VIEWS AND INDEXES

EGCO321 DATABASE SYSTEMS

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

TYPE OF SUBQUERIES

- There are two main types of subqueries - nested and correlated.
- Subqueries are nested, when the subquery is executed first, and its results are inserted into Where clause of the main query.
- Correlated subqueries are the opposite case, where the main query is executed first and the subquery is executed for every row returned by the main query

NESTED QUERIES (TYPE I)

- Query inside a query
- Use in WHERE and HAVING conditions
- Executes one time
- No reference to outer query
- Also known as non-correlated or independent nested query
- Often used with IN, comparison operators, ALL, ANY and their negations.

CORRELATED QUERIES (TYPE II)

- Similar to nested loops
- Executes one time for each row of outer query
- Reference to outer query
- Also known as correlated query
- Used for difference problems - problems where elements belong to one set but not to another.
- Often used with [NOT] EXISTS, but also used with IN, comparison operators, ALL, ANY and their negations.

EXAMPLE 1

- Query using comparison operators.

```
SELECT name
  FROM Student s
  WHERE 3 < (SELECT COUNT(*)
             FROM Result r
             WHERE r.rollno=s.rollno;
```

- Query using EXISTS

```
SELECT staffNo, fName, lName, position
  FROM Staff s
  WHERE EXISTS (SELECT * FROM Branch b
                   WHERE s.branchNo = b.branchNo
                       AND city = 'London');
```

EXAMPLE 2

- Sub-Query: SELECT as a table source.

```
SELECT MIN(QTY) AS min, MAX(QTY) AS max FROM
  (SELECT s.StdSSN, COUNT(*) AS QTY
          FROM student s JOIN enrollment e
               ON s.StdSSN = e.StdSSN) t
```

Warning: Duplicate columns should not allowed in a sub-query table (SELECT)

EXAMPLE 3

- Sub-Query: SELECT as a table source.

```
SELECT CourseNo, CrsDesc, OffYear, OffTerm,
       COUNT(StdSSN) AS QTY
FROM (SELECT c.CourseNo, c.CrsDesc, OffYear,
      OffTerm, e.StdSSN FROM course c JOIN
      offering o ON c.CourseNo = o.CourseNo
      JOIN enrollment e ON
      o.OfferNo = e.OfferNo) t
GROUP BY CourseNo, OffYear, OffTerm
```

Warning: Duplicate columns should not allowed in a sub-query table (SELECT)

CONCLUSION (1)

	Nested Query	**Correlated Query**	**Join Operation**
Definition	A query is written inside another query and the result of the inner query is used in the execution of the outer query.	A query is nested inside another query and an inner query uses values from the outer query.	The join operation is used to combine data or rows from two or more tables based on a common field between them.
Approach	Bottom-up approach. Inner query runs first, and only once. The outer query is executed with result from the Inner query.	Top to Down Approach. Outer query executes first and for every Outer query row Inner query is executed.	It is basically cross product satisfying a condition.

CONCLUSION (2)

	Nested Query	**Correlated Query**	**Join Operation**
Dependency	Inner query execution is not dependent on Outer query.	Inner query is dependent on the Outer query.	There is no Inner Query or Outer Query. Hence, no dependency is there.
Performance	Performs better than Correlated Query but is slower than Join Operation.	Performs slower than both Nested Query and Join operations as for every outer query inner query is executed.	By using joins we maximize the calculation burden on the database but joins are better optimized by the server so the retrieval time of the query using joins will almost always be faster than that of a subquery.

VIEWS

VIEWS (1)

- The dynamic result of one or more relational operations operating on the base relations to produce another relation.
- A view is a virtual relation that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.

VIEWS (2)

- Because joins are hard (for some people)
- Because joins are irritating if you have to set them up all the time
- Because even some of the common things people want require reasonably in-depth knowledge of business objects
- Because you can make the database technicalities hidden to your users and let them get on at the level they are comfortable

TERMINOLOGY

- SQL – Structured Query Language
- Table – Where data is stored
- Field – Data structure for a piece of information in a table
- **View – A dynamic table**
- Function (Stored Procedure) – Reusable SQL logic that hides the steps and complexity from other code

CREATING A VIEW

- The format of the CREATE VIEW statement is:

```
CREATE VIEW ViewName [(newColumnName [, . . . ])]
AS subselect [WITH [CASCADED | LOCAL] CHECK OPTION]
```

EXAMPLE 4

- Create a horizontal view

  "Create a view so that the manager at branch B003 can see only the details for staff who work in his or her branch office."

```
CREATE VIEW Manager3Staff
AS SELECT *
   FROM Staff
   WHERE branchNo = 'B003';

SELECT * FROM Manager3Staff;
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

EXAMPLE 5

- Create a vertical view

  "Create a view of the staff details at branch B003 that excludes salary information, so that only managers can access the salary details for staff who work at their branch."

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
  FROM Staff
  WHERE branchNo = 'B003';

CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
  FROM Manager3Staff;
```

EXAMPLE 6

- Grouped and joined views
  "Create a view of staff who manage properties for rent, which includes the branch number they work at, their staff number, and the number of properties they manage."

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
   FROM Staff s, PropertyForRent p
   WHERE s.staffNo = p.staffNo
   GROUP BY s.branchNo, s.staffNo;
```

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

REMOVING A VIEW (DROP VIEW)

- A view is removed from the database with the DROP VIEW statement:

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

VIEW UPDATABILITY

- All updates to a base table are immediately reflected in all views that encompass that base table.
- Similarly, we may expect that if a view is updated then the base table(s) will reflect that change.
- However, consider again the view StaffPropCnt of Example 3. Consider what would happen if we tried to insert a record that showed that at branch B003, staff member SG5 manages two properties, using the following insert statement:

```
INSERT INTO StaffPropCnt VALUES ('B003', 'SG5', 2);
```

SINGLE-TABLE UPDATABLE VIEWS (1)

- Rules for Single-Table Updatable Views:
  - The view includes the primary key for the base table.
  - All required fields (NOT NULL) of the base table without a default value are in the view.
  - The view's query does not include the GROUP BY or DISTINCT keywords.

SINGLE-TABLE UPDATABLE VIEWS (2)

- Create a row and column subset view with the primary key.

```
CREATE VIEW Fac_View AS
  SELECT FacSSN, FacFirstName, FacLastName,
  FacRank, FacSalary, FacDept, FacCity,
  FacState, FacZipCode
    FROM Faculty
    WHERE FacDept = 'MS'
```

SINGLE-TABLE UPDATABLE VIEWS (3)

- Insert a new faculty now into the MS department.

```
INSERT INTO Fac_View
(FacSSN,FacFirstName,FacLastName,FacRank,FacSala
ry, FacDept, FacCity, FacState, FacZipCode)
VALUES ('999-99-8888', 'JOE', 'SMITH', 'PROF',
8000, 'MS', 'SEATTLE', 'WA', '98011-011')
```

- Give assistant professors in Fac_View a 10% raise.

```
UPDATE Fac_View
  SET FacSalary = FacSalary*1.1
    WHERE FacRank = 'ASST'
```

SINGLE-TABLE UPDATABLE VIEWS (4)

- Delete a specific faculty member from Fac_View.

```
DELETE FROM Fac_View
  WHERE FacSSN = '999-99-8888'
```

- Change the department of highly paid faculty members to the finance department.

```
UPDATE Fac_View
  SET FacDept = 'FIN'
    WHERE FacSalary > 100000
```

MULTIPLE-TABLE UPDATABLE VIEWS (1)

- Rules for 1-M Updatable Views:
  - The query includes the primary key of the child table.
  - For the child table, the query contains all required columns (NOT NULL) without default values.
  - The query does not include the GROUP BY or DISTINCT keywords.
  - The join column of the parent table should be unique (either a primary key or a unique constraint).
  - The query contains the foreign key column(s) of the child table.

MULTIPLE-TABLE UPDATABLE VIEWS (2)

- Create a 1-M updatable query with a join between the Course and the Offering tables.

```
CREATE VIEW Course_Offering_View1 AS
  SELECT Course.CourseNo, CrsDesc, CrsUnits,
    Offering.OfferNo, OffTerm, OffYear,
    Offering.CourseNo, OffLocation, OffTime,
    FacSSN, OffDays
  FROM Course INNER JOIN Offering
    ON Course.CourseNo = Offering.CourseNo
```

MULTIPLE-TABLE UPDATABLE VIEWS (3)

- This query is read-only because it does not contain *Offering.CourseNo*.

