнять многократные операции обновления для нескольких избыточных копий.

Избыточность поэтому обходится значительно дороже в тех случаях, когда при обработке файлов обновляется большое количество информации или, что еще хуже, часто вводятся новые элементы данных или уничтожаются старые.

3) вследствие того что различные копии данных могут соответствовать различным стадиям обновления, информация, выдаваемая системой, может быть противоречивой.

Если не использовать базы данных, то при обработке большого количества информации появится так много избыточных данных, что фактически станет невозможным сохранять их все на одном и том же уровне обновления.

Непрерывное расширение

Одной из наиболее важных характеристик большинства баз данных является их постоянное изменение и расширение.

По мере добавления новых типов данных или при появлении новой бизнес-логики должна быть обеспечена возможность быстрого изменения структуры базы данных.

Реорганизация базы данных должна осуществляться по возможности без перезаписи прикладных программ и в целом вызывать минимальное количество преобразований.

Простота изменения базы данных может оказать большое влияние на развитие приложений для баз данных, используемых в управлении производством.

- Требования к обработке данных обычно изменяются непредсказуемым образом. При этом если возникает необходимость модификации выбранных структур данных, то приходится соответственно перезаписывать и отлаживать прикладные программы.

- Чем большее количество прикладных программ имеется в наличии на установке, тем более дорогой становится эта процедура.

Поэтому одним из важных свойств базы данных является ***независимость данных и использующих их прикладных программ друг от друга*** в том смысле, что изменение одних не приводит к изменению других.

В действительности же полностью независимыми данные бывают так же редко, как и полностью неизбыточными.

Независимость данных определяется с различных точек зрения. Сведения, которыми должен располагать программист для доступа к данным, различны для различных баз данных.

Тем не менее независимость данных — это одна из основных причин использования систем управления базами данных.

Установление многосторонних связей

В том случае, когда один набор элементов данных используется для многих приложений, между элементами этого набора устанавливается множество различных взаимосвязей, необходимых для соответствующих прикладных программ.

Организация базы данных в значительной степени зависит от реализации ***связей между элементами данных и записями***.

В базе данных, используемой многими приложениями, должны быть установлены многочисленные промежуточные взаимосвязи между данными. В этом случае при хранении и использовании данных контролировать их правильность, обеспечивать их защиту и секретность труднее, чем при хранении данных в простых, несвязанных файлах.

Что касается обеспечения секретности данных и восстановления их после сбоев, то этот вопрос при конструировании баз данных является очень важным.

!!! В подавляющем количестве систем средства управления базами данных обеспечивают возможность пользователи использовать данные таким способом, который не был предусмотрен разработчиками системы. Это значит, что

**Пользователи могут обращаться к БД с запросами,
которые заранее в ней не предусматривались.**

Наличие такой возможности означает организацию данных в системе, при которой доступ к ним осуществляется по различным путям, причем одни и те же данные могут использоваться для ответа на различные запросы.

При организации данных в виде ориентированных на конкретное приложение схем, в которых доступ к данным производится одним и тем же определенным способом, описанная выше возможность отсутствует. Вся существенная информация об объектах должна сохраняться одновременно и полностью, а не только та ее часть, которая необходима для конкретного варианта использования.

→ L1.21-25

Идентификатор объекта

Программисту или администратору базы данных необходимо иметь возможность обращаться к записи или к кортежу, связанному с данным объектом.

Для этого необходимо иметь возможность идентифицировать запись или группу записей и располагать средствами их обнаружения в таблице.

С этой целью один из элементов данных обычно определяется в качестве идентификатора объекта.

Идентификатор объекта должен быть уникальным – никакой другой объект не может иметь то же значение данного элемента данных.

Идентификатором объекта **СЛУЖАЩИЙ** *может быть* **НОМЕР_СЛУЖАЩЕГО**.

Идентификатором объекта **СЧЕТ** *может быть* **НОМЕР_СЧЕТА**.

Иногда требуется более одного элемента данных для идентификации записи.

Например, для того чтобы идентифицировать банковскую операцию для записи **СЧЕТ**, необходимы идентификаторы **НОМЕР_СЧЕТА** и **НОМЕР_ОПЕРАЦИИ**.

Для идентификации записи о рейсах самолетов необходимы **НОМЕР_РЕЙСА** и **ДАТА**. Одного номера рейса недостаточно, так как вылет с одним и тем же номером рейса может происходить каждый день.

