**DB 期末**

CH08

Seven-step algorithm to convert the basic ER model constructs into relations

Step 1: Mapping of Regular(Strong) Entity Types.

Step 2: Mapping of Weak Entity Types

include as foreign key attributes of R the primary key attribute(s)

of the relation(s) that correspond to the owner entity type(s).

Step 3: Mapping of Binary 1:1 Relation Types

three possible approaches :

Foreign Key approach : one table has a pointer to the other table (totally participation)

Merged relation option(both participation are total) : merged two table into one table

Cross-reference or relationship relation option : create third table.

Step 4: Mapping of Binary 1:N Relationship Types.

Include as foreign key on the N side that references to the other one

Step 5: Mapping of Binary M:N Relationship Types.

create a new relation S to represent the relation

Step 6: Mapping of Multivalued attributes.

For each multi-valued attribute A, create a new relation R.

Step 7: Mapping of N-ary Relationship Types

For each n-ary relationship type R, where n>2, create a new relationship S to represent R

Step8: Options for Mapping Specialization or Generalization.

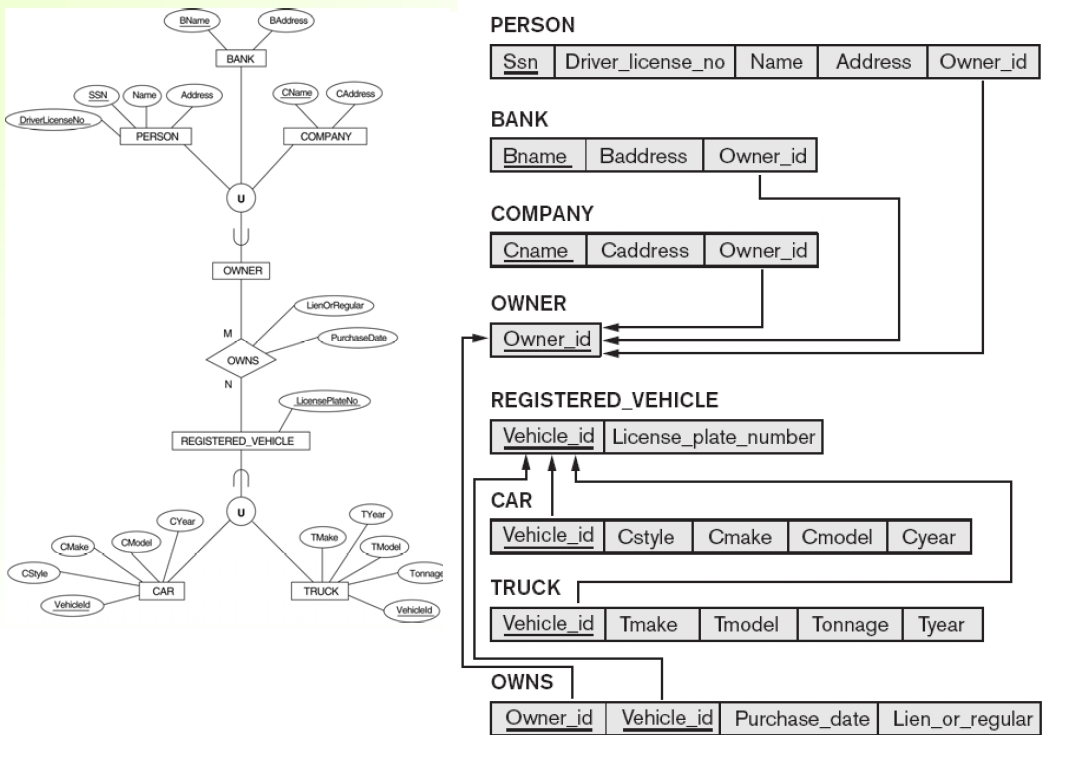
Option 8A: Multiple relations-Superclass and subclasses

This option works for any specialization

Option 8B: Multiple relations-Subclass relations only(Total Disjoint)

Option 8C: Single relation with one type attribute(Disjoint)

Option 8D: Single relation with multiple type attributes(Overlapping || Disjoint)

Step 9: Mapping of Union Types (Categories).

