Основы технологий хранения баз данных в памяти

Лекция 2

План

1. Сжатие
2. Макет данных
3. Разбиение
4. Вставка
5. Обновление
6. Удаление
7. Только вставка

1. Compression
2. Data Layout
3. Partitioning
4. Insert
5. Update
6. Delete
7. Insert Only

1. Кодирование словарей

Поскольку память является новым узким местом ("бутылочным горлышком") в системе, требуется минимизировать доступ к ней.

С одной стороны, этого можно достичь доступом к меньшему числу столбцов (столбец-ориентированное устройство БД), так что только необходимые атрибуты (столбцы) запрашиваются при операциях. С другой стороны, уменьшение числа битов, используемых для представления данных, может снизить как потребление памяти, так и доступ к ней в разы.

Кодирование словаря создает базу для ряда других методов сжатия, которые могут быть применены поверх закодированных столбцов. Основной эффект кодирования словаря является то, что длинные значения, такие как тексты, представляются в виде коротких целочисленных значений.

Кодирование словаря является относительно простым. Это означает не только то, что его легко понять, но и его легко реализовать и при этом не необходимости использовать сложные многоуровневые процедуры, которые могли бы ограничить или уменьшить прирост производительности.

В данной теме мы обсудим различия между горизонтальным, ориентированным на строки макете, и макете столбчатой планировки. Введем такие понятия, как сжатие и разбиение. Основываясь на этом, получим пояснение внутренних шагов, выполняемых в базе данных для выполнения фундаментальных реляционных операций вставки, обновления и удаления. Как вывод, поясним фундаментальное отличие базы данных «в памяти» SanssouciDB от большинства других баз данных: вставка только подход. Следуя данной концепции, мы обходим несколько ошибок, касающиеся реляционной целостности и дополнительно получаем основу для создания меньшего зазора для функций, требующих времени

1. Сжатие

Методы сжатия

- Тяжелый вес в сравнении методами легкого веса
- Фокус на легких весовых методах баз данных
- Для вектора атрибута
  - префикс кодирование
  - кодирование длин серий
  - кластерное кодирование
  - разреженное кодирование
  - косвенное кодирование
- Для словаря
  - сжатие Дельта для строк
  - другие типы данных хранятся в отсортированных массивах

Пример

recID	fname	lname	gender	country	city	birthday	2nd_nationality
0	Martin	Albrecht	m	GER	Berlin	08-05-1955	n/a
1	Michael	Berg	m	GER	Berlin	03-05-1970	n/a
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968	n/a
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992	US
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977	n/a
5	Martin	Schulz	m	GER	Mainz	06-04-1980	GER
6	Sushi	Pao	f	CN	Peking	09-12-1954	n/a
7	Chen	Su Wong	m	CN	Shanghai	27-06-1999	n/a
...	...	...	...	...	...	...	...

- 200 countries = 8 bit
- 1 million cities = 20 bit
- 100 different 2nd nationalities = 7 bit
- 5 million first names = 23 bit

1. Префикс - кодирование

В реальных базах данных, мы часто сталкиваемся с тем, что столбец содержит одно доминирующее значение, а остальные имеют низкую избыточность. В этом случае мы очень часто будем хранить одинаковые значения в несжатом формате. Префикс кодирование является самым простым способом более эффективно обрабатывать этот случай. Чтобы применить кодировку префикс, наборы данных должны быть отсортированы по колонке с преимущественным значением и вектор атрибута должен начинаться с преимущественного значения.

Сжатие колонки предусматривает, что преобладающее значение не должно храниться в явном виде каждый раз, когда оно встречается. Это достигается за счет сохранения количества вхождений преобладающего значения и наличия одного экземпляра самого значения в векторе атрибута. Таким образом, вектор атрибут префикс кодирования содержит следующую информацию:

- число вхождений преобладающего значения
- valueID преобладающего значения из словаря
- valueIDs остальных значений.

Префикс-кодирование:

- используется, если столбец начинается с длинной последовательности одного и того же значения;
- лишь одно преобладающее значение имеется в столбце, а остальные значения в основном уникальны или имеют низкое дублирование.