Первичный ключ

Идентификатор объекта рассматривается как ключ записи или группы записей. Такой ключ называется ***первичным ключом***. (Primary Key)

В тех случаях, когда в качестве ключа используются несколько элементов данных (полей, агрегатов), их обычно указывают как соединенные символом «плюс», например **НОМЕР_РЕЙСА + ДАТА**. Такое сочетание называется сцепленным ключом.

Иногда для формирования первичного ключа необходимы три, четыре и более элементов данных. Запись о кассовом сборе, например, в приведенной ниже группе идентифицируется комбинацией элементов **КИНОФИЛЬМ**, **КИНОТЕАТР**, **ДАТА**.

КИНОФИЛЬМ	КИНОТЕАТР	ДАТА	КАССОВЫЙ_СБОР

Первичным ключом поэтому является **КИНОФИЛЬМ+КИНОТЕАТР+ДАТА**.

***Первичный ключ* — это такой элемент данных или такая совокупность элементов данных, которая единственным образом идентифицирует одну запись или группу записей.**

Первичный ключ имеет большое значение, так как он используется для определения местоположения записи с помощью индексов или других методов адресации.

Избыточные значения

Значения атрибутов не обязательно должны запоминаться вместе с ключами таким способом, как это показано ниже.

Дело_№	Фамилия Имя Отчество	Пол	Звание	Рожд	Отдел	Должность
58231	Сидоров Иван Петрович	муж	06	1971.03.12	002	Начальник станции
12874	Смирнова Анна Ивановна	жен	08	1988.10.11	014	Инженер
31774	Бутусова Ульяна Алексеевна	жен	11	1999.08.18	002	Инженер-программист

Звание

ID	Name
04	Полковник
05	Подполковник
...	
17	Кадет
18	Вольноопределяющийся

Отдел

ID	Name
002	Директорат
003	Бухгалтерия
...	
014	ПФО
015	ОЗХО

Должность

ID	Name
08	Начальник станции
09	Инженер-программист
10	Оператор
...	
58	Повар

Почти всегда наблюдается существенная избыточность значений атрибутов в тех случаях, когда они хранятся так, как показано в атрибуте **Должность**.

Для того чтобы устранить избыточность, значения атрибутов могут храниться отдельно и снабжаться указателями на них со стороны ключей (Звание).

Такие указатели называются ***внешними ключами*** (Foreign Key).

Вторичные ключи (secondary key)

Можно использовать ключ, который идентифицирует не уникальную запись или кортеж, а все записи или группы, имеющие определенное свойство.

Такой ключ называется *вторичным*.

Значение атрибута **ЦВЕТ**

В качестве вторичного ключа может быть использован атрибут **`Отдел.Name`**.

Этот ключ может быть использован для идентификации тех служащих, которые числятся в определенном отделе.

Иногда таблица имеет много внешних ключей, которые используются для поиска записей с данными характеристиками.

Связь вторичного ключа с элементами данных или с группами, к которым он относится, может быть реализована различными способами. Один из таких способов – использование вторичного индекса. Индекс использует вторичный ключ как вход, а на выходе предоставляет первичный ключ, в результате чего может быть идентифицирована нужная запись или группа записей.

Элементарная форма вторичного индекса — инвертированный список или файл.

Звание	Дело_№
05	13875
05	76014
06	58231
06	32960
08	12874

Фамилия Имя Отчество	Дело_№
Сидоров Иван Петрович	58231
Смирнова Анна Ивановна	12874
Бутусова Ульяна Алексеевна	31774

Инвертированный список содержит все значения вторичного ключа и хранит вместе с каждым его значением соответствующие идентификаторы записи (первичные ключи).

Инвертированные файлы

Существуют два основных способа, с помощью которых данные могут быть организованы и использованы.

1) Первый способ определяется тем, что каждый кортеж содержит значения атрибутов данного объекта или N:1 ссылки на справочные таблицы, содержащие атрибуты (LUT– LookUp Table).

2) Второй способ является инверсией первого. С помощью этого способа могут быть получены идентификаторы объектов, связанных с данным атрибутом.

Первый способ хранения данных полезен для ответа на вопрос: *каковы свойства данного объекта?*

Второй — для ответа на вопрос: *какие объекты имеют данное свойство?*

Полностью инвертированный файл — это такой файл, который хранит идентификаторы объектов, связанные с конкретным значением каждого атрибута.

