Chapter 6: Physical Database Design and Performance

Modern Database Management

6th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden

Robert C. Nickerson

ISYS 464 – Spring 2003

Topic 23

Database Development Process

- Database planning
- Database requirements analysis
- Conceptual database design
- Logical database design
- Physical database design
- Database implementation

Physical Database Design

- Purpose - translate the logical description of data into the *technical specifications* for storing and retrieving data
- Goal - create a design for storing data that will provide *adequate performance* and insure *database integrity*, *security* and *recoverability*

Physical Design Process

Inputs

- Normalized relations
- Volume estimates
- Attribute definitions
- Response time expectations
- Data security needs
- Backup/recovery needs
- Integrity expectations
- DBMS technology used

Leads to

Decisions

- Attribute data types
- Physical record descriptions (doesn't always match logical design)
- File organizations
- Indexes and database architectures
- Query optimization

Designing Fields

- Field: smallest unit of data in database
- Field design
  - Choosing data type
  - Controlling data integrity

Choosing Data Types

Choose data type for field so as to:

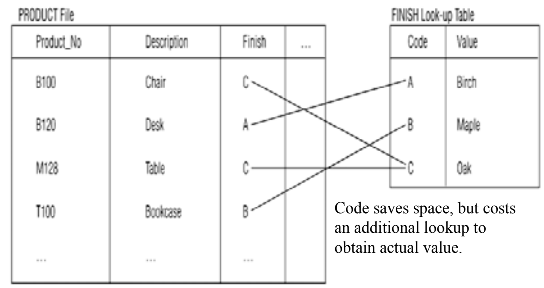
- Minimize storage space (smallest possible field)
- Represent all values (large enough field)
- Improve data integrity (type of data allowed)
- Support needed data manipulation (type of data)

Use coding, compression, encryption if necessary

Oracle Data Types

- CHAR – fixed-length character
- VARCHAR or VARCHAR2 – variable-length character (memo)
- SMALLINT, INTEGER – integer number
- DEC or NUMBER – number with decimal positions
- DATE – actual date
- BLOB – binary large object (good for graphics, sound clips, etc.)

Figure 6.2
Example code-look-up table (Pine Valley Furniture Company)



Controlling Data Integrity

Data validation controls

- Data type provides some control of type of data that can be entered into field
- Default value - assumed value if no explicit value (DEFAULT option in SQL)
- Range control – allowable value limitations (constraints or validation rules; CHECK option in SQL)
- Null value control – allowing or prohibiting empty fields (NOT NULL option in SQL)
- Referential integrity – range control (and null value allowances) for foreign-key to primary-key match-ups

Handling Missing Data

- Enforce NOT NULL constraint
- Assign a DEFAULT value
- Code application so as to ignore missing values (if value is not significant)
- Report any missing values for manual correction
- Don’t make up data

Denormalization

- Transforming ***normalized*** relations into ***unnormalized*** physical record specifications (higher NFs to lower NFs)
- Benefits:
  - Can improve performance (speed) be reducing number of table lookups (i.e *reduce number of necessary join queries*)
- Costs (due to data duplication)
  - Wasted storage space
  - Data integrity/consistency threats
  - Modification anomalies
- Common denormalization opportunities (create fewer tables)
  - One-to-one relationship (Fig 6.3)
  - Many-to-many relationship with attributes (Fig. 6.4)
  - Reference data (1:N relationship where 1-side has data not used in any other relationship) (Fig. 6.5)

Denormalization of Relations in 1:1 Relationship (Fig 6-3)

Normalized relations:

Student (Student_ID, Campus_Address)

Application (Application_ID, Application_Date, Qualifications, Student_ID)

Denormalized relation:

Student (Student_ID, Campus_Address, Application_ID, Application_Date, Qualifications)

Results in Nulls because application is optional

Denormalization of Relations in Associative Relationship (Fig 6-4)

Normalized relations:

Vendor (Vendor_ID, Address, Contact_Name)

Item (Item_ID, Description)

Price_Quote (Vendor_ID, Item_ID, Price)

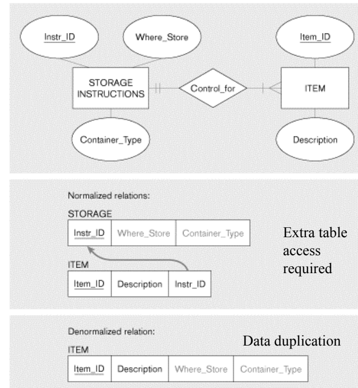
Denormalized relation:

Vendor (Vendor_ID, Address, Contact_Name)

Item_Quote (Vendor_ID, Item_ID, Description, Price)

Results in *significant* duplication of data in Item_Quote

Fig 6.5 – A possible denormalization situation: reference data (Where_Store, Container_Type)



