CHAPTER 1 :

Physical Database Design for Relational Databases

Sunisa Sathapornvajana

Semester 2/2019

Outline

- The purpose of physical database design.
- Logical vs. Physical database design.
- File organizations and Indexes.
- When to use secondary indexes to improve performance.
- Meaning of denormalization.
- When to denormalization to improve performance.

Logical vs. Physical Database Design

	Logical design	Physical design
Goal	Translate the conceptual to specific data model such as relational model	Improve the overall performance (minimize response time to access and change database)
Input	ER diagram	Table design
Output	Relational schema/ Table design	File organization Indexes Security View
Related to	What will be stored in database	How data are implemented in database
DBMS	Independent to target DBMS	Depend on a specific DBMS

Physical Database Design

- Final phase of database development process
- Transform a table design from logical design into an efficient implementation
  - Minimizes response time for limited resources (disk space and main memory)

Definition: Physical Database Design

- Process of producing a description of the implementation of the database on secondary storage

- Describe:
  - Base relations (derived data, general constraints)
  - File organizations
  - Indexes
  - Query optimization
  - Security measures

File Organizations

- Is one of the most important choices in physical database design
- Objective:
  - To store and access data in an efficient way (acceptable performance)

- Relations and tuples are stored in the secondary storage

Design File Organizations and Indexes

- Types of file organization available
  - Is dependent on the target DBMS

- Designer understand
  - Storage structures available provided by DBMS
  - How the target system uses the file structures
  - Nature of the data and its intended use
  - Typical workload supported by the database

Analyze transactions

- Before choosing file organization and indexes
  - Need to know the transactions or queries that will run on the database
- Analyze transactions to identify performance criteria:
  - the transactions run frequently and have significant impact on performance
  - the transactions are critical to business
  - the time that have high demand on the database (peak load)

Analyze transactions

- The information helps
  - to identify the part of database that may cause performance problems.
  - to select suitable file organization and indexes

- Information includes:
  - Attributes were updated
  - Criteria used to restrict the tuples (WHERE clause)

Analyze transactions

- Impossible to analyze all transactions
- Should at least investigate the "most important" transactions

- Use 80/20 rule
  - The most active 20% of user queries cause 80% of the total data access

Analyze transactions

- To analyze transactions, we use
  - Transaction relation cross-reference matrix
    - Show the relations that each transaction accesses
  - Transaction usage map
    - Show the diagram that indicates which relations are potentially heavily used.
- To focus on areas:
  - Map all transaction paths to relations
  - Relations are frequently accessed by transactions
  - Data usage of selected transactions for the relations frequently accessed

Cross-referencing transactions and relations

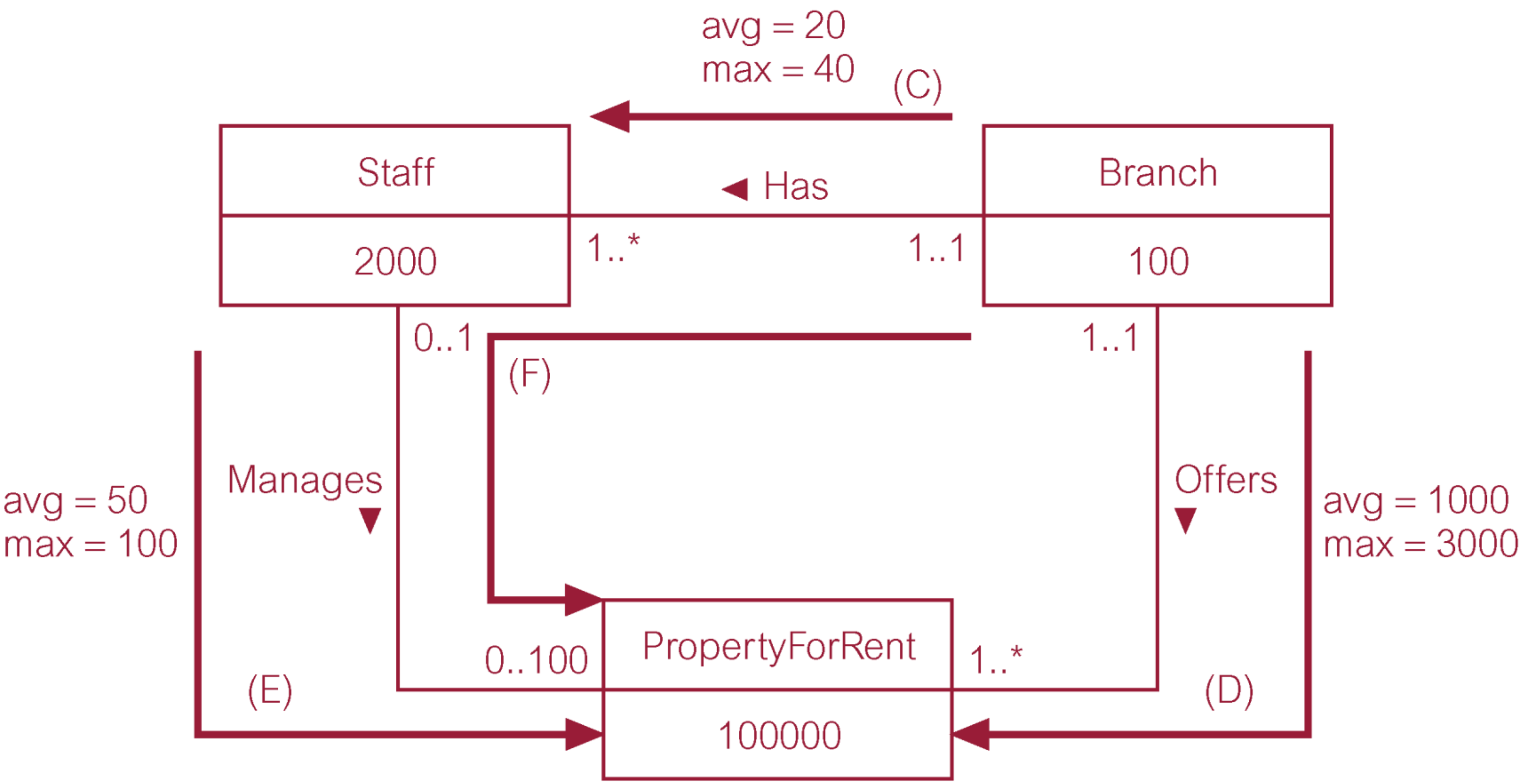
Table 17.1 Cross-referencing transactions and relations.

Transaction/ Relation	(A)				(B)				(C)				(D)				(E)				(F)			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch										X				X								X		
Telephone																								
Staff		X				X				X								X				X		
Manager																								
PrivateOwner	X																							
BusinessOwner	X																							
PropertyForRent	X					X	X	X						X				X				X		
Viewing																								
Client																								
Registration																								
Lease																								
Newspaper																								
Advert																								

I = Insert; R = Read; U = Update; D = Delete

Which relation is most accessed by transactions?

Transaction usage map



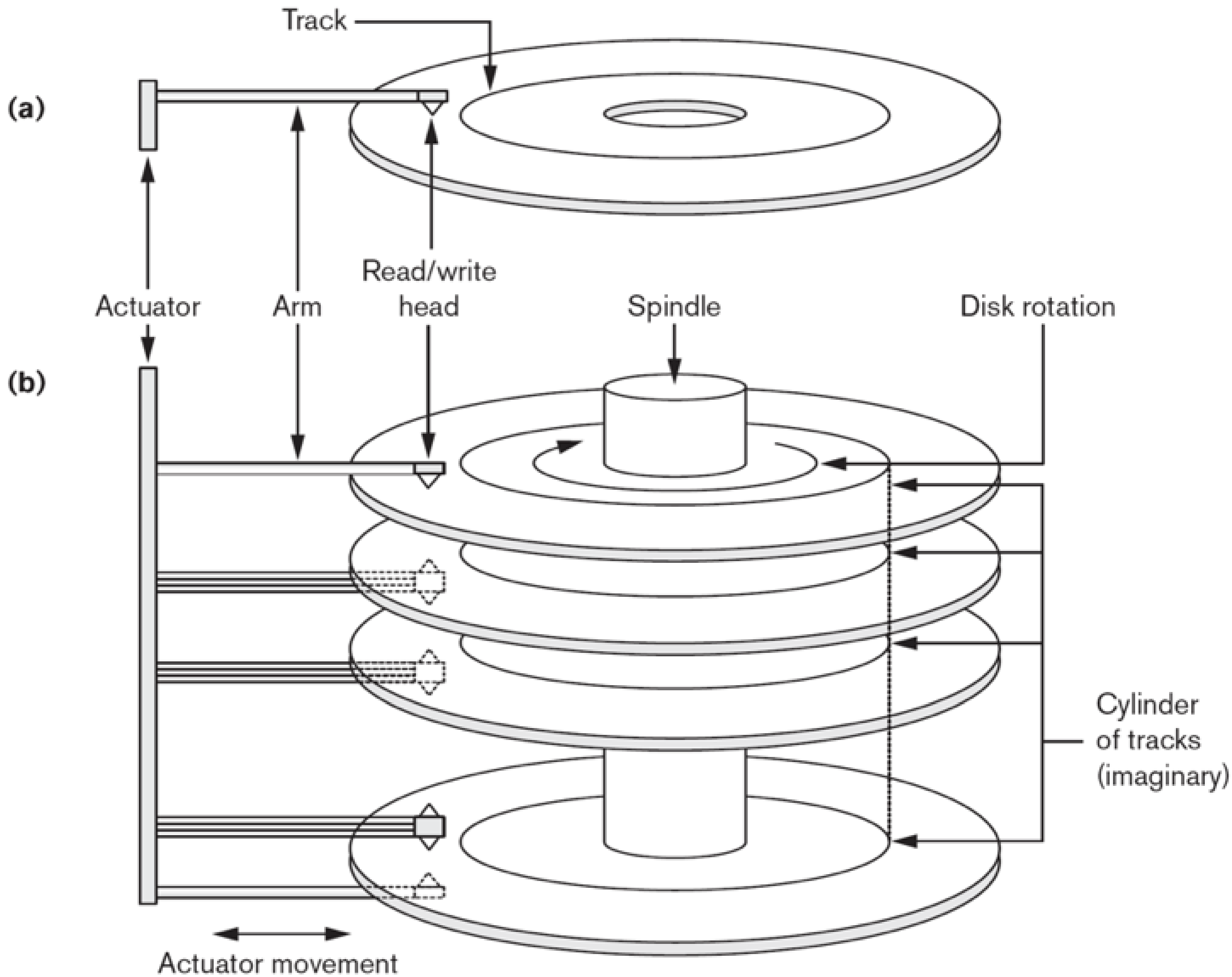
What should be analyze in data usage?

- To determine:
  - Relations and attributes accessed by transaction
  - Type of access (Insert/Update/Delete/Query)
  - Attributes used in predicates (WHERE clause)
  - Attributes in the join of two or more relations
  - Expected frequency at which transaction will run (50 times per day)
  - Performance goals for transaction (Complete within 1 second)

File Organizations

- File Organizations
  - Determines the order of records are stored and accessed in the file
  - The physical arrangement of data in a file into records and blocks/pages on secondary storage
- Main types
  - Heap (unordered)
  - Sequential (ordered)
  - Hash

Figure 17.1
(a) A single-sided disk with read/write hardware.
(b) A disk pack with read/write hardware.



