

# إدارة الملفات File Management

## الدرس الأول

### مقدمة :

لبناء نظم برمجية ناجحة وقوية نحتاج لفهم كيفية بناء البيانات وتخزينها والوصول إليها ومعالجتها بصورة فعالة وبالكفاءة المطلوبة.

تهتم مادة ادارة الملفات بدراسة طرق تنظيم و تخزين البيانات وكذلك دراسة وسائط التخزين المختلفة وخصائص ومميزات كل نوع وطريقة استخدامه .

وبما ان البيانات تحفظ في صورة ملفات فان هذه المادة تهتم ايضا بتنظيم هذه الملفات داخلياً حيث تعنى بتركيبها وبنيتها وطرق الوصول الى محتوياتها وطرق البحث والفهرسة مما يؤدي لخلق أنواع عديدة من الملفات .

أما تنظيمها خارجياً فيعني وضع كل نوع في وسط التخزين المناسب له . ولأن سرعة الوصول الى البيانات الموجودة بالقرص او الذاكرة الثانوية تعتبر أكثر ببطئاً من سرعة الوصول الى البيانات في الذاكرة الرئيسية نجد ان مادة ادارة وتنظيم الملفات تهتم ايضاً بكيفية تحسين سرعة الوصول الى البيانات .

ويمكننا أن نجمل أهداف إدارة وتنظيم الملفات في الآتي :

- 1- سرعة الوصول الى المعلومات داخل الملف .
- 2- الإستخدام الأمثل لوسائط التخزين الخارجي .
- 3- سهولة عملية التخزين للبيانات .

مقدمة عن وسائط التخزين الفيزيائية

### Overview of physical storage media

يتم تخزين البيانات فيزيائياً في وسيط تخزين storage medium حتى تتمكن البرامج من التعامل مع هذه البيانات بالإسترجاع والتعديل، وعموماً يركز تصنيف وسائط التخزين على ثلاث عوامل أساسية :

- 1- سرعة الوصول للبيانات الموجودة بها .

The speed of which data can be accessed

- 2- التكلفة

the cost per unit of data to buy the medium

- 3- الإعتدالية او العمر الافتراضي the medium's reliability .

وسائط التخزين هذه يمكن تقسيمها لنوعين أساسيين هما :

- 1- وسائط التخزين الأولية primary storage :-

وهي تشمل وسائط التخزين التي تتعامل مباشرة مع المعالج مثل الذاكرة الرئيسية Main Memory والـ Cache Memory وهذه الوسائط تمتاز بسرعتها العالية ولكن حجمها صغير مقارنة بالذاكرة الثانوية .

## 2. وسائط التخزين الثانوية secondary storage :-

وهذا النوع يضم الأقراص الممغنطة Magnetic Disks والأقراص الضوئية optical Disk والأشرطة الممغنطة Magnetic Tapes وتتميز بسعاتها التخزينية الكبيرة وتكلفتها القليلة نسبياً ، ولكن سرعتها بطيئة مقارنة بالوسائط الأولية .  
البيانات المخزنة بوسائط التخزين الثانوية لا يمكن للمعالج أن يتعامل معها مباشرة ولكن يجب أولاً أن تنتقل إلى وسيط التخزين الأولى .

## أنواع الذاكرة ووسائط التخزين Types of storage media

يمكن تنظيم أنواع وسائط التخزين المختلفة في شكل هرمي بناء على سرعتها وتكلفتها وكلما انحدروا من قمة الهرم إلى أسفل تقل السرعة والتكلفة.

الـ Primary Storage وهي أسرع أنواع وسائط التخزين - مثل الـ cache والـ Main memory - وتوجد في قمة الهرم . وفي المستوى الثاني من الهرم توجد الـ secondary storage مثل الـ Magnetic disks وفي أدنى مستوى من الهرم تأتي الـ Tertiary storage مثل الـ magnetic tape والـ optical disk jukebox Epps .

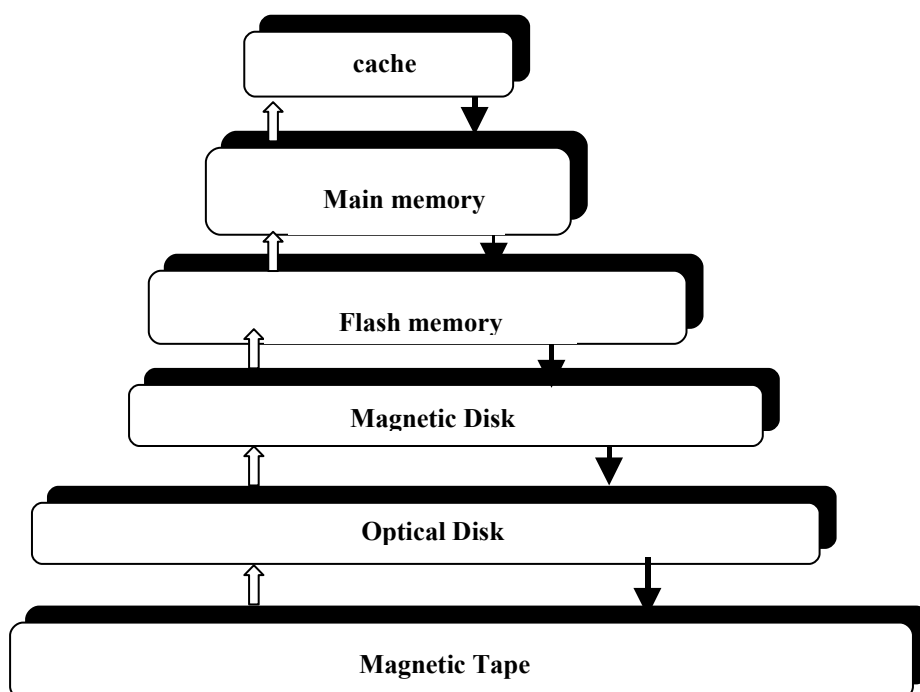


Figure (1) : Storage –device hierarchy

بالإضافة لسرعة وتكلفة وسائط التخزين يوجد عامل إضافي هام وهو هل الذاكرة او وسط التخزين متطاير ام غير متطاير volatile or Nonvolatile storage . والذاكرة المتطايرة تعني أنها ذاكرة مؤقتة تفقد

محتوياتها بمجرد انقطاع التيار الكهربى عنها ولحفظ البيانات يجب ان تكون الذاكرة Nonvolatile Storage بالرجوع الى الشكل الهرمى (1) : من الذاكرة الرئيسية Main Memory والى اعلى عبارة عن Nonvolatile Storage اما الانواع التى تقع اسفل الـ Main Memory جميعها Nonvolatile Storage .

وحدات قياس التخزين المختلفة فى الحاسوب :

1 byte = 8 bits

1 k (kilo byte) = 1000 bytes =  $10^3$

1 M (mega byte) = 1 million bytes =  $10^6$

1 G (Giga byte) = 1 billion bytes =  $10^9$

1 T (Tera byte) = 1000 Giga byte =  $10^{12}$

وستتناول أنواع وسائط التخزين بشيء من التفصيل ابتداءً من أسرعها واغلاها وهي الـ Cache وحتى أقلها تكلفة وأبطأها وهي الـ Magnetic Tape .

**1- Cache Memory** : هي أسرع واغلى وسائط التخزين الأولية وهي عبارة عن Static RAM تستخدمها الـ CPU لزيادة سرعة تنفيذ البرامج . وتتم إدارة الـ Cache وتنظيمها بواسطة نظام التشغيل نفسه .

**2- Main Memory** : تمثل منطقة العمل الرئيسية للمعالج التى يحتفظ فيها بالبيانات والبرامج العاملة حالياً وهي عبارة عن Dynamic RAM .

ميزتها الأساسية سعرها قليل نسبياً مقارنة مع الـ Cache حيث نلاحظ ان سعرها متناقص . أما عيبها الأساسي فهو أنها متطايرة Volatile Memory . وايضا بطأها نسبياً مقارنة بالـ Cache .

**3-Flash Memory** : تعرف أيضاً بـ ( EEPROM )

Electrically Erasable programmable Read Only Memory

وهي نوع وسط بين القرص والذاكرة الرئيسية تختلف عن الذاكرة الرئيسية Main Memory فى انها غير متطايرة Nonvolatile . وهي الاسرع فى استرجاع البيانات مقارنة بالـ Main Memory لكن عملية الكتابة واعادة الكتابة بها تعتبر الاكثر تعقيداً . ولاعادة الكتابة بها يجب مسح كل الـ Block أولاً ثم بعد ذلك الكتابة من جديد ويكمن العيب الأساسي للـ Flash Memory فى أنها لا تتيح مسح إلا أجزاء محدودة منها .

وتعتبر الـ Flash Memory هي الأكثر استخداماً بدلاً عن القرص فى أنظمة الكمبيوتر الصغيرة المضمنة فى العديد من الأجهزة والتي لا تتطلب وسائط تخزين كبيرة جداً .

**4-Magnetic Storage** : الاقراص المغناطيسية تستخدم لحفظ كميات البيانات الكبيرة وتعتبر هي

الوسيط الاساسى لتخزين البيانات لفترات طويلة Long term online storage وحجم القرص يحدد

بسعته بالـ Bytes مثلاً القرص المرن Floppy Disk سعته 1.4 MB والقرص الصلب Hard

Disk سعته فى تزايد مستمر وتتراوح الان من بين بضعة M Bytes الى عشرات الـ G.Byte . لمعالجة البيانات يتم نقلها أولاً من القرص الى الذاكرة الرئيسية وبعد المعالجة تعاد اليه ثانية ليتم حفظها فيه .

ويطلق على الأقراص المغناطيسية Direct-access Storage لأن الوصول للبيانات الموجودة على القرص يتم مباشرة دون المرور على البيانات الموجودة قبلها .

وتتمتاز الأقراص المغناطيسية بأنها غير متطايرة Nonvolatile وتحفظ ببياناتها رغم انقطاع التيار عنها أو تعطل النظام .. وناوياً ماتصاب الأقراص بأعطال جسيمة تؤدي إلى تدمير البيانات وضياعها .

**5-Optical Storage** : أكثرها شيوعاً الـ Disk (CD-ROM) Compact Disk Read Only Memory تخزن فيها البيانات بطريقة يستخدم فيها الضوء (optically) وتقرأ بواسطة الليزر وهي تحتوي على بيانات مكتوبة لا يمكن مسحها لذا تسمى (WORM) Write once Read Many وتستخدم كوسيط لتخزين بيانات الأرشيف Archival Storage Of Data وتسمح بكتابة البيانات مرة واحدة فقط وقراءتها عدد من المرات دون عرضها للمسح .

والـ Optical jukebox Memories عبارة عن منظومة من CD-ROMs تحمل على الـ Drive عند الطلب سعتها كبيرة حيث تسع مئات الـ GBs ولكنها أبطأ من الـ Magnetic Disk ، هذا النوع لم ينتشر كثيراً نسبة لزيادة القرص الصلب وقلة سعره المضطرب .

أما الـ (DVD) Digital Video Disks فهو نوع جديد من الأقراص الضوئية optical disk تصل سعته إلى 15 GB .

**6.Tape Storage** : تستخدم الـ Magnetic Tape نسبة لانها تتمتاز بقلّة التكلفة مقارنة بالأقراص المغناطيسية وإمكانية تخزينها لكميات كبيرة من البيانات قد تصل لعدد من الـ Tera Bytes استخدامها الأساسي لحفظ نسخ احتياطية الـ Back up ولحفظ وأرشيف البيانات الـ Archival Data ويمكن عيبها الرئيسي في بطئها الشديد فهي عبارة عن Sequential –Access Storage بمعنى أنه للوصول لبيانات محددة محفوظة بالشريط يجب المرور على جميع البيانات التي تسبقها تسلسلياً من بداية الشريط وحتى الوصول إليها . وعموماً فإن ساعات وسائط التخزين المختلفة في زيادة وأسعارها في نقصان .

يجب تخزين وحفظ قواعد البيانات Databases بصورة دائمة في إحدى وسائط التخزين الثانوية وذلك للآتي :

- 1.البيانات قد تكون كثيرة بحيث لاتسعها الذاكرة الرئيسية .
2. احتمال ضياع البيانات في الذاكرة الرئيسية Volatile Storage أكبر من احتمال ضياعها من القرص.
- 3.تكلفة التخزين في الوسيط الثانوي أقل من تكلفة التخزين في الذاكرة الرئيسية .

### تمرين :-

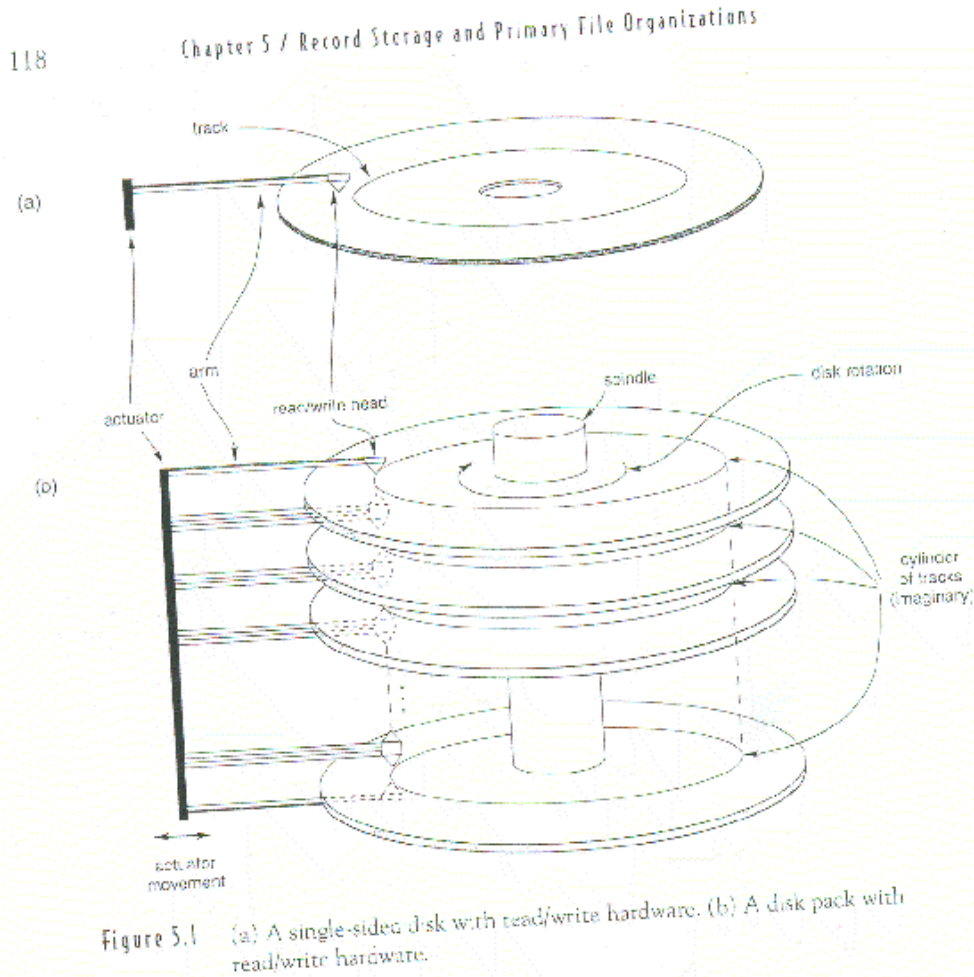
- 1.ما هي الفروقات بين طرق التخزين الأولية Primary Storage وطرق التخزين الثانوية ؟  
Secondary Storage ؟
- 2.علل لايمكننا تخزين قواعد البيانات الكبيرة جداً في الذاكرة الرئيسية ؟

## الدرس الثاني

### الأقراص المغناطيسية : Magnetic Disks

وتعتبر الأقراص المغناطيسية -تحديداً الأقراص الصلبة Hard Disks - أكثر الوسائط الثانوية استخداماً في الحاسبات . حيث ان البيانات بها تكون متاحة On-line مقارنة بوسائط التخزين الثانوية الأخرى والتي تكون Off-line بمعنى انها تكون غير جاهزة وتحتاج لعمل يدوي أو ميكانيكي لتوصيلها بالنظام . لذا سنتناول الأقراص بشيء من التفصيل :

### خصائص الأقراص الفيزيائية ( physical Characteristics of Disks ) :



يتكون القرص من مجموعة الـ Platters وهي شرائح مغناطيسية دائرية رقيقة مصنوعة من مادة صلبة كما في الأقراص الصلبة Hard Disks أو تكون مصنوعة من مادة بلاستيكية كما في الأقراص المرنة Floppy Disks .