```
CREATE VIEW Course_Offering_View2 AS
  SELECT CrsDesc, CrsUnits, Offering.OfferNo,
    Course.CourseNo, OffTerm, OffYear,
    OffLocation, OffTime, FacSSN, OffDays
  FROM Course INNER JOIN Offering
    ON Course.CourseNo = Offering.CourseNo
```

MULTIPLE-TABLE UPDATABLE VIEWS (4)

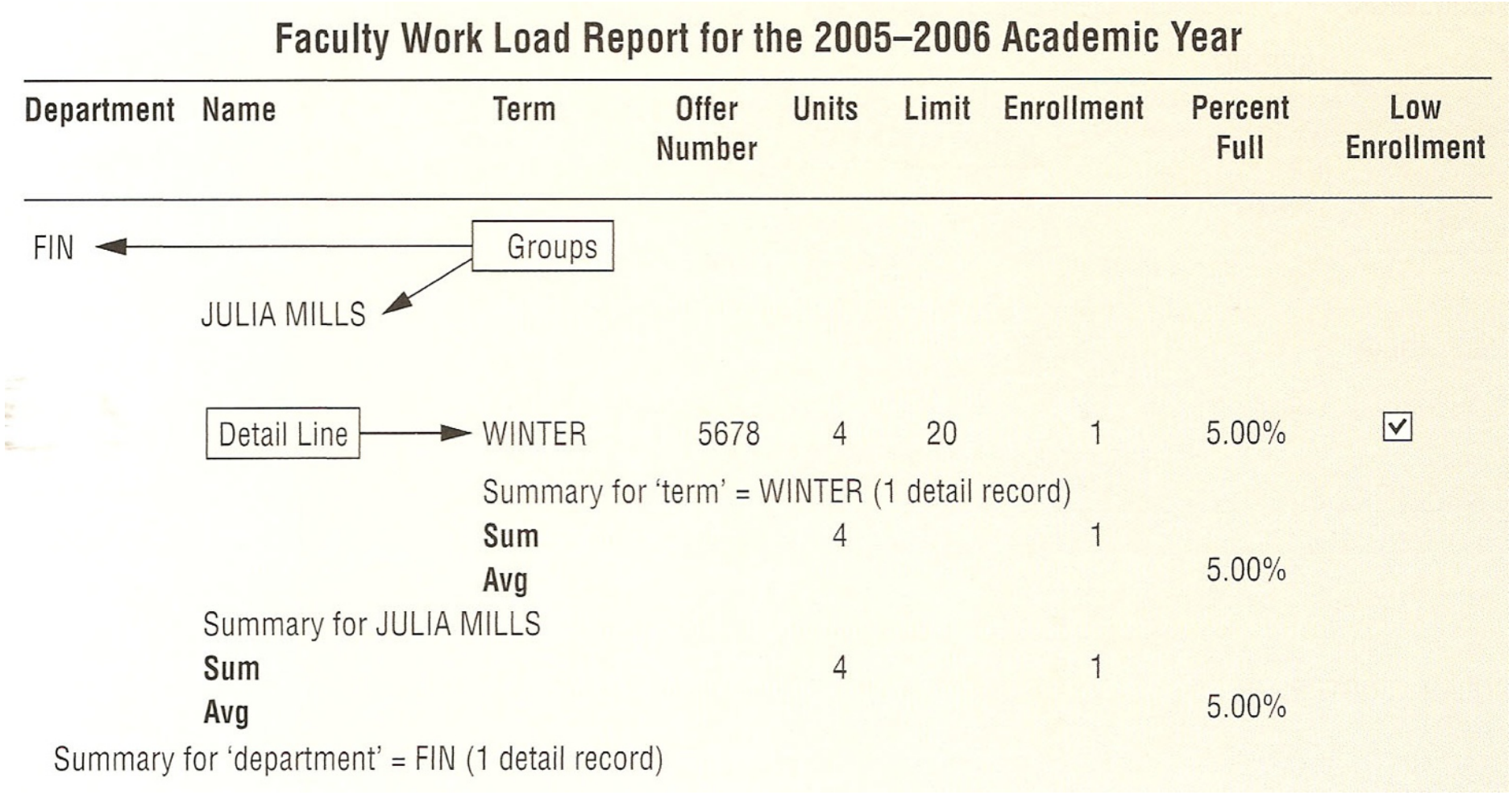
- Insert a new row into Offering as a result of using Course_Offering_View1.