Records

- Fixed and variable length records
- Records contain fields which have values of a particular type
  - E.g., amount, date, time, age
- Fields themselves may be fixed length or variable length
- Variable length fields can be mixed into one record:
  - Separator characters or length fields are needed so that the record can be "parsed."

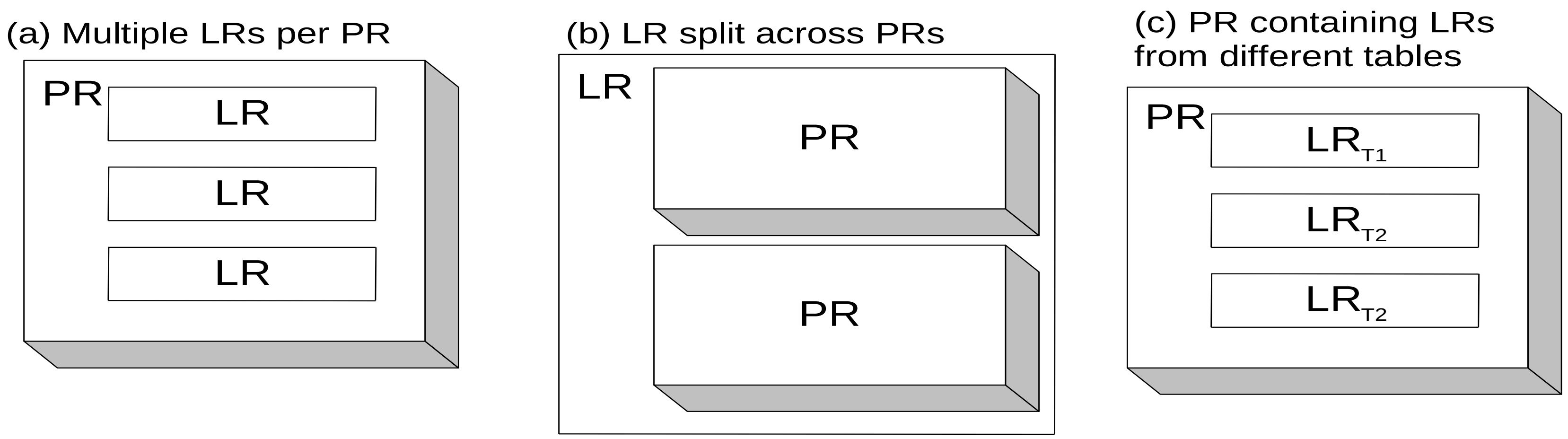
Blocking

- Blocking:
  - Refers to storing a number of records in one block on the disk.
- Blocking factor (**bfr**) refers to the number of records per block.
- There may be empty space in a block if an integral number of records do not fit in one block.
- Spanned Records:
  - Refers to records that exceed the size of one or more blocks and hence span a number of blocks.

Physical record

- A Physical record
  - An unit of transfer between disk (nonvolatile) and primary storage (volatile/memory)
  - Consists of more than one logical record (tuple)
  - Known as Block or Page

Physical Record vs. Logical Record

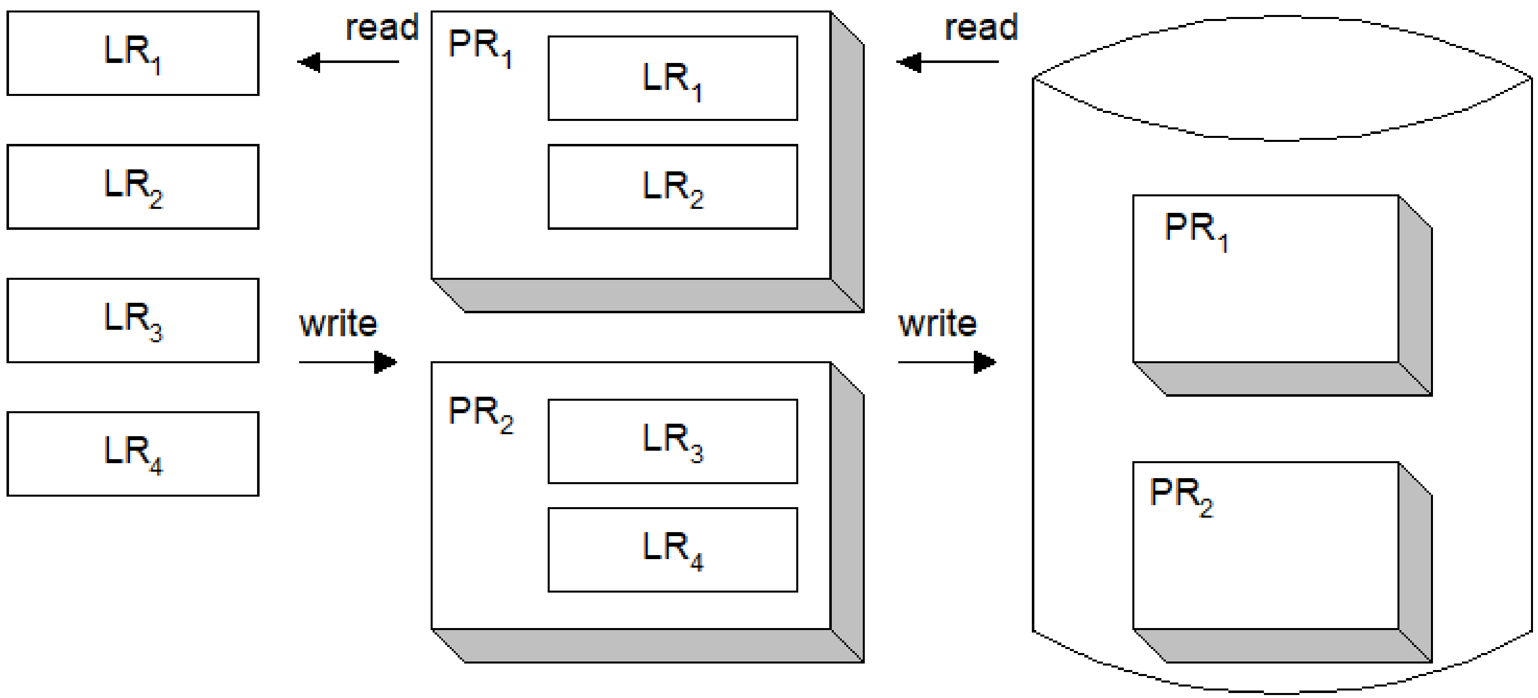


Application loads logical records from the physical records

Application buffers: Logical Records (LRs)

DBMS buffers: Logical Records (LRs) inside of Physical Records (PRs)

Operating system: Physical Records (PRs) on disk



Application Buffer

DBMS Buffer

Disk

Heap file (unordered)

- Simple type
- Records are place in the file as the same order as they are inserted
- New record is inserted in the last page/block of file
- Efficient for insert operation
- No ordering for specific field
- Search using linear search (reading from the first record until the required record is found)
- Delete a record will marked as deleted and is not reused
- Performance will be slow down when more deletion (Need reorganization by DBA)
- Best for bulk loading data into a table

Sequential File (Ordered)

- Records are sorted on the value of one or more fields
- The field(s) that file is sorted on called the Ordering field
- If the ordering field is key, it is called the ordering key
- Search using a binary search (more efficient than a linear search)
- Inserting and deleting records require record reorganization in order to maintain the order of records
- Inserting and deleting could be time-consuming
- Use overflow (or transaction) file when insert a new record and the overflow file is merged with the main sorted file

Sequential File (Ordered)

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

Hash file

- Use hash function to calculate the address of the page in which the record is inserted
- The address is based on one or more fields in the record called hash field
- If the has field is the key field, it is called hash key
- Records in hash file will be randomly distributed across the available file space
- Called random or direct file
- Can cause the collision problem
- Collision solution
  - Multiple hashing
  - Dynamic hashing

Hash file

- Limitations
  - Not good for retrievals based on pattern matching or ranges
  - Not good for retrievals based on other fields (not hash field)

What is an Index?

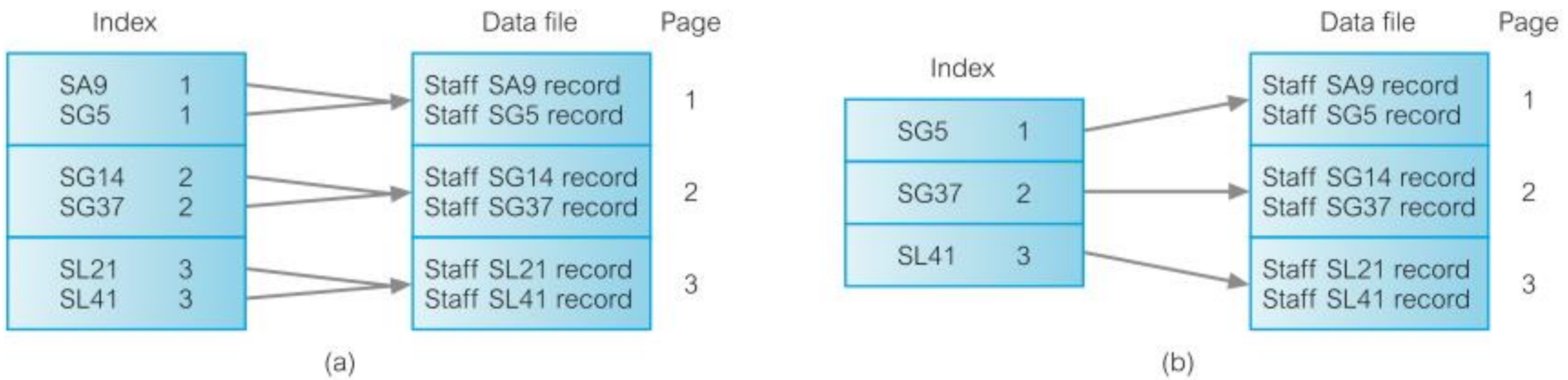
- Techniques for making the retrieval of data more efficient
- Is a data structure that helps DBMS:
  - To locate particular records in a file more quickly
  - To speed response to user queries
- Is similar to an Index in a book
- Is an auxiliary structure file (index/key file) associated with a file (data file)
- An index structure (index file) consists of:
  - A key value (indexing field value)
  - The address of logical records (data file)
- The values in index file are ordered by the indexing field

Types of Index

- Main types of Index
  - Primary Index
    - Is ordered by an ordering key field to guaranteed to have a unique value
  - Clustering Index
    - Is ordered by on a non-key field
  - Secondary Index
    - Is ordered by a non-ordering field