Частично инвертированный файл является более простым и хранит идентификаторы объектов, связанные со значениями некоторых (но не всех) атрибутов.

Предикаты

предикат – это простое условие (терм) или логическая комбинация *простых условий*;
простое условие – это операция сравнения двух *выражений*;
выражение состоит из операций с константами и именами полей;
константа – это значение предопределенного типа, например целое число или строка.

Запросы. Обобщенный синтаксис.

A – атрибут; E – объект; V – значение атрибута; Rel – предикативное выражение.

Форма	Тип запроса	Пример
A(E) = ?	Обычный запрос атрибута. Каково значение атрибута A объекта E?	Заработок торгового агента № 271 за последний месяц
A(?) Rel V Rel: {=, ≠, <, >, >=, ⩽, ⩾}	Какие объекты имеют заданное значение атрибута. Запрос в инвертированный файл: Какой объект E имеет значение атрибута A, равное (неравное, меньше, больше, ...) V?	Кто из торговых агентов заработал больше 2000 долл. за последний месяц?
?(E) Rel V	Перечислить все атрибуты, имеющие заданный набор значений для данного объекта. Запрос менее простой: ?(E)=V — Какой атрибут или какие атрибуты объекта E имеют значение V?	За какие месяцы заработки торгового агента No 271 превысили 2000 долл?
?(E)=?	Запрос на получение всей информации о данном объекте. Запрос на значения всех атрибутов объекта E.	Сообщить всю хранимую информацию о торговом агенте No 271
A(?)=?	Перечислить значения данного атрибута для каждого объекта. Запрос на значения атрибута A для всех объектов.	Перечислить заработки за последний месяц каждого торгового агента
?(?) Rel V	Перечислить все атрибуты объектов, имеющие данное значение. Сложный запрос на все атрибуты всех объектов, имеющие значение V.	Для каждого торгового агента определить месяц, когда его заработок превышал 2000 долларов.

Индексирование

При выполнении запроса к таблице пользователя часто интересуют только некоторые записи в ней, например записи, имеющие определенное значение в некотором поле.

Индекс – это файл, помогающий движку базы данных быстро найти такие записи, не просматривая всю таблицу.

Наиболее распространенными способами реализации индексов являются:

- статическое хеширование;
- расширяемое хеширование;
- B-деревья.

Эффективность выполнения некоторых запросов может значительно повысить подходящая организация таблиц. Представим телефонный справочник – по сути, большая таблица, записи которой содержат имена, адреса и номера телефонов абонентов.

Фамилия Имя Отчество	Адрес	Тел. номер

Эта таблица отсортирована сначала по фамилиям, а затем по именам.

Предположим, что нам нужно узнать номер телефона конкретного человека.

Существенно ускорить поиск помогает тот факт, что записи отсортированы по имени.

Например, можно выполнить бинарный поиск, который в худшем случае потребует просмотреть log2N записей, где N – общее число записей в справочнике. Это очень быстро.

Например, если N = 1 000 000, то log2N < 20, то есть чтобы найти нужного человека в справочнике, содержащем номера миллиона человек, никогда не придется просматривать больше 20 записей.

Телефонный справочник отлично приспособлен для поиска абонента по имени, но не подходит для быстрого поиска, например по номеру телефона или по адресу.

Единственный способ получить эту информацию из телефонной книги – просмотреть каждую запись в ней, что потребует просмотреть в среднем N/2 записей.

Для эффективного поиска абонентов по номеру телефона нужен справочник, отсортированный по номерам телефонов (такие справочники еще называют «*обратными телефонными справочниками*»). Однако такой справочник удобен, только если известен номер телефона. Если необходимо найти в таком справочнике номер телефона конкретного абонента, то снова придется просмотреть каждую запись.

Этот пример наглядно иллюстрирует важное ограничение организации таблиц — таблицу можно организовать (упорядочить) только каким-то одним способом. Если необходимо, чтобы поиск выполнялся быстро и по номеру телефона и по имени абонента, потребуются две отдельные копии телефонной книги, каждая со своей организацией. А если понадобится возможность быстро находить номер телефона по известному адресу, потребуется третий экземпляр телефонного справочника, отсортированный по адресу. Это справедливо и в отношении таблиц в базе данных.

Чтобы иметь возможность эффективно находить в таблице записи с определенным значением некоторого поля, нужна версия таблицы, организованная по этому полю.

Механизмы баз данных это удовлетворяют эту потребность, поддерживая *индексы*.