CH14

Bottom-up design methodology (design by synthesis)

分析需求

Top-down design methodology (design by analysis)

通通變一個 再慢慢分開

**Database Design Goals**

Information Preservation

Minimum Redundancy

**Measures of quality**

Making sure attribute semantics are clear

Reducing redundant information in tuples

Reducing NULL values in tuples

Disallowing possibility of generating spurious tuples

**Guideline 1**

Design relation schema so that it is easy to explain its meaning

Do not combine attributes from multiple entity types and relationship types into a single relation

**anomalies**

1.insertion

2.deletion

3.update

**Guideline 2**

Design base relation schemas so that no update anomalies are present in the relations

Note anomalies clearly

Make sure that the programs that update the database will operate correctly

**Problems with NULLs**

1.Wasted storage space

2.Problems understanding meaning

3.The results become unpredictable with JOIN and aggregation operations

**Guideline 3**

Avoid placing attributes in a base relation whose values may frequently be NULL

**Guideline 4**

The relations should be designed to satisfy the lossless join condition.

No spurious tuples should be generated by doing a natural-join of any relations

**Functional Dependency (FDs)**

Formal tool for analysis of relational schemas

Definition :

A set of attributes X functionally determines a set of

attributes Y if the value of X determines a unique value for Y

**Normalization process**

The process of decomposing unsatisfactory "bad" relations

by breaking up their attributes into smaller relations

**Decomposition Goals**

Lossless join property (Nonadditive Join Property)

Dependency preservation property

**Prime attribute** : the attribute in candidate key

**Nonprime attribute** : not a prime attribute

**First Normal Form**

Disallows

composite attributes

multivalued attributes

nested relations

Only attribute values permitted are single **atomic values**

**Achieve first normal form**

1.place in separate relation along with the primary key

2.Expand the key

3.use several atomic attributes

**Full functional dependency**:

a FD X → Y where removal of any attribute A from X means that the FD does not hold any more

**Partial dependency :**

if some attribute A ∈ X can be removed from X and the dependency still holds

**Second Normal Form**

Every non-prime attribute A in R is fully functionally dependent on the primary key of R

**Third Normal Form**

Satisfies 2NF and Not allow transitive functional dependency in a table

**1st normal form**

All attributes depend on the key

**2nd normal form**

All attributes depend on the whole key

**3rd normal form**

All attributes depend on nothing but the key

**BCNF**不允許 attribute dependent on non-prime attribute 。

CH15

**Binary Decomposition**

Decomposition of a relation R into two relations.

**Property NJB** (Nonadditive join test for binary decompositions)

A decomposition D = {R1, R2} of R has the lossless join property with respect to a set of

functional dependencies F on R if and only if either

The FD ((R1 ∩ R2) R1- R2)) is in F+, or

The FD ((R1 ∩ R2) R(R2 - R1)) is in F+.**(交集能決定一邊的差集即可)7**

CH16

**seek time + rotational delay + block transfer time = disk read/write time**

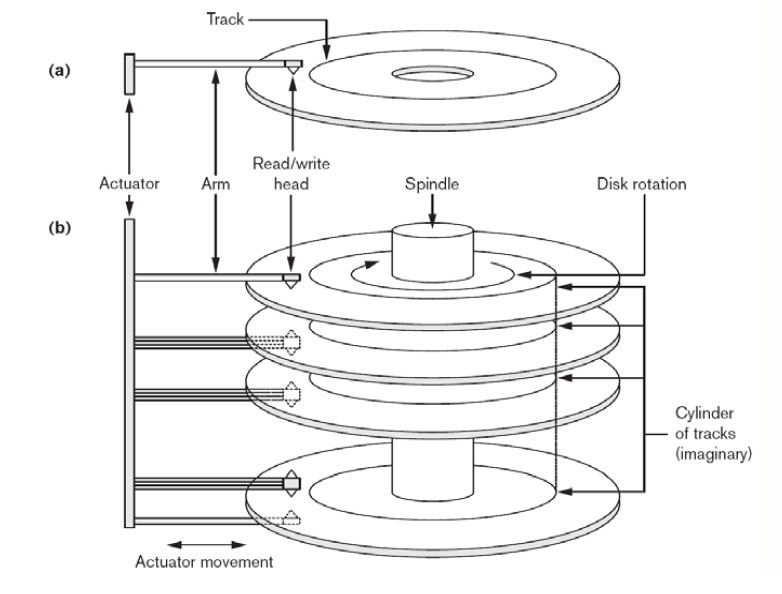
1. Seek time: position the read/write head on the correct track (bottleneck)