Partitioning

- "Denormalize" to create more tables (not fewer as before)
- Horizontal Partitioning: Distributing the rows of a table into several separate files
  - Useful for situations where different users need access to different rows
  - Example: Partition customer data by sales region (can create supertype/subtype relationship)
- Vertical Partitioning: Distributing the columns of a table into several separate files
  - Useful for situations where different users need access to different columns
  - Example: Partition customer data into sales related columns and billing related columns
  - The primary key must be repeated in each file (1:1 relationship)
- Combinations of Horizontal and Vertical

Partitions often correspond with User Schemas (user views)

Partitioning

- Advantages of Partitioning:
  - Records used together are grouped together
  - Each partition can be optimized for performance
  - Security, recovery
  - Partitions stored on different disks: reduces contention
  - Take advantage of parallel processing capability
- Disadvantages of Partitioning:
  - Slow retrievals across partitions
  - Complexity
  - Data duplication across partitions

Data Replication

- "Denormalize" to create duplicate data
- Purposely storing the same data in multiple locations of the database
- Improves performance by allowing multiple users to access the same data at the same time with minimum contention
- Sacrifices data integrity due to data duplication
- Best for data that is not updated often
- Sometimes used for clients that are disconnected from the system at times
- Requires data to be synchronized periodically

Three-Level View of Database

- External view (multiple): logical view of part of the database made available to a user group (subschema)
- Conceptual view: logical view of entire database (schema)
- Internal view: physical view of database as stored by the DBMS

Internal View: Physical Records

- Physical Record: A group of fields stored in adjacent memory locations and retrieved together as a unit by the DBMS; may be one or more rows, or part of a row
- Page (Block): The amount of data read or written in one I/O operation by the OS; may be one or more physical records
- Blocking Factor: The number of physical records per page (block)

I/O Process

OS retrieves first page
- OS passes first physical record in page to DBMS
  - DBMS processes rows in first physical record
- OS passes next physical record in page to DBMS
  - DBMS processes rows in next physical record
- Etc.

When there are no more physical records in page, OS retrieves next page

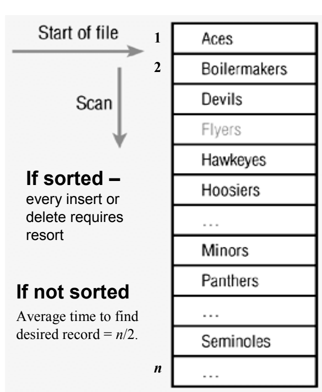
Designing Physical Files

- Physical File:
  - A named portion of secondary memory allocated for the purpose of storing physical records
- Constructs to link two pieces of data:
  - Sequential storage – one record physically follows another on disk.
  - Pointers – physical location (address) of record on disk.
- File Organization:
  - How the files are arranged on the disk.
- Access Method:
  - How the data can be retrieved based on the file organization.

Sequential File Organization

- Records physically stored in sequence usually according to primary key
- Records accessed in sequence
- Accessing a specific record: all records that physically come before the desired record must be accessed first. Average access time = N/2
- Inserting a new record requires rewriting the file
- Deleting a record may require rewriting the file
- Updating the key field of a record requires rewriting the file

Figure 6-7 (a)
Sequential file organization



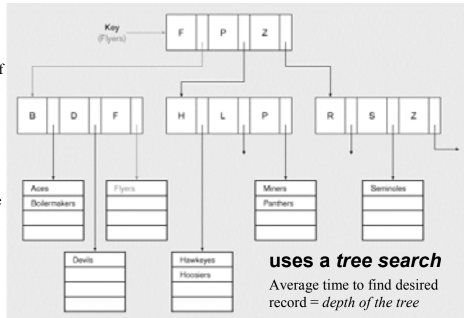
- Records of the file are stored in sequence by the primary key field values.

Indexed File Organization

- Index – a separate table that contains location of records based on a column or combination of columns
- Primary keys are automatically indexed
- Oracle has a CREATE INDEX operation, and MS ACCESS allows indexes to be created for most field types
- Indexing approaches:
  - Basic concept – SQL Indexes slides
  - Multi-level (tree) index, Fig. 6-7b
  - B+ tree

Fig. 6-7b – Multi-level (tree) index

In this example:
- Pointer points to values <= value to left of pointer
- Last level is page with multiple physical records
- Leaves of the tree are all at same level → consistent access time



B+ Tree

- B tree: A type of multiple level index (balanced tree)
- B+ tree: A type of B tree
- Benefits:
  - Each access requires the same amount of time
  - Modifications of database require only changes in index (data in database does not have to be rewritten)
  - Modifications of database do not change access time

B+ Tree

Each index node consists of:

Ptr1 Key1 Ptr2 Key2 … KeyN PtrN+1

Ptr1 = pointer to index node for values <= Key1

Prt2 = pointer to index node for values > Key1 and <= Key2

PtrN+1 = pointer to index node for values > KeyN

Last level contains pointers to physical records

See B+ tree transparency/handout

Hashed (Direct, Random) File Organization

- Address of each record determined by a hashing (randomizing) algorithm that converts the primary key into a disk address

Hashing Algorithm

Division remainder algorithm: Divide primary key by nearest prime number to size of file and use remainder to indicate disk address

Ex: File size = 5000   Nearest prime = 4999

PK = 85274   85274/4999 = 17 remainder 291

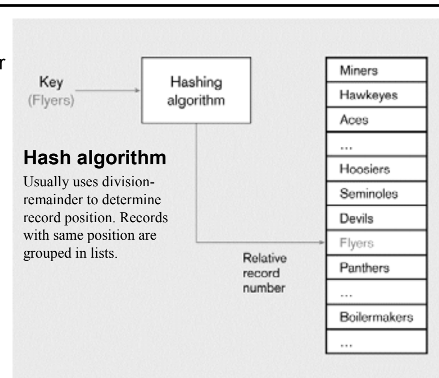
Store row with PK = 85274 as record 291 in sequence from the beginning of file

Problem: two PKs can give same disk address

Ex: PK = 90273 90273 = 18 remainder 291

Called *collision*. Collision handling algorithm needed

Fig 6-7c **Hashed** file or index organization



Comparison of File Organizations

	Sequential	Indexed	Hashed
Storage utilization	Best	In between	Worst
Speed of sequential access	Best	OK	Can't be done
Speed of random access	Can't be done	OK	Best

Fig 6-9 **Join** Index – speeds up join operations

Customer

RowID	Cust#	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
...				

Store

RowID	Store#	City	Size	Manager
20001	S4266	Dayton	K2	E2166
20002	S2654	Columbus	K3	E0245
20003	S3789	Dayton	K4	E3330
20004	S1941	Toledo	K1	E0874
...				

Join Index

CustRowID	StoreRowID	Common Value*
10001	20001	Dayton
10001	20003	Dayton
10002	20002	Columbus
10003	20002	Columbus
10004	20004	Toledo
...		

Order

RowID	Order#	Order Date	Cust#(FK)
30001	O5532	10/01/2001	C3861
30002	O3478	10/01/2001	C1062
30003	O8734	10/02/2001	C1062
30004	O9845	10/02/2001	C2027
...			

Customer

RowID	Cust#(PK)	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
...				

Join Index

CustRowID	OrderRowID	Cust#
10001	30004	C2027
10002	30002	C1062
10002	30003	C1062
10004	30001	C3861
...		

Rules for Using Indexes

Most DBMSs use some sort of indexed file structure (B+ tree)

When to use indexes?

1. Use on larger tables
2. Index the primary key of each table (automatic in Oracle)
3. Index frequently searched fields (fields frequently in WHERE clause)
4. Fields in SQL ORDER BY and GROUP BY commands
5. When there are a variety of values for a column; >100 values but not when there are <30 values

Rules for Using Indexes

6. DBMS may have limit on number of indexes per table and number of bytes per indexed field(s)
7. Null values will not be referenced from an index
8. Use indexes heavily for non-volatile databases; limit the use of indexes for volatile databases

   Why? Because modifications (e.g. inserts, deletes) require updates to occur in index files

Query Optimization

- Parallel Query Processing – specify extent of parallelism
- Override Automatic Query Optimization - maybe
- Data Block Size -- Performance tradeoffs:
  - Block contention – smaller block better
  - Random access speed – smaller block better
  - Sequential access speed – larger block better
  - Row size – block size should be multiple of row size
  - Overhead – larger block size better

Query Optimization

- Wise use of indexes
- Compatible data types in comparisons
- Simple queries
- Avoid query nesting (subqueries)
- Temporary tables for query groups
- Select only needed columns
- No sort without index

Database Implementation

Implement physical design of database

- Result is the conceptual view of database
- Code the conceptual view (schema) description (CREATE TABLE commands in SQL)
- Populate the database with test data (INSERT commands in SQL)
- Test the conceptual view of the database (data manipulation commands in SQL)

Database Implementation

Implement external views

- Code each external view (subschema) description (CREATE VIEW commands in SQL)
- Test the external views (data manipulation commands in SQL)

Database Implementation

Enhance performance

- Create indexes to improve database performance, if necessary (CREATE INDEX commands in SQL)

Provide access for system developers

- Grant privileges to analyst/programmers developing other parts of information system (GRANT commands in SQL)

Database implementation

Prepare for installation

- (After all parts of information system have been developed)
- Populate database with actual ("live") data (INSERT commands in SQL)
- Grant privileges to users and user groups (GRANT commands in SQL)