```
INSERT INTO Course_Offering_View1
(Offering.OfferNo, Offering.CourseNo, OffTerm,
OffYear, OffLocation, OffTerm, FacSSN, OffDays )
VALUES ( 7799, 'IS480', 'SPRING', 2000,
'BLM201', #1:30PM#, '098-76-5432', 'MW' )
```

USING VIEWS IN HIERARCHICAL FORMS

- Hierarchical form is a formatted window for data entry and display using a fixed (main form) and a variable (subform) part.
- One record is show in the main form and multiple, related records are show in the subform.

USING VIEWS IN HIERARCHICAL REPORTS



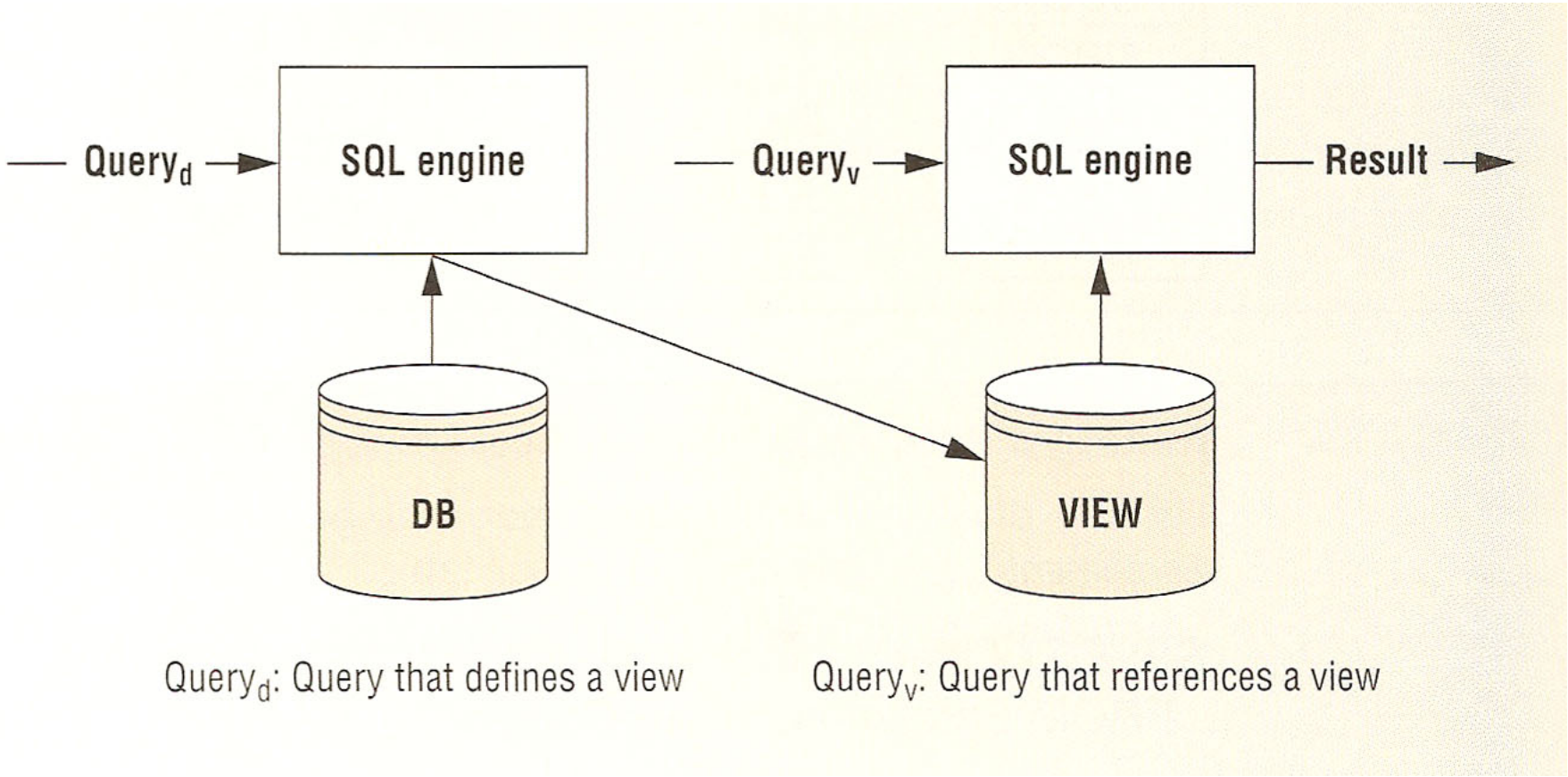
Faculty Work Load Report for the 2005–2006 Academic Year

Department	Name	Term	Offer Number	Units	Limit	Enrollment	Percent Full	Low Enrollment
FIN								
	JULIA MILLS							
		WINTER	5678	4	20	1	5.00%	☑
		Summary for 'term' = WINTER (1 detail record)						
		Sum		4		1		
		Avg					5.00%	
	Summary for JULIA MILLS							
	Sum			4		1		
	Avg						5.00%	
Summary for 'department' = FIN (1 detail record)								

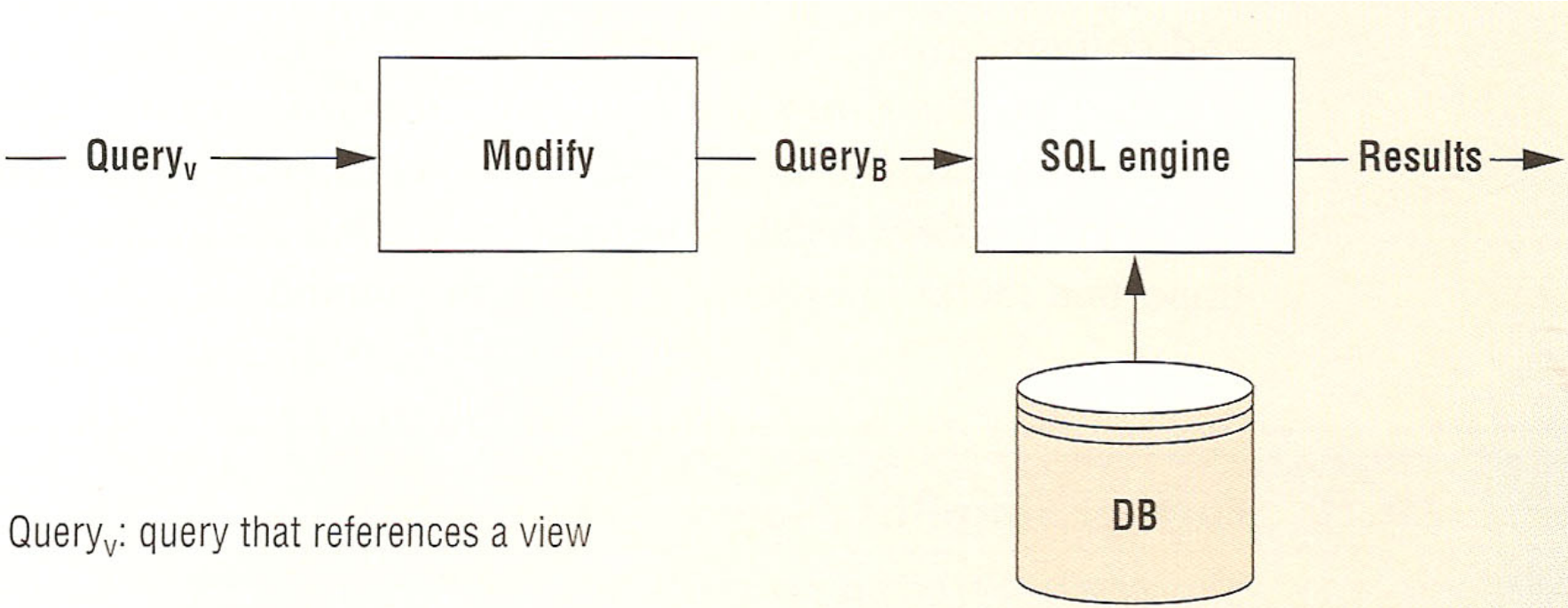
PROCESSING QUERIES WITH VIEW REFERENCES

- **View Materialization** is a method to process a query on a view by executing the query directly on the stored view. The stored view can be materialized on demand (when the view query is submitted) or periodically rebuild from the base tables.
- **View Modification** is a method to process a query on a view involving the execution of only one query. A query using a view is translated into a query using base table by replacing references to the view with its definition.

PROCESS FLOW OF VIEW MATERIALIZATION



PROCESS FLOW OF VIEW MODIFICATION



Query$_V$: query that references a view

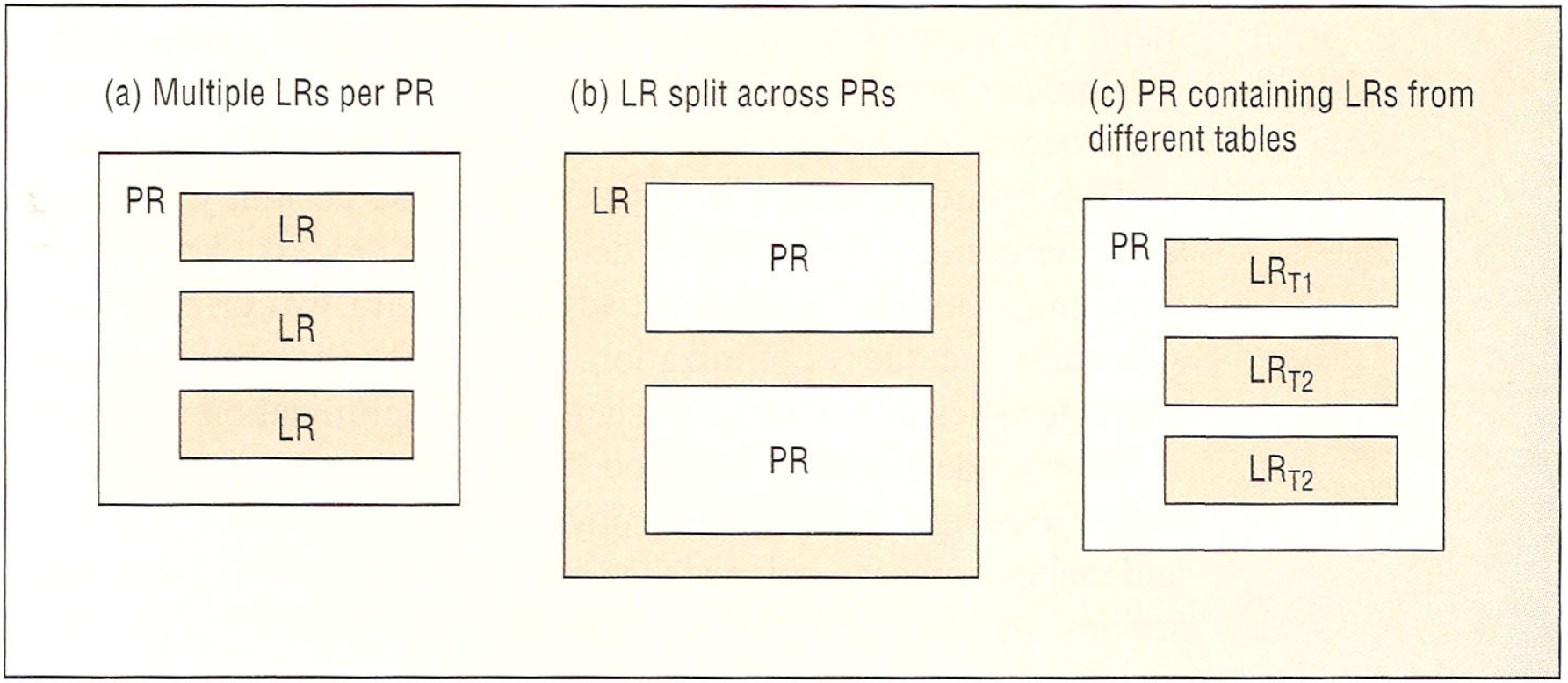
Query$_B$: modification of Query$_V$ such that references to the view are replaced by references to base tables.

INDEXES

STORAGE LEVEL OF DATABASE

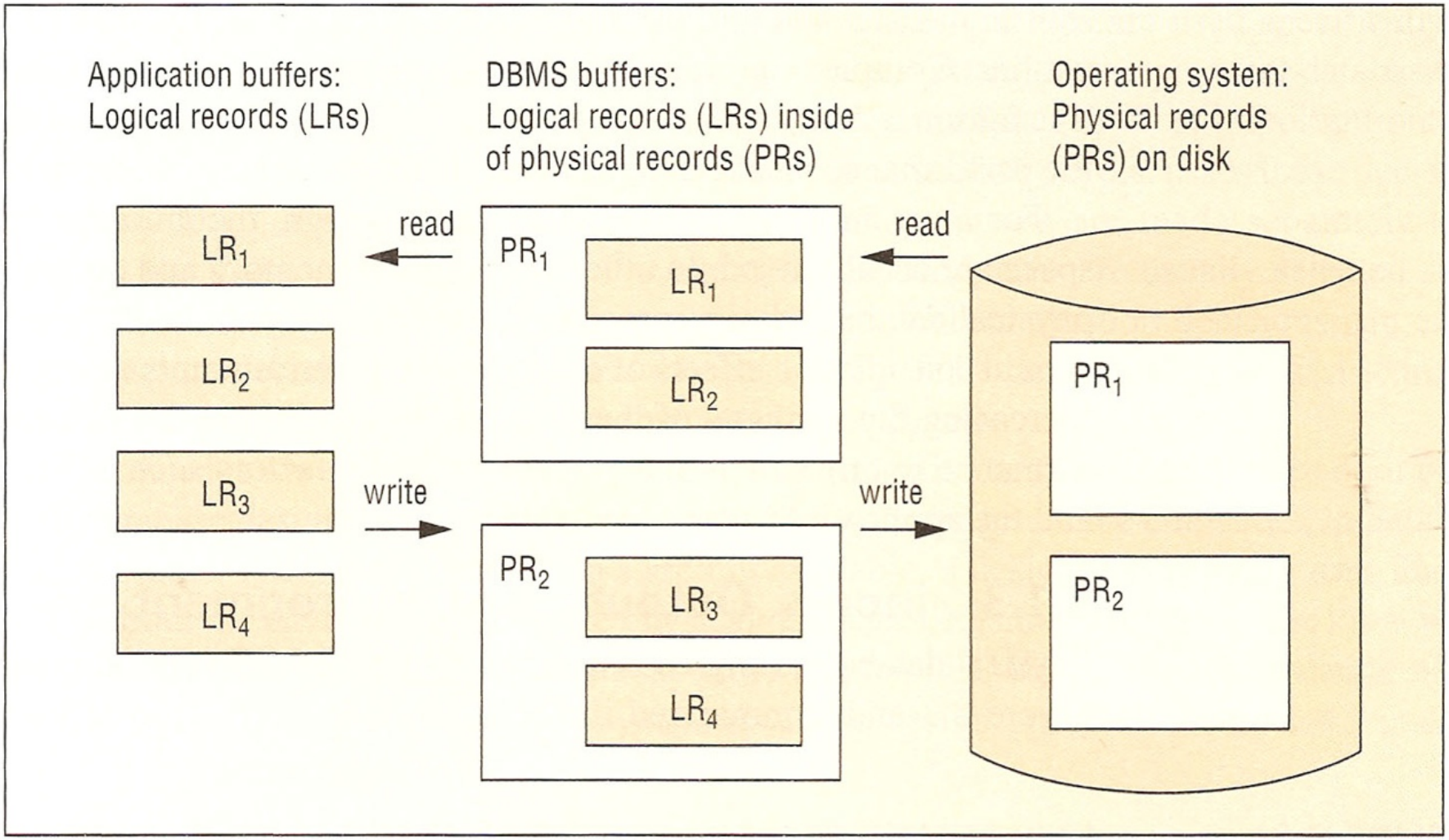
- Physical record is the collection of bytes that are transferred between volatile storage in main memory and stable storage on a disk.
- The number of physical record accesses is an important measure of database performance.

RELATIONSHIP BETWEEN LR AND PR

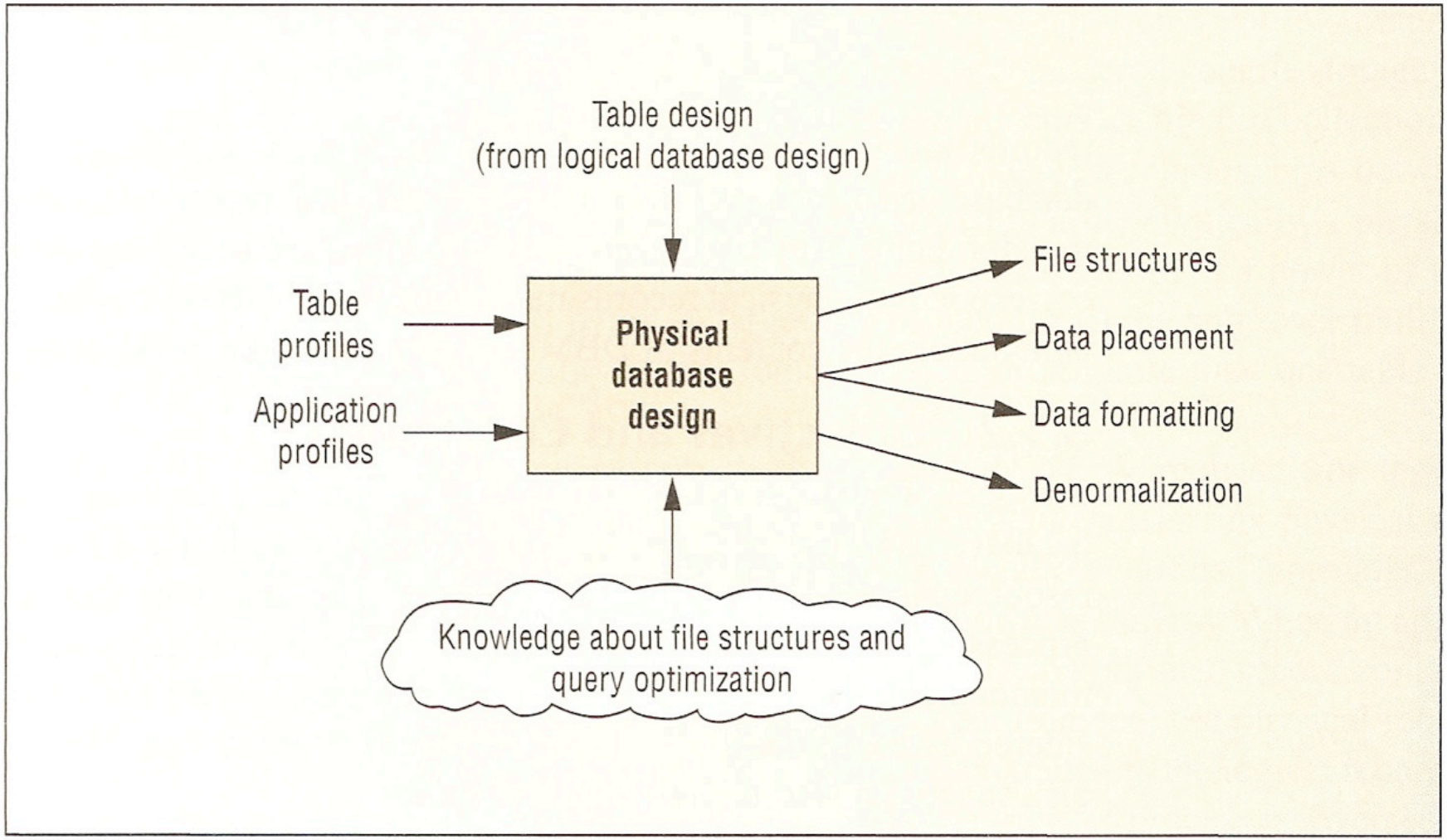


- LR is Logical Records
- PR is Physical Records