Пример: колонка стран, таблица отсортирована по численности населения страны.

Дан вектор атрибута столбца стран из таблицы населения мира, который отсортирован по численности населения стран в порядке убывания. Таким образом, 1,4 млрд граждан Китая перечислены вначале, затем граждане Индии и так далее. ValueID для Китая, который принял значение 37 в словаре (см. 7.1a), хранится 1,4 миллиарда раз в начале вектора атрибута в несжатом формате. В сжатом формате valueID 37 будет записано только один раз, а затем остальные valueIDs для других стран, как и в несжатом. Число вхождений “1,4 миллиарда'' для Китая будет храниться в явном виде. Рисунок 7.1b представляет примеры несжатых и сжатых векторов атрибутов.

Следующий расчет показывает степень сжатия. Прежде всего, количество бит, необходимых для хранения всех 200 стран рассчитывается как $\log_2(200)$, что дает 8 бит.

Без сжатия вектор атрибут сохраняет 8 бит для каждого valueID 8 миллиардов раз:

8 миллиардов * 8 бит ~ 8 миллиардов байт ~ 7,45 ГБ

Если применить к столбцу стран префикс-кодирование, valueID для Китая сохраняется только один раз в 8 битах вместо 1,4 миллиарда раз по 8 бит. Дополнительное 31-битовое поле добавляется для хранения количества вхождений :

$$\log_2 (1{,}4 \text{ млрд}) = 31 \text{ бит}$$

Следовательно, вместо того, чтобы хранить 1,4 миллиарда раз 8 бит, только 31 бит + 8 бит = 39 бит действительно

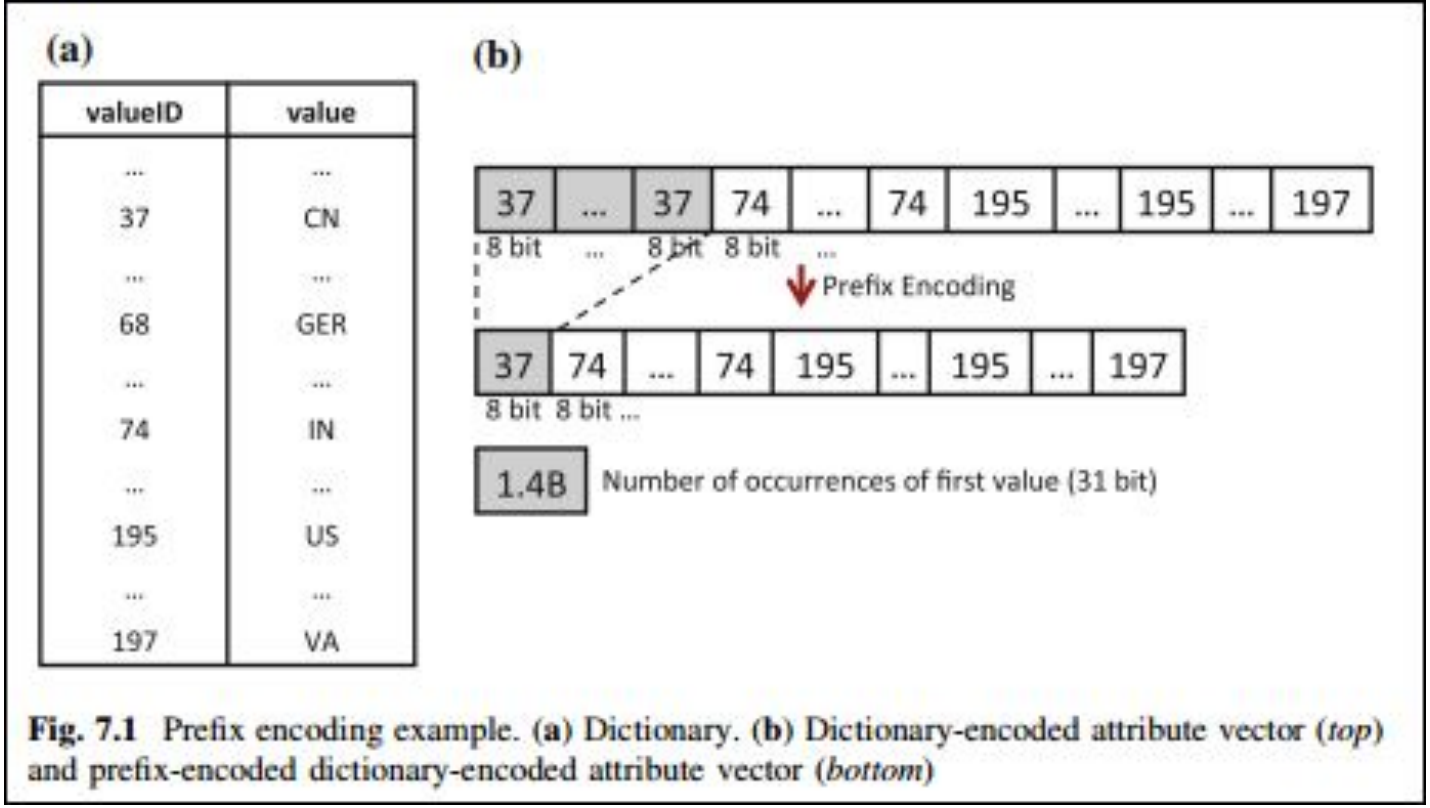
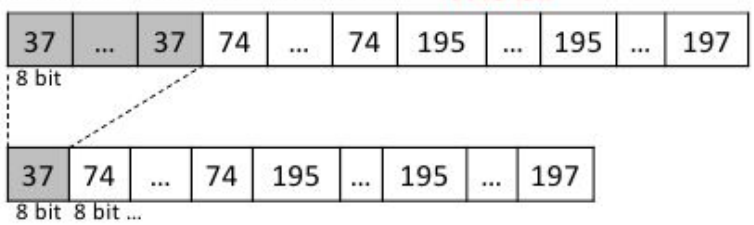