يتم تخزين البيانات على أسطح هذه الشرائح فإذا كان التخزين على وجه واحد فقط من وجهي الشريحة يطلق على القرص Single Sided disk بينما الـ Double Sided Disk يتيح إمكانية التخزين على وجهي الشريحة .

بالنسبة للقرص الصلب توجد منظومة من الشرائح Disk Pack تضم عدة شرائح يقسم كل سطح من هذه الشرائح منطقياً إلى مجموعة من المسارات الدائرية Tracks هذه المسارات بدورها تنقسم إلى وحدات صغيرة تسمى ( Sectors or Blocks ) .

اعتماداً على نوع القرص تختلف أحجام الـ Sectors وعدد المسارات Tracks في الشريحة من قرص إلى آخر . وغالباً ما يتراوح عدد الـ Tracks بين مئات أو إلى عدة آلاف من الـ Tracks في الشريحة الواحدة بينما يتم تقسيم المسار track إلى عدد من الـ Sectors بواسطة نظام التشغيل أثناء تهيئة القرص Formatting لذا فإن حجم الـ Sector ثابت ولا يمكن تغييره وغالباً ما يتراوح الـ Sector الواحد بين 32 KB إلى 4096 KB . جدير بالذكر أن هذه الـ Sectors تفصل عن بعضها بمساحات محددة ( فراغات ) تسمى بـ ( Interblock gaps ) هذه المساحات تضم بيانات تحكم خاصة تكتب بها أثناء عملية التهيئة الـ initialization هذه البيانات تستخدم لتحديد موضع الـ Sector في الـ Track .

وعموماً هنالك تحسن كبير في صناعة الأقراص الصلبة من ناحية السعة التخزينية ومن ناحية السرعة في استرجاع البيانات أن أسعارها في انخفاض مستمر .

#### • الحركة الميكانيكية :

الوحدة الآلية الحقيقية التي تقوم بالقراءة والكتابة هي رأس القراءة والكتابة Read/Write head . حيث يوجد رأس قراءة وكتابة على سطح الـ platter وهو الذي يقوم بتسجيل البيانات بطريقة مغناطيسية في Sector محدد .

لأي وجه من وجهي الـ Platter رأس قراءة وكتابة يتحرك على سطح Platter للوصول للـ Tracks المختلفة .

يضم القرص العديد من الـ Platters وبالتالي العديد من رؤوس القراءة والكتابة مثبتة على مجمع رؤوس واحد يعرف بـ Disk Arm وهي بدورها مثبتة على حامل يسمى Boom يتحكم في حركته محرك Motor يتحرك في اتجاه أفقي حتى يتم وضع الرؤوس على الـ tracks المحددة فعند حركة أحد الرؤوس الأخرى للوصول للـ track رقم (I) الموجود في جميع الـ Platters الأخرى. كل الـ tracks رقم I في جميع الـ platters تعرف بـ Cylinder رقم (I)

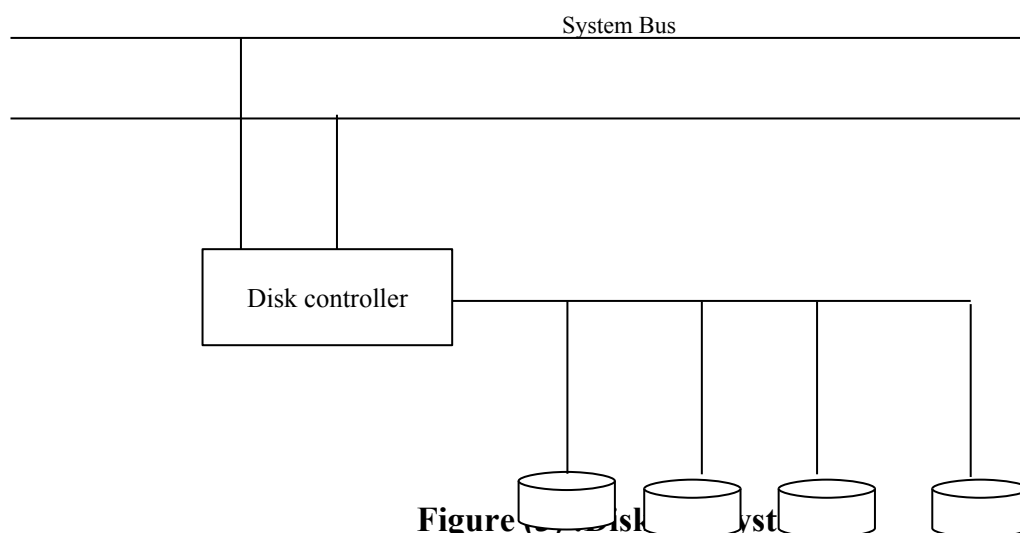
منظومة الشرائح ( platters ) الـ Disk Pack مثبتة على محور Spindle ويحتوي الـ Disk Drive على محرك Motor يحرك هذه الشرائح حركة دائرية حول محورها . يطلق على النوع السابق من انواع الأقراص الصلبة Movable-head disks .

هنالك أنواع من الأقراص لها رؤوس ثابتة وبعدد الـ Tracks (راس لكل Track)، حيث يكون بالذراع Arm الواحدة رؤوس وبعدد الـ tracks على السطح الواحد. هذا النوع يعرف بمصطلح Fixed-head disk في هذا النوع من الأقراص يتم تحديد الـ track أو Cylinder بواسطة نظام مفتاح الكتروني Electronic Switching يحدد الراس المطلوب والمقابل للـ track المحدد الكترونيات بدلاً من حركة ميكانيكية حقيقية ونتيجة لهذا يعتبر هذا النوع أسرع كثيراً من الأول ولكن نسبة للعدد الزائد من الرؤوس فتكلفته أعلى .

وكذلك يوجد نوع آخر من انواع الأقراص به أكثر من Disk Arm واحد في نفس الـ Platter يتيح الوصول لأكثر من Track واحد في نفس الزمن .

هذا النوع من الأقراص الـ (Fixed head disk) غير شائع ويستخدم فقط في الـ High – performance mainframe systems .

### • Disk Controller :



هذه الوحدة تعتبر جزء من Disks وهي تحكم فيه وتشكل الواجهة interface بينه وبين نظام الكمبيوتر. واحد من أنواع الواجهات interface القياسية والمستخدم حالياً مع الحاسبات الشخصية تسمى بـ (small computer storage interface) SCSI (Controller) تتلقى الـ أوامر القراءة والكتابة المكتوبة بلغات المستوى العالي لتتخذ بناء عليها الحركة المناسبة لوضع الذراع الـ Arm وبالتالي وضع راس القراءة والكتابة في الموضع المطلوب لحدوث عملية القراءة أو الكتابة .

كذلك تقوم الـ Controller بإلحاق وحدة تسمى Checksum بكل Sector يتم كتابة عدد البيانات المخزنة بها .

وعند قراءة هذه البيانات يتم حساب حجم البيانات التي تمت قراءتها ويقارن هذا الحجم بقيمة الـ Checksum المخزنة في كل Sector فإذا لم تتطابق القيمتان فإن هناك خطأ Error ، في هذه الحالة تكرر الـ Controller المحاولة عدة مرات ، فإذا استمر الخطأ في الحدوث تنبه الـ controller نظام التشغيل إلى وجود مشكلة في عملية القراءة Read Failure .

### •العنونة ونقل البيانات :

عند نقل البيانات من أو الى القرص يتم التعامل بوحدة تسمى الـ Disk Block وهو يمكن ان يكون Sector واحد أو مجموعة من الـ Sectors المتتالية في نفس Track حيث يتم تحديد عنوان الـ Block في القرص والوصول اليه مباشرة. وعنوان الـ Block يتكون من رقم السطح الـ Platter ، رقم المسار الـ Track (داخل السطح) ، ورقم الـ Block (داخل الـ Track) .  
وتقوم وحدة I/O Hardware الموجودة بالقرص نفسه بتحديد هذا العنوان .

ذاكرة الـ Buffer الخاصة بالقرص عبارة عن جزء من الذاكرة الرئيسية يتم نقل البيانات بينها وبين القرص فعند عملية القراءة يتم نقل البيانات بينها وبين القرص . فعند عملية القراءة يتم نسخ البيانات من الـ Block الى الـ Buffer وعند الكتابة يتم نقل محتويات الـ Buffer الى القرص الصلب . احيانا يتم نقل البيانات في شكل مجموعة من الـ Block كوحدة واحدة تسمى الـ Cluster في هذه الحالة يجب زيادة حجم الـ Buffer حتى تسع كل الـ Cluster .

تنتقل البيانات بين القرص والذاكرة الرئيسية في شكل Blocks ويقوم الـ File System Manager بتحويل عنوان الـ Block لرقم الـ Platter والـ Cylinder والـ Track والـ Block المطلوب التعامل معه .

### مقاييس أداء الأقراص ( Performance Measure of Disks ) :

المقاييس الأساسية لأداء القرص هي :

- |                      |               |
|----------------------|---------------|
| 1.Capacity           | 2.Access Time |
| 3.Data Transfer Rate | 4.Reliability |

#### 1. Capacity : السعة

ونجد أن الحاسبات الحديثة توفر ساعات تخزينية ضخمة تتيح تخزين كميات كبيرة من البيانات .

#### 2. Access Time : زمن الوصول للبيانات:

هو الزمن المطلوب لتحديد موضع الـ Block المطلوب قراءته أو الكتابة به لتبدئ عملية نقل البيانات بين القرص والذاكرة الرئيسية ويكون من :-

##### (a) Seek Time (s) زمن البحث

هو الزمن المطلوب لتحريك رأس القراءة والكتابة الى الـ Track المطلوب في الـ Movable-head disks أما في الـ Fixed-head disks فهو الزمن المستغرق لتحديد الرأس المطلوب إلكترونياً { Electronically switch to the appropriate head } في الـ Movable-head disks يختلف هذا الزمن على حسب موضع الرأس الحالي والموضع المطلوب الانتقال اليه وعادة ماتحدد الشركات المصنعة هذا الزمن كمتوسط Average seek Time بالملي ثانية وهو عادة مايتراوح بين 10-60 msec في الحاسبات الشخصية ومن 8-9msec في اجهزة الـ servers .

##### (b) Rotational Delay (or latency) Time (rd) :



عندما يكون راس القراءة والكتابة في الـ track المطلوب يجب الانتظار حتى تدور الـ Platter لتصل لبداية الـ Block المطلوب أسفل الراس هذ الزمن يمكن ان يكون صفر اذا كان الـ Block المطلوب تحت الراس بمجرد وصول الراس للـ Track المعنى أو يمكن أن يكون زمن دورة كاملة للـ Platter اذا كان الـ المطلوب هو الـ Block قبل الـ Block الموجود حالياً تحت الراس .

اما بالنسبة للـ Fixed-head disk يعتبر الـ Seek Time صغير لذا تكون الـ rd هي القيمة المؤثرة على سرعة القراءة والكتابة . اذا الـ Access time هو مجموع الـ seek time + rd .

### 3. (Data Transfer Rate) أو Block Transfer Rate:

هو معدل نقل البيانات من وإلى القرص بعد وصول الراس لبداية الـ Block المطلوب نقله. هناك زمن مطلوب لنقل البيانات من الـ Block إلى الذاكرة الرئيسية وزمن نقل الـ ((Block Transfer time Block (btt) يعتمد على حجم الـ Block وحجم الـ Track وسرعة دوران القرص . وبهذا يكون الزمن الكلي المطلوب لتحديد موضع الـ Block ونقل محتواه هو مجموع :

$$\text{Seek time} + \text{Rotational delay} + \text{Block transfer rate}$$

مثال :-

قرص صلب معدل نقل بياناته يساوي 10 m sec وزمن البحث seek time هو 5 m sec إذا علمت أن قيمة الـ Rotational delay time هي صفر ، احسب الزمن اللازم للوصول إلى block ونقل محتواه ؟

الحل :-

$$\begin{aligned} \text{Access time} &= \text{Seek time} + \text{Rotational delay} + \text{Block transfer rate} \\ &= 5 + 0 + 10 = 15 \text{ m sec} \end{aligned}$$

وعموماً الـ Seek Time والـ Rotational delay زمنيهما اكبر من الـ Block Transfer Rate حيث أن الزمن الأكبر للحركة الميكانيكية ، لذا ولزيادة كفاءة وسرعة نقل البيانات توضع الـ Block التي يراد التعامل معها (قراءة أو كتابة ) في اسطوانة واحدة Cylinder لتقليل الحركة الميكانيكية . وعلى كل حال يعتبر الزمن المطلوب للوصول إلى هذه البيانات المحددة على القرص كبيراً مقارنة بومن تشغيلها في الـ main memory .

### 4. Reliability الإعتدائية: -

يتم قياس الـ Reliability لقرص بمتوسط زمن تعطله عن العمل a) mean time of failure is a measure of reliability of the disk ) وهو متوسط الفترة الزمنية التي يتوقع ان يعمل فيها القرص باستمرار دون توقف (متوسط زمن السقوط) .

