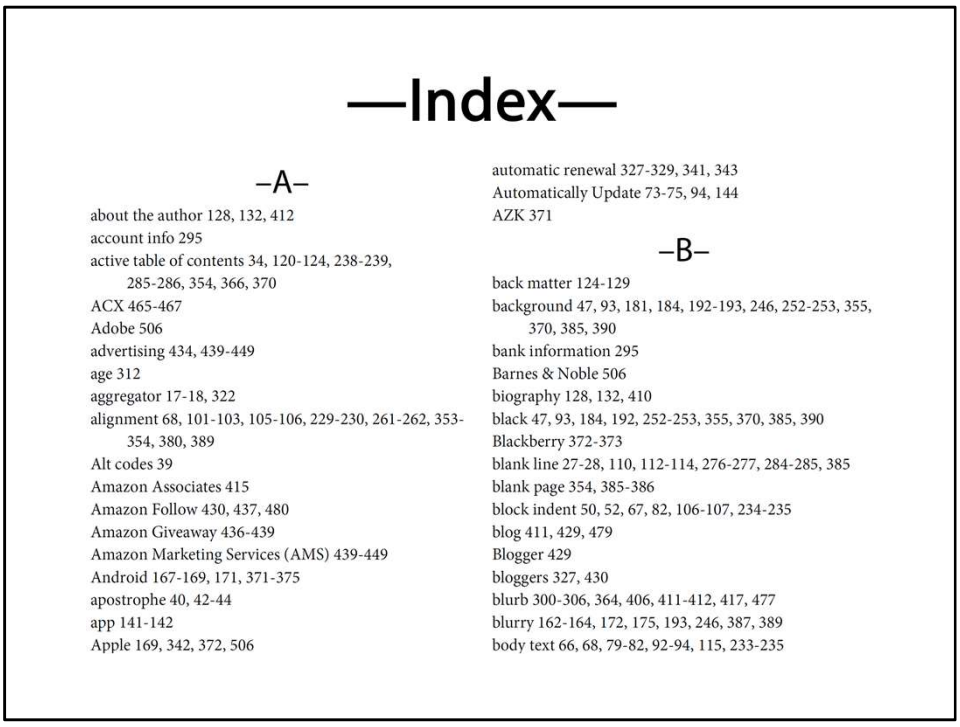
Database 101: Index

Sam Wong 2019-12-02 (CC-SA 3.0)



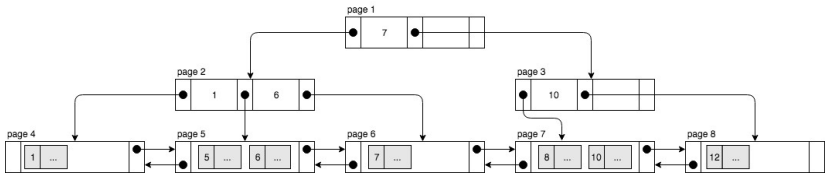
—Index—

–A–

–B–

在現實生活中，書本上就會有Index。Database的Index其實是同樣的概念。

InnoDB: Clustered Index / 聚集索引 (Primary Key / 主鍵)