2. Rotational delay or latency: the beginning of the desired block rotates into position under

the read/write head

3. Block transfer time: transfer the data of the desired block

(Bulk transfer rate: time required to transfer consecutive blocks)



**Double Buffering**

1. CPU start processing once data is in main memory, while I/O processor can read and

transfer from disk

2. Permits continuous reading or writing of data on consecutive disk blocks, which eliminates

the seek time and rotational delay for all but the first block transfer

**Record**

consists of a collection of related data

Records contain fields

**BLOB (binary large object)**

used for data items consisting of large unstructured objects

**Blocking factor (bfr)**

refers to the number of records per block

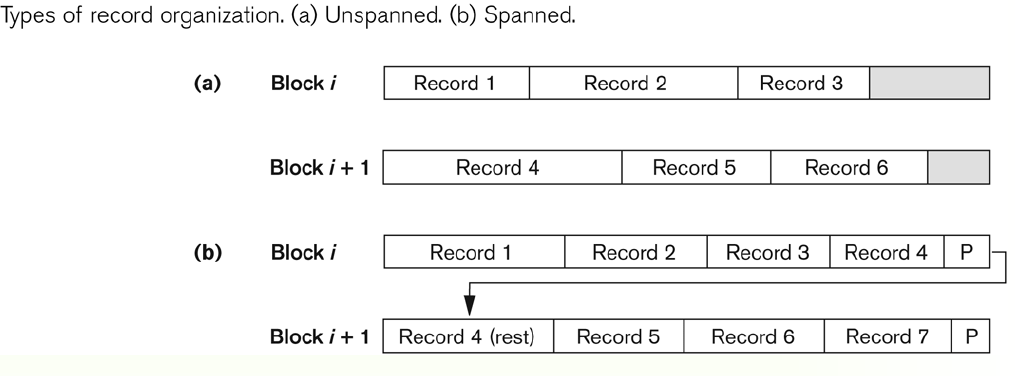
= Floor(B/R)

The unused space in each block = BlockSize – (bfr \* RecordSize)

**Spanned Records**

Refers to records that exceed the size of one more blocks and hence span a number of blocks.

a pointer is used if rest of the record is not stored in the next consecutive block.



**File**

a sequence of records

**File descriptor (or file header)**

the field names, their data types, the addresses of the file blocks on disk.

**Blocks needed for a file** (contains r records) = Ceiling(r/bfr)

**Reason for variable-length records**

1. fields are of varying size

2. repeating field

3. optional field

4. records of different types

**Physical disk type**

1. continuous

2. linked

3. indexed

**File organization**

refers to the organization of the data of a file into records, blocks, and access structures

**Access method**

provides a group of operations that can be applied to a file, such as open, reset, find, read,

**Ordered Files = Sequential file**

insert is expensive,records must be inserted in the correct order.

binary search can be used.

**Hash file (direct file)**

hash field 經過 hash function 可得到一個hash key。

單筆操作較快，多筆操作較慢。

CH17

**index**

= index field + pointer

**dense index (secondary may be)**

has an index entry for every search key value (and hence every record) in the data file.