TRANSFERRING PHYSICAL RECORDS



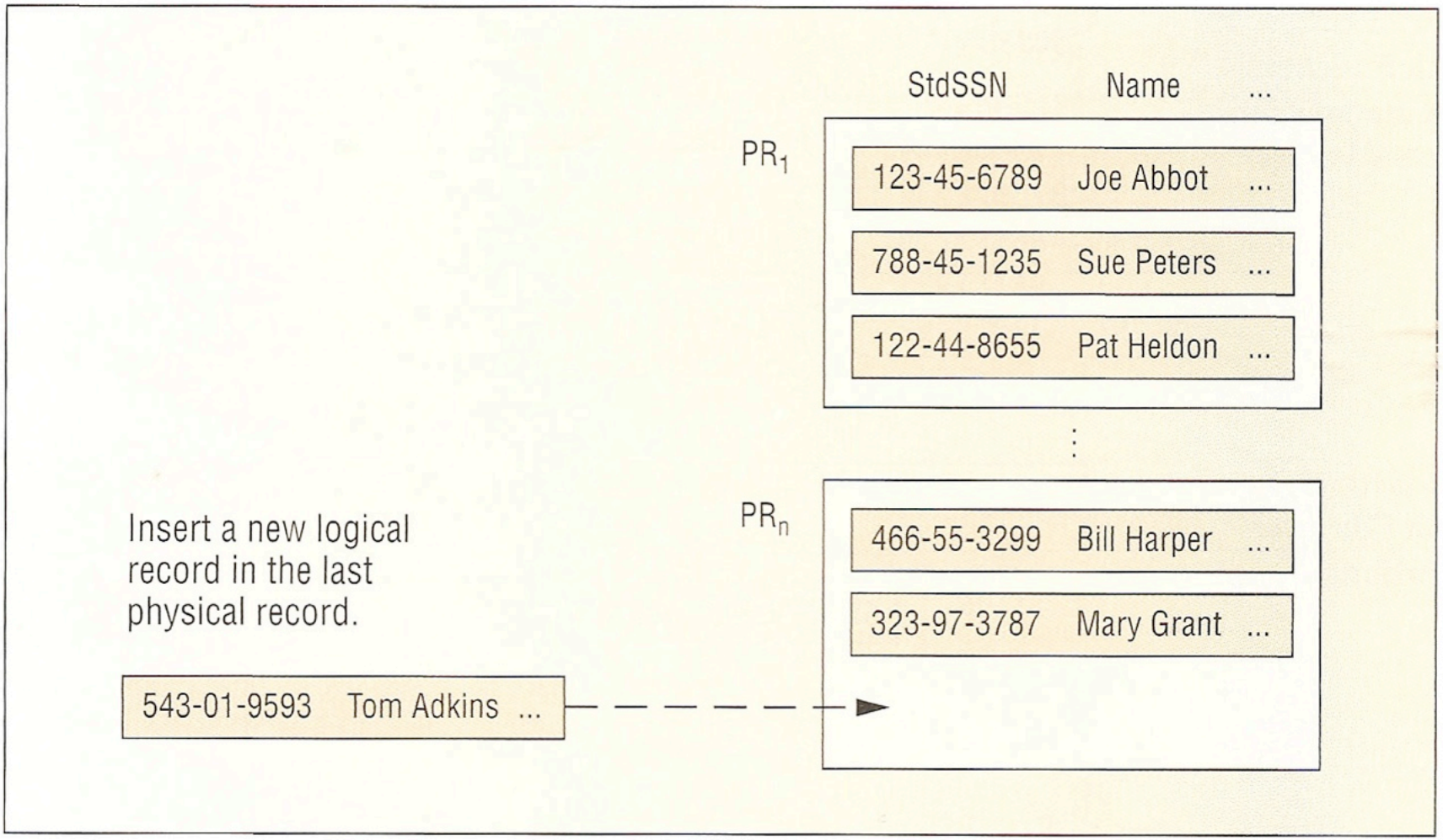
INPUTS, OUTPUTS, AND ENVIRONMENT



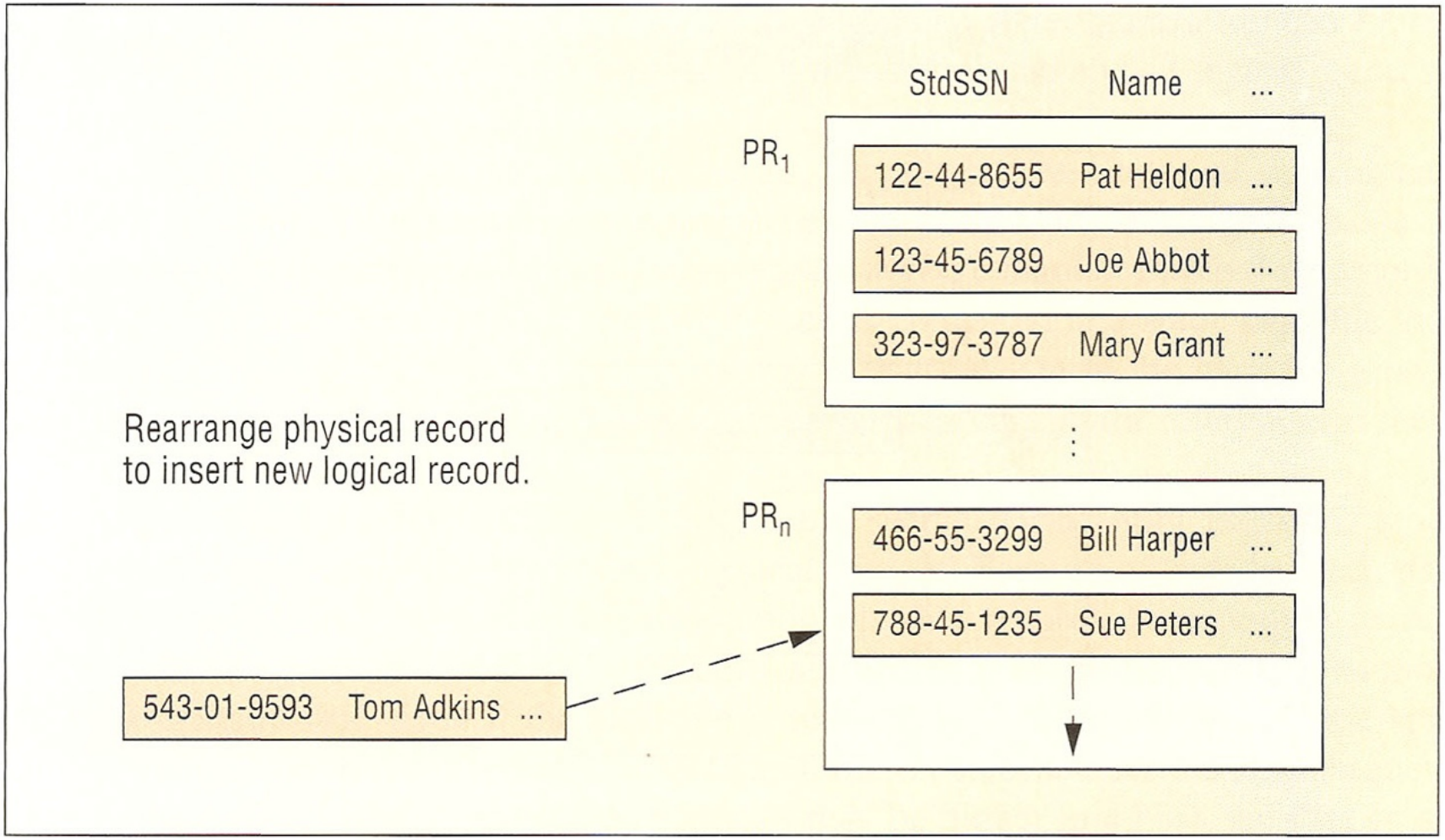
SEQUENTIAL FILES (1)

- Sequential file is a simple file organization in which records are stored in insertion order or by key value. Sequential files are simple to maintain and provide good performance for processing large numbers of records.

SEQUENTIAL FILES (2)



SEQUENTIAL FILES (3)

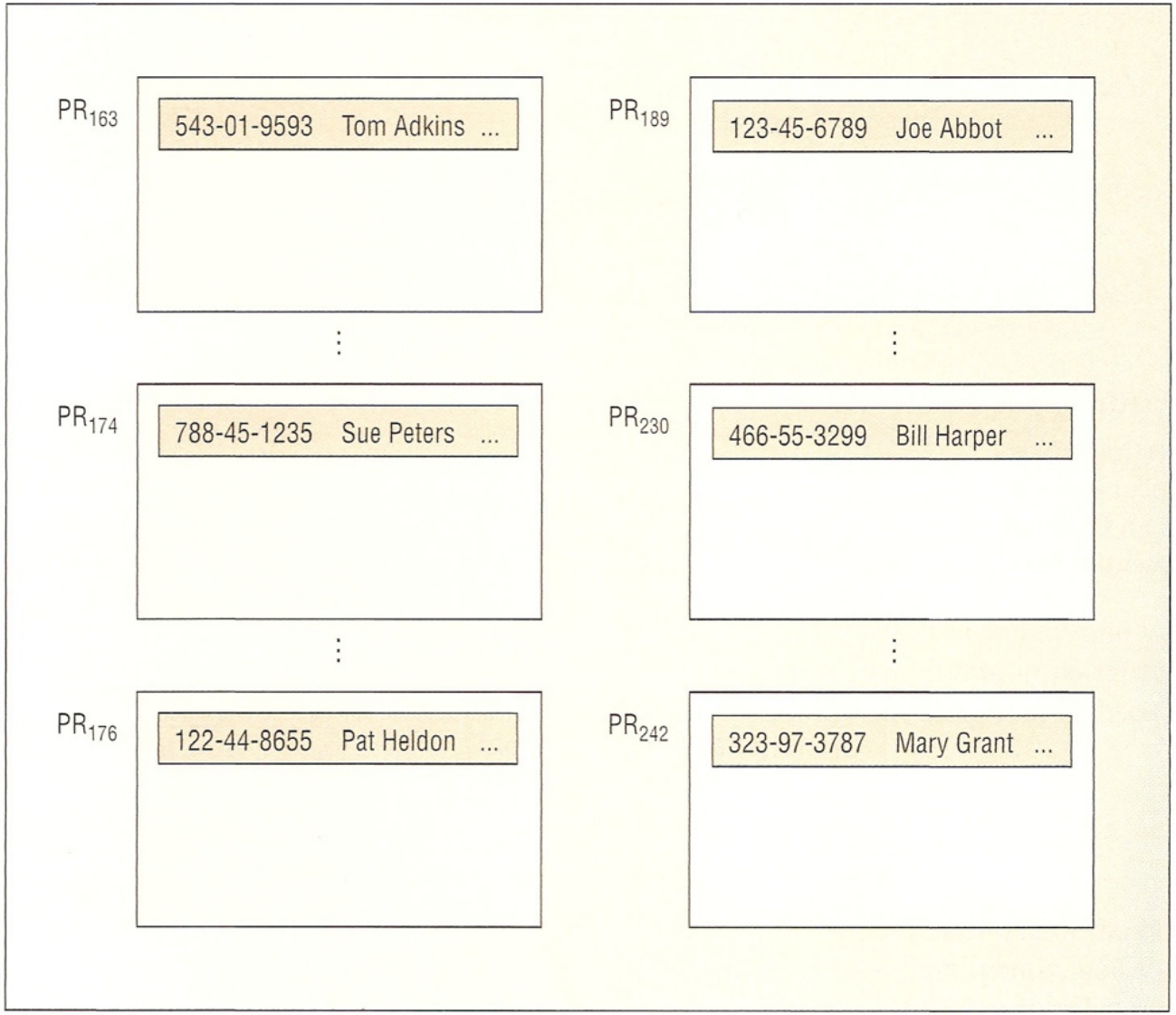


HASH FILES (1)

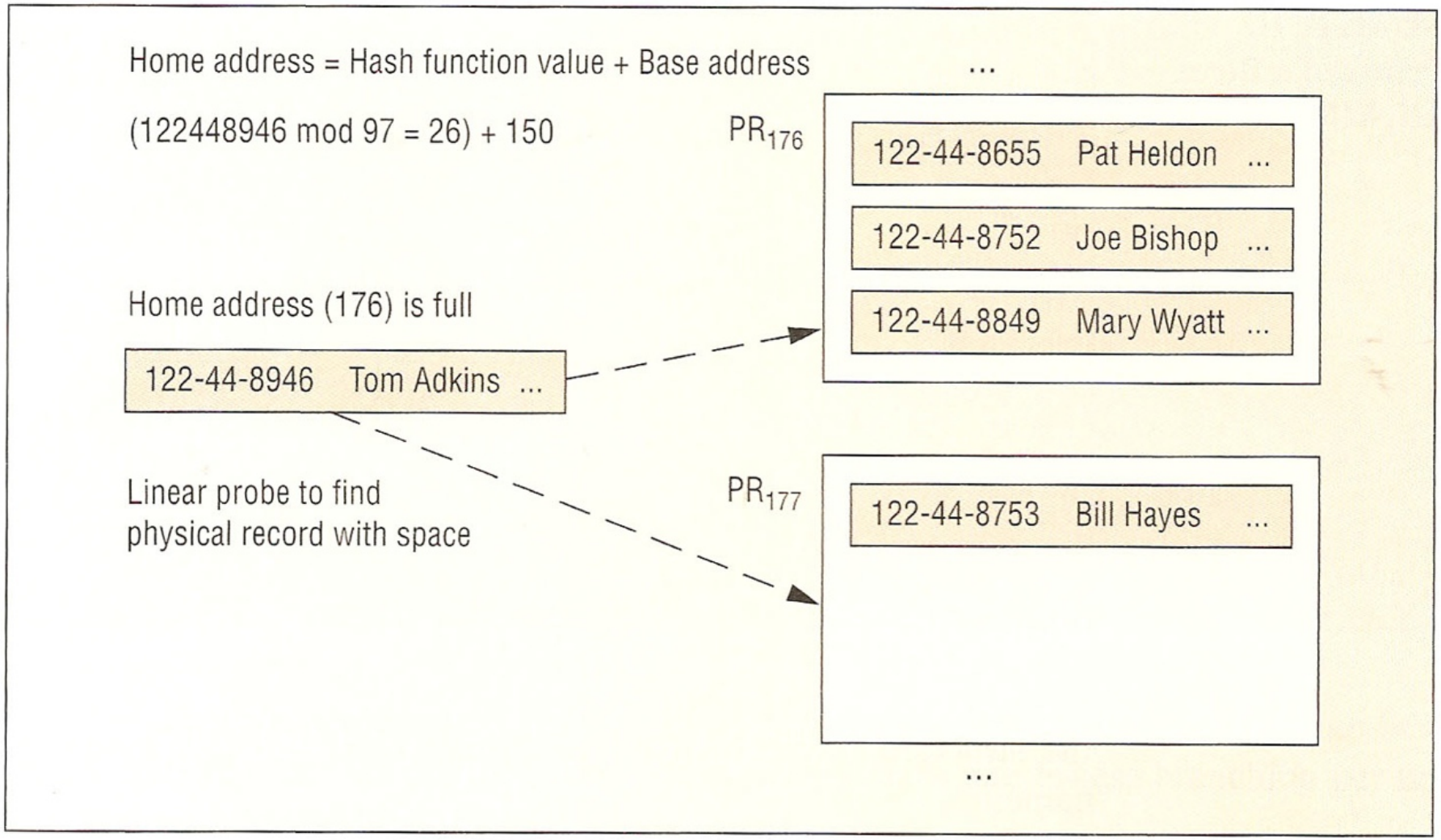
- Hash file is a specialized file structure that supports search by key. Hash files transform a key value into an address to provide fast access.

StdSSN	StdSSN Mod 97	PR Number
122448655	26	176
123456789	39	189
323973787	92	242
466553299	80	230
788451235	24	174
543019593	13	163

HASH FILES (2)

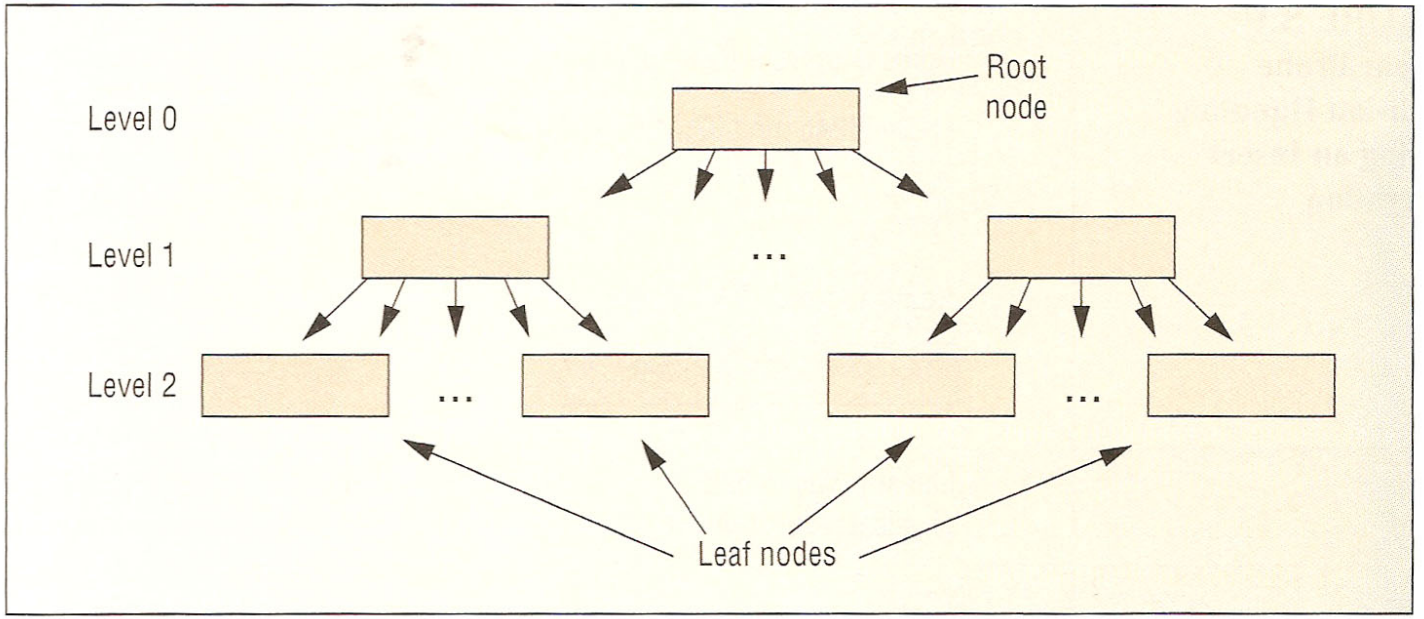

PR163
543-01-9593 Tom Adkins ...
PR189
123-45-6789 Joe Abbot ...
PR174
788-45-1235 Sue Peters ...
PR230
466-55-3299 Bill Harper ...
PR176
122-44-8655 Pat Heldon ...
PR242
323-97-3787 Mary Grant ...

HASH FILES (3)


Home address = Hash function value + Base address
(122448946 mod 97 = 26) + 150
Home address (176) is full
122-44-8946 Tom Adkins ...
Linear probe to find
physical record with space
PR176
PR177
122-44-8655 Pat Heldon ...
122-44-8752 Joe Bishop ...
122-44-8849 Mary Wyatt ...
122-44-8753 Bill Hayes ...

MULTIWAY TREE (B-TREE) FILES (1)

- B-Tree file is a popular file structure supported by most DBMSs because it provides good performance both on key search as well as sequential search.
- A B-Tree file is a balanced, multiway tree.



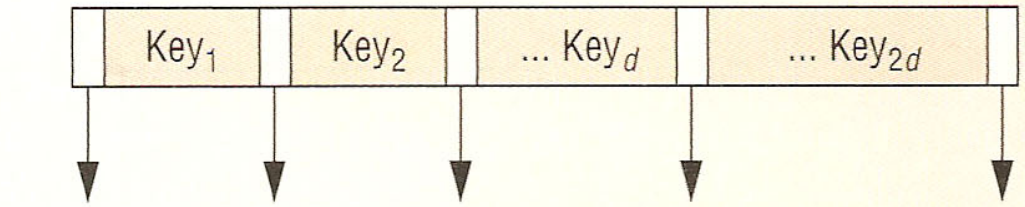
MULTIWAY TREE (B-TREE) FILES (2)

- Some of the characteristics are possible meanings for balanced tree.
  - Balanced: all leaf nodes resize on the same level of the tree.
  - Bushy: the number of branches from a node is large, perhaps 50 to 200 branches. Multiway, meaning more than two, is a synonym for bushy.
  - Block-Oriented: each node in a B-Tree is a block or physical record. To search a B-Tree you start in the root node and follow a path to a leaf node containing data of interest.

MULTIWAY TREE (B-TREE) FILES (3)