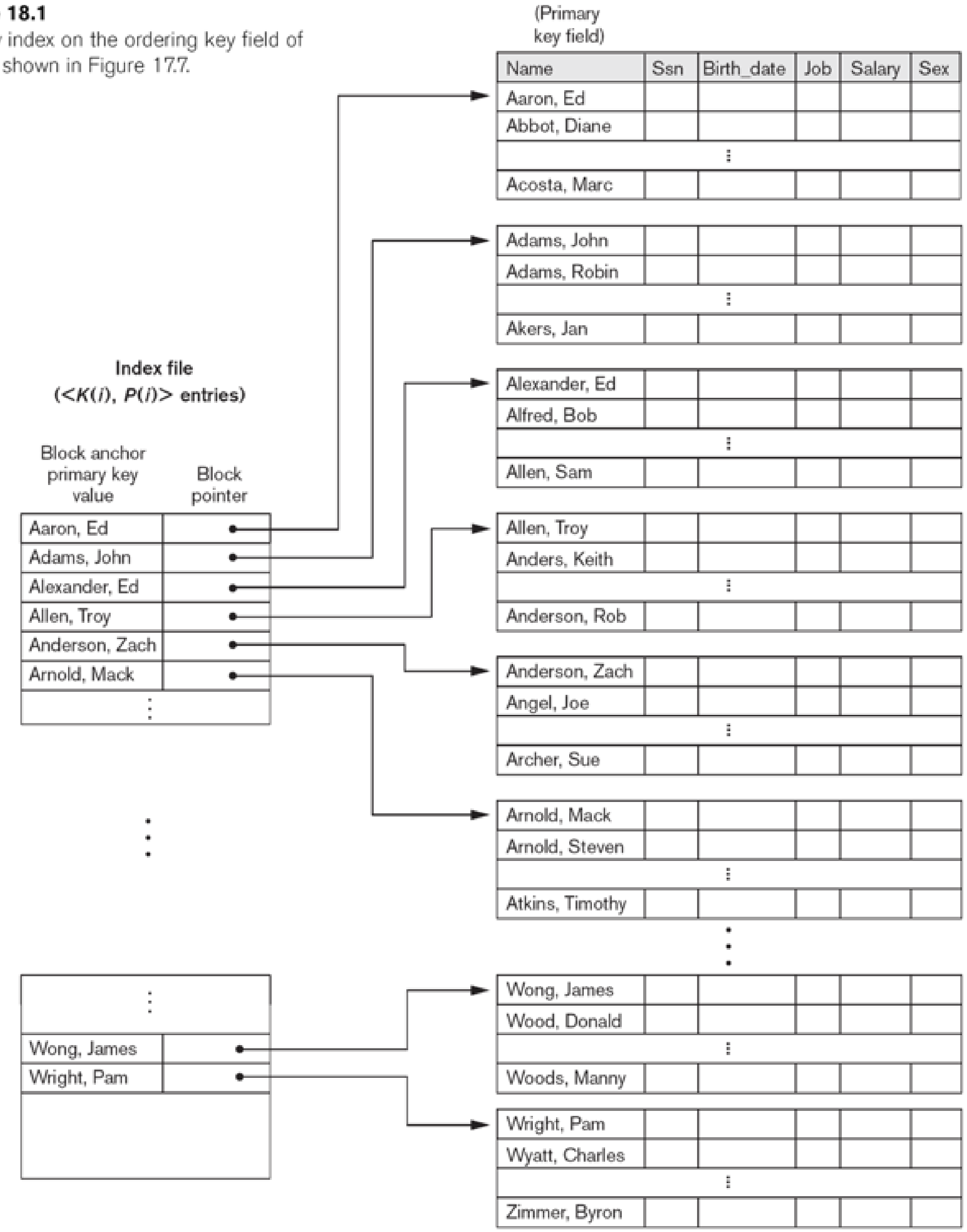
A file can have
  - At least one primary index or one clustering index
  - Zero or several secondary indexes
- Sparse (some search keys) vs. Dense (every search key)

Dense vs. Sparse Indexes



Primary Index on the Ordering Key Field

Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



A Clustering Index Example

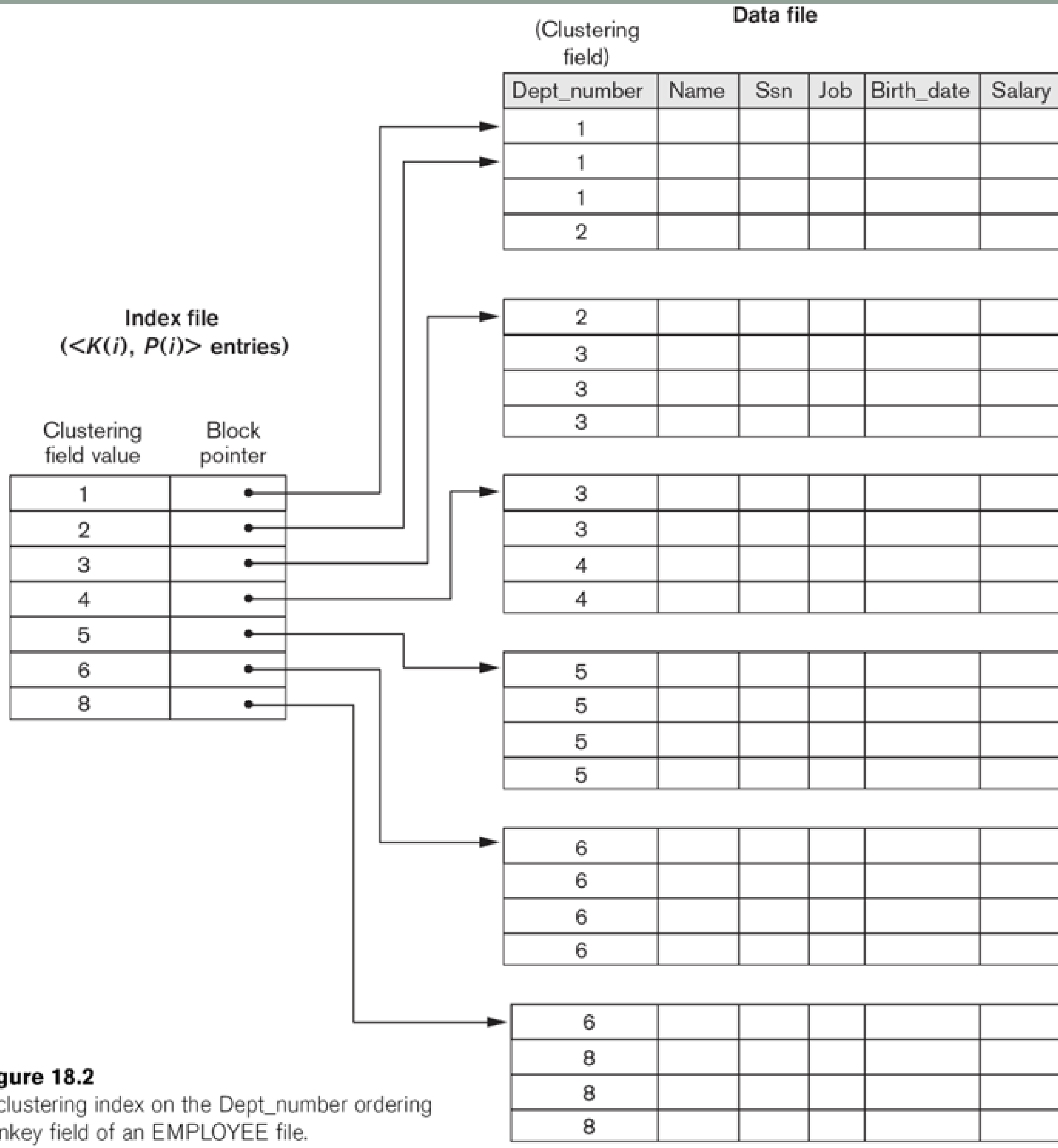


Figure 18.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

Indexed Sequential Files

- Records in the data file are stored sequentially (sequential file)
- Records are sorted and stored in a primary index (indexed file)
- Records can be processed sequentially or individually access using a search key value using the index file
- Such as ISAM file (IBM)

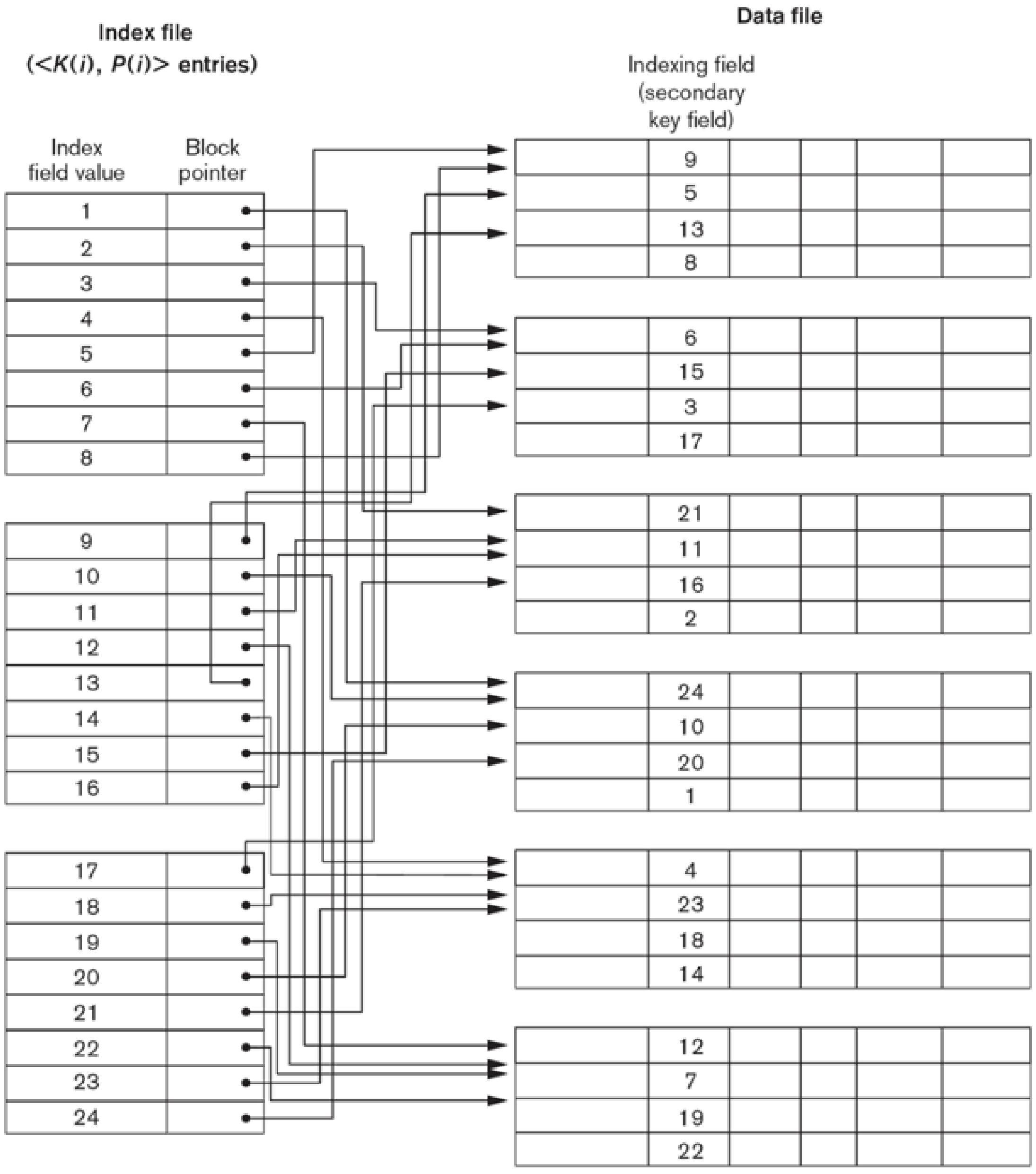
- Is more versatile structure which has
  - A primary storage area
  - A separate index or indexes
  - An overflow area

Secondary Indexes

- Is sorted by non-ordering field
- May or may not contain unique value
- Help to improve the performance of queries that use attributes other than the primary key
- Need to be balanced against overhead involved in maintaining the indexes while the database is being updated.
- The Overhead includes
  - Adding an index record when a tuple is inserted
  - Updating a secondary index when the tuple is updated
  - The increasing in disk space for storing the secondary index
  - possible performance degradation during query optimization to consider all secondary indexes.