**sparse (or nondense) (index, primary)**

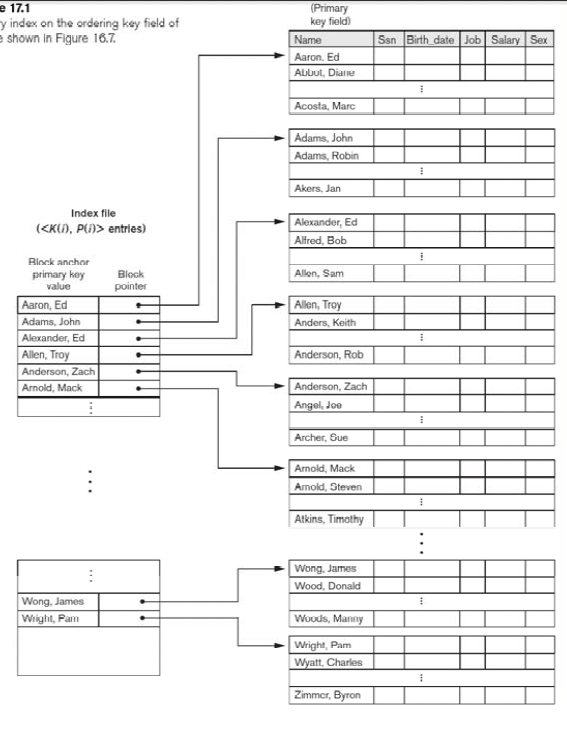
has index entries for only some of the search values

**Primary Index** on the ordering key field of an ordered file (number of blocks)

Includes one index entry for each block;

block anchor, the index entry has the key field value for the first record in the block

insert, delete 麻煩，對每個block做overflow file解決

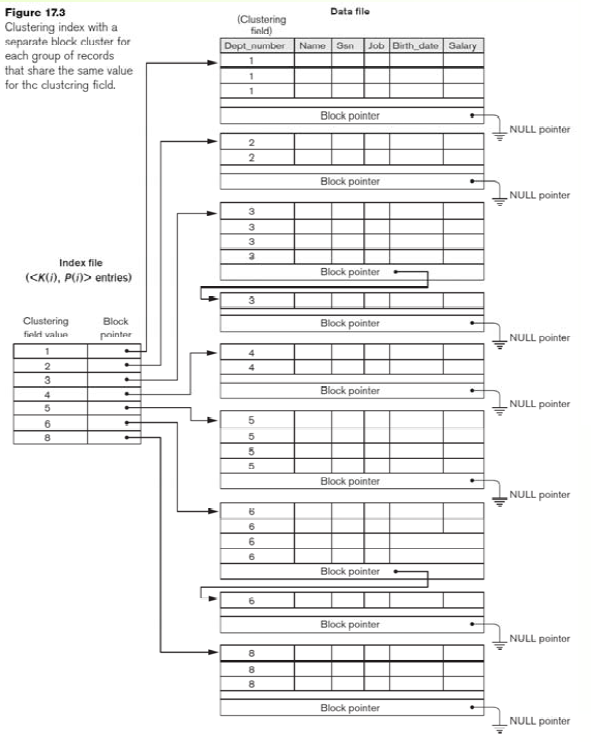


**Clustering Indexes** on the ordering nonkey field of a file (number of distinct value)

Includes one index entry for each distinct value of the field;

the index entry points to the first data block that contains records with that field value

insert, delete 麻煩，每個block下面做block pointer指到另一個block



**Secondary Indexes**  on any field(key or nonkey, ordering or nonodering) of a file

key → dense (one pointer for each record)

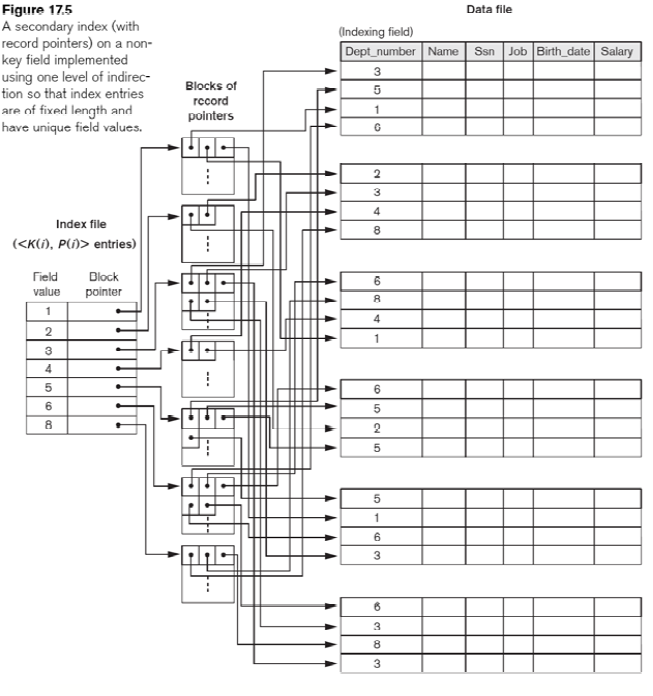
option1 : nonkey → dense (duplicate index entries)