- Dynamic: the shape of a B-Tree changes as logical records are inserted and deleted.
- Ubiquitous: the B-Tree is a widely implemented and used file structure.

MULTIWAY TREE (B-TREE) FILES (4)



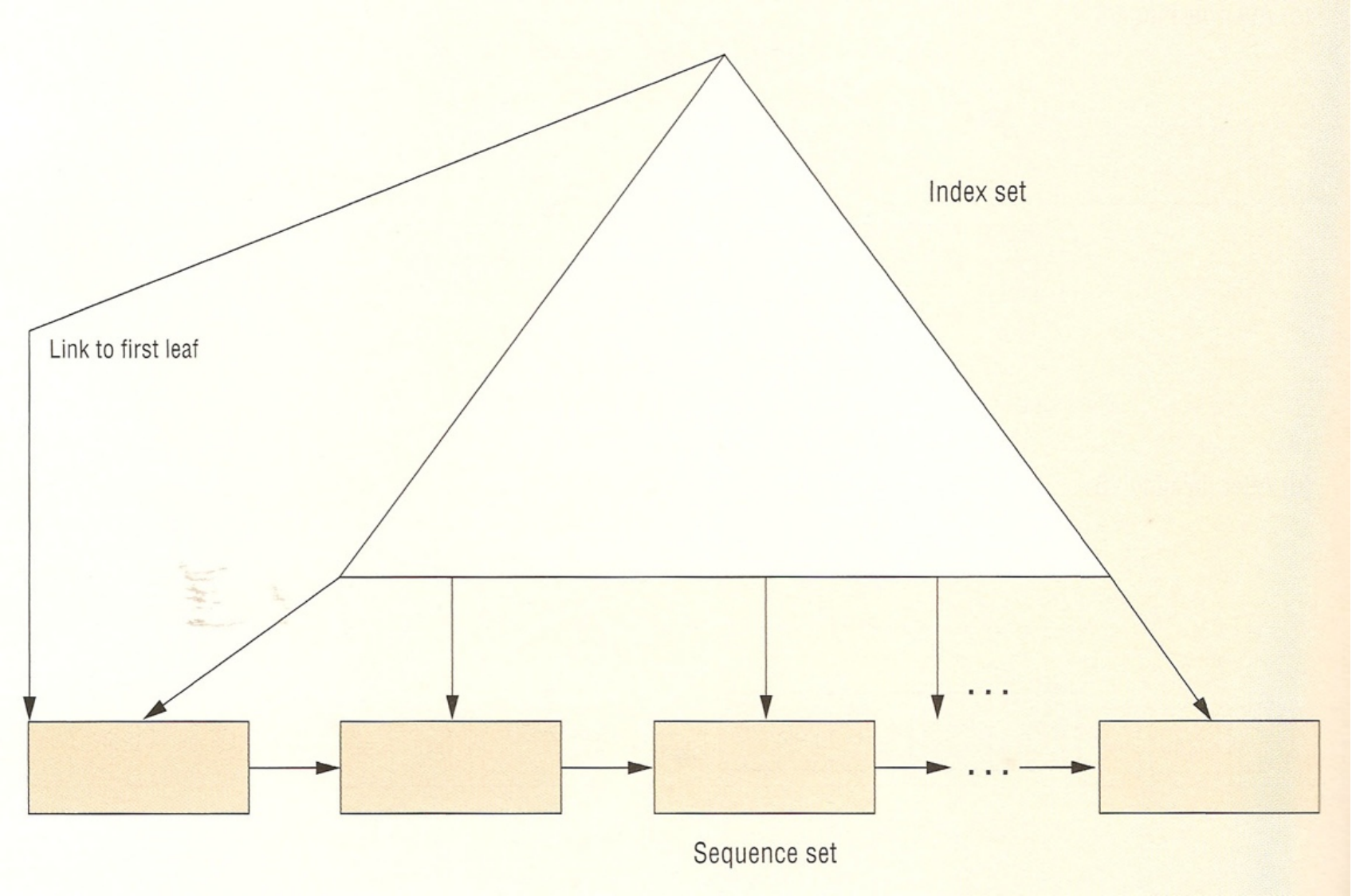
Each nonroot node contains at least half capacity (d keys and d+1 pointers).

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

B+TREE FILE (1)

- The most popular variation of the B-Tree. In a B+Tree, all keys are redundantly stored in the leaf nodes.
- The B+Tree provides improved performance on sequential and range searches.

B+TREE FILE (2)



BITMAP INDEXES (1)

- Bitmap index is a secondary file structure consisting of a column value and a bitmap.
- A bitmap contain one bit position for each row of a referenced table.
- A bitmap column index references the rows containing the column value.
- A bitmap join index references the rows of a child table that join with rows of the parent table containing the column.
- Bitmap indexes work well for stable columns with few values typical of tables in a data warehouse.

BITMAP INDEXES (2)

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 Column Index on FacRank

FacRank	Bitmap
Asst	110010010001
Assc	001000100100
Prof	000101001010

MYSQL STORAGE ENGINES

- A storage engine or "database engine" is the underlying software component that a database management system (DBMS) uses to create, read, update and delete (CRUD) data from a database.
- MyISAM and InnoDB are two popular storage engines.

MYISAM VS INNODB

- The major difference between MyISAM and InnoDB is in referential integrity and transactions.
  - Referential Integrity
  - Transactions and Atomicity
  - Table-locking vs Row-locking
  - Transactions and Rollbacks
  - Reliability (Crash Recovery)
  - FULLTEXT Indexing
  - Disk and Memory Use

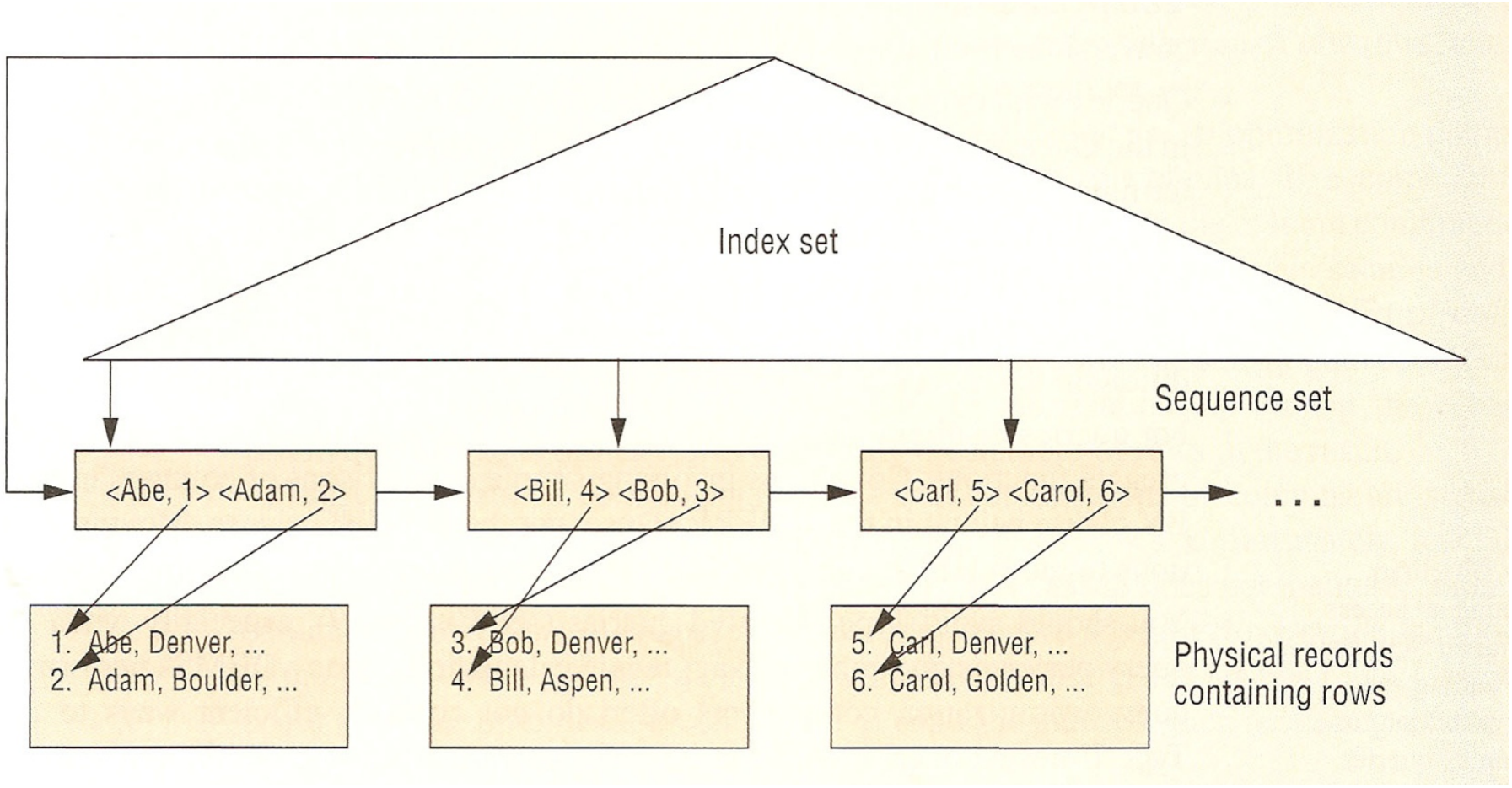
TYPE OF INDEXES

- Type of indexes in mysql
  - PRIMARY
  - UNIQUE
  - INDEX (Simple)
  - SPATIAL
  - FULLTEXT
  - DESCENDING (MySQL 8.0 or Above)
- Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees. Exceptions are that indexes on SPATIAL data types use R-trees, and that MEMORY tables also support hash indexes.

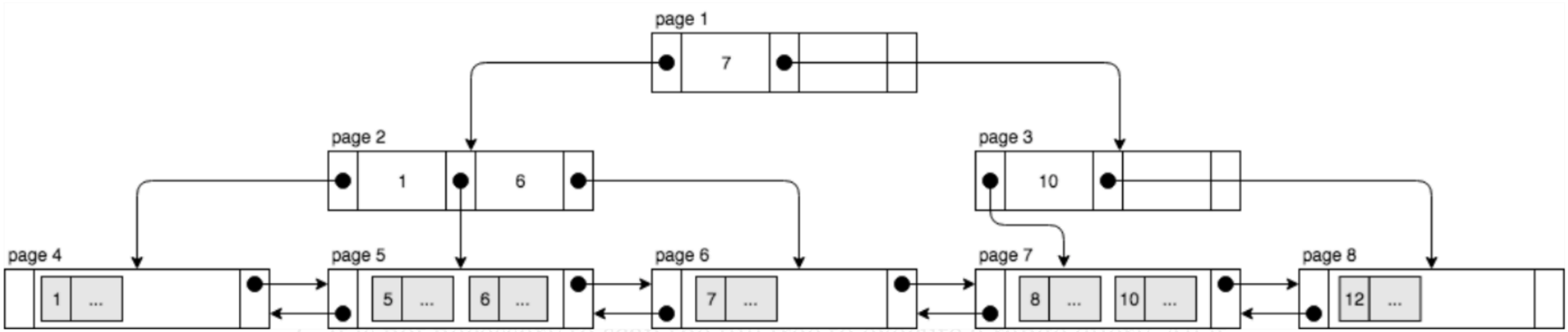
CLUSTER AND NON-CLUSTER