### تمارين

1. عرف كل من المصطلحات الآتية :-

1- block

2-cylinder

3-sector

4-interblockgap

## 5- Cluster

2. باختصار أذكر الطريقة الميكانيكية التي تتم بها قراءة البيانات من القرص الصلب ؟
3. ماذا نعني بمتوسط زمن السقوط ؟

### الدرس الثالث

## RAID Technology

مع زيادة احجام وسرعات الذاكرة الرئيسية والمعالجات كان لابد من زيادة احجام وسرعات وحدات التخزين الثانوية لمقابلة هذه الزيادة ، ولمقابلة احتياجات التطبيقات الجديدة وكان من اميز التطورات في وحدات التخزين الثانوية الـ RAID

### (RAID) Redundant Arrays of Inexpensive Disk

الحرف I احيانا يرمز للـ Independent كان السبب الاساسي للتفكير في الـ RAID هو أن الزيادة في تحسين أداء الـ Disk لم يكن بنفس سرعة الزيادة في تحسين اداء الذاكرة والمعالج . فكان الحل إنشاء منظومة ( Array ) من الاقراص المفصولة عن بعضها لتعمل كلها كقرص (منطقي) واحد ليكون عالي الكفاءة باستعمال مفهوم Data Striping والذي يحقق مفهوم العمل على التوازي parallelism . عملية الـ Data Striping هي عبارة عن توزيع البيانات على الاقراص لتعمل كلها كقرص واحد وسريع . الشكل التالي يوضح كيفية توزيع البيانات لملف واحد وعلى عدد 4 اقراص .

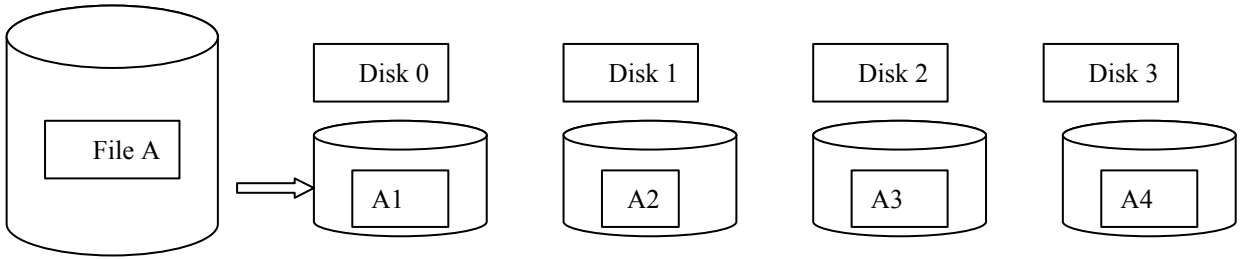


Figure (4) : Data striping : file A is striped across 4 disk

عملية توزيع البيانات Data Striping تعمل على تحسين الاداء الكلي بالاتي :

1. تنتج لأكثر من عملية  $O(1/multiple)$  (قراءة او كتابة ) ان تعمل على التوازي مما يحسن عملية نقل البيانات transfer rate الكلي .

2. يقوم الـ RAID بتوزيع العمل على كل الـ Disks .

3. يمكن تحسين الاعتمادية reliability بتكرار كتابة البيانات على اقراص مختلفة.

### تحسين الإعتمادية باستخدام الـ RAID Improving Reliability with RAID

باستعمال منظومة أقراص Disks مكونة من n قرص احتمال العطل n مرة احتمال تعطل القرص الواحد . فمثلا لمنظومة مكونة من 100 قرص اذا كان العمر الافتراضي للقرص 200.000 ساعة أي 22.8 سنة .. يكون العمر الافتراضي للمنظومة فقط 2000 ساعة أي 83.3 يوم وبفرض ان هناك نسخة واحدة من البيانات مقسمة على كل الأقراص .

اذا الحل هو كتابة نسخة إضافية من البيانات ولكن عيب هذا الحل هو المساحة الإضافية المطلوبة لهذه النسخة إضافة للعمليات الإضافية (I/O) اللازمة لكتابة البيانات أكثر من مرة .

إذا فالتقنية الشائعة هي الـ **Mirroring or Shadowing** حيث تكتب البيانات في قرصين متطابقين في نفس الوقت والذان يعاملان منطقياً كقرص منطقي واحد. عند قراءة البيانات تقرأ من أسرع القرصين .. وعند تعطل أحدهما تقرأ من الآخر حتى يتم إصلاح الأول .

### **تحسين الأداء باستخدام الـ RAID RAID in Improving Performance:**

توزيع البيانات على منظومة الأقراص يزيد من سرعة نقل البيانات . وبما أن البيانات تقرأ بواقع block واحد في اللحظة فإنه يمكن تطبيق الـ **disk striping** لادنى مستوى بتقسيم الـ byte إلى 8 ثنائيات (bits) وتوزيع هذه الـ bits على 8 أقراص مختلفة حيث يكتب الـ bit رقم  $j$  في القرص رقم  $j$  وبهذه تزيد كمية البيانات المقرؤة في اللحظة **bit-level data striping** .

ويمكن تطبيق الـ **data striping** على مستوى أعلى من الـ bits حيث يمكن تقسيم الملف إلى blocks وكل block يكتب في قرص منفصل **block level striping** بهذه الحالة يمكن الوصول إلى Block مختلفة في نفس اللحظة مما يزيد كفاءة زمن الوصول إلى البيانات عموماً كلما زاد عدد الأقراص زادت كفاءة المنظومة وإيضاً زاد احتمال حدوث العطل وبالتالي زادت الحاجة إلى **Mirroring** .

### **RAID Organization and Levels :**

تقسم الـ RAID من ناحية تنظيمية إلى أنواع أو مستويات **Levels** يعتمد هذا التصنيف على عاملين أساسيين وهما :-

\* **Granularity of data striping**

\* **Pattern used to compute redundant information**

في البداية كان التصنيف من المستوى الأول إلى الخامس **level 1** وحتى **level 5** ومؤخراً أضيفت **level 0** و **level 6** .

**level 0** : البيانات لا تتكرر ولهذا سرعة الكتابة وتعديل البيانات عالية لأن البيانات لا تكتب مرتين . أما سرعة القراءة فهي أقل مما عليه في المستوى **level 1** حيث تكتب البيانات مكررة لأنها تستعمل تقنية **mirrored disks** ، حيث تكون في الأخيرة سرعة القراءة عالية وبحيث يتم تحسين الأداء بجدولة عمليات القراءة ويتم تنفيذ العملية التي تتطلب زمن بحث أقل .

**Level 2** : تستعمل ما يسمى بالـ **memory-style redundancy** والتي توفر **parity bits** وبهذا فهي تحتاج لثلاث أقراص إضافية **three redundant disk** لكل أربعة أقراص أصلية، مقارنة بأربعة أقراص مقابل أربعة أصلية في **level 1** وهي أيضاً تتيح إمكانية اكتشاف وتعديل الخطأ **error detection and correction** .

**Level 3** : تستعمل قرص واحد لتخزين الـ **parity** وأربعة أقراص لتخزين البيانات معتمدة على الـ **controller** لتحديد أي الأقراص قد تعطل .

**Level 4 , level 5** : تستعمل الـ **block level data striping** وفي **level 5** توزع البيانات والـ **parity information** عبر كل الأقراص .

**Level 6** : تطبق ما يسمى بالـ **P+Q redundancy** ، حيث توجد **(2bit for redundant data are store for every 4 bits of data)** .

مما سبق من مقارنة يتضح ان اعادة بناء البيانات اسهل في level 1 في حالة تعطل أي قرص .. اما في بقية المستويات فهي تحتاج الى عملية معقدة والقراءة من كل الاقراص الاخرى . ولهذا فان level 1 يستعمل في التطبيقات المهمة والحرية مثل حفظ log of transactions اما level 3 و level 5 فهي تستعمل في البيانات كبيرة الحجم حيث تقدم level 3 سرعة عالية لنقل البيانات . اذا مصممي الـ RAID يجب ان يضعوا في الحسبان العديد من العوامل : مثل الـ RAID level و عدد الاقراص المطلوبة ، طريقة الـ parity المستعملة .

بمناقشة الـ RAID Technology يتضح لنا التقدم في مجال تخزين البيانات وكيفية توظيف الافكار نفسها يكون حسب التطبيقات .

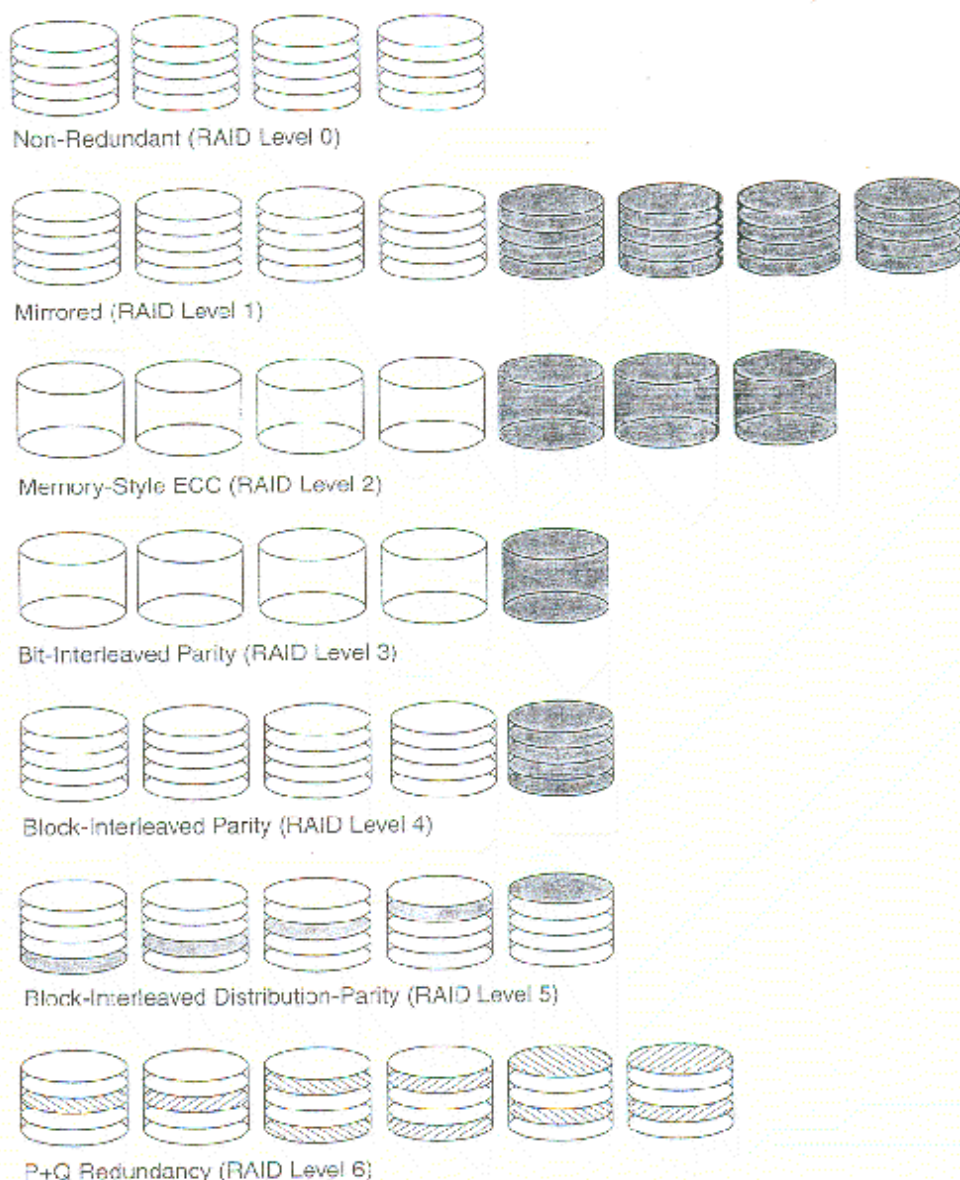


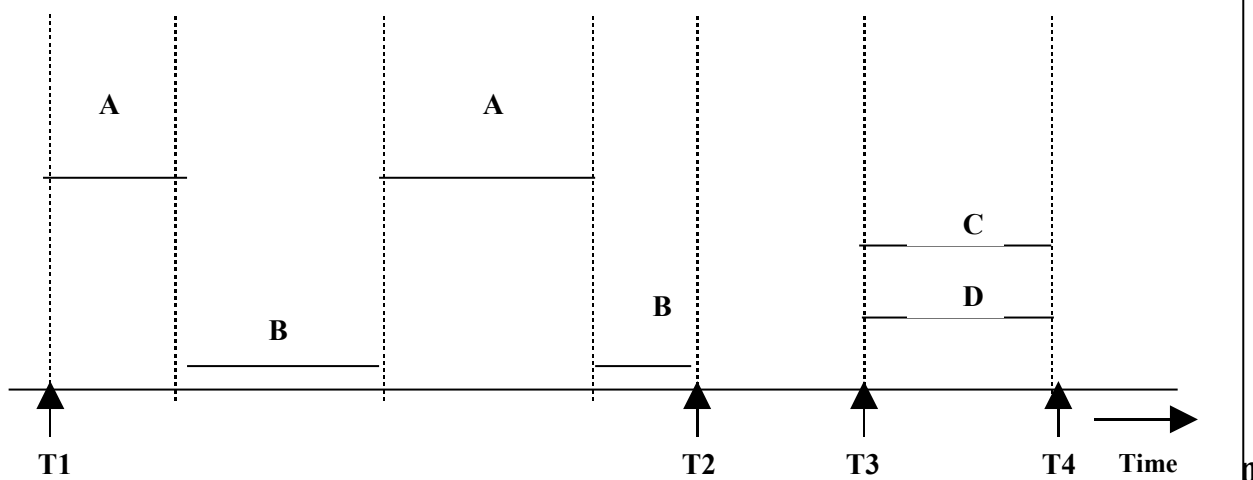
Figure 5.4 Multiple levels of RAID. From Chen, Lee, Gibson, Katz, and Patterson (1994), ACM Computing Survey, Vol. 26, No. 2 (June 1994). Reprinted with permission.

## Buffering of Blocks

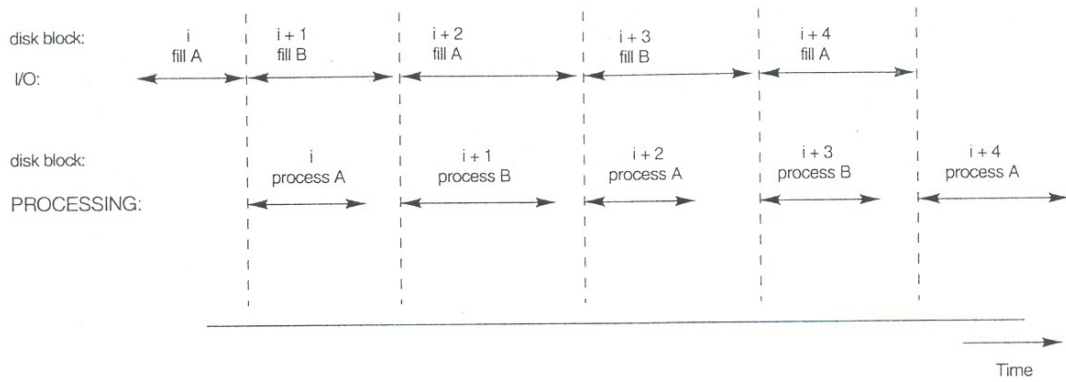
إذا كان هناك العديد من الـ Blocks يراد نقلها من القرص الى الذاكرة الرئيسية وكلها معروفة العناوين. يمكن حجز اكثر من موضع في الذاكرة Several buffers لزيادة سرعة نقل البيانات يستخدم الـ buffer كوسيط بين الذاكرة والقرص عند عملية نقل البيانات. انشاء قراءة او كتابة one buffer يمكن للمعالج ان يشغل buffer اخرى . وهذا يمكن بوجود (disk I/O processor او controller) والذي يعمل على التوازي وبمنعزل من الـ CPU . الشكل (6) يوضح كيف يمكن لمعالجين ان يعملان على التوازي العمليتان A,B تعملان على التوالي ولكن بصورة Inter leaved بينما C,D تعملان على التوازي ( in a parallel fashion)

**Interleaved concurrency of operations A & B**

**Parallel Execution of operations C & D**



في حالة تحكم الـ CPU على عدة عمليات يكون التشغيل على التوازي parallel execution غير ممكنا . ولكن يمكن لتلك العمليات ان تعالج in an interleaved way ، وفي حالة تشغيل العمليات على التوازي in a parallel fashion تكون عملية الـ buffering مفيدة جدا اما لوجود معالج القرص الخاص (disk I/O processor) controller) او لوجود عدة CPU . الشكل يوضح كيف يمكن لعمليتي القراءة والمعالجة ان تعملان على التوازي عندما يكون زمن معالجة الـ Block على الذاكرة اقل من الزمن المطلوب لقراءة ونقل الـ blocks التالي الى الذاكرة .