Example of a Dense Secondary Index

Figure 18.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



Guidelines for choosing indexes

1. Do not index small relations
2. Index the primary key if it is not a key of file organization
3. Add an index to a foreign key that is accessed frequently
4. Add an secondary index to any attribute that is heavily used
5. Add an index on attributes that are frequently involved in:
    - Selection (WHERE clause) and join criteria
    - ORDER BY
    - GROUP BY
    - Other operations involving sorting (UNION or DISTINCT)

Guidelines for choosing indexes

6. Add an index on attributes used in built-in aggregate functions (or built-in functions)

    Select branchno, avg(salary)

    From staff

    Group by branchno ;

    Create index staff_sal_idx on staff(branchno,salary) ;

    This may allow DBMS to perform the query data in the index alone (called index-only plan)

6. Add an index on attributes for an index-only plan
7. Do not index an attribute or relation frequently updated
8. Do not index an attribute that query returns a lot of rows
9. Do not index attributes of long character strings

Guidelines for choosing indexes

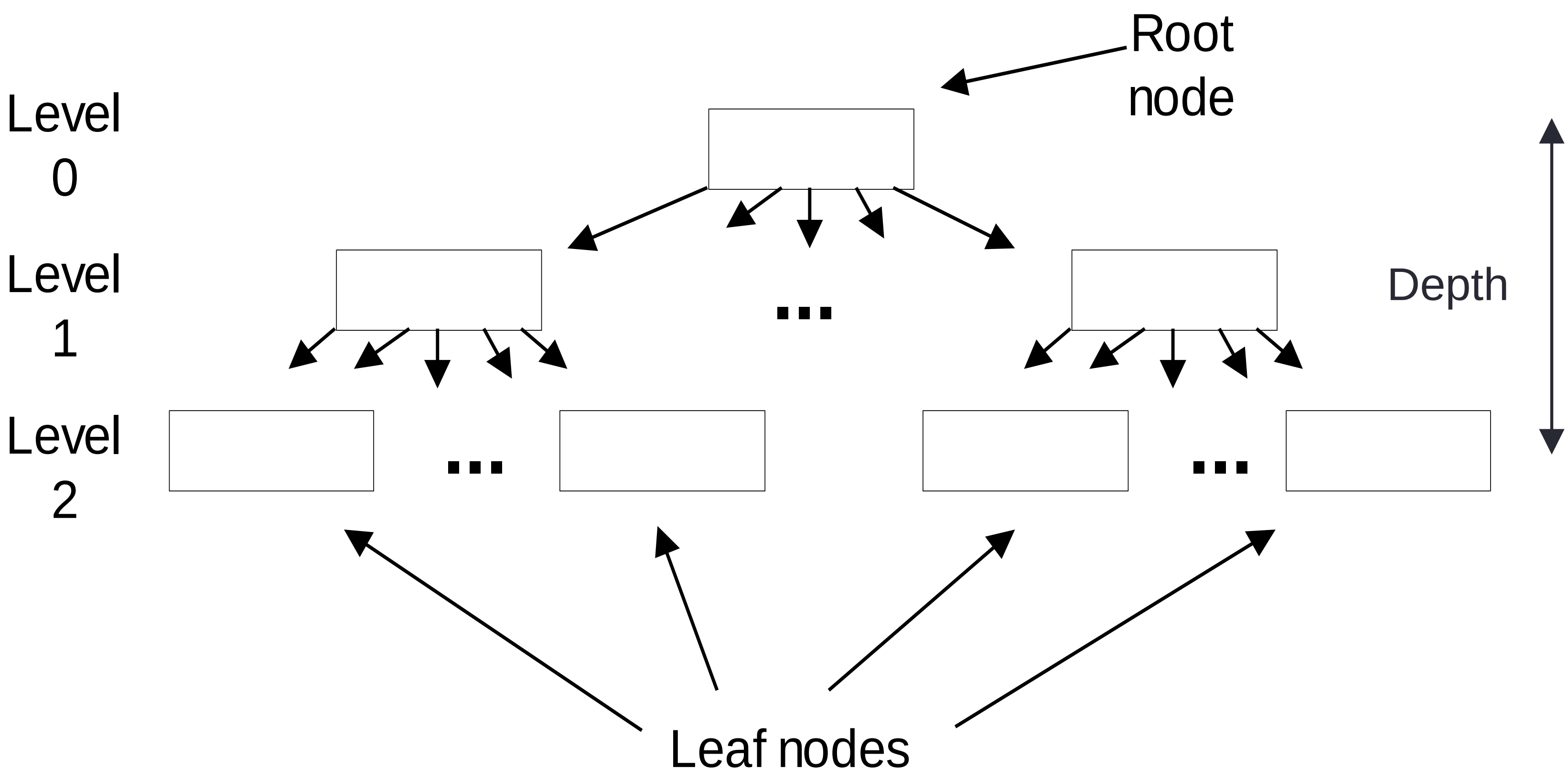
Additional guidelines:

1. A combination of columns used together in query conditions may be good candidates for nonclustering indexes if the joint conditions return few rows.
2. Volatile tables (lots of insertions and deletions) should not have many indexes
3. Stable columns with few values (low cardinality) are good candidates for bitmap indexes if the columns appear in WHERE conditions
4. Avoid indexes on combinations of columns. Most optimization components can use multiple indexes on the same table. An index on a combination of columns is not as flexible as multiple indexes on individual columns of the table

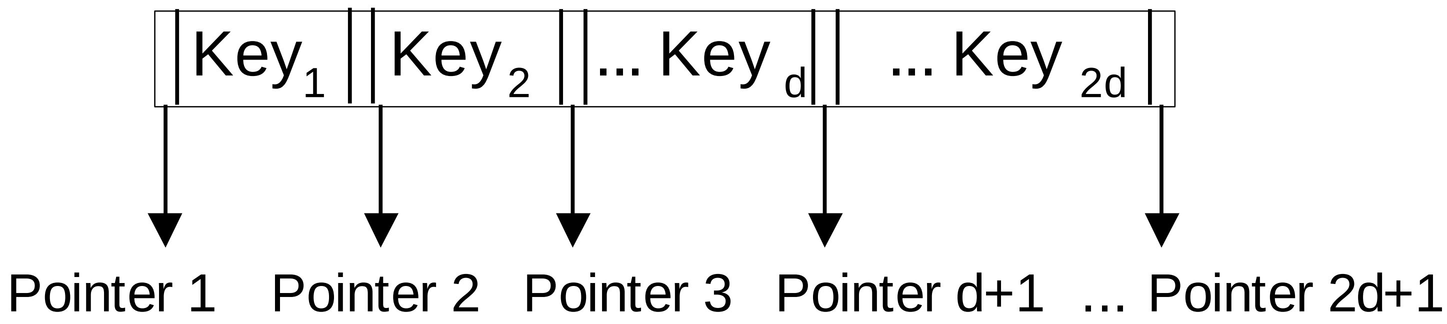
B-tree

- Most DBMSs use data structure called B-tree to hold data and indexes
- B-tree provides good performance on both sequential search and key search.
- B-tree characteristics:
  - Balanced : The same depth from root to each leaf node (Depth)
  - Bushy : The number of branches from a node is large (Degree/Order)
  - Block-oriented : Each node in a B-tree is a block or physical record
  - Dynamic : The shape of a B-tree changes When records are inserted or deleted

Structure of B-tree



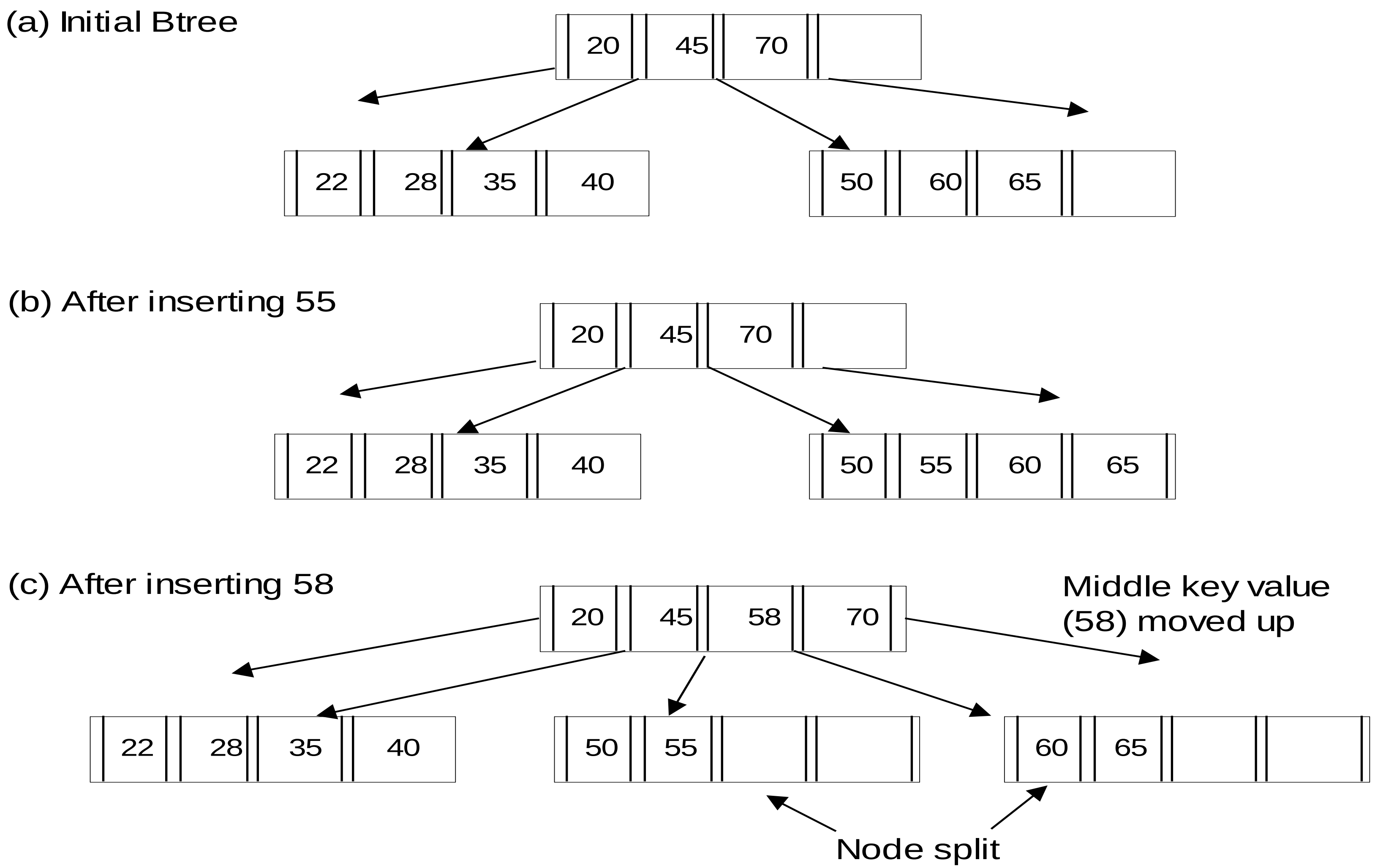
B-tree Node Containing Keys and Pointers



Each non root node contains at least half capacity (d keys and d+1 pointers).