- Index is a secondary file structure that provides an alternative path to the data.
- In a clustering index, the order of the data records is close to the index order.
- In a non-clustering index, the order of the data records is unrelated to the index order.

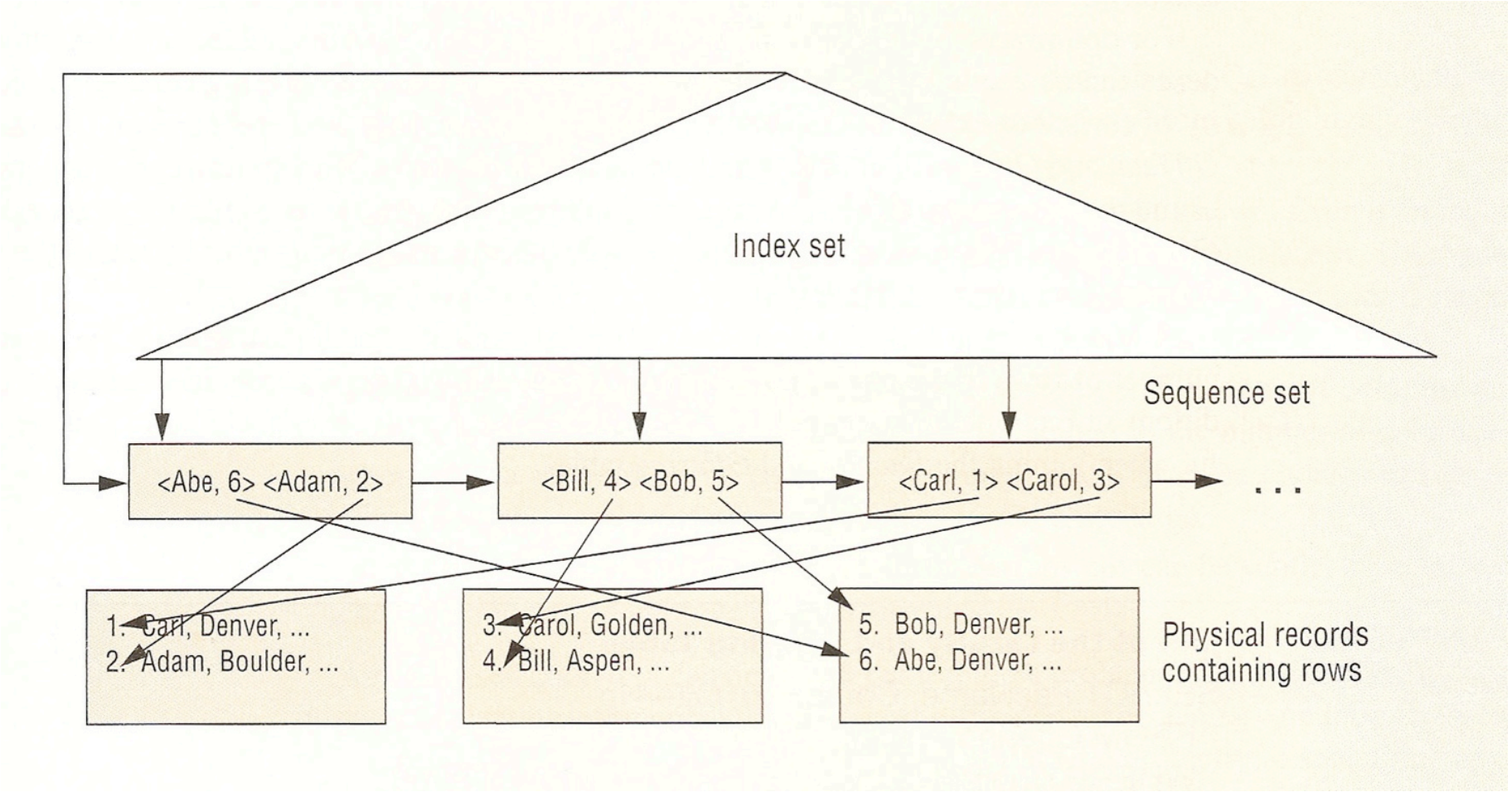
CLUSTERING INDEX EXAMPLE



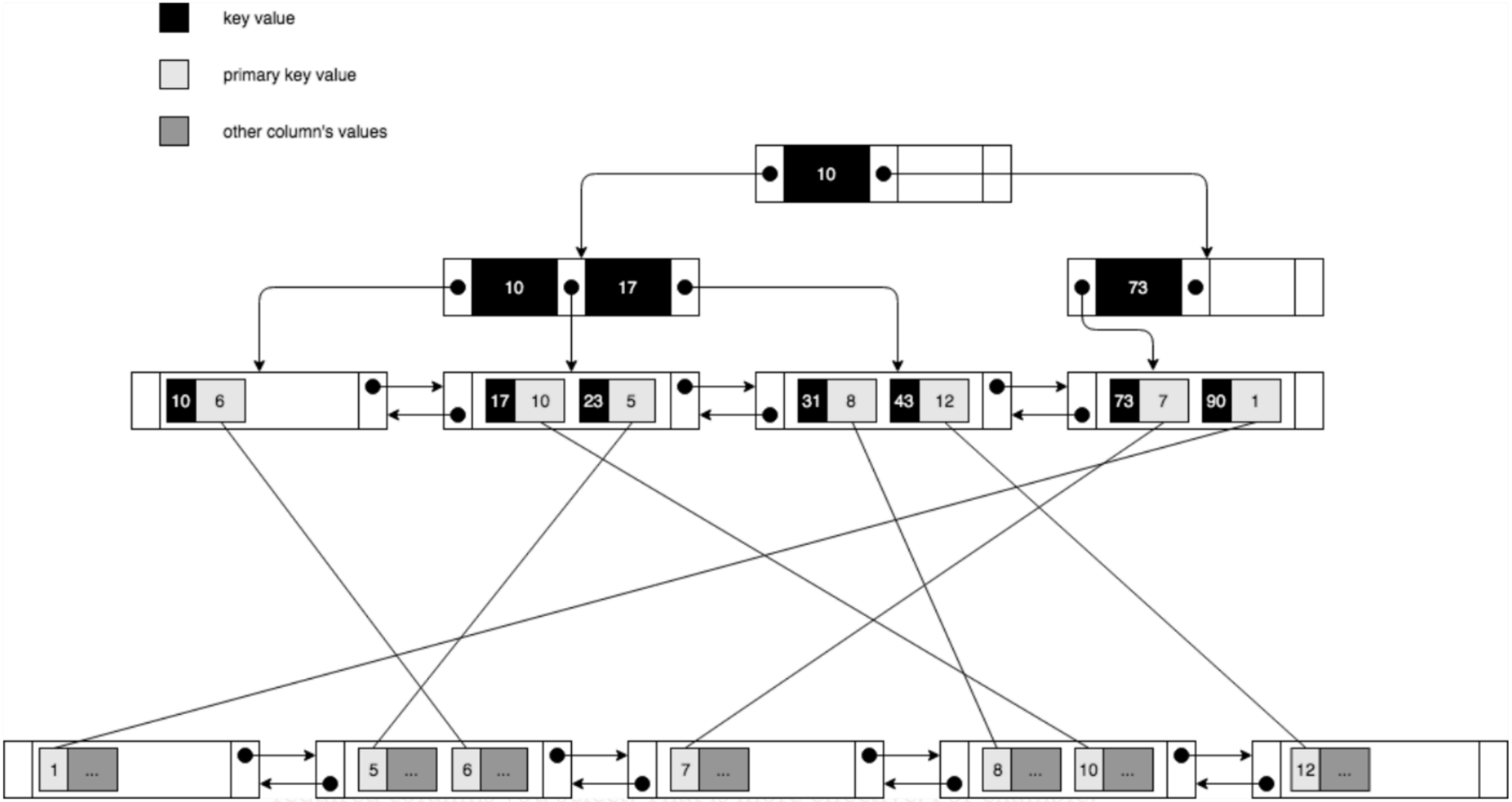
CLUSTERING INDEX EXAMPLE



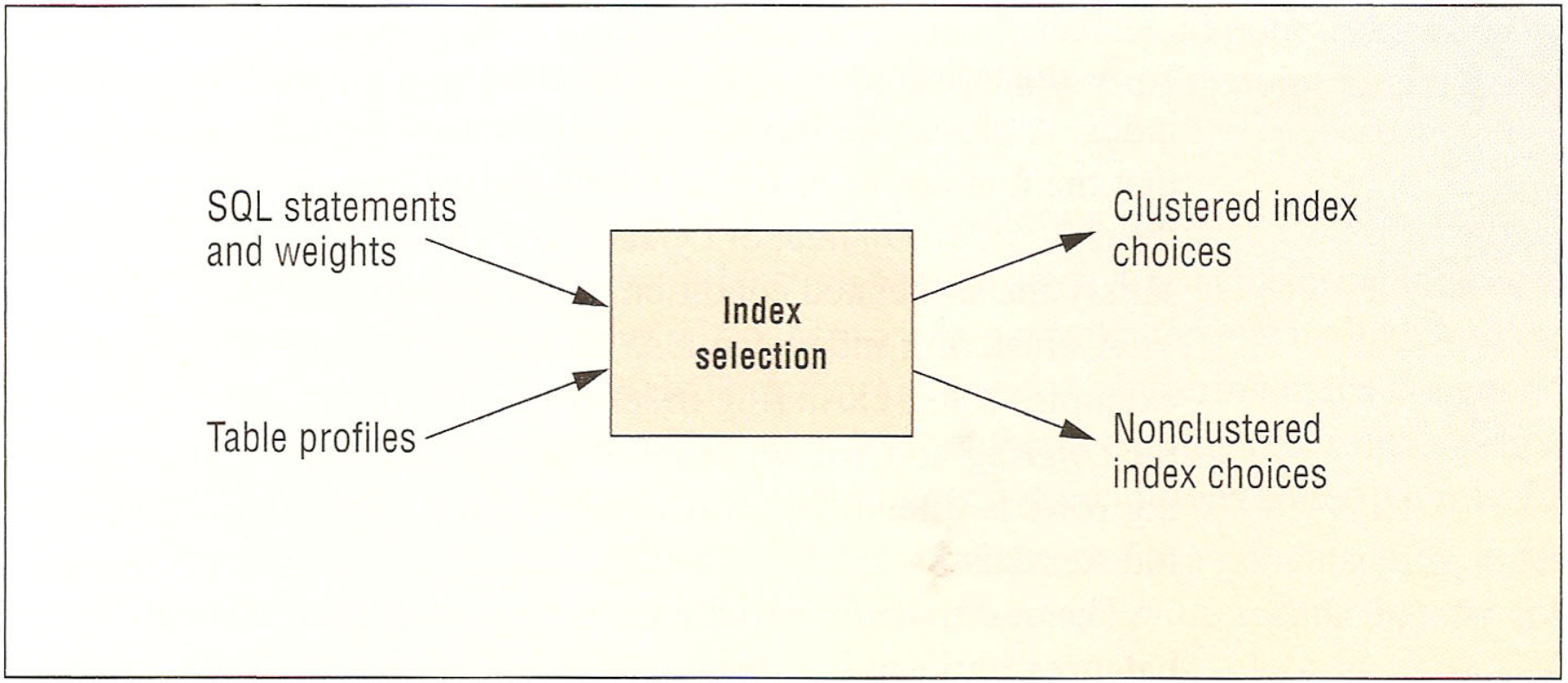
NON-CLUSTERING INDEX



NON-CLUSTERING INDEX EXAMPLE



INPUTS AND OUTPUTS OF INDEX SELECTION



SELECTION RULES (1)

- **Rule 1:** A primary key is a good candidate for a clustering index.
- **Rule 2:** To support join, consider indexes on foreign keys. A non-clustering index on a foreign key is a good idea when there are important queries with highly selective conditions on the related primary key table. A clustering index is a good choice when most joins use a parent table with a clustering index on its primary key, and the queries do not have highly selective conditions on the parent table.

SELECTION RULES (2)

- **Rule 3:** A column with many values may be a good choice for a non-clustering index if it is used in equality conditions. The term many values means that the column is almost unique.
- **Rule 4:** A column used in highly selective range conditions is a good candidate for a non-clustering index.
- **Rule 5:** A combination of columns used together in query conditions may be good candidates for non-clustering indexes if the join conditions return few rows, the DBMS optimizer supports multiple index access, and the columns are stable. Individual indexes should be created on each column.

SELECTION RULES (3)

- **Rule 6:** A frequently updated column is not a good index candidate.
- **Rule 7:** Volatile tables should not have many indexes.
- **Rule 8:** Stable columns with few values are good candidates for bitmap indexes if the columns appear in WHERE conditions.
- **Rule 9:** 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 column of the table.

EXAMPLE

Table	Number of Rows	Column (Number of Unique Values)
Student	30,000	StdSSN (PK), StdLastName (29,000), StdAddress (20,000), StdCity (500), StdZip (1,000), StdState (50), StdMajor (100), StdGPA (400)