Fig. 7.1 Prefix encoding example. (**a**) Dictionary. (**b**) Dictionary-encoded attribute vector (*top*) and prefix-encoded dictionary-encoded attribute vector (*bottom*)

Dictionary encoded attribute vector **7.45 GB**



1.4B
# of occurrences of first valueID (64 bit)

Using China saves 1.4 billion × 8 bit:
(8 billion – 1.4 billion) × 8 bit + 64 bit + 8 bit
≈ 6.15 GB

Dictionary

valueID	value
...	...
37	CN
...	...
68	GER
...	...
74	IN
...	...
195	US
...	...
197	VA

Direct access!

Таким образом, 1,3 ГБ, то есть 17% пространства в памяти экономится. Еще одно преимущество кодирования префикс - это прямой доступ с вычислением номера строки. Например, чтобы найти всех мужчин - граждан Китая, система управления базой данных может определить, что только последовательности с номерами строк от 1 до 1,4 млрд следует рассматривать, а затем отфильтровать по гендерному значения.

Несмотря на то, что мы видим, что мы сократили необходимый объем оперативной памяти, очевидно, что мы по-прежнему храним много избыточной информации для всех других стран.

Поэтому рассмотрим метод кодирования длин серий (Run-Length Encoding).

2. Кодирование длин серий (Run-Length encoded)

Кодирование длин серий представляет собой метод сжатия, который работает лучше всего, если вектор атрибута состоит из нескольких различных значений с большим числом вхождений. Для получения максимальной степени сжатия, столбец должен быть отсортирован, так чтобы все одинаковые значения были расположены вместе. При кодировании длин серий, значение последовательности с одинаковым значением заменяются одним экземпляром значения и

(А) либо его количеством вхождений

(Б) либо его исходным положением в виде смещения.