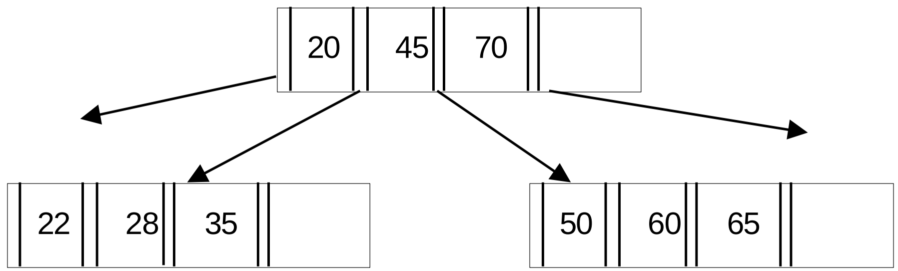
Each non root node contains at most full capacity (2d keys and 2 d+1 pointers).

Btree Insertion Examples

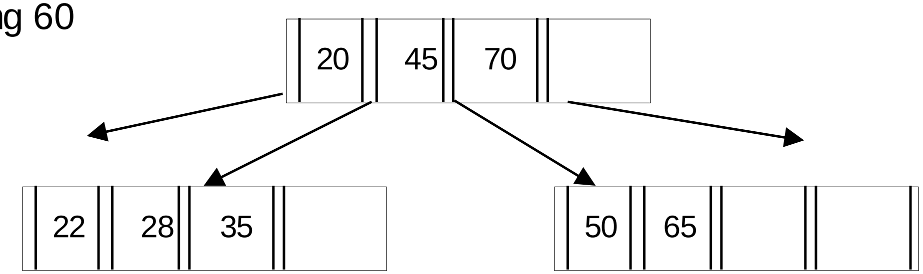


Btree Deletion Examples

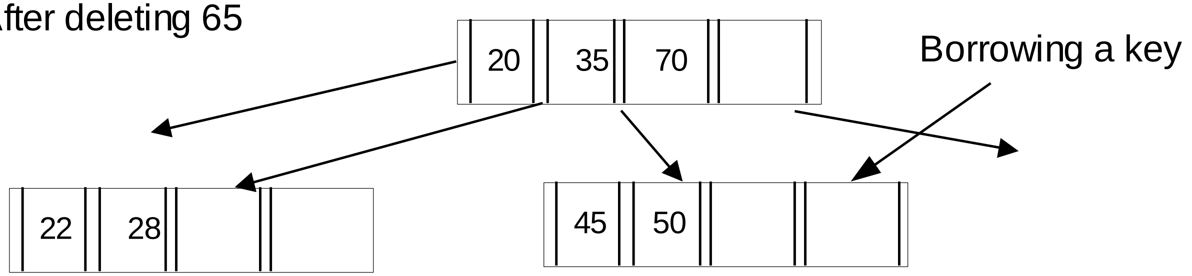
(a) Initial Btree



(b) After deleting 60



(c) After deleting 65



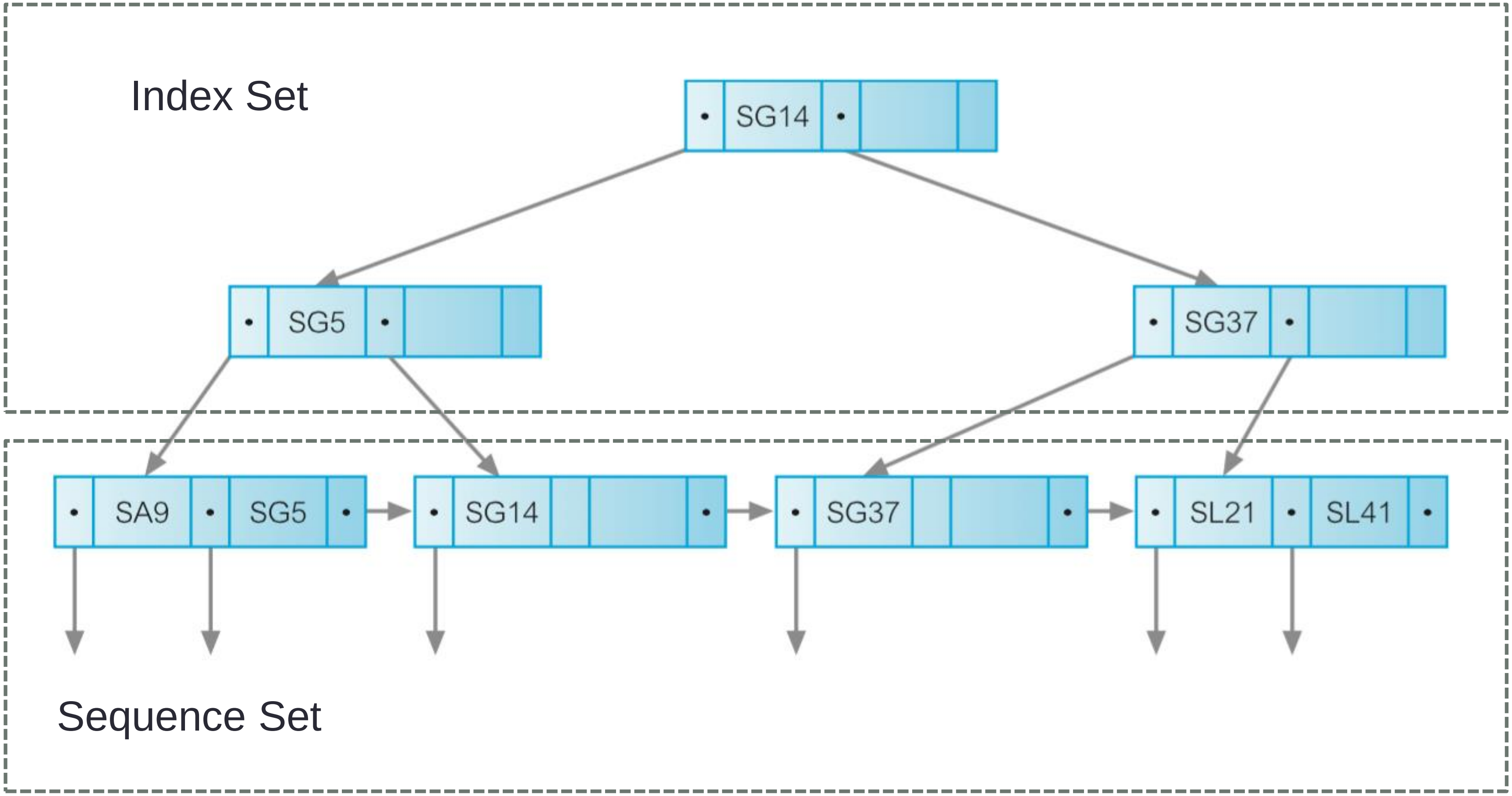
B+-tree

- Provides improved performance on sequential and range searches.
- In a B+tree, all keys are redundantly stored in the leaf nodes.
- To ensure that physical records are not replaced, the B+tree variation is usually implemented.

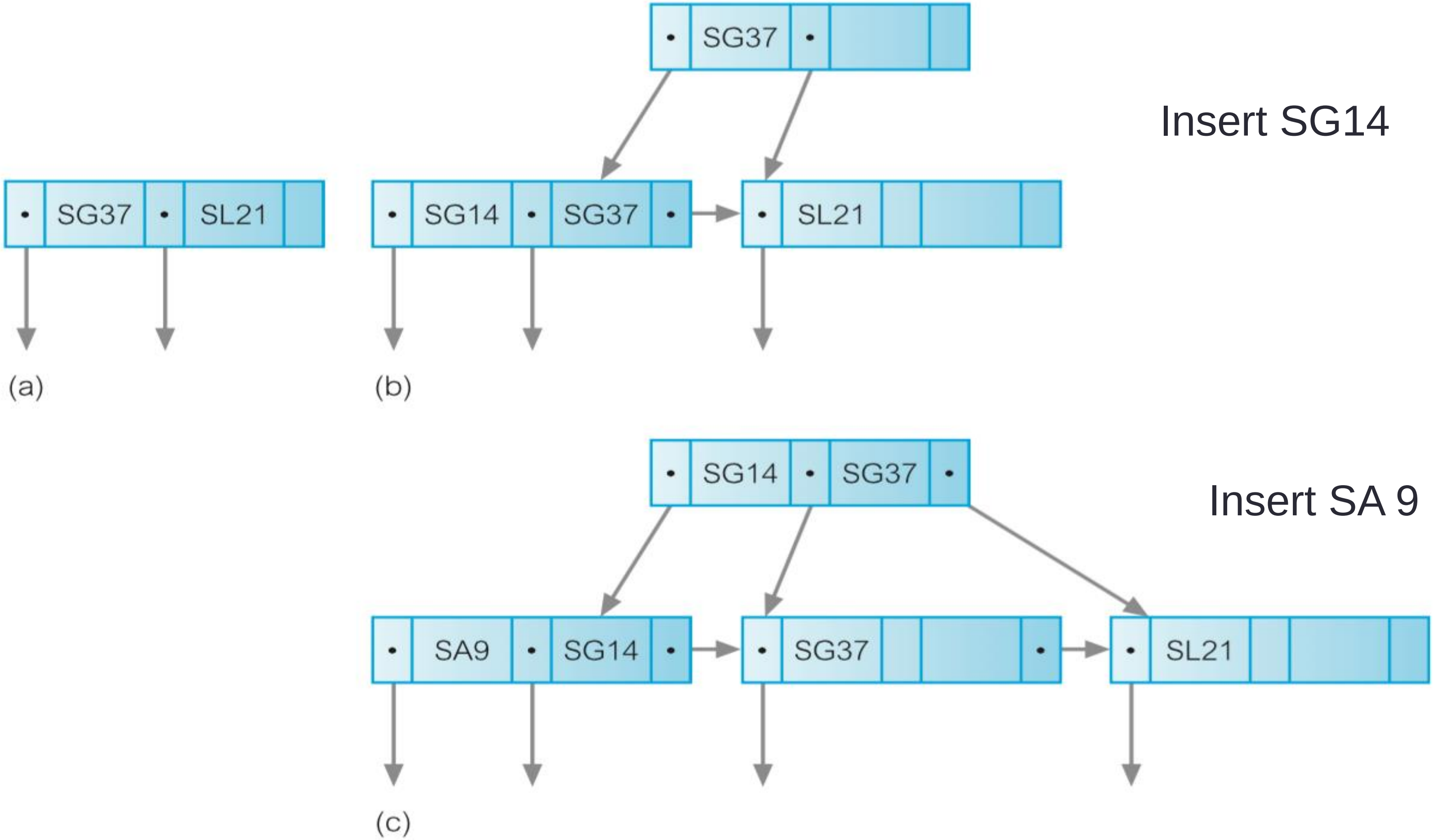
B-tree vs. B+ tree

- B-tree has record pointers in all of index nodes
- B+tree has record pointer only in leaf nodes
- B+tree leaf nodes are linked together to form a sequential file