**Figure (7) use of two buffers A and B for reading from Disk**

يمكن للـ CPU أن تبدأ عملها على الـ Block بمجرد وصوله الى الذاكرة الرئيسية وفي نفس الوقت تقوم disk I/O processor (controller) { بنقل وقراءة block اخر على buffer اخرى . هذه الطريقة تسمى double buffering وايضاً تستعمل لكتابة فيض من الـ blocks المتتالية من الذاكرة الى الـ disk عملية الـ double buffering تسمح بالقراءة والكتابة المتتالية للبيانات الموجودة على blocks متجاورة مما ينقص الـ Rotational delay seek time & لكل الـ blocks عدا الـ Block الاول زيادة على ذلك البيانات تكون جاهزة للتشغيل مما يقلل زمن الانتظار للبرنامج.

#### تمرين :-

1. ماهو الهدف الأساسي الذي أدى لضرورة ظهور تقنية الـ RAID؟
2. كيف ساهمت طريقة الـ Mirroring في زيادة الاعتمادية ؟
3. لماذا نجد ان الـ RAID Level 1 هي الأفضل عند التعامل مع البيانات المهمة جداً والتي قد تتطلب زمن قراءة سريع للبيانات ؟

#### الدرس الرابع

## : Placing File Records on Disk

في هذا الجزء نناقش مفهوم السجل record وأنواع السجلات والملفات ثم نناقش بعض طرق وضع هذه السجلات على القرص .

### : السجلات records

هي نوع من أنواع البيانات المرتبة، ولا يشترط أن تكون العناصر المكونة لها من نفس النوع .. ويقال لعناصر البيانات الفردية انها حقول Fields داخل السجل فمثلا سجل الطالب يتكون من عدة حقول (اسمه وهو حرفي ، رقمه وهو رقمي ، تاريخ ميلاده وهو تاريخ )  
نقوم بدراسة التطبيقات لهذه المادة باستخدام لغة الباسكال لشيوعها وتمكن جميع الطلاب منها .

### تعريف السجل :

نتيح لغة باسكال طريقتين لتعريف السجلات :

```
;VAR record name :RECORD field1; field 2; ...field n (1)  
;END
```

مثال لها نعرف سجل لبيانات زبون

```
VAR Customer: RECORD  
;Custno :integer  
;Custtype:char  
Custbalance:real  
;END
```

(2) نعرف نوع بيانات جديد ان سجل ثم بعد ذلك يمكن توضيح ان متغيرا من هذا النوع :

```
TAYPE name = RECORD field field2 ...field n END
```

: مثلا لنفس سجل الزبون في المثال السابق

```
Type Customer = RECORD  
Custno :integer;  
Custtype: char;  
Custbalance:real  
END;  
VAR cust:customer ;
```

. وهذه الطريقة اعم لانها تسمح تسمح بادخال متغيرات سجلات اضافية

### **: RECORD PROCESSING تشغيل السجل**

اذا اردنا ان نحدد لسجل ما قيمه ما لسجل اخر . اولاً يجب ان تكون السجلات من نفس النوع مثلاً

```
,VAR: old customer, New customer: customer
```

```
:Begin
```

```
New Customer: =Old Customer
```

```
END
```



وهذا يتم نقل جميع بيانات السجل Old Customer الى New customer .  
او يمكن نقل هذه البيانات من سجل لآخر .. حقلاً حقلاً

```
; New customer. Custno := old customer.Custno  
; New customer. custtype := oldcustomer .custtype  
; New customer .custbalance := oldcustomer.custbalance
```

تشغيل السجلات عملياً يحتاج لهياكل بيانات اكبر لتحتوي السجلات، مثلاً ان يكون لدينا ملف زبائن او منظومة تحوي عدداً محدداً من الزبائن فهنا نحتاج لتعريف منظومة Array عناصرها سجلات وبهذا يتسنى لنا الوصول الى السجل الموجود داخل المنظومة . تعريف مثل هذه المنظومة كالآتي :

```
; VAR Cust:Array [1..100] of customer
```

فمثلاً يمكننا نقل بيانات الزبون رقم 35 الى سجل الزبون رقم 20

```
; [ Cust [20] := cust [35
```

او نقل بيانات حقل فقط من حقول زبون الى الحقل المقابل لزبون آخر مثلاً :

```
;Cust [20].Custtype :=cust [35 ].custtype
```

في بعض التطبيقات قد يكون بالسجل العديد من الحقول ونريد تشغيلها كلها . فهنا يكون من الافضل استعمال عبارة WITH بدلاً من ان يكتب :

```
; New customer. Custno := 16  
New customer. custtype := 'A';  
; New customer .custbalance := 315.6
```

يمكن أن نكتب :

WITH Newcustomer DO

Begin

```
Custno := 16 ;  
Custtype:= 'A ' ;  
Custbalance := 315.6 ;
```

### أنواع السجلات Variable length & Fixed length Record

يمكن أن تكون جميع السجلات المكونة للملف ذات احجام متساوية Fixed length Records and

variable length Records وفي بعض الحالات يمكن ان تكون أحجام هذه السجلات غير متساوية

وتكون سجلات الملف من النوع ( Variable length records ) لعدة أسباب منها الاسباب الآتية :-

- وجود حقل أو أكثر من حقول السجل احجامها مختلفة ( Variable length fields ) فمثلاً حقل الاسم  
يمكن ان يختلف حجمه من سجل لآخر .

- وجود حقل أو أكثر من حقول السجل يمكن ان تكون حقول اختيارية optional fields .

- وجود حقل أو أكثر من قيمة ويسمى بـ repeating field .

### **:Record Blocking and Spanned Versus Unsnapped Records**

السجلات المكونة للملف يجب تسكينها (أو وضعها) في disk block لان الـ Block هي وحدة نقل البيانات بين القرص والذاكرة الرئيسية (Block is the unit of data transfer between disk and memory) اذا كان حجم الـ Block هو B بالنسبة لـ File of fixed length records بحجم R bytes لكل سجل ، وفي حالة ان  $B > R$  يكون المعامل bfr يساوي :

$$\text{bfr} = B / R \text{ records per block}$$

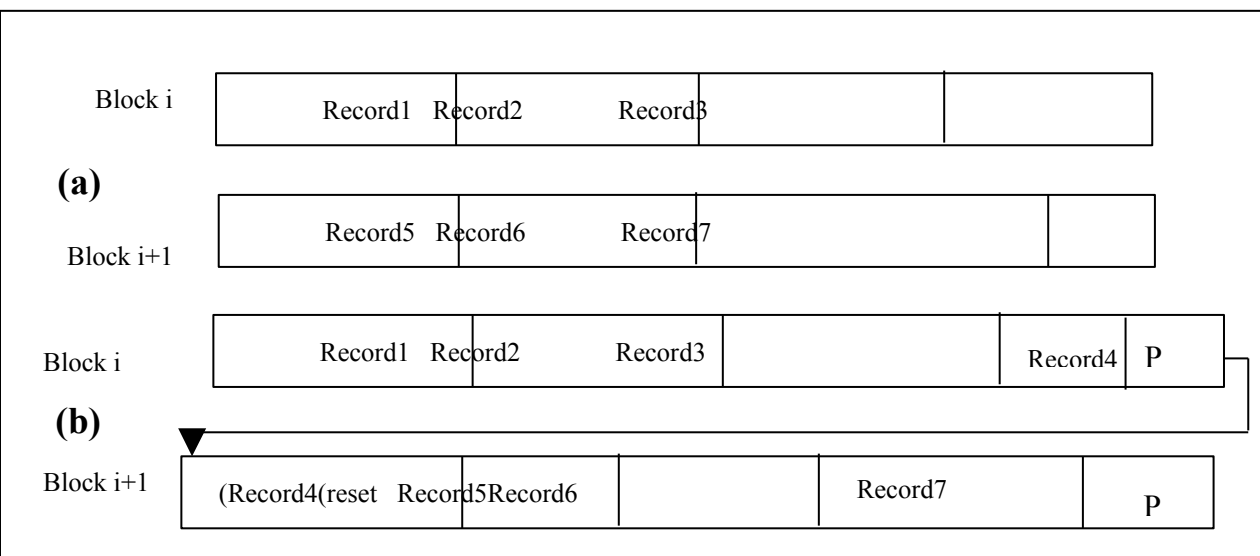
حيث الـ bfr هي الـ Blocking factor for the file ( اقصى عدد من السجلات التي يمكن تخزينها في الـ block الذي حجمه B اذا كان طول السجل R )

وعموماً تنتج بعض المساحات غير المستغلة في كل block هذه المساحة تساوي :

$$B - (\text{bfr} * R) \text{ bytes}$$

ولاستغلال هذه المساحات يمكننا تخزين جزء من السجل في هذه المساحة المتبقية من الـ Block وتخزين باقي السجل في Block آخر مع وجود مؤشر pointer في نهاية الـ Block الاول يؤشر للـ Block الاخر الذي يحوي باقي السجل هذه التنظيم يعرف بالـ Spanned لان في هذه الحالة السجل يمتد على اكثر من block واحد .

في حالة حجم السجل اكبر من حجم الـ Block يجب استخدام الـ Spanned organization. الـ Unsnapped organization لا يسمح بتجاوز حدود Block لتخزين سجلات الملف .



**Figure (8) Types of record organization. (a) Unsnapped. (b) Spanned**

البيانات المخزنة هذه تنظم في شكل ملفات من السجلات File of record ولهذا يجب تخزين هذه السجلات بداخل الملفات بطريقة تسهل التعامل معها وهناك العديد من طرق تنظيم الملفات Primary file organization والتي توضح كيف توضع السجلات فيزيائياً في الملفات وفي القرص ، وبالتالي كيف يمكن استدعاء السجلات . منها الـ heap file وهذا النوع يضع السجلات بدون ترتيب حيث يضيف فقط السجل في نهاية الملف اما الملف المرتب Sequential or Sorted file فيحفظ البيانات في شكل مرتب بناء حقل معين يسمى مفتاح الترتيب Sort key .

## Allocating File Block on Disk

هناك عدة طرق لتسكين (وضع) كتل الملف File Blocks في القرص. فمثلا في طريقة contiguous allocation توضع الكتل (blocks) في Disk blocks متتالية ، مما يجعل عملية قراءة الملف سريعة جدا باستعمال double buffering ولكن بالمقابل تكون الزيادة في الملف (الإضافة) صعبة . في طريقة linked allocation أى كتلة من الملف file block تحتوى على مؤشر يؤشر للكتلة التالية.. وعكس سابقتها هذه الطريقة تجعل الإضافة فى الملف سهلة ولكن عملية القراءة للملف كاملا تكون بطيئة جدا.

أما طريقة Clusters of consecutive disk blocks فهي عبارة عن هجين من الطريقتين السابقتين، حيث تجتمع الكتل blocks المتجاورة بوحدة أكبر تسمى ال cluster.

### -:File header

بإدئة الملف أو واصفة الملف تحتوى على معلومات عن الملف والتي تحتاج لها البرامج التي تحتاج للوصول الى السجلات بالملف.

البإدئة تحتوى على معلومة عن عناوين ال blocks المكونة للملف وطريقة تمثيل السجلات record format description والتي تحتوى على طول الحقول بالإضافة لترتيب الحقول فى السجلات من النوع fixed length unspanned وأيضا تحتوى على نوع الحقل ، الفاصلات separators ونوع السجل للسجلات من النوع variable length record.

للبحث عن سجل بالقرص يجب نقل block واحد او اكثر one or more block إلى الذاكرة buffer ثم تقوم البرامج بالبحث عن السجل فى الذاكرة بإستعمال المعلومات الموجودة فى بإدئة الملف . فإذا كان عنوان السجل المطلوب غير معروف فإن البحث يكون liner search ال blocks ، حيث ننقل كل السجلات إلى الذاكرة حتى يتم العثور على السجل المطلوب أوينتهى الملف وهذه العملية تأخذ وقت طويل للغاية..

الهدف من التنظيم الجيد للملفات هو وضع ال blocks التي تحتوى على السجل المطلوب بأقل عدد ممكن من حركة او نقل ال ( blocks transfer blocks ).

### تمرين :-

1. اذكر الأسباب التي أدت لوجود سجلات من النوع variable length record؟

2. ملف حجم ال block له 20 byte وطول السجلات في الملف 6byte:

-أحسب المعامل bfr

-بالرسم وضح شكل ال Block اذا علمت ان التخزين كان من النوع Spanned organization

-ارسم شكل ال block مرة أخرى إذا كان التخزين من النوع Unspanned Organization

3. عرف سجل طالب Student بلغة باسكال يحتوي علي الحقول التالية :-

رقم الطالب ، إسم الطالب ، تاريخ الميلاد ، السنة الدراسية  
- إستخدم التعريف في نقل بيانات سجل إحد الطلاب لسجل طالب آخر

## الدرس الخامس

### Operations on files

يمكن تقسيم العمليات على الملفات الى مجموعتين رئيسيتين هما :

1- عمليات استرجاع Retrieval operations

2- عمليات تعديل update operations الى Retrieval operation لاتقوم باي تعديل على

الملف ولكن تقرأ فقط السجلات المحددة . أما الى update operations فهي تجري التعديل على

الملف اما بالإضافة او الحذف او تعديل حقل في السجلات.

إذاً في كلا الطريقتين نحتاج لاختيار سجل او سجلات بناء على شرط اختيار معين selection or

filtering condition والسجل او السجلات التي تستوفي الشرط السابق المحدد .

مثلا سجل الموظف الذي حقوله Name , SSN, Salary , Jop, department ;

فيمكن ان يكون شرط الاختيار البسيط ( simple selection condition ) Equality

operation علامة تساوي : مثلاً

(SNN = 1324) او ('Department = 'Account)

او (Salary > 5000) : (Comparison operation) ....