Таблица может иметь один или несколько индексов, каждый из которых определен для отдельного поля. Каждый индекс действует подобно версии таблицы, организованной по соответствующему полю.

Индекс — это файл с индексными записями, содержащий одну индексную запись для каждой записи в соответствующей таблице. Каждая индексная запись имеет два поля, которые хранят значение индексируемого поля и значение первичного ключа соответствующей записи в таблице.

ID	Name	Address	Phone

→

Name	ID

Address	ID

Phone	ID

На поля в индексной записи можно условно ссылаться по именам `record_id` и `field_value`.

```
struct index_s {
    field_t   field_value;
    primkey_t record_id;
};
```

Движок БД организует записи в файле индекса по полю `field_value`.

Можно считать для простоты, что записи в индексе по полю `field_val` отсортированы. В этом случае для того, чтобы найти запись по значению `field_val`, нужно выполнить бинарный поиск и извлечь из поля `record_id` . первичный ключ искомой записи.

Аналогично можно найти *все записи* со значением `field_val`. Бинарный поиск выдаст `record_id` первой записи с атрибутом `field_val.` Поскольку в индексе все записи отсортированы по `field_val`, остается только последовательно читать следующие записи в индексе, пока получаемый `field_val` будет удовлетворять поставленному условию.

18, 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	3	5	6	9	9	10	13	18	18	**18**	18	32	37	?
b0							b1				b2				e0
b0							b1				b2	b4	b3		e0

Насколько эффективно такое применение индексов?

В отсутствие индексов лучшее, что можно предпринять при обработке любого запроса, – выполнить последовательный поиск в таблице.

Если атрибут в таблице равномерно распределен и имеет вид «A = константа», справедливо правило — полезность индекса для поля A в таблице пропорциональна количеству уникальных значений в этом поле.

Согласно этому правилу, индекс наиболее полезен, когда индексируемое поле является первичным ключом таблицы, потому что каждая запись имеет свое уникальное значение ключа.

И наоборот, индекс будет бесполезен, если число уникальных значений в поле A меньше количества записей в блоке чтения.

Но, есть нюансы (прямой доступ против последовательного).

Статические хеш-индексы

Статическое хеширование – это, пожалуй, самый простой способ реализации индексов.
Это не самая эффективная стратегия, но достаточно простая и понятная для реализации.

Статическое хеширование

Статический хеш-индекс использует фиксированное число N ячеек, пронумерованных от 0 до N–1. Индекс также использует хеш-функцию, отображающую значения в ячейки. По результатам хеширования поля `field_val` каждая индексная запись помещается в свою ячейку.

Статический хеш-индекс действует следующим образом:

- чтобы сохранить индексную запись, ее нужно поместить в ячейку, вычисленную хеш-функцией;
- чтобы найти индексную запись, нужно вычислить хеш-функцию ключа поиска и просмотреть соответствующую ячейку;
- чтобы удалить индексную запись, нужно сначала найти ее (как указано выше), а затем удалить из ячейки.

Стоимость поиска с помощью хеш-индекса ***обратно пропорциональна*** количеству ячеек.

Если индекс содержит B блоков (единиц чтения) и N ячеек, то на каждую ячейку будет приходиться около B/N блоков, поэтому для поиска в ячейке потребуется обратиться к B/N блоков.

N = 17, B = 8 ячейка полностью в одном блоке -> гарантированно одно чтение
N = 17, B = 20 ячейка располагается в двух блоках -> может быть два чтения

Расширяемое хеширование

Стоимость поиска при использовании индексов на основе статического хеширования обратно пропорциональна количеству ячеек – чем больше ячеек используется, тем меньше блоков в каждой из них.

Наиболее оптимально, когда каждую ячейку приходится ровно один блок.

Если бы размер индекса никогда не изменялся, то рассчитать это идеальное количество ячеек легко. Но на практике индексы растут по мере добавления новых записей в базу данных.

Если исходить из текущего размера индекса, то впоследствии, при его увеличении, каждая ячейка будет содержать несколько блоков индекса.

Если выбрать большее количество ячеек, ориентируясь на потребности в будущем, то пустые и почти пустые в данный момент ячейки будут напрасно расходовать значительный объем дискового пространства до тех пор, пока индекс не вырастет и их не заполнит.

Эту проблему решает стратегия, известная как *расширяемое хеширование*.