B+-tree Structure



Insert of B+-tree index



Insert SG14

Insert SA 9

Insert of B+-tree index

Insert SG5

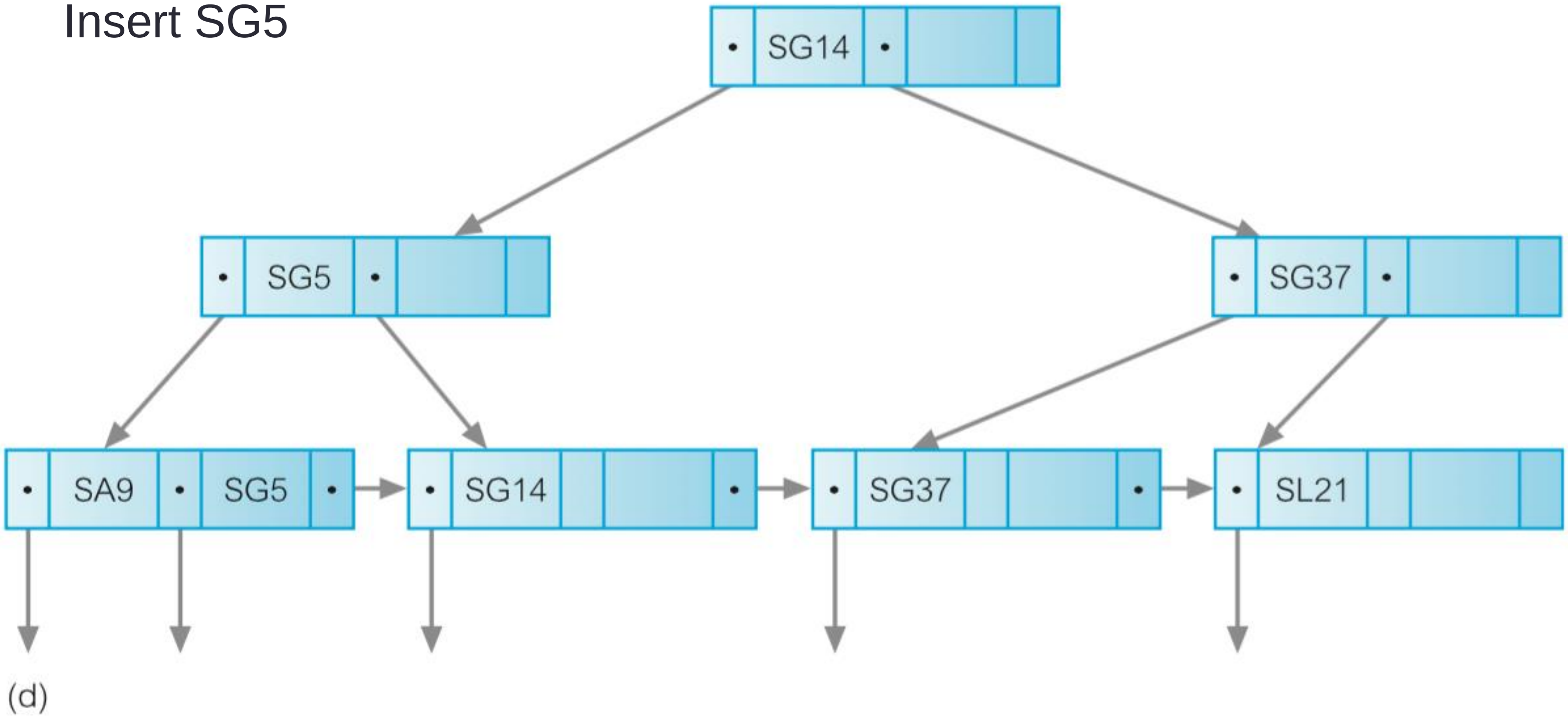


Figure F.12 Insertions into a B⁺-tree index: (a) after insertion of SL21, SG37; (b) after insertion of SG14; (c) after insertion of SA9; (d) after insertion of SG5.

Bitmap Indexes

- Are becoming increasing popular (Data warehouse)
- Are generally used on attributes that have a parse domain (a few values)
- Rather than storing the actual value of attribute
  - It stores a bit vector for each attribute to indicate which records contain the value
- Are very space-efficient
- Bitmap:
  - String of bits : 0 (no match) or 1 (match)
  - One bit for each row
- Bitmap index record:
  - Column value
  - Bitmap
  - DBMS converts bit position into row identifier