تعتمد عمليات البحث في الملف عموماً على شرط اختيار بسيط simple selection condition وإذا

كان الشرط معقد فيجب ان تقوم DBMS او المبرمج بتجزئته لشروط بسيطة يمكن ان تستعمل للوصول

الى السجل ثم يختبر بعد ذلك اذا كان هذا السجل يحقق بقية الشروط فمثلاً اذا كان الشرط:

(Department = 'compute') and (Salary > 5000)

فإننا نستعمل أولاً الشرط 'Department = 'account' فبعد الوصول للسجلات التي تحقق هذا الشرط

نقوم باختبار الشرط الآخر وهو Salary > 5000 .

إذا كانت هناك عدة سجلات تحقق شرط البحث فاول سجل نصل اليه يكون هو الى current record ثم

ننتقل منه الى بقية السجلات .

العمليات على الملف للوصول الى السجلات والعمل عليها تختلف على حسب نظم ادارة قواعد البيانات

الى DBMS او اللغة المستعملة ولكن على العموم يمكن حصرها كالآتي :-

**Open**: تجهز الملف لعملية القراءة او الكتابة ويتم حجز عدد ملائم من الى buffer على الاقل اثنين

لوضع الى blocks بها تقوم بقراءة الى file header ووضع مؤشر الملف في بداية الملف.

**Reset** : تضع مؤشر الملف المفتوح في بداية الملف .

**(Find (or locate** : تبحث عن أول سجل يحقق شرط البحث وتنتقل الى block الذي يحوي السجل

الذي يحقق الشرط في الذاكرة buffers ويقوم مؤشر الملف بالاشارة الى السجل في الى buffer ليصير

السجل الحالي current record .

**( Read (or Get** : تنسخ السجل الحالي current record الموجود في الى buffer الى البرنامج

الذي يطلبه، ايضاً قد تحرك مؤشر الملف الى السجل التالي الذي قد يكون مطلوب بعد هذا السجل .

**Find Next**: تبحث عن السجل التالي الذي يحقق شرط البحث، تنتقل إلى block الحاوي لهذا السجل إلى buffer ( إذا لم يكن أصلاً موجوداً بها ) ، يوضع السجل في الـ buffer ويصبح السجل الحالي current record .

**Modify**: تعدل قيمة حقل بالسجل الحالي current record ثم تعدل الملف على القرص .  
**Delete**: تُمسح السجل الحالي current ثم تعدل الملف على القرص لعكس عملية التعديل to reflect the modification .

**Insert**: تضيف سجل جديد بالملف حيث تحدد أولاً إلى block الذي يدخل به السجل، ثم تنتقل هذا إلى block إلى الـ buffer (إن لم يكن موجوداً) تكتب السجل على الـ buffer ثم تكتب محتوى الـ buffer في القرص .

**Close**: اكمل عملية الوصول والتعامل مع الملف بتحرير الـ buffers .

كل العمليات السابقة (عدا open and close) تسمى record at a time operation لان أي عملية تنفذ على سجل واحد يمكن ان نجعل العمليات Read, find, find next في عملية واحدة تطبق على الملف وهي scan وهذه المجموعة من العمليات تسمى set at time operation .

**Scan**: اذا تم فتح الملف في هذه اللحظة فان الـ scan تعطي اول سجل ، وفي غير ذلك تعطى السجل التالي . اذا تم تحديد شرط مع هذه العملية فهي تعطي اول سجل يحقق الشرط ثم التالي . وهكذا . وتوجد أيضاً امثلة للعمليات من النوع set at time operation .

**Find all**: تحدد كل السجلات بالملف والتي تحقق الشرط .

**Find ordered**: تسترجع كل السجلات بالملف بترتيب محدد .

**Reorganize**: تبدأ عملية التنظيم كما سنرى لاحقاً بعض تنظيمات الملف تحتاج إعادة ترتيب دورياً ... مثلاً إعادة ترتيب الملف بناءً على حقل آخر .

نميز فيما يلي بين مصطلحين وهما :

### **Access method and File organization :**

**File organization**: تنظيم الملف نعني به تنظيم البيانات داخل الملف إلى سجلات و blocks ونعني به أيضاً هياكل الوصول إلى البيانات access structures ونعني به أيضاً طريقة وضع السجلات والـ blocks في وسائط التخزين .

**Access method**: طريقة الوصول إلى البيانات تضم مجموعة من العمليات التي تتيح التعامل مع الملف (مثل العمليات الموضحة سابقاً) على كل حال يمكن ان نطبق عدد من Access method على تنظيم معين للملفات فمثلاً الـ indexed access لايمكن ان يطبق الا على الملف المفهرس أصلاً .

عادة نختار تنظيم الملف وطريقة الوصول حسب طبيعة العمل في الملف فمثلاً بعض الملفات تكون Static بمعنى أن التعديل فيها لا يذكر بينما بعض الملفات Dynamic بمعنى أن التعديل فيها مستمر، بعض الملفات نحتاج فيها لقراءة كمية من السجلات في اللحظة الواحدة .

مثلاً إذا حدد المستخدم انه يبحث عن بيانات الموظف برقمه اذا يقوم المبرمج ببناء الملف مفهرساً برقم الموظف أو نحتاج الوصول للبيانات بالقسم فبالتالي نرتبها بالقسم وهذا ما سيتم توضيحه في الجزء الخاص بالفهرسة .

## **: Files of Unordered Records Heap File (1)**

يعتبر هذا أبسط نوع من أنواع تنظيم الملفات File Organization حيث تدخل السجلات الى الملف حسب ترتيب ادخالها فيه .... السجل الجديد يدخل في نهاية الملف ، يسمى هذا النوع من الملفات heap file or pile file وهذا النوع يستخدم لاستعمالات خاصة مثل secondary indexes او حفظ البيانات لاستعمال مستقبلي في هذا النوع من الملفات :

● عملية الاضافة لسجل سريعة جداً ويمكن وصفها في الخطوات الآتية:

1. يتم نسخ الـ block الاخير من الملف الى الـ buffer .

2. يضاف السجل الجديد الى هذا الـ block .

3. يكتب الـ block مرة اخرة في القرص rewritten

● عنوان الـ block الاخير من الملف يحفظ في الـ File header .

\* عملية البحث عن أي سجل باي شرط يتطلب المرور على كل الـ blocks واحداً تلو الآخر Liner search .. وهي عملية مكلفة اذا كان هناك سجل واحد يحقق الشرط فيجب على البرنامج أن يقرأ نصف الـ blocks (في المتوسط) قبل ان يجد السجل المطلوب فمثلاً لملف حجمه B Blocks نبحث في عدد  $B/2$  Blocks اما اذا كان هناك اكثر من سجل يحقق الشرط أو لا يوجد أي سجل يحقق الشرط فيجب على البرنامج ان يقرأ كل الـ B blocks لبحث في كل السجلات .

● عملية الحذف تكون كالاتي :

1. ابحث عن السجل المراد مسحه .

2. انسخ الـ block الذي يحوي هذا السجل الى الـ buffer .

3. امسح السجل من الـ buffer .

4. أعد كتابة الـ buffer في Disk .

عملية المسح هذه تترك فراغات في الـ block .

هناك آلية اخرى لعملية المسح ..... وهي ان يكون هناك bit or byte مصاحبة لكل سجل تسمى علامة المسح deletion marker عند مسح السجل فقط توضع قيمة محددة في هذه العلامة (مثلاً صفر تعني ان السجل محذوف والقيمة 1 تعني ان السجل موجود ) ، برامج البحث العاملة على مثل هذا النوع من الملفات التي تستعمل هذا النوع من الحذف تراعي هذا المؤشر. كلا هذين النوعين من آليات الحذف تحتاج ملفاتها لاعادة تنظيم دورياً (Reorganization) لإعادة استغلال المساحات الفارغة في الـ block .

● اعادة تنظيم الملفات تتم دورياً لاعادة استغلال المساحات الفارغة أثناء هذه العملية تتم قراءة الـ blocks على التوالي . ثم تجمع السجلات الموجودة في الـ blocks من جديد وتلغي السجلات المحذوفة. بعد اكتمال عملية اعادة التنظيم تكون المساحة بافضل استغلال .

يمكن بآلية أخرى لاعادة التنظيم ان نستعمل مكان السجلات المحذوفة للكتابة فيها مرة أخرى ، ولكن هذه العملية تحتاج لمعرفة الاماكن الفارغة التي يمكن ان نكتب بها بكل الـ blocks .

يمكن ان نستعمل spanned or unspanned organization كتظيم للملف غير المرتب (heap file ) كما يمكن ان تستعمل الـ fixed or variable length records .

عملية التعديل في الـ variable length records قد تتطلب مسح السجل القديم ثم إدخال السجل الجديد (بنهاية الملف) لان السجل الجديد قد يكون اكبر من القديم وبالتالي لاتسعه المساحة القديمة .

● لقراءة جميع السجلات بترتيب معين ننشئ نسخة جديدة من الملف مرتبة وعملية الترتيب للملف مكلفة كلما كان حجم الملف اكبر .

الملفات غير المرتبة اذا استعملت (fixed length records using unspanned blocks and contiguous allocation ) يكون الوصول الى أي سجل بموضعه في الملف فاذا كانت السجلات مرقمة 0,1,2,...,r-1 والسجلات داخل الـ block أيضا مرقمة 0,1,2,...,bfr-1 (حيث bfr هو الـ blocking factor ) : السجل رقم I في الملف يحدد موضعه كالآتي :

● في block رقم  $(I/bfr)$

● وداخل الـ block في سجل رقم  $(I \bmod bfr)$

هذا النوع من الملفات ولان السجلات يمكن الوصول اليها حسب موضعها يسمى relative or direct . file

### **:(II) Files of Ordered Record (Sorted files)**

يمكن ترتيب سجلات الملف فيزيائياً physically بناءً على قيمة احد حقول الملف ويسمى حقل الترتيب ordering field . وهذا يعطى ملفاً مرتباً ordered or sequential File فإذا كان حقل الترتيب هذا هو حقل مفتاحي في الملف key field أي لاتتكرر قيمته في اي سجل آخر فهو ( فريد او Unique )، نقول انه ordering Key . الشكل التالي يبين ملف مرتب وحقل المفتاح key field به هو الاسم وهو أيضاً حقل الترتيب ordering field اذن نقول ان الحقل name هو ordering key field .



	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
block 2	Acosta, Marc					
	Adams, John					
	Adams, Robin					
block 3	⋮					
	Albers, Jan					
	⋮					
block 4	Alexander, Ed					
	Allred, Bob					
	⋮					
block 5	Allen, Sam					
	⋮					
	⋮					
block 6	Allen, Troy					
	Anders, Keith					
	⋮					
block 7	Anderson, Rob					
	⋮					
	⋮					
block 8	Anderson, Zach					
	Angel, Joe					
	⋮					
block 9	Archer, Sue					
	⋮					
	⋮					
block 10	Arnold, Mack					
	Arnold, Steven					
	⋮					
block 11	Atkins, Timothy					
	⋮					
	⋮					
block n-1	Wong, James					
	Wood, Donald					
	⋮					
block n	Woods, Manny					
	⋮					
	⋮					
block n+1	Wright, Pam					
	Wyatt, Charles					
	⋮					
block n+2	Zimmer, Byron					
	⋮					
	⋮					

Figure(9) : Some blocks of an ordered (sequential) file of Employee Records With Name as the ordering Key field

الملفات المرتبة افضل من الملفات غير المرتبة للآتي :-

1. قراءة السجلات على حسب مفتاح الترتيب تكون سريعة جدا لاننا لانتاج لاجراء عملية ترتيب للبيانات.

2. الوصول الى السجل التالي من السجل الحالي على حسب ترتيب المفتاح، غالباً لا يحتاج لقراءة block اخر من ملف وذلك لان السجل في نفس الـ block الحالي ( إلا أن يكون السجل الحالي هو الاخير في الـ block ) . ( بحيث لا يوجد بعده سجل مخزن )

3. استعمال مفتاح الترتيب ordering key كشرط للبحث Search condition يؤدي لسرعة عملية البحث . وذلك عند استخدامنا لخوارزميه البحث الثنائي Binary search ، وهي أفضل من خوارزمية الـ linear search . ( والتي تقوم بالبحث داخل السجلات واحدا تلو الآخر ) .

يمكن اجراء عملية البحث الثنائي Binary search في الملفات على مستوى الـ Blocks بدلاً من السجلات لتحديد الـ block المطلوب ومن ثم نقوم بالبحث داخل الـ block للوصول الى السجلات . افترض وجود ملف مكون من عدد b من blocks مرقمة من 1,2,...,b بحيث تكون السجلات مرتبة حسب مفتاح الترتيب داخل هذه الـ blocks فاذا كنا نبحث عن سجل موظف يبدأ اسمه بالحرف k وبفرض أن عناوين الـ disk blocks موجودة في file header ، فان الخوارزمية التالية تصف عملية البحث الثنائي .  
خوارزمية البحث الثنائي :

Algorithm : Binary Search on an Ordering key of a disk file

$L \leftarrow 1; u \leftarrow b$  ; ( \* b is the number of file blocks \* )

While (  $u \geq L$  ) do

Begin  $i \leftarrow (L+u) \text{ div } 2$  ;

Read block i of the file into the buffer ;

if  $K < (\text{ordering key field value of the first record in block } i)$ .

then  $u \leftarrow i - 1$

else if  $K > (\text{ordering key field value of the last record in block } i)$

then  $L \leftarrow i + 1$

else if the record with ordering key field value = K is in the buffer then go to found

else go to not found

end;

go to not found

تقوم خوارزمية البحث الثنائي بقراءة عدد  $(\log_2 b)$  من الـ block في حالة وجود السجل ام لا . بينما في خوارزمية الـ linear search نقرأ عدد  $(b/2)$  blocks في المتوسط اذا وجد السجل وعدد b من الـ blocks اذا لم يوجد .

### مثال :-

ملف عدد الـ Blocks به تساوي block 120 ، أحسب عدد الـ block access إذا تم البحث عن سجل داخل الملف وذلك في حالة :-

1. الملف مرتب بناء علي حقل ترتيب.

2. الملف غير مرتب.

### الحل :-

1. الملف مرتب إذن نستخدم خوارزمية البحث الثنائي للبحث عن السجل :

$$\text{Block access} = \log_2 b = \log_2 120 = 7 \text{ block access}$$

2. الملف غير مرتب

في افضل الاحوال وفي حالة وجود السجل :

$$\text{block access} = b/2 = 120/2 = 60 \text{ block access}$$

في حالة المرور علي كل السجلات وعدم وجود السجل المطلوب :

$$\text{Block access} = b = 120 \text{ block access}$$

تتضمن طريقة البحث الـ شروط  $\geq$  ،  $\leq$  ،  $<$  ،  $>$

وذلك بناء على مفتاح الترتيب حيث تكون اكثر كفاءة . حيث أن الترتيب الحقيقي للسجلات يضمن أن كل السجلات التي تحقق شرط البحث تكون في blocks متتالية مثلا بالرجوع الى الشكل (9) اذا كان شرط البحث ( $G < \text{Name}$ ) بمعنى بالترتيب الهجائي (قبل الـ G) .. هذا يعني أن السجلات التي تحقق الشرط تكون من بداية الملف وقبل اول سجل يبدأ فيه حقل الاسم بالحرف G .