Пример: колонка стран, таблица отсортирована по численности населения страны

На рисунке 7.2 приведен пример кодирования длин серий с использованием стартовой позиции в виде смещения. Сохранение начальной позиции ускоряет доступ. Адрес конкретного значения может быть прочитан в столбце непосредственно, вместо вычисления его с самого начала столбца, таким образом, обеспечивая прямой доступ.

Применительно к нашему примеру столбца стран отсортированного по населению, вместо того, чтобы хранить все 8 миллиардов значений (7,45 ГБ), мы сохраняем два вектора:

- один со всеми различными значениями: 200 раз по 8 бит
- другой со стартовой позицией: 200 раз по 33 бит (33 бита необходимы для хранения смещения по 8 млрд ($\log_2$(8 млрд) = 33 бит). Дополнительное 33-битовое поле в конце этого вектора хранит число вхождений для последнего значения (последней страны).

Следовательно, размер вектора атрибута может быть значительно уменьшен примерно до 1 Кбайт без потери информации:

200 * (33 бит + 8 бит) + 33 бит ~ 1 КВ

Если хранить вектор числа вхождений (вариант а), одно поле из 33 бит может быть сэкономлено, но возникнет потеря возможности прямого доступа с двоичным поиском. Потеря прямого доступа приводит к увеличению продолжительности отклика, что нежелательно для управления корпоративными данными.