Example : Bitmap index

```
SELECT staffNo, salary
FROM Staff
WHERE position = 'Supervisor' AND branchNo = 'B003';
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

(a)

Manager	Assistant	Supervisor
1	0	0
0	1	0
0	0	1
0	1	0
1	0	0
0	1	0

B003	B005	B007
0	1	0
1	0	0
1	0	0
0	0	1
1	0	0
0	1	0

(b)

Example : Bitmap Index

Faculty Table

RowId	FacSSN	…	FacRank
1	098-55-1234		Asst
2	123-45-6789		Asst
3	456-89-1243		Assc
4	111-09-0245		Prof
5	931-99-2034		Asst
6	998-00-1245		Prof
7	287-44-3341		Assc
8	230-21-9432		Asst
9	321-44-5588		Prof
10	443-22-3356		Assc
11	559-87-3211		Prof
12	220-44-5688		Asst

Bitmap Index on FacRank

FacRank	Bitmap
Asst	110010010001
Assc	001000100100
Prof	000101001010

Join Indexes

- Are another type of index that is becoming increasingly popular, particularly in data warehouse
- Are indexes on attributes from two or more relations that come from the same domain
- This type of query could be common in data warehousing applications when attempting to find out facts about related pieces of data
  - Example : To find how many properties come from a city that has an existing branch

Example : Join Index

Branch

rowID	branchNo	street	city	postcode
20001	B005	22 Deer Rd	London	SW1 4EH
20002	B007	16 Argyll St	Aberdeen	AB2 3SU
20003	B003	163 Main St	Glasgow	G11 9QX
20004	B004	32 Manse Rd	Bristol	BS99 1NZ
20005	B002	56 Clover Dr	London	NW10 6EU
20006	. . .			

Join Index

branchRowID	propertyRowID	city
20001	30002	London
20002	30001	Aberdeen
20003	30003	Glasgow
20003	30004	Glasgow
20003	30005	Glasgow
20003	30006	Glasgow
20005	30002	London
20006	. . .	

(b)

PropertyForRent

rowID	propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
30001	PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
30002	PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
30003	PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
30004	PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
30005	PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
30006	PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003
30007	. . .									

(a)

Join Indexes

- The join index precomputes the join of the Branch and PropertyForRent relations based on the city attribute
- Thereby removing the need to perform the join each time the query is run, and improving the performance of the query.
- Join index is particularly important when query has a high frequency.
- Oracle combines the bitmap index and the join index to provide a bitmap join index.

Clustered and Nonclustered Tables

- Some DBMSs support clustered and nonclustered tables
- Clustered tables
  - Are groups of one or more tables physically stored together
  - Share common columns (often used together) called cluster key
  - Related records are physically stored together
  - Disk access time is improved

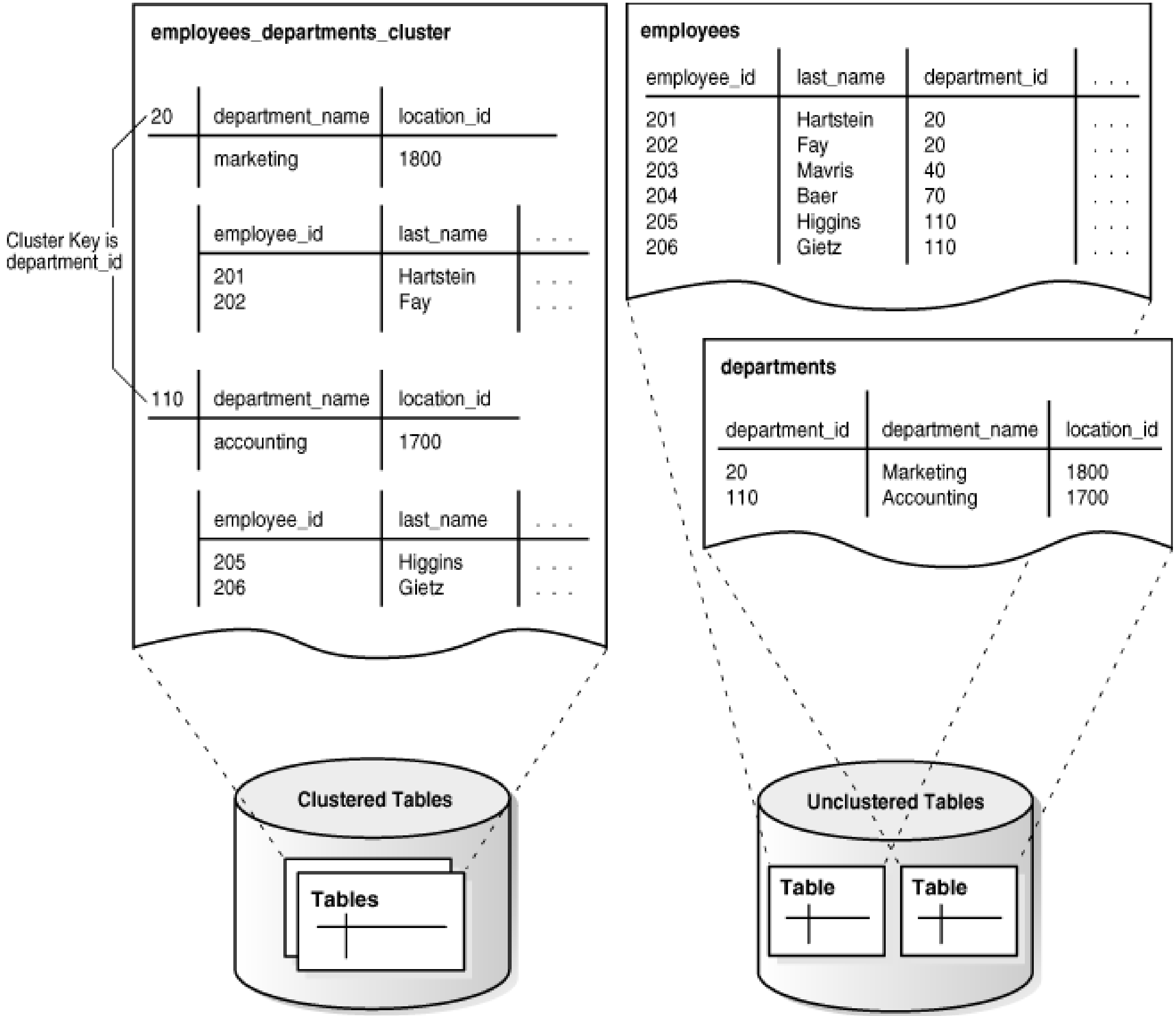
Example : Clustered Table

street	city	postcode	branchNo	staffNo	fName	lName	position	sex	DOB	salary
22 Deer Rd	London	SW1 4EH	B005	SL21	John	White	Manager	...	...	30000
				SL41	Julie	Lee	Assistant			9000
163 Main St	Glasgow	G11 9QX	B003	SG37	Ann	Beech	Assistant	...	...	12000
				SG14	David	Ford	Supervisor			18000
				SG5	Susan	Brand	Manager			24000

Staff table

Branch table

Cluster key


employees_departments_cluster
20
department_name
location_id
marketing
1800
employee_id
last_name
201
202
Hartstein
Fay
Cluster Key is department_id
110
department_name
location_id
accounting
1700
employee_id
last_name
205
206
Higgins
Gietz
employees
employee_id
last_name
department_id
201 Hartstein 20
202 Fay 20
203 Mavris 40
204 Baer 70
205 Higgins 110
206 Gietz 110
departments
department_id
department_name
location_id
20 Marketing 1800
110 Accounting 1700
Clustered Tables
Tables
Unclustered Tables
Table
Table

Normalization in database design

- The result of normalization is a design that:
  - Is structurally consistent
  - Has minimal redundancy

- However, sometimes a normalized database does not provide maximum processing efficiency

Denormalization

- Thus, some situations may be necessary to accept loss of some benefits of a fully normalized design in order to improve performance

- Denormalization should be considered only when:
  - It is estimated that the system will not be able to meet its performance requirements

Factors to be considered when:

- Denormalization:
  - Makes implementation more complex
  - Often less (loose) flexibility
  - May speed up retrievals but slows down updates

Denormalization means:

A refinement to relational schema:

- The degree of normalization for a modified relation is less than the degree of at least one of the original relations

Combine two relations into a new relation:

- The new relation is still normalized but contains more nulls than the original relations

Rule of thumb

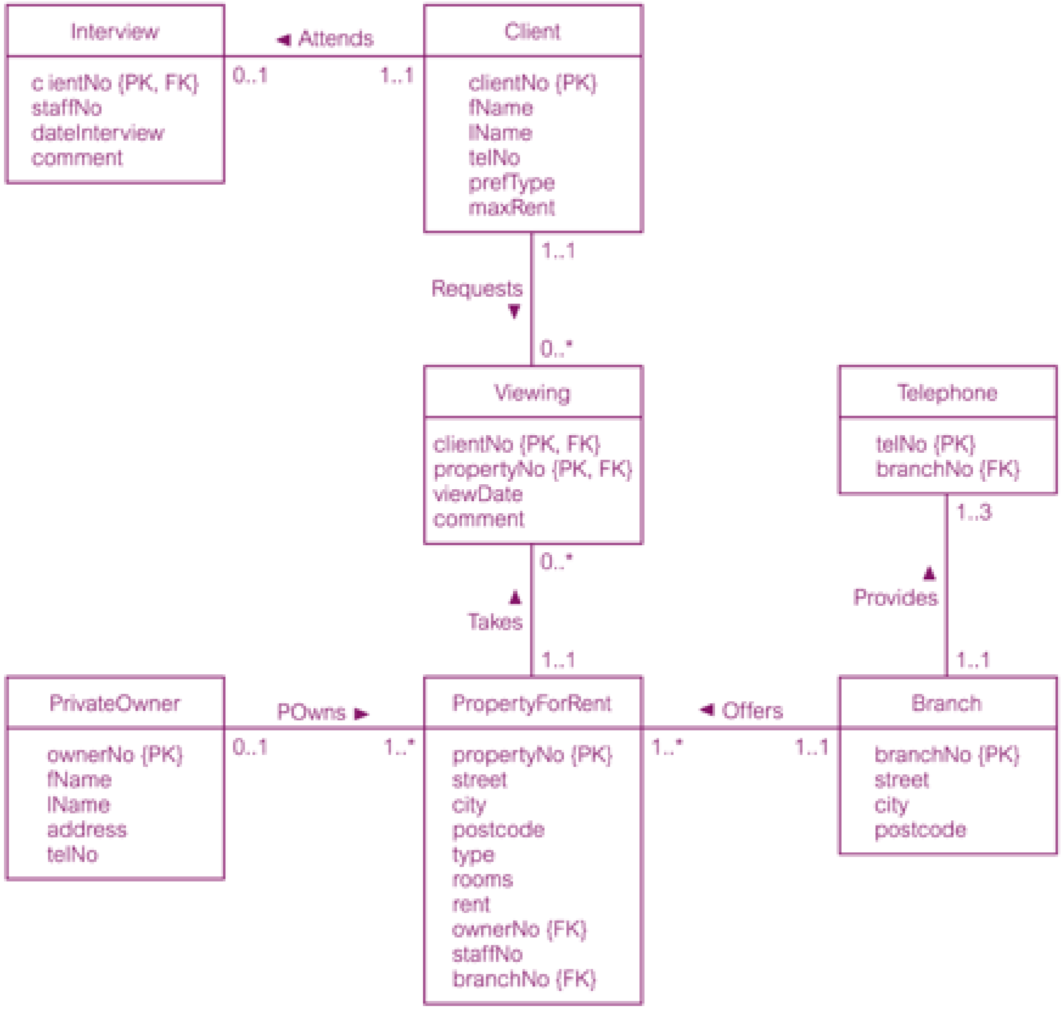
- If criteria are met:
  - Performance is unsatisfactory
  - A relation has a low update rate
  - A relation has a very high query rate
- Denormalization may be a good option

- The transaction/relation cross-reference matrix can provide useful information in this step

Common situations for denormalization

1. Combining 1:1 relationships
2. Duplicating non-key attributes in 1:* relationship to reduce joins
3. Duplicating foreign key attributes in 1:* relationships to reduce joins
4. Duplicating attributes in *:* relationships to reduce joins
5. Introducing repeating groups
6. Creating extract tables
7. Partitioning relations

Sample Relation Diagram



Sample Relations

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Telephone

telNo	branchNo
0207-886-1212	B005
0207-886-1300	B005
0207-886-4100	B005
01224-67125	B007
0141-339-2178	B003
0141-339-4439	B003
0117-916-1170	B004
0208-963-1030	B002

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Client

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	425
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

Interview

clientNo	staffNo	dateInterview	comment
CR56	SG37	11-Apr-00	current lease ends in June
CR62	SA9	7-Mar-00	needs property urgently

PrivateOwner

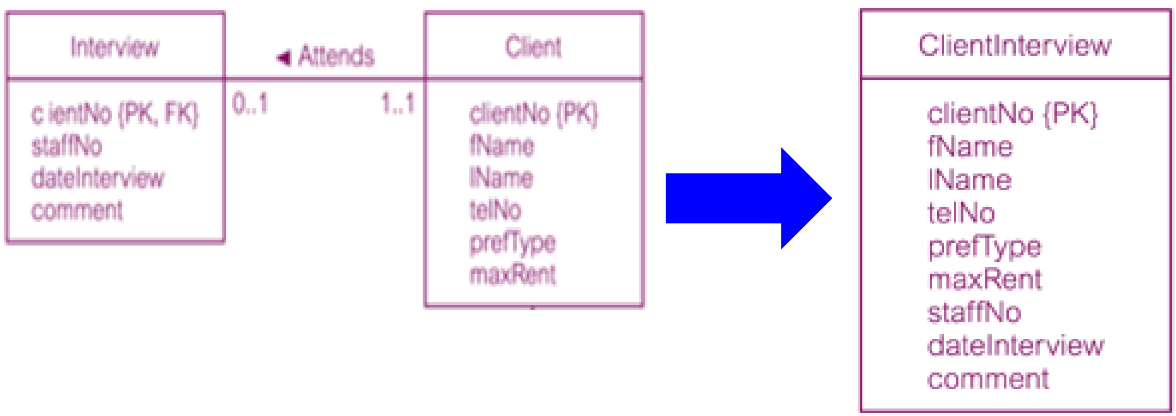
ownerNo	fName	lName	address	telNo
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-01	too small
CR76	PG4	20-Apr-01	too remote
CR56	PG4	26-May-01	
CR62	PA14	14-May-01	
CR56	PG36	28-Apr-01	no dining room

(b)

1. Combining 1:1 relationships



(a)

ClientInterview

clientNo	fName	lName	telNo	prefType	maxRent	staffNo	dateInterview	comment
CR76	John	Kay	0207-774-5632	Flat	425			
CR56	Aline	Stewart	0141-848-1825	Flat	350	SG37	11-Apr-03	current lease ends in June
CR74	Mike	Ritchie	01475-392178	House	750			
CR62	Mary	Tregear	01224-196720	Flat	600	SA9	7-Mar-03	needs property urgently

(b)

Figure 18.2 Combined Client and Interview: (a) revised extract from the relation diagram; (b) combined relation.

Combining 1:1 relationships

- Combining the relations into a single relation only for:
  - Relations are frequently referenced together and infrequently referenced separately
- If the participation is optional, there may be a significant number of nulls in the combined relation
- Therefore, if the client relation is large, there will be a significant amount of wasted space

2. Duplicating non-key attributes in 1:* relationship to reduce joins

- Whenever the PropertyForRent relation is accessed, it is very common for the owner's name to be accessed at the same time