الترتيب على اساس حقل معين غير الحقل المفتاحي لايفيد عند اجراء عملية البحث الثنائي في الملف، بمعنى أنه اذا كان الملف مرتب بحقل الاسم ordering field Name وكنا نبحث بحقل المهنة job فان هذا الترتيب لا يفيد ، وعلينا عمل نسخة أخرى من الملف مرتبة حسب الـ job أو البحث بـ linear search في الملف القديم .

• في حالة ان الملف مرتب يمكننا اجراء عملية البحث الثنائي Binary search اما اذا كان

غير ذلك فيتم البحث عن طريق linear search .

### الإضافة والحذف في الملف المرتب :-

عملية الإضافة والحذف تعتبر مكلفة جدا في الملف المرتب لان السجلات يجب ان تبقى مرتبة في الملف عند الإضافة او الحذف physically ordered .

لإضافة سجل يجب اولاً أن نجد له المكان المناسب الصحيح في الملف بناء على قيمة الحقل المفتاحي له . ثم نوجد مساحة لإضافة هذا السجل وذلك بإجراء عملية إزاحة للسجلات، وهذه العملية تكون اصعب كلما زاد حجم الملف لانه في المتوسط نحتاج لإزاحة نصف السجلات لايجاد المكان الفارغ للسجل الجديد . وهذا يعني أن نصف عدد الـ blocks المخزنة تتم قراءتها من الـ disk ثم تعاد كتابتها مرة أخرى .

في عملية الحذف المشكلة اقل فيمكن أن نستعمل الـ **deleting marker** وهي علامة خاصة توضع في نهاية السجل للدلالة على حذفه. ودورياً نقوم باعادة التنظيم للملف reorganization للتخلص من السجلات المحذوفة .

ويمكننا لزيادة كفاءة عملية الاضافة ان نبقى على بعض المساحة غير المستغلة في كل block تحسباً لاضافة سجلات لاحقاً . ولكن هذا حل مؤقت فبمجرد امتلاء هذه المساحة الفارغة تظهر مشكلة الاضافة مرة اخرى .

هنالك طريقة اخرى وهي انشاء ملف مؤقت غير مرتب لنكتب فيه السجلات الجديدة ويسمى بملف الفيضان transaction or overflow file . وبينما يسمى الملف الاصلي بـ main or master file تدخل السجلات الجديدة في نهاية ملف الفيضان بدلاً من إدخالها في مكانها الصحيح حسب الترتيب في الـ main file ودورياً تتم عملية ترتيب لملف الفيضان ودمج الملف الرئيسي مع ملف الفيضان ليرتب الملف الجديد مرة أخرى ويتم هذا أثناء عملية إعادة التنظيم reorganization دورياً . وبهذا تصبح عملية الإضافة أكثر كفاءة ... ولكن في المقابل زاد التعقيد في عملية البحث ، حيث تكون عملية البحث كالآتي :-  
ابحث في الملف الرئيسي (المرتّب) باستخدام خوارزمية binary search .  
إذا وجد السجل المطلوب

- اذهب الى النهاية .

- والا ابحث في ملف الفيضان ( غير المرتب) باستخدام خوارزمية linear search

#### عملية التعديل في الملف المرتب :

تعديل قيمة حقل ما في ملف مرتب تعتمد على عاملين :-

1. شرط البحث للوصول الى السجل .

2. الحقل المراد تعديله .

فإذا كان شرط البحث يعتمد على حقل المفتاح يمكن الوصول الى السجل باستخدام خوارزمية الـ binary search وفيما عدا ذلك يجب ان تجري خوارزمية الـ linear search .

اما اذا كان الحقل المراد تعديله حقل غير ترتيبى non ordering field فهو يعدل ثم تعاد كتابته في نفس موقعه في الملف the same physical location باعتبار أننا نعمل على fixed length record أما تعديل حقل الترتيب نفسه قد يستلزم أن السجل يجب أن يعدل موضعه الحقيقي في القرص مما يتطلب مسح السجل القديم واطافة السجل الجديد.  
**عملية القراءة :**

بتجاهل ملف الفيضان تكون عملية القراءة سريعة جدا وذلك لان الـ blocks تقرأ بالتتالي باستعمال الـ double buffering . أما بأخذ ملف الفيضان في الحسبان يجب ان :-

1. ترتيب ملف الفيضان Overflow أولاً .

2. دمج الملف الرئيسي مع ملف الفيضان للحصول على ملف جديد مرتب مع مراعاة حذف

السجلات المؤشرة . ( وهي السجلات المحذوفة سابقاً وبها علامة Deletion marker ) .

## الدرس السادس

### تقنية البعثة Hashing Technique

إحدى طرق تنظيم الملفات الأساسية ترتكز على فكرة ال hashing والتي توفر سرعة في الوصول للسجلات بناءً على شرط بحث معين وهذا النوع من التنظيم يعرف بالـ hash file أو ال Direct file.

شرط البحث يجب أن يكون شرط تساوي اعتماداً على حقل واحد من حقول السجل يعرف بالـ Hash field للملف. وفي أغلب الحالات يكون ال Hash field أيضاً هو حقل مفتاحي key field للملف وفي هذه الحالات يعرف بالـ hash key.

الفكرة الأساسية لل hashing هي وجود الدالة h والتي تعرف بالـ hash function أو ال Randomize function ، تستخدم الدالة قيمة ال hash field للسجل لإنتاج عنوان ال block الذي سيتم تخزين السجل فيه على القرص . أما البحث عن السجل في ال block المعين فيتم في الذاكرة الرئيسية main memory buffer ، مع معظم السجلات نحتاج فقط للوصول الى block واحد Single –block access للوصول للسجل المعنى .

#### **: Internal hashing**

ويتم تطبيق ال hashing هنا في صورة جدول يسمى بـ hash table من خلال استخدام مصفوفة سجلات array of record . افترض ان المصفوفة مكونة من عدد M record بالترقيم 0,1,...,M-1 ولدينا عدد M خانة بالعناوين المقابلة لعناوين المصفوف، نختار دالة Hash function المناسبة لتحويل قيمة ال hash field لقيمة صحيحة Integer تقع في المدى بين 0...m-1 ، وواحدة من اشهر دوال ال Hash function الشائعة الدالة :

$$h(k) = k \text{ Mod } M$$

التي تعيد باقي قسمة العدد K على العدد M . وهي القيمة التي تستخدم كعنوان للسجل .

(a)

	NAME	SSN	JOB	SALARY
0				
1				
2				
3				
M-2				
M-1				

(b)

	data fields	overflow pointer	
0		-1	address space
1		M	
2		-1	
3		-1	
4		M+2	
M-2		M+1	overflow space
M-1		-1	
M		M+5	
M+1		-1	
M+2		M+4	
M+O-2			
M+O-1			

- null pointer = -1.
- overflow pointer refers to position of next record in linked list.

(10): Internal hashing data structure

(a) Array of M position for use in internal hashing (b) Collision resolution by chaining records

بالنسبة للأرقام الغير صحيحة للـ hash field كالحروف مثلاً يتم تحويلها لأرقام صحيحة integer قبل تطبيق دالة الـ Mod باستخدام الـ ASCII code المقابل لها .

### مثال :

الـ K hash field لسجل نوعية بياناته حرفيه من 20 حرف string of 20 characters ،  
K = array [ 1 ..20 ] of char ؛ ، الخوارزمية التالية تستخدم لحساب الـ hash address (الدالة  
تعيد الـ ASCII code للحرف ) :

Algorithm (a) : applying the mod hash function to a character string K:

Temp  $\leftarrow$  1 ;

For i : 1 to 20 Do

Temp  $\leftarrow$  Temp \* code (k [i]) mod m ;

Hash -address  $\leftarrow$  Temp Mod M ;

وتوجد انواع عديدة لدوال الـ hash function تقوم بحساب العناوين بطرق مختلفة.

### **حدوث التصادم :-**

مشكلة اغلب هذه الدوال انها لاتضمن للقيمة الناتجة عنوان فريد (لايتكرر) خاص بها وذلك لان الـ hash  
field space (وهو مدى القيم التي يمكن أن تاخذها الـ hash field ) دائماً اكبر من الـ address  
space ( وهو عدد العناوين المتاحة لتخزين السجلات ) مما يؤدي لحدوث التصادم او collision عندما  
تنتج الدالة بناء على قيمة hash field لسجل معين قيمة، وهذه القيمة تمثل عنوان للتخزين يحتوي على  
سجل آخر . في هذه الحالة يجب ايجاد موقع جديد ، عملية ايجاد موقع جديد للسجل تعرف بالـ  
Collision Resolution حيث توجد طرق عديدة للـ collision Resolution ومنها :-

1- open addressing

2-chaining

3-multiple hashing

الخوارزمية (b) توضح إحدى طرق Collision resolution :

Algorithm (b) : Collision resolution by open addressing :

i  $\leftarrow$  hash \_ address (k) ; a  $\leftarrow$  i ;

If location I is occupied then

Begin

i  $\leftarrow$  (i+1) Mod M ;

While (i  $\neq$  a ) and location i is occupied do

i  $\leftarrow$  (i+1) Mod M ;

If (i = a ) then all position are full

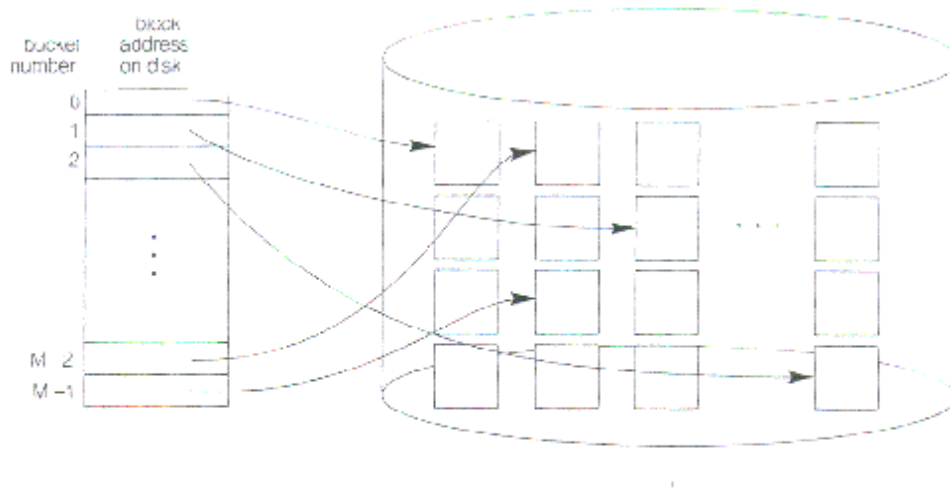
Else new \_ hash \_ address  $\leftarrow$  i ;

End ;

الغرض من الـ hashing function الجيدة توزيع السجلات بصورة مثلى على العناوين المتوفرة  
وتقليل التصادم collision بالإضافة لتقليل حجم المواقع غير المستغلة.

### External Hashing for disk files: -

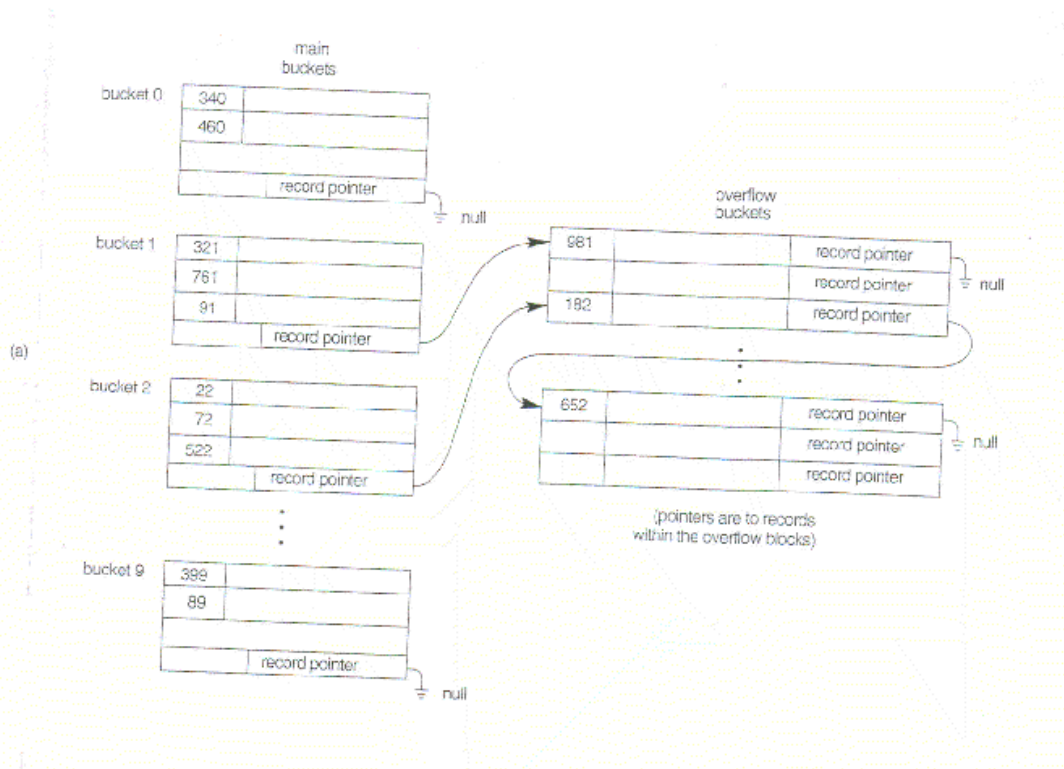
عملية الـ Hashing علي القرص تسمى بـ External Hashing وللملائمة خصائص القرص تم  
عنونة المساحات عن طريق ما يعرف بـ Buckets . الـ Bucket عبارة عن block واحد أو  
مجموعة من الـ blocks المتجاورة من السجلات Records .  
(A bucket is either one disk or a cluster of continues blocks)  
تقوم دالة الـ Hashing function بإنتاج key وربطه برقم الـ bucket الذي له علاقة برقم الـ  
key الناتج.  
ويوجد جدول بالـ file header لتحويل رقم الـ bucket لرقم أو عنوان الـ disk block  
المناظر .  
شكل رقم 11 .



Figure(11) : Matching Bucket numbers to disk block addresses

ويقل حدوث مشكلة التصادم collision مع الـ buckets وذلك لان العديد من الـ blocks يمكن أن  
توضع في الـ bucket من غير حدوث مشاكل (اكثر من block لها عنوان واحد يمثل عنوان الـ  
block المخزنة داخله ) وفي حالة امتلاء الـ bucket ووصول block جديد ليضاف للـ bucket  
يمكن استخدام الـ variation of chaining حيث يتم استخدام مؤشر pointer موجود بالـ  
bucket يشير لسلسلة متصلة من السجلات الفائضة عن bucket (overflow records) .  
المؤشرات في السلسلة عبارة عن مؤشرات لسجلات ( record pointers ) هذه المؤشرات تحوي عنوان  
الـ block وموقع السجل داخل الـ block .