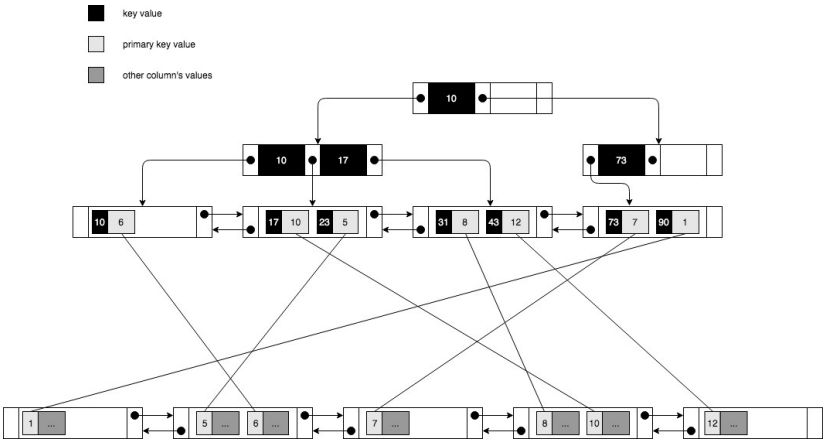


Key透過B+ Tree儲存，能讓DB Engine快速以範圍作搜索，或以Key作排序。
InnoDB的主資料表是按Primary Key排列存放。

註: 每個Page在硬碟上不是按順序存放

(這頁看不懂不需要深究…隨便看看就好)

InnoDB: Secondary Index / 二級索引 (PostgreSQL: 所有索引同一結構)



InnoDB: 其他普通的Index (Secondary Index)，是指向Clustered Index。
PostgreSQL: 資料沒有以Clustered Index方式儲放。Primary Key其實和一般Unique Index一樣。

(這頁看不懂不需要深究…隨便看看就好)

用途

- 搜索定位
- 分組、排序 (Group By, Order By)
- 行數預測 – 影響Query Plan
  - Join順序
  - 讀取方式
- 直接提供資料

頭兩點很明顯，就像之前所說的。
後兩點在下面說明。

缺點

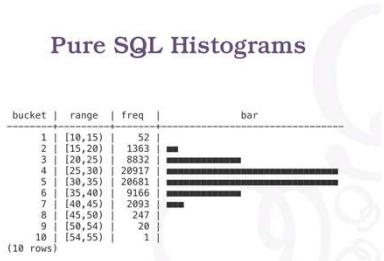
- 時間 (更新/刪除會更內)
- 儲存空間

搜索性能和更新性能之間要取平衡

行數預測 – 影響Query Plan

- Cardinality / 勢
  - 決定JOIN的順序
- Histogram / 直方圖 (限 PostgreSQL)
  - 更準確的行數預測
  - 決定查全表還是查索引
- 多字段相關性 (限 PostgreSQL 10+)

強制刷統計資料的方法 –
SQL: Analyze Table



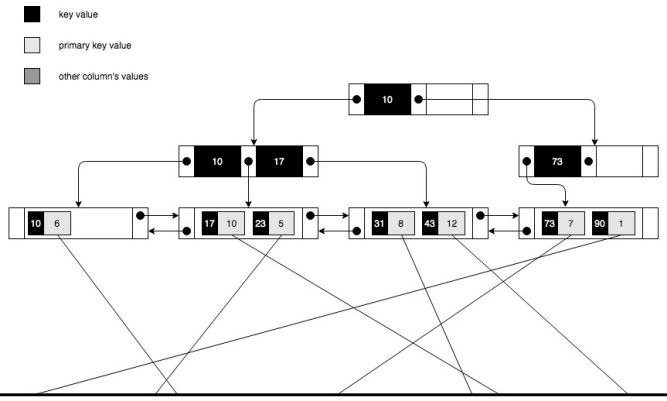
決定JOIN的順序 - Engine會希望由越小的Table/Resultset先join起。
行數也影響要動用的Algorithm – Inner Join、Merge Join、Hash Join或索性全表搜索之類

理論上DB會自動定期更新 (例如在改動了10%資料之後)，但如果使用中出現奇怪的Query Plan，可以用Analyze Table強制刷一次統計資料試試。

各DB都支持查詢現有統計資料，具體語句請參考各自的手冊。

直接提供資料

- Covering Index
- MIN()/MAX()最佳化



若SELECT需要的字段都在Index裏儲存了，就不需要讀取主資料的部份。

註: 詳細還有一些MVCC問題但不必拘於小節～

善用Explain

```
  select_type: SIMPLE
        table: SalesOrder
   partitions: NULL
         type: ref
possible_keys: statusPaymentTimeIndex
          key: statusPaymentTimeIndex
      key_len: 187
          ref: const,const
         rows: 468
     filtered: 100.00
        Extra: Using index
```