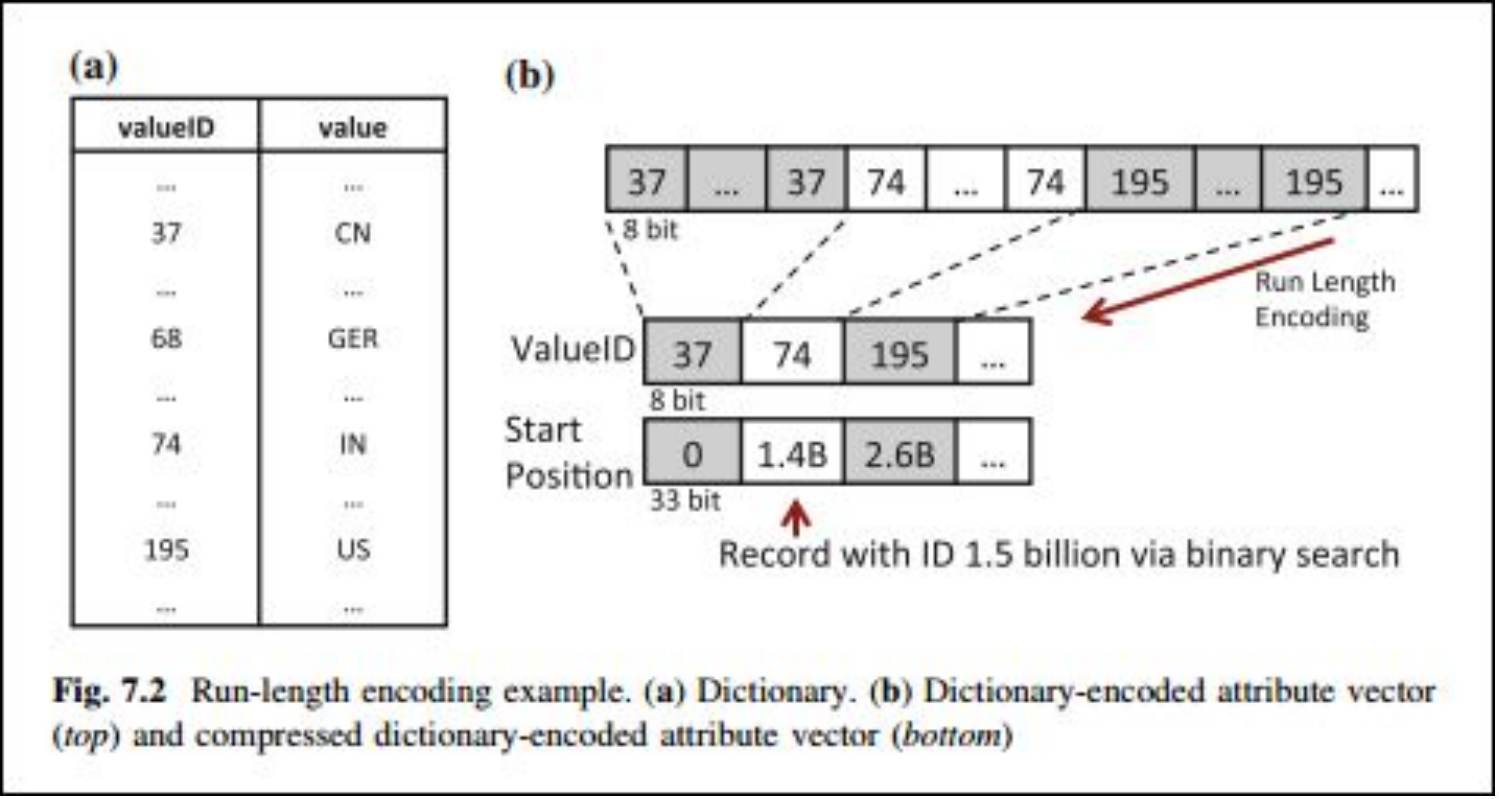
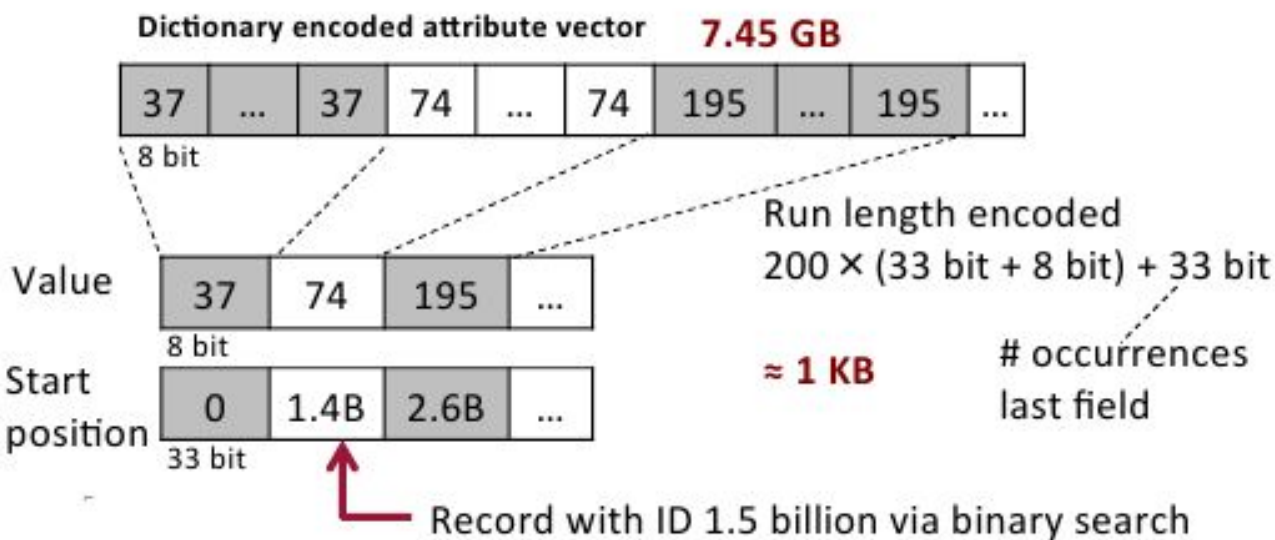


Fig. 7.2 Run-length encoding example. (**a**) Dictionary. (**b**) Dictionary-encoded attribute vector (*top*) and compressed dictionary-encoded attribute vector (*bottom*)

Direct access!