Figure(12) :Handling overflow for buckets by chaining

تمكننا تقنية الـ hashing من سرعة الوصول والاسترجاع للسجلات بناءً على قيمة الـ hash field الخاص بها .

الوصف السابق لتقنية الـ hashing يسمى بالـ static hashing حيث يتم تسكين عدد محدد  $m$  من الـ buckets وهذا يعتبر مشكلة خطيرة عند التعامل مع الملفات ذات الأحجام المتغيرة Dynamic File . افترض انه تم تسكين عدد  $M$  buckets وكل bucket يمكن أن يسع  $m$  record إذا عموماً سيكون لدينا عدد  $(m * M)$  لملء كل  $M$  buckets .

وإذا قل عدد السجلات records عن الـ  $(m * M)$  ستظهر العديد من المساحات غير المستقلة ومن الناحية الأخرى إذا زاد عدد السجلات record عن  $(m * M)$  يمكن أن يحدث الـ (Collisions) . أيضاً عملية استرجاع السجلات سيصبح أبطأ بسبب القوائم الطويلة من سجلات الفيضان overflow records ، إذن في كلا الحالتين نحتاج لتعديل عدد  $M$  من الـ bucket واستخدام hashing function جديدة لتوزيع السجلات .

عمليات إعادة التنظيم هذه تستغرق وقت طويل بالنسبة للملفات الطويلة. توجد عمليات تنظيمية جديدة للـ dynamic file باستخدام تقنية الـ hashing تسمح بزيادة عدد الـ buckets ديناميكياً باستخدام عمليات إعادة تنظيم محليه localized reorganization .

عملية البحث عن سجل عند استخدام الـ external hashing عن طريق أحد الحقول الأخرى غير حقل الـ hash field تكون عملية مكلفة جداً . وذلك كما يحدث في حالة الـ unordered file .

حذف السجل يتم بحذفه من الـ buckets اما اذا كان للـ buckets سلسلة فيضان overflow chain يمكن نقل احد سجلات الفيضان الى الـ bucket ليحل محل السجل المحذوف . اما اذا كان السجل المراد حذفه عبارة عن سجل فيضان overflow record فيمكن حذفه بسهولة من السلسلة وتعديل السلسلة المتصلة .

اما بالنسبة لعملية تعديل حقل في سجل فاننا نعلم ان التعديل يعتمد على عاملين : شرط البحث والحقل المراد تعديله .

فاذا كان شرط البحث هو (مساواة) مع الـ hash field فذلك يتيح الوصول بكفاءة للسجل باستخدام الـ hashing field والا فيجب في غير ذلك استخدام الـ linear search .

أما تعديل الـ non hash field يتم بتعديل السجل واعادة كتابته مرة أخرى في نفس الـ bucket بينما تعديل الـ hash field فيعني ان السجل يمكن ان ينتقل الى bucket اخرى وهذا يتطلب حذف السجل القديم واطرافه السجل المعدل .

كما ذكرنا سابقا ان العيب الاساسي للـ static hashing يكمن في ان الـ hash address space محدودة مما يجعل عملية التغيير بالتمدد او الانكماش في حجم الملف صعب للغاية . ويوجد حلان لهذه المشكلة :

- 1-Extendable hashing .
- 2-Linear hashing .

## الدرس السابع

### الفهارس indexes

في هذا الفصل نفترض ان هناك ملف موجود صلاً بتركيب معروف مثل الملف المرتب او غير المرتب .  
وهنا نقوم بدراسة هيكل بيانات جديد مساعد يسمى الفهرس Index والذي يساعد على زيادة سرعة  
استرجاع البيانات في الملف بناء على شروط بحث محدد.

الفهرس يوفر طريق اضافي للوصول الى البيانات دون ان يؤثر على الوضع الطبيعي او الحقيقي  
physical للبيانات في الملف على القرص ، وانما فقط يمكن الوصول السريع للسجلات بناء على مفتاح  
الفهرسه indexing fields ، وحقيقة أي حقل في الملف يمكن ان يستعمل لإنشاء فهرس أو فهارس  
متعددة multiple indexes وبهذا قد يكون هناك العديد من الفهارس على مختلف الحقول في الملف  
الواحد .د

للاوصول الى سجل او سجلات محددة في الملف معتمدين على شرط بحث محدد وباستخدام الفهرس المناسب  
indexing field ، يجب أولاً ان ندخل الى الفهرس نفسه والذي بدوره يشير الى كتلة او مجموعة من  
الكتل في الملف one or more blocks والتي تحوي السجل المطلوب.  
أقوى أنواع الفهارس تعتمد على الملف المرتب (single level indexes) . كما يمكن للفهارس ان تبني  
باستخدام البعثرة hashing او أي هيكل بيانات يساعد على عملية البحث .

وفيما يلي ندرس مختلف انواع الـ single level ordered index وهي ، clustering  
primary , secondary و بالنظر الى أن الـ single level index كلف مرتب يمكن أن نعمل  
عليه هو نفسه فهرس محققين بذلك فكرة الـ multi level index وهناك طريقة فهرسه مشهورة تسمى (   
Indexed Sequential Access Method ) ISAM تعتمد على هذه الطريقة.

#### - Types of Single – level Ordered Index

فكرة الفهرس المرتب تشبه تماماً فكرة الفهرس بنهاية الكتب حيث نجد المصطلحات الواردة في الكتاب مرتبة  
حسب الحروف الهجائية مع رقم الصفحة لكل مصطلح . وفي حالة عدم وجود الفهرس بالكتاب نضطر للبحث  
في كل كلمات الكتاب حتى نصل إلى المصطلح المطلوب وهذه العملية مثل البحث الخطي linear search  
في الملف .

يحتوي الملف على عدة حقول ، يُعَوَّف الفهرس على حقل واحد فقط ويسمى حقل الفهرسة indexing  
field or attribute ، الفهرس نفسه عبارة عن هيكل بيانات (يمكن ان يكون ملف ) تخزن به أي قيم ،  
حقل الفهرسة معه مؤشرات pointers للكتل disk blocks التي تحوي سجلات لها قيمة حقل الفهرسة

القيم بالفهرس مرتبه (قيم حقل الفهرسة) ، وحجم الفهرس نفسه اقل بكثير من حجم ملف البيانات ولهذين  
السببين يمكن استعمال البحث الثنائي binary search في الفهرس .  
هناك عدة أنواع من الفهارس المرتبة وهي :

1. Primary index.
2. Clustering index.
3. secondary index.

### 1. Primary index :

هو عبارة عن ملف مرتب مكون من حقلين:

الاول من نفس نوع بيانات المفتاح الاساسي لملف البيانات data file وهو ال primary key .  
 الثاني هو المؤشر pointer والذي يُوْشر الى (block address (disk block).  
 ولهذا فهناك سجل واحد فقط ، index entry في ملف الفهرس مقابل كل كتلة block من ملف البيانات  
 أي سجل في الفهرس index entry يحوي بحقل المفتاح الاساسي قيمة حقل البحث لاول سجل بالكتلة  
 التي يُوْشر لها ، ومؤشر لتلك الكتلة block وهو عبارة عن عنوان هذا ال block . وفيما يلي نرسم  
 لسجل الفهرس (i) index entry ، (i) كآلاتي :

$$(k(i), p(i))$$

الشكل رقم (13) يوضح مثلاً لفهرس أُولى primary index بإعتبار ان حقل الاسم name هو المفتاح  
 الاولي primary key لملف البيانات وبفرض ان الاسم لايتكرر unique ، تم اختيار الاسم ليكون هو  
 primary key للفهرس

□ بالفهرس قيمة الاسم والمؤشر لكل كتلة ، وكمثال لاول ثلاث سجلات في الفهرس index entry

$$\begin{aligned} k(1) &= (\text{Aaron}, \text{Ed}) , & p(1) &= \text{address of block 1} \\ k(2) &= (\text{Adams}, \text{John}) , & p(2) &= \text{address of block 2} \\ k(3) &= (\text{Alexander}, \text{Ed}) , & p(3) &= \text{address of blocks 3} \end{aligned}$$

الشكل رقم (13) يوضح هذا الفهرس وبيانات الملف ، عدد السجلات بالفهرس يساوي عدد الكتل بالملف .

السجل الاول باي كتلة يسمى السجل الخطأ في للكتلة او anchor record .

dense او كثيفه sparse الفهارس يمكن تصنيفها الى خفيفه

Dense Index(a) : لها سجل index entry لكل سجل في ملف البيانات .

Sparse Index : وايضاً تسمى non dense index وهي لها سجل index entry واحد فقط

لبعض السجلات في ملف البيانات ، وبهذا فالـ primary index هو من هذا النوع لان لها index

entry لكل block في ملف البيانات .

ملف الفهرس يكون حجمه اقل بكثير من حجم ملف البيانات وذلك لسببين :-

1. عدد الـ index entries به اقل من عدد السجلات بملف البيانات .

2. حجم الـ index entry اقل من حجم السجل في ملف البيانات لان به فقط حقلين هما المفتاح

والمؤشر .

ولهذا فان ملف الفهرس يمكن ان يخزن في عدد بسيط من الكتل disk blocks مقارنة بعدد الكتل لملف

البيانات وبهذا تكون عملية البحث الثنائي في ملف الفهرس اكثر كفاءة لان حجمه صغير .

## البحث في الـ primary Index :-

السجل الذي يكون مفتاحه الأساسي يساوي  $k$  يقع في الكتلة  $\text{block}$  والذي عنوانه  $(i, p)$  حيث  $k(i) \leq k < k(i+1)$  والكتلة رقم  $I$  تضم بملف البيانات كل السجلات التي تكون قيمة مفتاحها قريبه من الـ  $k$  وبهذا نسبة للترتيب الحقيقي للملف  $\text{physical order}$  .  
وللوصول الى السجل بمعلومية القيمة  $k$  بحقل المفتاح نقوم بالبحث الثنائي  $\text{binary search}$  في ملف الفهرس نجد الـ  $(i, \text{index entry})$  المناسب وبالتالي نصل إلى الكتلة  $\text{disk block}$  والتي عنوانها  $(i, p)$  .  
مثال رقم (1) :

افترض أن لدينا ملف مرتب مكوّن من عدد  $r=30,000$  سجل مخزن في قرص حجم الـ  $\text{block}$  به يساوي  $B=1024\text{byte}$  . السجلات به من النوع  $\text{fixed Size \& Unspanned}$  وطول السجل  $R=100\text{ byte}$  .  
الـ  $\text{blocking factor}$  تساوي:

$$\text{bfr} = (B/R) = 1024 / 100 = 10 \text{ Record/block}$$

عدد الكتل  $\text{blocks}$  التي تكفي الملف:

$$B = [r / \text{bfr}] = [30,000 / 10] = 3000 \text{ blocks}$$

□ البحث الثنائي في ملف البيانات يحتاج لـ

$$[\log_2 b] = [\log_2 3000] = 12 \text{ block access}$$

والآن نقوم بالحساب اذا استعملنا الفهرس ، فنفرض أن حقل الترتيب حجمه  $v=9\text{ bytes}$  ، المؤشر حجمه  $p=6\text{ bytes}$  وقمنا بإنشاء فهرس للملف تكون تفاصيله كالآتي :-

$$\text{index entry } R_i = 9 + 6 = 15 \text{ byte}$$

وبهذا يكون الـ  $\text{blocking factor}$  يساوي :

$$\text{bfr} = 1025 / 15 = 68 \text{ entries/block}$$

والعدد الكلي للـ  $\text{index entries}$  يساوي عدد الـ  $\text{blocks}$  بملف البيانات وهو يساوي  $r_i = 3000$

□ عدد الـ  $\text{blocks}$  التي نحتاجها:

$$b_i = [r_i / \text{bfr}_i] = 3000 / 68 = 45 \text{ block}$$

ولكي نقوم بالبحث الثنائي في هذا الفهرس نحتاج لـ :

$$[\log_2 b_i] = [\log_2 45] = 6 \text{ block access}$$

إضافة للـ  $\text{block}$  الذي يحتوي على البيانات نفسها .

$$\text{العدد الكلي} = 6 + 1 = 7 \text{ block accesses}$$

□ تم تحسين النتيجة من 12 الى 7 .

المشكلة الأساسية في الـ  $\text{primary index}$  هي مشكلة كل الملفات المرتبه وهي مشكلة إدخال وحذف

سجل ، وهنا المشكلة مركبه حيث يظهر اثرها على :

● ملف البيانات حيث تحرك البيانات ، للمحافظة على الترتيب بعد عملية الحذف أو الاضافه .

• ملف الفهرس حيث تدخل **entries** جديدة ويعدل الفهرس بما يتوافق مع البيانات المدخلة او المحذوفه .  
 ايضا تحل هذه المشاكل بإستعمال ملف الفيضان **over flow** او استعمال **linked list of over flow** record لأي **block** في ملف البيانات . وحذف البيانات بإستعمال الـ **deletion marker** .

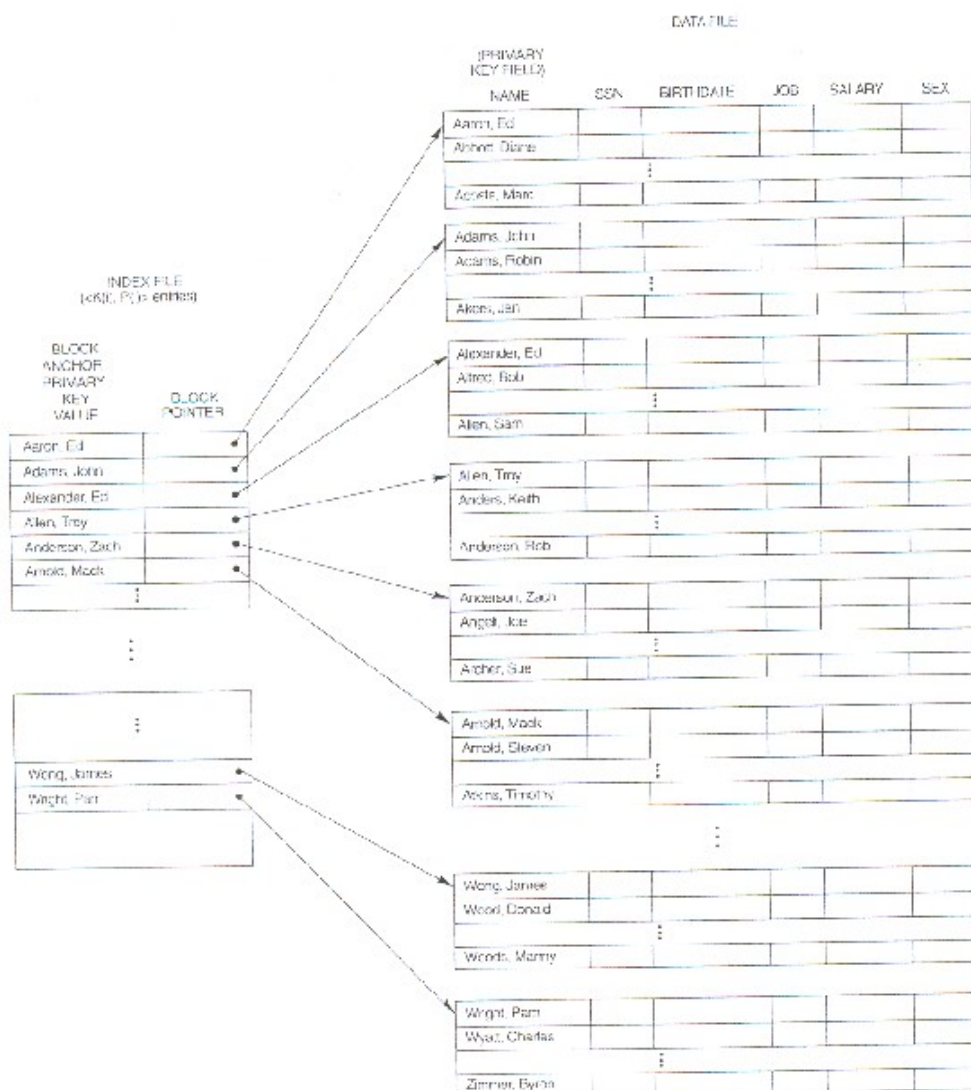


Figure 6.1 Primary index on the ordering key field of the file shown in Figure 5.9.