```
SELECT p.*, o.lName
FROM PropertyForRent p JOIN PrivateOwner o
    ON p.ownerNo = o.ownerNo
WHERE branchno = 'B003' ;
```

Duplicate lName in PropertyForRent

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	lName	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	Keogh	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	Farrel	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40	Murphy		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	Shaw	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	Farrel	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	Shaw	SG14	B003

SELECT p.*

FROM PropertyForRent p

WHERE branchno = ‘B003’ ;

Problems from the duplicated non-key attributes

- The benefits that result from this change have to be balanced against the problems
  - When the duplicated data is changed
  - Additional time for maintaining consistency
  - Increase in storage space resulting from the duplication

Lookup table

PropertyType

type	description
H	House
F	Flat

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	H	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	F	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	F	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	F	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	H	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	F	4	450	CO93	SG14	B003

- Lookup table
  - sometimes called a reference table or pick list)
  - Contains a code and a description

The advantages of using a lookup table:

- Reduction in the size of the child relation
- Easier changing the description in the lookup table instead of changing many records in the child relation
- Can be used to validate user input (foreign key)

Duplicate description in the child relation

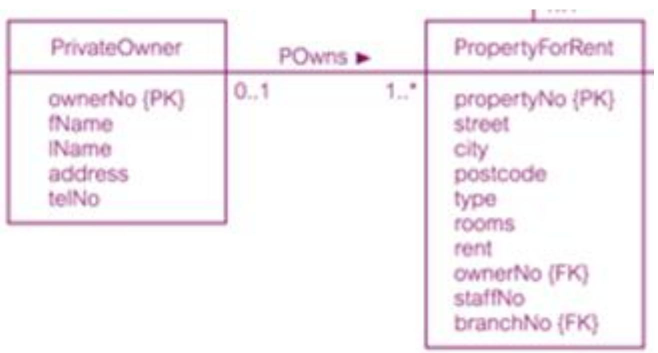
PropertyForRent

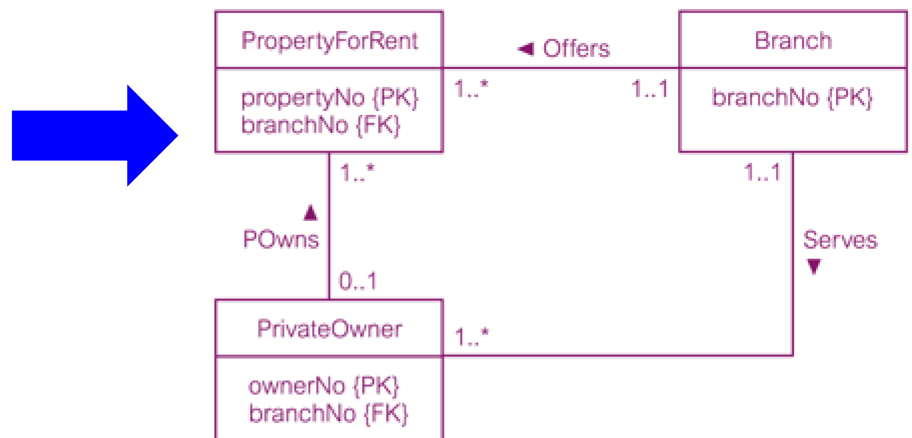
propertyNo	street	city	postcode	type	description	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	H	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	F	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	F	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	F	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	H	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	F	Flat	4	450	CO93	SG14	B003

Figure 18.5 Modified PropertyForRent relation with duplicated description attribute.

- If the lookup table is used in frequent or critical queries and the description is unlikely to change
- Should duplicating the description attribute in the child relation

Duplicating foreign key attributes in 1:* relationships to reduce joins





(a)

SELECT o.lName

FROM PropertyForRent p JOIN PrivateOwner o

ON p.ownerNo = o.ownerNo

WHERE branchNo = 'B003' ;

Simplify the SQL:

SELECT o.lName

FROM PrivateOwner o

WHERE branchNo = 'B003' ;

PrivateOwner

ownerNo	fName	lName	address	telNo	branchNo
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	B007
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	B003
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	B003
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	B003

(b)

Duplicating attributes in *:* relationships to reduce joins

- The *.* relationship between Client and PropertyForRent has been decomposed by introducing the intermediate Viewing relation.

- Case: DeamHome sales staff should contact clients who have still to make a comment on the properties they have viewed. However, the sales staff need only the street attribute of property when talking to the clients.

```
SELECT      p.street, c.*, v.viewdate
FROM   Client c JOIN viewing v ON  c.clientNo = v.clientNo
        JOIN PropertyForRent p ON v.propertyNo = p.propertyNo
WHERE       comment is NULL ;
```

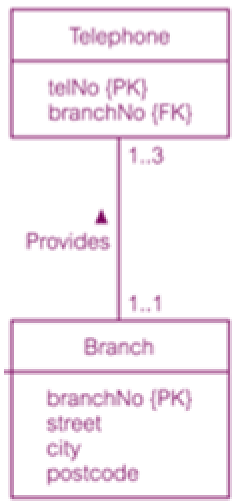
Duplicate street attribute in Viewing

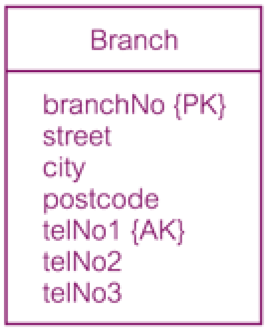
Viewing

clientNo	propertyNo	street	viewDate	comment
CR56	PA14	16 Holhead	24-May-04	too small
CR76	PG4	6 Lawrence St	20-Apr-04	too remote
CR56	PG4	6 Lawrence St	26-May-04	
CR62	PA14	16 Holhead	14-May-04	no dining room
CR56	PG36	2 Manor Rd	28-Apr-04	

```
SELECT      v.street, c.*, v.viewdate
FROM   Client c JOIN viewing v ON  c.clientNo = v.clientNo
WHERE       comment is NULL ;
```

5. Introducing repeating groups





(a)

Branch

branchNo	street	city	postcode	telNo1	telNo2	telNo3
B005	22 Deer Rd	London	SW1 4EH	0207-886-1212	0207-886-1300	0207-886-4100
B007	16 Argyll St	Aberdeen	AB2 3SU	01224-67125		
B003	163 Main St	Glasgow	G11 9QX	0141-339-2178	0141-339-4439	
B004	32 Manse Rd	Bristol	BS99 1NZ	0117-916-1170		
B002	56 Clover Dr	London	NW10 6EU	0208-963-1030		

(b)

- If access to this information is important or frequent, it may be more efficient to combine the relations and store the telephone details in the original Branch relation with one attribute for each telephone

6. Creating extract tables

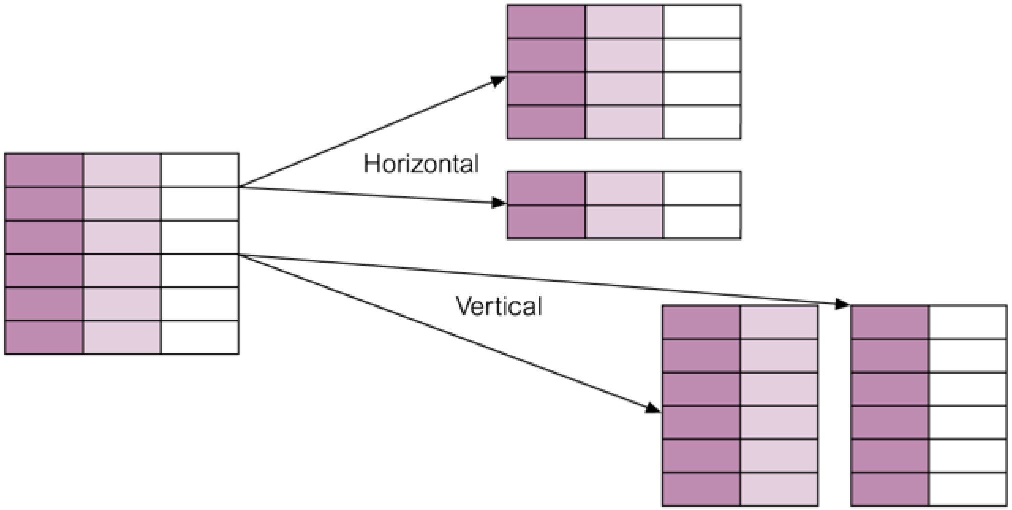
- During peak times during the day
  - Reports have to run
    - Access derived data
    - Perform multirelation joins on the same set of base relations (static or may not have to be current)

- Possible to create a single, highly denormalized extract table bases on the relations required by the report and allow users to access extract table directly instead of base relations

- The most common technique:
  - To create and populate the tables in an overnight batch run when the system is lightly loaded

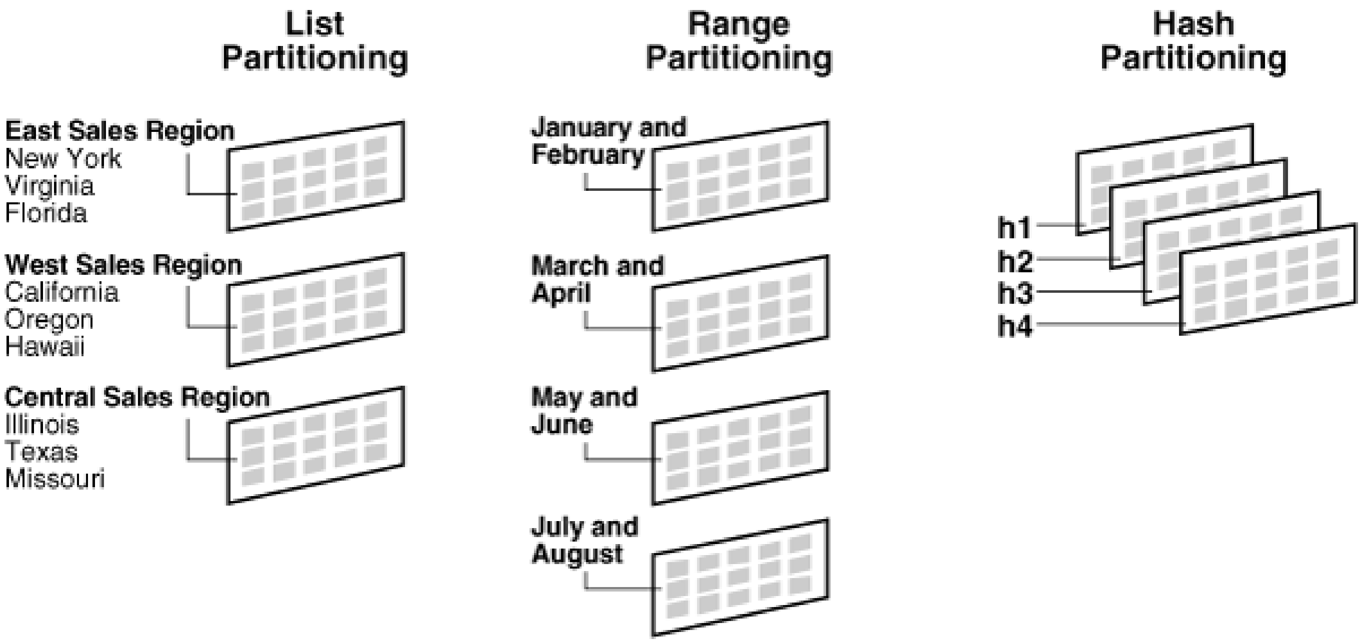
7. Partitioning relations

- To decompose relations into a number of smaller and more manageable pieces called partitions
- Two main types:
  - Horizontal partitioning
  - Vertical partitioning



Partitioning relations

Types of partitioning:
- Hash
- Range
- List



Advantages and Disadvantages of partitioning

- Advantages:
  - Improved load balancing
  - Improved performance
  - Increased availability
  - Improved recovery
  - Security

- Disadvantages:
  - Complexity
  - Reduced performance (queries that combine data from more than one partition)
  - Duplication

Advantages and disadvantages of denormalization

Table 18.1 Advantages and disadvantages of denormalization

Advantages	Disadvantages
Can improve performance by:	May speed up retrievals but can slow down updates.
■ precomputing derived data; ■ minimizing the need for joins;	Always application-specific and needs to be re-evaluated if the application changes.
■ reducing the number of foreign keys in relations;	Can increase the size of relations.
■ reducing the number indexes (thereby saving storage space); ■ reducing the number of relations.	May simplify implementation in some cases but may make it more complex in others.
	Sacrifices flexibility.

References

- Chapter 18-19, Thomas Connolly, Carolyn Begg , “Database Systems, A Practical Approach to Design, Implementation, and Management”, 6th Edition, Pearson, 2015
- Disk storage and index structure, R. Elmasri,S. Navathe “Fundamentals of Database Systems”, 7th edition, Pearson, 2016