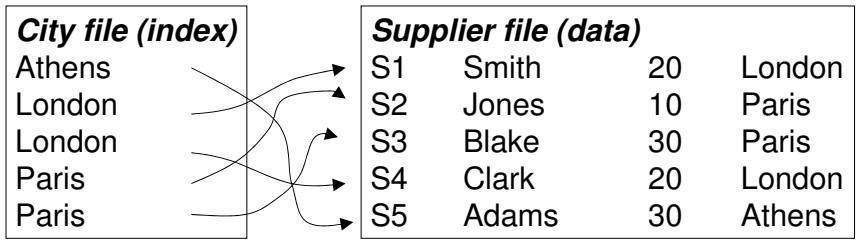
Indexing

Basic Concept

Supplier file (data)			
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Basic Concept

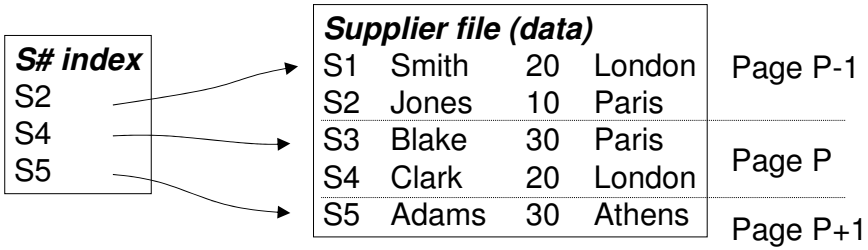


Index

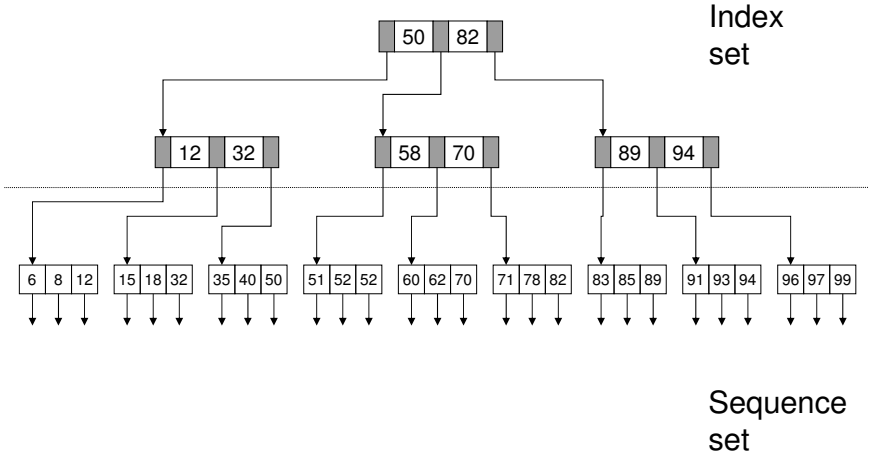
- ***Advantage***
  - Speed up Retrieval

- ***Disadvantage***
  - Slow down update

Nondense index



B-trees



Hashing

- Hashing-addressing (not hash indexing)
- Hash function
- Hash address

Hashing Example

Hash address = remainder of (part no / 13)

Part no.	S100	S200	S300	S400	S500
remainder	9	5	1	10	6

Hashing

Advantage

- No physical stored for index
- Position was found by computation
- Fast

Disadvantage

- Collision

ORACLE

Index คืออะไร?

- เป็นหนึ่งใน *schema object*
- ถูกใช้โดย *Oracle Server* เพื่อที่จะทำการ *retrieve* ข้อมูลได้เร็วขึ้น
- สามารถลด *disk I/O* เพราะสามารถเข้าถึงข้อมูลโดยใช้เส้นทางที่รวดเร็ว
- เป็นอิสระจาก *table* ที่ถูก *index*
- จะถูกเรียกใช้และบำรุงดูแลรักษาโดย *Oracle server*