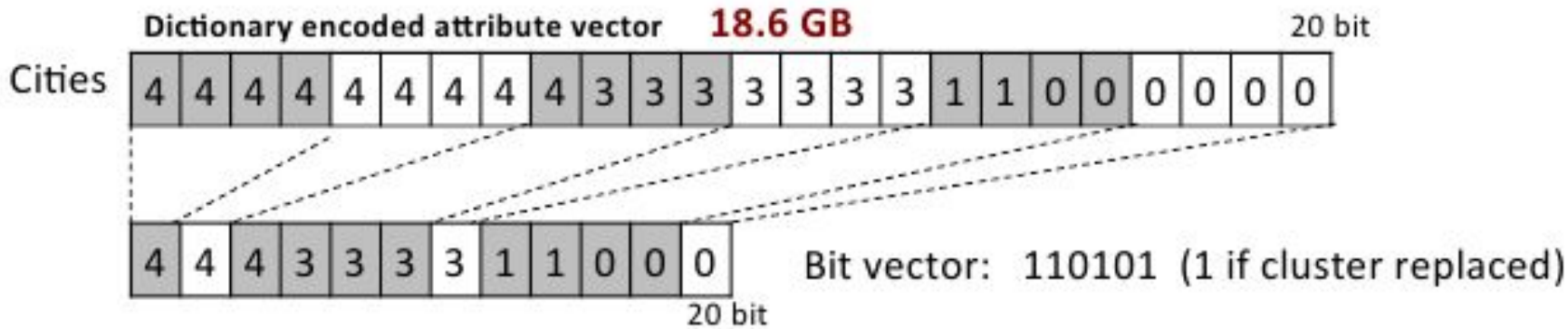


Dictionary

valueID	value
...	...
37	CN
...	...
68	GER
...	...
74	IN
...	...
195	US
...	...

Кластерное кодирование

- вектор атрибутов разбивается на N блоков фиксированного размера (обычно 1024)
- если кластер содержит только одно значение, оно заменено единственным возникновения этого значения
- бит вектор длины N указывает, какие кластеры были заменены одним



No direct access! Compute position via bit vector.