Enrollment	300,000	StdSSN (30,000), OfferNo (2,000), EnrGrade (400)
Offering	10,000	OfferNo (PK), CourseNo (900), OffTime (20), OffLocation (500), FacSSN (1,500), OffTerm (4), OffYear (10), OffDays (10)
Course	1,000	CourseNo (PK), CrsDesc (1,000), CrsUnits (6)
Faculty	2,000	FacSSN (PK), FacLastName (1,900), FacAddress (1,950), FacCity (50), FacZip (200), FacState (3), FacHireDate (300), FacSalary (1,500), FacRank (10), FacDept (100)

Column	Index Kind	Rule
Student.StdSSN	Clustering	1
Student.StdGPA	Nonclustering	4
Offering.OfferNo	Clustering	1
Enrollment.OfferNo	Clustering	2
Faculty.FacRank	Bitmap	8
Faculty.Dept	Bitmap	8
Offering.OffTerm	Bitmap	8
Offering.OffYear	Bitmap	8

INDEX MATCHING

- A composite index matches conditions according to the following rules:
  - The first column of the index must have a matching condition.
  - Columns match from left (MS) to right (LS). Matching stops when the next column in the index is not matched.
  - At most, one BETWEEN condition matches. No other conditions match after the BETWEEN condition.
  - At most, one IN condition matches an index column. Matching stops after the next matching condition. The second matching condition cannot be IN or BETWEEN.

INDEX MATCHING EXAMPLE

Condition	Index	Matching Notes
C1 = 10	C1	Matches index on C1
C2 BETWEEN 10 AND 20	C2	Matches index on C2
C3 IN (10, 20)	C3	Matches index on C3
C1 < > 10	C1	Does not match index on C1
C4 LIKE 'A%'	C4	Matches index on C4
C4 LIKE '%A'	C4	Does not match index on C4
C1 = 10 AND C2 = 5 AND C3 = 20 AND C4 = 25	(C1,C2,C3,C4)	Matches all columns of the index
C2 = 5 AND C3 = 20 AND C1= 10	(C1,C2,C3,C4)	Matches the first three columns of the index
C2 = 5 AND C4 = 22 AND C1 = 10 AND C6 = 35	(C1,C2,C3,C4)	Matches the first two columns of the index
C2 = 5 AND C3 = 20 AND C4 = 25	(C1,C2,C3,C4)	Does not match any columns of the index: missing condition on C1
C1 IN (6, 8, 10) AND C2 = 5 AND C3 IN (20, 30, 40)	(C1,C2,C3,C4)	Matches the first two columns of the index: at most one matching IN condition
C2 = 5 AND C1 BETWEEN 6 AND 10	(C1,C2,C3,C4)	Matches the first column of the index: matching stops after the BETWEEN condition

CREATE AN INDEX

- The format of the CREATE INDEX statement is:

```
CREATE [UNIQUE] INDEX IndexName
ON TableName (columnName [ASC | DESC] [, . . . ])
```

- Example:

```
CREATE UNIQUE INDEX StaffNoInd ON Staff (staffNo);
CREATE UNIQUE INDEX PropertyNoInd ON PropertyForRent (propertyNo);
```

SHOW INDEX

- Display index information of a table. The format of the SHOW INDEX statement is:

```
SHOW INDEX FROM table_name
[WHERE conditions]
```

REMOVING AN INDEX

- The format of the DROP INDEX statement is:

  ALTER TABLE table_name DROP INDEX index_name
  DROP INDEX table_name.index_name

DESIGNING INDEXES (1)

- Designing indexes effectively can make a huge difference in query performance, but over-indexing can hurt write performance and storage.
- Here are some detailed suggestions and principles for indexing in databases:
  - Understand Query Patterns
    - Index columns used in WHERE, JOIN, ORDER BY, GROUP BY clauses.
    - Avoid indexing columns that are rarely used in searches.
  - Consider Composite (Multi-Column) Indexes
    - When queries filter on multiple columns together, a composite index can be more effective than multiple single-column indexes.

DESIGNING INDEXES (2)

- Choose the Right Type of Index
  - B-Tree Index: Good for range queries, equality, sorting (=, <, >, BETWEEN).
  - Hash Index: Fast equality checks, but not good for ranges.
  - Bitmap Index: Efficient for low-cardinality columns (like gender, status) but not ideal for high-cardinality ones.
  - Full-Text Index: For text search across large string fields.
  - Spatial Index: For geographic data.
- Keep Index Size in Mind
  - ndexes consume disk space and memory, and can slow down writes (INSERT, UPDATE, DELETE).
  - Avoid indexing very large columns unnecessarily (e.g., long text fields)

DESIGNING INDEXES (3)

- Monitor and Tune Indexes
  - Remove indexes that are rarely used, since they only add overhead.
- Rule of Thumb:
  - Primary Key usually automatically creates a unique index.
  - Foreign Keys are often good candidates for indexing to speed up joins.
  - Indexes can improve query performance, but too many can degrade insert/update performance.

"Index columns frequently filtered or joined, avoid over-indexing, and always monitor usage."