Oracle Support

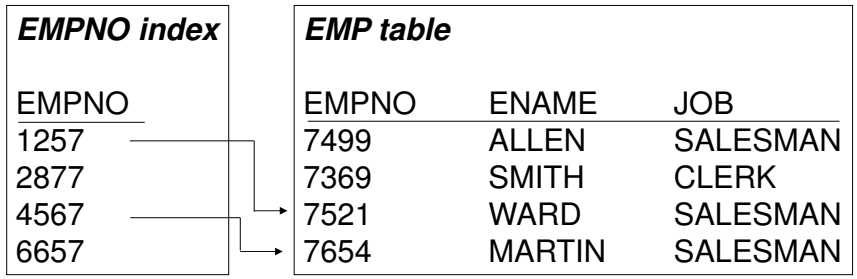
- B tree
- Hashing
- BIT MAP
- REVERSE KEY INDEX

Bit Map

S#	Color
S1	White
S2	Red
S3	White
S4	Green
S5	Brown
S6	Brown

Index	
White	101000
Red	010000
Green	000100
Brown	000011

Reverse key index



Reverse key index

- ในการบันทึกข้อมูลลง *database* โดยข้อมูลเรียงกันจากน้อยไปหามาก อาจจะเกิดคอขวดในการ *update index* เพราะจะ *update* ในที่เดียวกันใน *index tree*
- *Reverse key index* จะทำการกลับค่าใน *column* ที่ทำการ *index* จึงทำให้ การ *update index* กระจายทั่วๆ ไป
- *Reverse key index* ไม่สามารถค้นในลักษณะ *range searches*

เปรียบเทียบ *B-Tree* และ *Bitmap Indexes*

B-tree

- เหมาะกับ *Columns* ที่มีค่าใน *column* แตกต่างกันมาก
- การ *update* ค่าใน *columns* จะไม่แพง
- ไม่เหมาะกับ *query* ที่มีการใช้ *OR*
- เหมาะกับงาน *OLTP*

Bitmap

- เหมาะกับ *Columns* ที่มีค่าใน *column* แตกต่างกันน้อย
- การ *update* ค่าใน *columns* จะแพงมาก
- เหมาะกับ *query* ที่มีการใช้ *OR*
- เหมาะกับงาน *DSS*

INDEX ถูกสร้างได้อย่างไร *(Oracle)*

- *Automatically: index* จะถูกสร้างโดยอัตโนมัติเมื่อเราสั่ง *PRIMARY KEY* หรือ *UNIQUE constraint* ใน *table definition*
- *Manually: users* สามารถสั่งสร้าง *index* ได้เองเพื่อเพิ่มความเร็วในการเข้าถึงข้อมูล

ตัวอย่างคำสั่งสร้าง *B-Tree Indexes*

- *CREATE INDEX fname_idx ON emp(firstname)*

ตัวอย่างคำสั่งสร้าง *Reverse Key Indexes*

- *CREATE INDEX ordno_idx ON order(ordno) REVERSE*

ตัวอย่างคำสั่งสร้าง *Bitmap Indexes*

- *CREATE BITMAP INDEX city_idx ON emp(city)*

เมื่อไหร่ที่ควรสร้าง *INDEX*

- *Column* นั้นถูกใช้บ่อยใน *WHERE clause* หรือ *JOIN condition*
- *Column* มีค่าที่กว้างมาก *(wide range of value)*
- *Column* มีค่า *null* มาก
- เมื่อ 2 *columns* หรือมากกว่า ถูกใช้บ่อยใน *WHERE clause* หรือ *JOIN condition*
- *Table* มีขนาดใหญ่ และการ *queries* ต้องการแค่ 2-4% ของจำนวน *rows* ทั้งหมด

เมื่อไหร่ที่ไม่ควร สร้าง *INDEX*

- *Table* มีขนาดเล็ก
- *Columns* นั้นๆ ไม่ค่อยได้ถูกใช้ในการ *query*
- *Queries* ที่ได้ผลลัพธ์มากกว่า 2-4% ของจำนวน *rows* ทั้งหมด
- *Table* ที่ถูก *update*บ่อย

Summary

- *Index* ช่วยให้ *retrieve* เร็วขึ้นในขณะที่ทำให้การ *update* ช้าลง
- เลือก *index* ให้เหมาะสมกับงาน
- เพื่อเพิ่มประสิทธิภาพเราอาจจะสั่งให้หยุดการ *index* ชั่วขณะ เช่นตอน *import data* แล้วค่อยสั่ง *rebuild index*