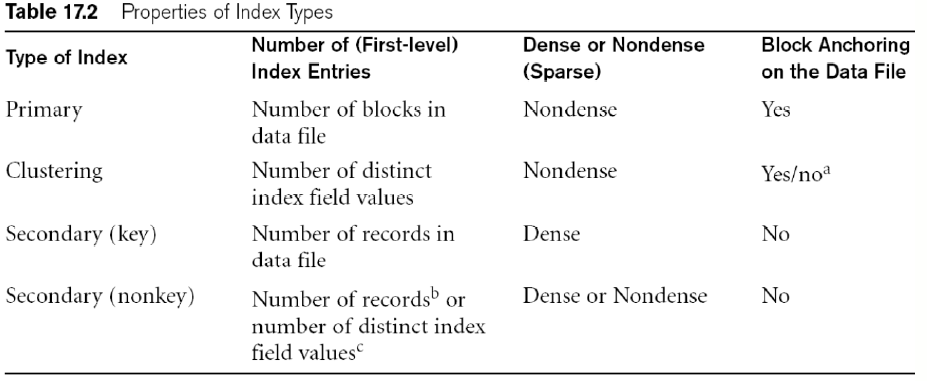
option2 : nonkey → nondense (variable length record, pointer list)

option3 : nonkey → indirect index, nondense (fixed length and have a single entry for each

index field value, extra level of indirection)

A file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, but not both.





**Example**

record size R = 150 bytes

block size B = 512 bytes

r = 30000 records

blocking factor Bfr (一個block可以存多少個record)

= B div R = 512 div 150 = 3 records/block

number of file blocks b = (r/Bfr)

= (30000/3)= 10000 blocks

log(10000) = 14

Simple index

if select field size with 9 bytes and pointer size 7 bytes

total index size = 9 +7 = 16

bfr = ceiling(512 / 16) = 32

number of file blocks b = (r/Bfr)

= 30000/32 = 938

log(938) + 1 = 11

record size R = 100 bytes

block size B = 1024 bytes

r = 30000 records

Primary index

bfr = floor(block size/record size) = 1024 / 100 = 10

number of file blocks b = (r/Bfr) = 30000/10 = 3000

number of index = 3000

size of each index = 15

index bfr = floor(block size/index size) = 1024/15 = 68

number of file blocks b = (number of index/index Bfr) = 3000 / 68 = 45

log(45) + 1 = 7

secondary index

bfr = floor(block size / record size) = 1024 / 100 = 10

number of file blocks b = (r/Bfr) = 30000/10 = 3000

select the key as index → number of index = 30000

size of each index = 15

index bfr = floor(block size/index size) = 1024/15 = 68

number of blocks for index = 30000 / 68 = 442

log(442) + 1 = 10

**Multi-Level Indexes**

幫index建index，讓最上層僅剩一個index

**fan-out (fo)**

bfr of multilevel index

**(logfobi) block accesses**

**Exapmle**

block size B = 1024 bytes

Suppose that the dense secondary index is converted into a multilevel index.

1.First level

fan-out = index bfr = floor(block size/index size) = 1024/15 = 68

number of blocks for index = 30000 / 68 = 442

2.Second level

number of blocks for index = 442 / 68 = 7

3.Third level

number of blocks for index = 7 / 68 = 1

total 4 level = 3 level index + original file level

log 68 (30000) = time