Figure (13): Primary Index on the ordering key field of the file

## : clustering Index (2)

إذا كانت السجلات بالملف مرتبة فيزيائياً **physically** على أساس حقل غير مفتاحي **non key field** والذي ليس له قيمة فريدة لأي سجل ، ويسمى هذا الحقل **clustering field** .  
 يمكن ان ننشئ نوع جديد من الفهارس يسمى **clustering index** ليتوافق مع هذا النوع .  
 هذا النوع يختلف عن الـ **primary Index** بأنه ليس له قيمة فريدة **unique** لكل سجل بملف البيانات  
 الـ **clustering Index** هو ملف مرتب مكون من حقلين :  
 \* الاول : من نفس نوعية بيانات الـ **clustering field** بملف البيانات .

\* الثاني : مؤشر block pointer

وهناك مدخل واحد (سجل واحد) one entry في الفهرس لكل قيمه منفصلة Distinct value لحقل clustering field ويحتوي علي :-

1. القيمة

2. مؤشر لاول block في الملف يحوي هذه القيمة في حقل الـ clustering field

الشكل رقم (14) يوضح مثال لـ clustering index .

نلاحظ أن الملف هنا أيضاً فيزيائياً مرتب physically وبهذا فهو يعاني من مشاكل الحذف والإدخال

ولكن لتلافى مشكلة الإدخال نخصص مجموعة من الـ blocks او cluster لكل قيمه جديده للـ

clustering field أي لانبداً القيمة الجديدة للـ clustering field الا في كتله (block) جديد وهذا يوضحه الشكل رقم (15) .

الـ clustering index هو مثال جديد لـ parse or non dense index ، لان له مدخل entry

في الفهرس لكل قيمة وحيدة Distinct value في ملف البيانات (بدلاً عن لكل سجل من ملف البيانات

كما في الـ primary Index )



Figure 6.2 A clustering index on the DEPTNUMBER ordering nonkey field of an EMPLOYEE file.

Figure(14) :A clustering index on the DeptNumber ordering non key field of an Employee File

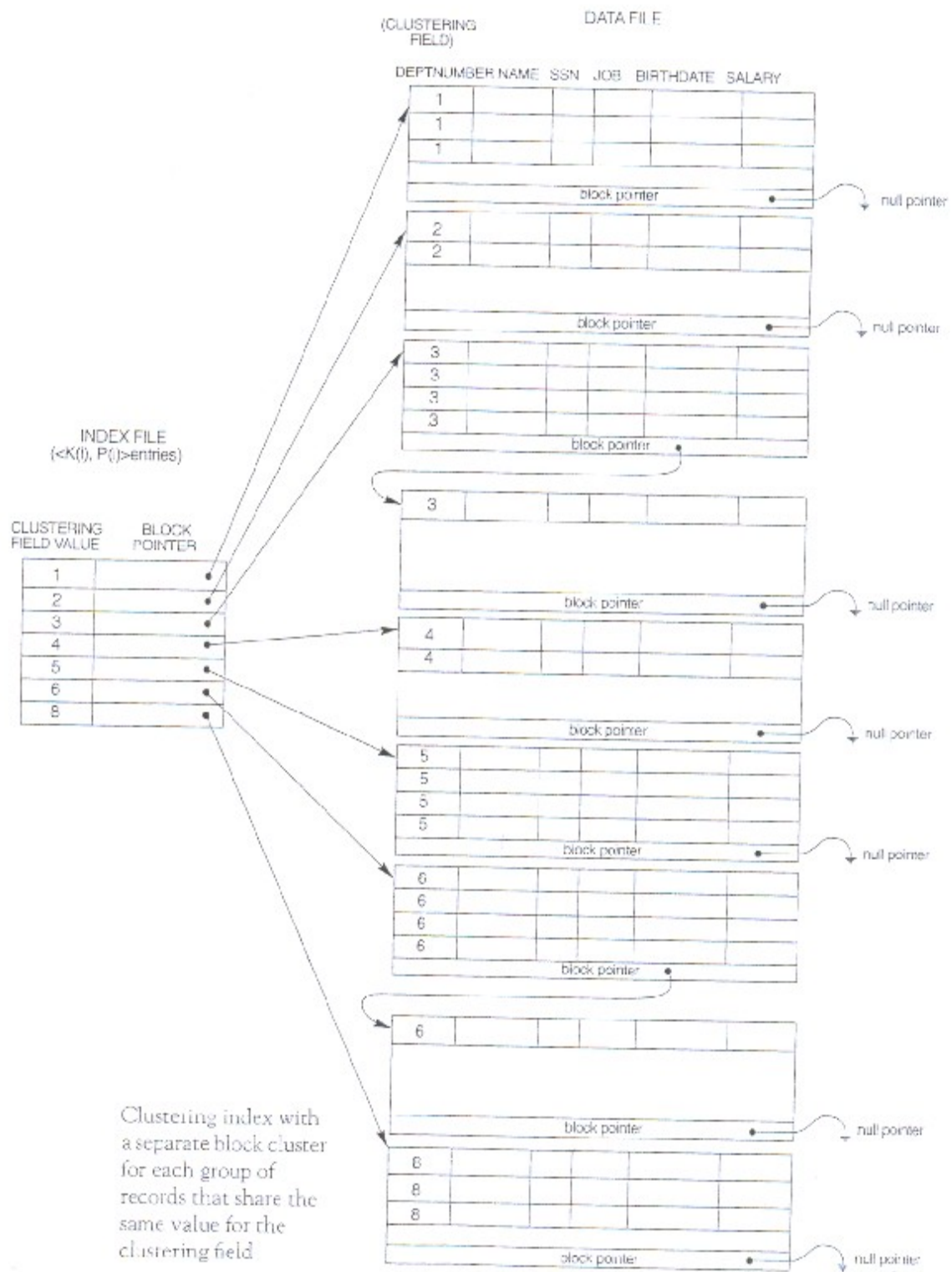


Figure (15)



### (3) secondary Index :

هو أيضاً ملف مرتب من حقليين:

الأول : حقل فهرسة indexing field هو من نفس نوع بيانات أحد الحقول بملف البيانات غير الحقل

الترتيبي بالملف non ordering field of the data file .

الثاني : مؤشر اما لكتلة block pointer او لسجل record pointer .

يمكن أن يكون لملف البيانات الواحد عدة فهرس ثانويه many secondary indexes .

ويمكن أن ينشأ فهرس ثانوي secondary Index على حقل مفتاحي key field والذي له قيمه فريده Distinct value لكل سجل بالملف.

وفي هذه الحالة يسمى الحقل secondary key .. و في هذه الحالة فهناك سجل فهرس index entry لكل سجل بيانات في ملف البيانات . ويتكون هذا ال Index entry من حقليين:

الأول: secondary key

والثاني: مؤشر للكتلة block التي يوجد بها هذا السجل او للسجل نفسه وبالتالي فهو Dense index .

مرة اخرى نرمز لمكونات سجل الفهرس index entry بالزوج  $p(i) > k(i)$  وسجلات الفهرس هذه مرتبه حسب قيمة الـ  $k(i)$  حيث يمكن إجراء بحث ثنائي Binary search عليها .

ولان السجلات بملف البيانات ليست مرتبه not physically order حسب الـ secondary key

فلا يمكن أن نستعمل block anchors بل يجب أن هناك مؤشر لكل سجل في ملف البيانات (وليس لكل

block كما في الـ primary index) . الشكل رقم ( 16 ) يوضح secondary index

والذي به المؤشرات هي block pointer ( نلاحظ ان هناك مؤشر لكل سجل في ملف البيانات ولكن هذا المؤشر يؤشر للـ block الذي يحوي هذا السجل ) .

وعند الوصول للـ block ونقله للذاكرة يتم البحث عن السجل المطلوب هناك .

الفهرس الثانوي secondary index يحتاج لمساحة تخزينية أكبر و زمن البحث فيه اطول مقارنة

بالفهرس الاولي primary index وهذا لان عدد المدخلات entries فيه اكبر .

ولكن وكما سنوضح في المثال التالي يكون التحسن في البحث كبيراً جداً عند استعمال الـ secondary index والمقارنة التالية توضح ذلك .

#### مثال رقم (2) :

بالرجوع للملف في مثال رقم (1):

-  $r = 30,000$

- Record size  $R = 100$  byte block size  $B = 1024$

- Number of blocks needed  $b = 3000$  block

• عند اجراء بحث خطي linear search (لان الملف غير مرتب ) نحتاج

$$b / 2 = 3000 / 2 = 1500 \text{ block accesses}$$

• باستعمال secondary index وكان حجم Pointer size ، non ordering key  $V = q$

$$p = b \text{ bytes}$$

$$\square \text{ entry size} = (9 + 6) = 15 \text{ byte}$$

$$- bfr_i = \lfloor (B / R_i) \rfloor = 1024 / 15 = 68 \text{ entries per block}$$

وحيث ان الفهرس من النوع الكثيف dense index ، إذن يكون عدد (ri) index entries

$$r_i = 30,000 \text{ يساوي عدد السجلات بملف البيانات}$$

□ يكون عدد الـ blocks التي نحتاجها لتخزين الـ index

$$b_i = \lceil (r_i / bfr_i) \rceil = \lceil (30,000 / 68) \rceil = 442 \text{ blocks}$$

وباستعمال البحث الثنائي على هذا الفهرس نحتاج لـ:

$$\log_2 b_i = \log_2 442 = 9 \text{ block access}$$

اضافة لكتلة block من ملف البيانات نفسه المحتوية على السجل فيكون العدد الكلي

$$\text{secondary index block access} \quad 9 + 1 = 10 \text{ مقارنة بـ } 1500 \text{ بدون استعمال}$$

ملخص المقارنه بين انواع الفهارس :-

جدول رقم (1)

Non ordering field	Ordering	Field type
(Secondary index (non key	Primary index	Key field
(Secondary index (non key	Clustering index	Non key field

جدول رقم (2)

Block anchoring on Data file	Dense or Non Dense	Number of (first –level ) index entries	Type of index
Yes	Non dense	Number of blocks in data file	Primary
Yes/ no(1)	Non dense	Number of distinct index field values	Clustering
No	Dense	Number of records in data field	Secondary (key)
No	Dense or Non dense	Number of record (2) or Number of distinct Index field values(3)	Secondary( non key)

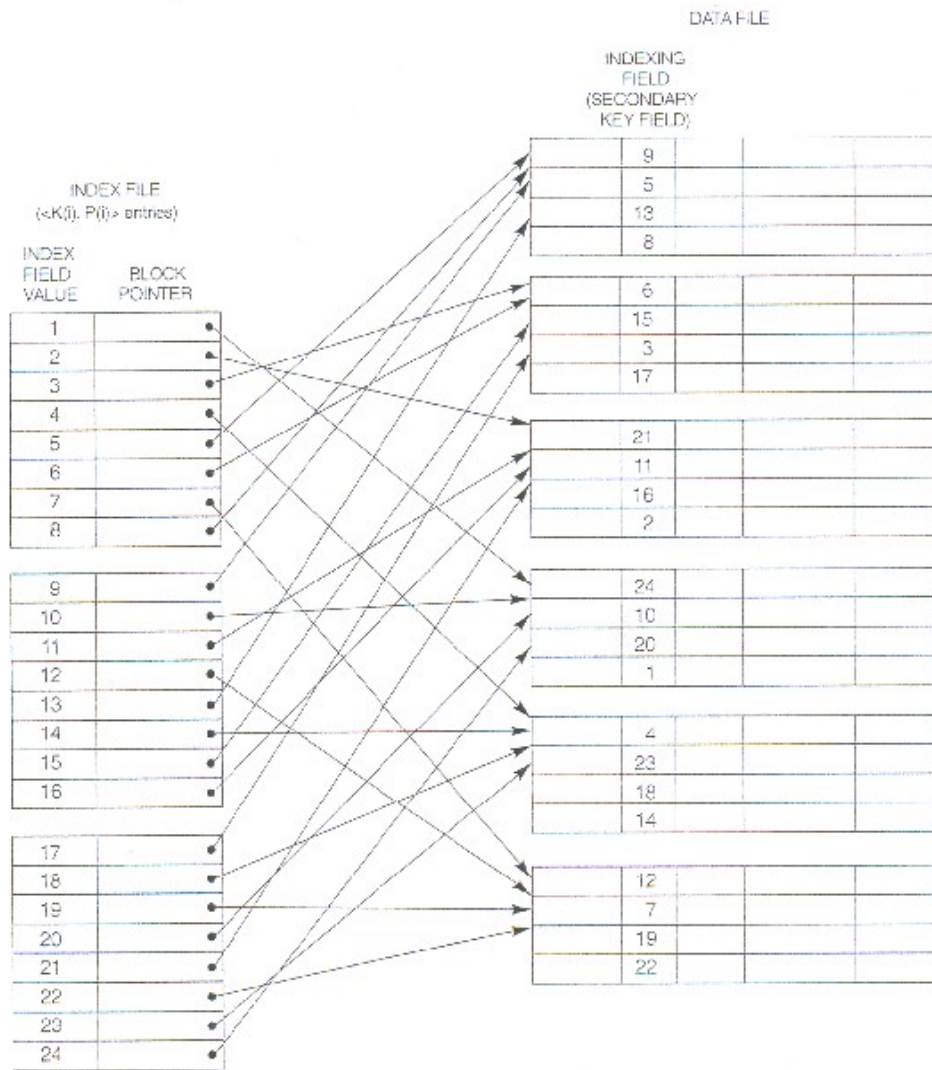
Property indexes types :

1- yes : if every distinct value of the ordering field starts anew block .

no : other wise

2- option 1

3- options 2 and 3



Figure(16): A dense secondary index (with block pointers) on a non ordering key field of a file