總之有沒有用到Index，最好就是用Explain看一下:
key代表用了那個index。
rows是預測的行數，如果偏離太多 – 例如應該是幾百但出來幾十萬，就是因為用不上。
Extra的都是寫什麼: 詳細看手冊

常見陷阱

問

```
CREATE TABLE Jobs (
  id int(11) NOT NULL AUTO_INCREMENT,
  type varchar(50) NOT NULL,
  isProcessed tinyint(1) NOT NULL,
  payload text(1000) NOT NULL,
  PRIMARY KEY (id),
  KEY isProcessedIndex (isProcessed, type)
)
```

```
SELECT count(*)
FROM Jobs
WHERE type='FOOBAR'
```

解

```
CREATE TABLE Jobs (
  id int(11) NOT NULL AUTO_INCREMENT,
  type varchar(50) NOT NULL,
  isProcessed tinyint(1) NOT NULL,
  payload text(1000) NOT NULL,
  PRIMARY KEY (id),
  KEY isProcessedIndex (isProcessed, type)
)
```

```
SELECT count(*)
FROM Jobs
WHERE
isProccesed=0
AND
type='FOOBAR’
```

Composite Index必須要由左至右用

書本實物比喻: 要你是找ilike ”A%”當然可以查Index，但若是找ilike “_A%”等同整個Index都翻一篇，那等於沒有…

問

某產品於1月1日首個変更日志

```
CREATE TABLE StockLog (
  id int(11) NOT NULL AUTO_INCREMENT,
  productId int(11) DEFAULT NULL,
  quantityFrom int(11) NOT NULL,
  quantityTo int(11) NOT NULL,
  created datetime(11) NOT NULL,
  PRIMARY KEY (id),
  KEY createdIndex (created),
  KEY productCreatedIndex (productId,created)
)
```

```
SELECT *
FROM StockLog
WHERE
created>'2019-01-01’
AND
productId=46709394
ORDER BY created
LIMIT 1
```

解

```
CREATE TABLE StockLog (
  id int(11) NOT NULL AUTO_INCREMENT,
  productId int(11) DEFAULT NULL,
  quantityFrom int(11) NOT NULL,
  quantityTo int(11) NOT NULL,
  created datetime(11) NOT NULL,
  PRIMARY KEY (id),
  KEY createdIndex (created),
  KEY productCreatedIndex (productId,created)
)
```

```
SELECT *
FROM StockLog
WHERE
created>'2019-01-01’
AND
productId=46709394
ORDER BY
productId, created
LIMIT 1
```

查出來的結果是一樣的，因為WHERE condition其實限死了。
但DB Engine就是這麼笨，所以要明確告訴它WHERE和ORDER都可以用上productCreatedIndex才行。

問

沒有這樣的業務場景but anyway也是出見過的bug…

```
CREATE TABLE StockLog (
  id int(11) NOT NULL AUTO_INCREMENT,
  productId int(11) DEFAULT NULL,
  quantityFrom int(11) NOT NULL,
  quantityTo int(11) NOT NULL,
  created datetime(11) NOT NULL,
  PRIMARY KEY (id),
  KEY createdIndex (created),
  KEY productCreatedIndex (productId,created)
)
```

```
SELECT *
FROM StockPlatformLog
ORDER BY
  productId DESC,
  created ASC
LIMIT 1
```

解

```
CREATE TABLE StockLog (
  id int(11) NOT NULL AUTO_INCREMENT,
  productId int(11) DEFAULT NULL,
  quantityFrom int(11) NOT NULL,
  quantityTo int(11) NOT NULL,
  created datetime(11) NOT NULL,
  PRIMARY KEY (id),
  KEY createdIndex (created),
  KEY productCreatedIndex (productId,created)
)
```

```
SELECT *
FROM StockPlatformLog
ORDER BY
  productId DESC,
  created ASC
LIMIT 1
```