Пример: колонка город таблицы, отсортированной по странам, городам

Размер кластера: 1024 элементов → 7.8 млн блоков

В худшем случае предположение: 1 несжатый блок на город

Несжатые блоки: 1 млн x 1024 x 20 бит 2 441 Мб

Сжатые блоки: (7.8 – 1) млн x 20 бит + 16 Мб

Бит вектор: 7.8 млн x 1 бит + 1 Мб ~ 2.4 Гб

Вычисление позиции с помощью бит вектора/

Разреженное кодирование

• Удалить значение V, который появляется чаще всего

Бит вектор указывает, из каких позиций V удаляли из исходной последовательности

Пример: 2-я национальность, колонка независит от порядка сортировки таблицы

Предположение: 99% людей не имеют 2-й национальности

Бит вектор 8 млрд x 1 бит 954 Мб

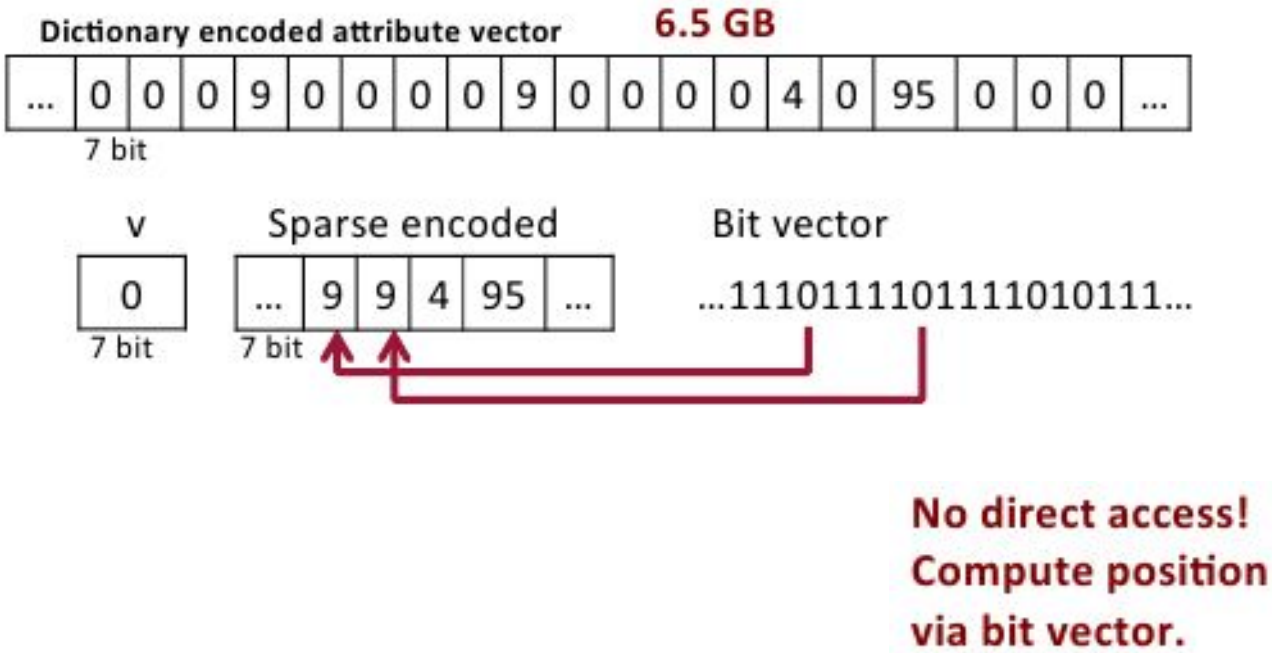
Вектор атрибут разреженного кодирования

8 млн x 7 бит +67 Мб

ıт ~1Гб

Dictionary

valueID	value
0	n/a
...	...
4	CO
...	...
9	GER
...	...
95	US
...	...



Косвенное кодирование

Последовательность разбивается на N блоков размером S (обычно 1024)

- Если блок содержит только несколько различных значений, используется дополнительный словарь для кодирования значений в этом блоке
- Дополнительно: ссылки на новые словари + блоки, которые имеют словарь

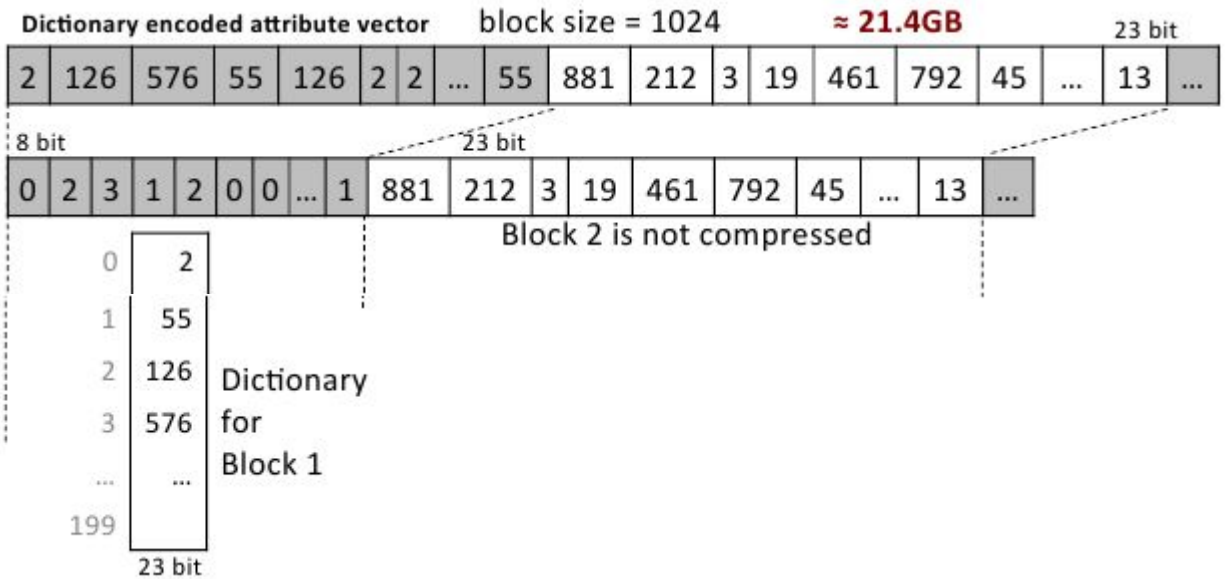
Пример: колонка Fname, таблица отсортировано по странам