Суть этой стратегии заключается в использовании достаточно большого количества ячеек, чтобы гарантировать, что каждая ячейка никогда не будет содержаться более чем в одном блоке, или, что эквивалентно, в одном блоке будет содержаться несколько ячеек.

Поскольку в одном блоке могут располагаться несколько ячеек, это позволяет большому количеству ячеек совместно использовать меньшее количество блоков и тем самым не допустить напрасного расходования дискового пространства.

Совместное использование блоков ячейками обеспечивается с помощью двух файлов – файла ячеек и каталога ячеек.

Файл ячеек содержит блоки индекса, а каталог ячеек отображает ячейки в блоки.

Каталог можно рассматривать как массив целых чисел, по одному для каждой ячейки. Пусть это массив `Dir`. Тогда если индексная запись хешируется в ячейку b, то запись будет сохранена в блоке `Dir[b]` файла ячеек.

Допустим, что в блоке размещаются 3 ячейки, всего ячеек 8, хеш-функция h(x)=x mod 8, в индексе семь записей с идентификаторами 1, 2, 4, 5, 7, 8, 12.

Каталог ячеек: [0 1 2 1 0 1 2 1]

Файл ячеек: [(4, r4) (8, r8) (12, r12)] [(1, r1) (5, r5) (7, r7)] [(2, r2)]

Надо заметить, что этот подход не работает, когда имеется слишком много записей с одинаковым значением `field_val`. Поскольку эти записи всегда будут хешироваться в одну и ту же ячейку, стратегия хеширования никак не сможет распределить их по нескольким ячейкам. В этом случае ячейка будет занимать столько блоков, сколько потребуется для хранения этих записей.

Индексы на основе B-дерева

Предыдущие две стратегии индексирования были основаны на хешировании.

Индексы на основе B-дерева — подход на основе сортировки. Основная идея заключается в сортировке индексных записей по значениям `field_val`.

Индексный файл – это последовательность индексных записей с полями `field_val` и `record_id`. При работе с индексным файлом для нас важно иметь возможность как можно быстрее находить идентификаторы записей (`record_id`) по значениям индексированного поля (`field_val` .

Кроме поиска поддерживается вставка и удаление.

Низкоуровневые операции логического уровня в таблицах

```
TABLE LIST (ID, NAME, DATE, ...)
TABLE ITEM (ID, LIST_ID, PART_NO, NAME, PROVIDER, QUANT, ...)

PRIMKEY LIST:ID (ID)
PRIMKEY ITEM:ID (ID)

INDEX LIST:BY_NAME        (NAME)
INDEX LIST:BY_DATE        (DATE)
INDEX ITEM:BY_NAME        (NAME)
INDEX ITEM:BY_NAME_INLIST (LIST_ID, NAME)
INDEX ITEM:BY_PNO_INLIST  (LIST_ID, PART_NO)    # кандидат в PrimKey
```

Простейшие сериальные операции

```
SET(TABLE) – установка курсора (точки доступа) в «сыром» порядке записей
NEXT(TABLE) – просмотр таблицы в направлении от начала к концу
PREV(TABLE) – просмотр таблицы в направлении с конца к началу
INSERT(RECORD, TABLE) – вставка в таблицу без обновления индексов (APPEND)
```

Прямой доступ по первичному ключу

GET(TABLE, KEY) – получить строку (запись) из таблицы по значению первичного ключа
PUT(RECORD, TABLE, KEY) – обновить строку (запись) в таблице по значению первичного ключа

Добавление/удаление с обновлением индексов

ADD(RECORD, TABLE) – добавить строку (запись) в таблицу и обновить все индексы
DEL(**RECORD**, TABLE, KEY) – удалить строку (запись) из таблицы и обновить все индексы

Прямой доступ по ключу

GET(TABLE, KEY, KEY_VALUE) — получить строку по значению ключа
PUT(RECORD, TABLE, KEY, KEY_VALUE) — обновить строку по значению ключа
ADD(RECORD, TABLE) — добавить строку (запись) в таблицу и обновить все индексы
DEL(RECORD, TABLE, KEY) — удалить строку (запись) из таблицы и обновить все индексы

Сериальные операции в порядке индексов

SET(TABLE, KEY, KEY_VALUE) – установить курсор доступа в порядке индекса
NEXT(TABLE, KEY) – просмотр таблицы в направлении с начала к концу
PREV(TABLE, KEY) – просмотр таблицы в направлении с конца к началу