Composite Index中某一個字段倒序時是用不了Index

如果只是order by created DESC，或order by productId DESC, created DESC倒時可以用Index – DB Engine並未至於那麼笨，它能夠反過來讀 (但會有性能損耗)。
如果都肯定大部份是反過來排的話，直接建一個DESC Index好了。

例子中的Composite Index只是某一個字段倒序，把Index反過來讀也是解決不了問題，所以等同沒有合適的Index。

問

```
CREATE TABLE SalesOrder (
  id int(11) NOT NULL AUTO_INCREMENT,
  platformId varchar(60) DEFAULT NULL,
  status int(11) NOT NULL,
  paymentTime datetime(11) NOT NULL,
  payload text(1000) NOT NULL,
  PRIMARY KEY (id),
  KEY statusIndex (status,platformId,paymentTime)
)
```

```
SELECT count(*)
FROM SalesOrder
WHERE
status=1
AND
platformId=100070
```

解

```
CREATE TABLE SalesOrder (
  id int(11) NOT NULL AUTO_INCREMENT,
  platformId varchar(60) DEFAULT NULL,
  status int(11) NOT NULL,
  paymentTime datetime(11) NOT NULL,
  payload text(1000) NOT NULL,
  PRIMARY KEY (id),
  KEY statusIndex (status,platformId,paymentTime)
)
```

```
SELECT count(*)
FROM SalesOrder
WHERE
status=1
AND
platformId='100070'
```

字符串字段要用字符串查。(但數字字段的可以用字符串查)

PostgreSQL直接安全地爆錯。

MySQL會默默執行，但會發現它很慢。
因為只是platformId用不上但status用上了，所以錯誤會正確的例如果Explain來看的話，”key”都會顯示有用到statusIndex，只有看rows才會發現問題。

為什麼呢 - 因為”100070”, “00100070”, “100070.00”, “100070E+0”等無限種可能的字符串，若類型轉成數字都能等於100070。所以不能從數字100070這個條件去推敲索引的範圍。
若status = ‘1’這樣倒是可以，因為’1’轉成數字都只有一個可能性。
但最好還是不要以身犯險，用正確的類型比較好。

書本實物比喻: Index是不區大小寫去編寫，A/a/á/â/ã/ä/å都會歸屬同一個順序，如果明確要找大寫like “A%”，那index的會變的不太用得上。

在SQL中，不一樣collation (charset和case sensitivity)的字符串索引也有同樣的問題，如果不一樣的話也會導致join的時候走不了索引。

問

某監控程式想看看系統沒有處理的Jobs

```
CREATE TABLE Jobs (
  id int(11) NOT NULL AUTO_INCREMENT,
  type varchar(50) NOT NULL,
  isProcessed tinyint(1) NOT NULL,
  payload text(1000) NOT NULL,
  PRIMARY KEY (id),
  KEY entityIdIndex (entityId),
  KEY isProcessedIndex (isProcessed, type)
)
```

```
SELECT * FROM Jobs
WHERE
isProcessed=0
ORDER BY id
LIMIT 3
```

解

```
CREATE TABLE Jobs (
  id int(11) NOT NULL AUTO_INCREMENT,
  type varchar(50) NOT NULL,
  isProcessed tinyint(1) NOT NULL,
  payload text(1000) NOT NULL,
  PRIMARY KEY (id),
  KEY isProcessedIndex (isProcessed, type)
)
```

```
SELECT * FROM Jobs
WHERE
isProcessed=0
LIMIT 3
```

因為不能同時用多個索引 (SQL也好書本也好)，原SQL會讓DB Engine很尷尬。
重新考慮業務上是否需要，移除不必要的排序。

Limit 3提示也只是在報警中除隨抓一些ID做例子，Order By本意是想方便排查問題 - 越老的Jobs越未有處理越是有問題。
但如果真的滯後，有上千上萬個Jobs未完成的話，這個時候再讓DB排序會雪上加霜。
有時候想一想，排序是可以犧牲掉。

如果業務去不掉，那起碼要加合適的Index。