Предположение: каждый набор 1024 человек в той же стране в среднем содержит 200 различных имен

Словари: (200 x 23 бит + 64 бита) x #block 4.2 Гб

Адрес словаря 7.8 млн

Вектор сжатия: 8 млрд x 8 бит 7.6 Гб ~ 11.8 Гб



Direct access!

Дельта кодирование для словаря

- Для строковых отсортированных значений
- Блок-мудрое сжатие (как правило, 16 строк на блок)

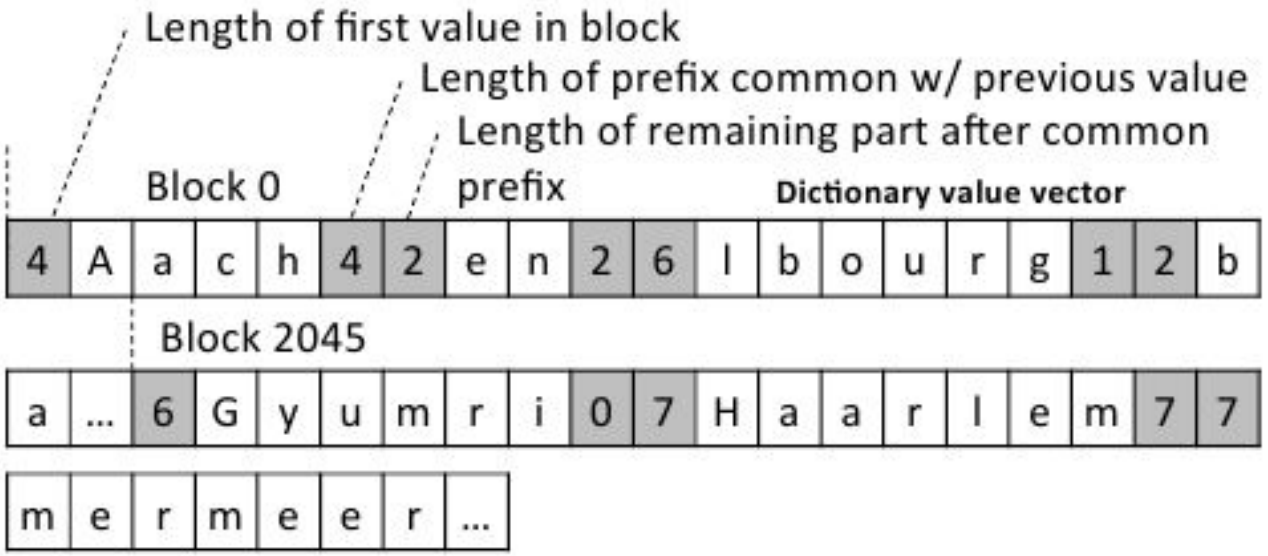
Dictionary:

1 million cities à 49 byte

≈ 46.7 MB

Dictionary

	valueID	value
Block 0	0	Aach
	1	Aachen
	2	Aalbourg
	3	Aba
	...	...
Block 2045	32720	Gyumri
	32721	Haarlem
	32722	Haarlemmer-meer
	...	...



Assumptions: average length of city names 7
average overlap of 3 letters

For the "numbers": longest city name 49 letters = 6 bit

Size of block × #blocks

(encoding numbers + 1st city + 15 other cities) × #blocks

((1+15 × 2) × 6 bit + 7 × 1 byte + 15 × (7-3) × 1 byte) × 62500

≈ 5.4 MB

Что необходимо иметь в виду, что большинство методов сжатия требуют отсортированных наборов для задействования всего своего потенциала, но таблицы базы данных могут быть отсортированы только по одному столбцу или каскадно, если не используются никакие другие вспомогательные структуры данных.