Physical Database Design Process

Physical Database Design Process

- The last stage of the database design process.
- A process of mapping the logical database structure developed in previous stages into internal model.
- The objective is to implement the database as a set of stored records, files, indexes to provide adequate performance and ensure database integrity, security and recoverability.

Major Inputs to Physical Database

- Logical database structures, such as network, hierarchical or relational data model that were developed during logical design.
- User processing requirements that were identified during requirements definition. This includes the size, frequency of use, requirement for response times, security, backup etc.
- Characteristics of the database management system.

Components of Physical Database Design

- Data Volume and Usage Analysis
- Data Distribution Strategy
- File Organizations

Data Volume and Usage Analysis

- Estimates of database size are used to select physical storage devices and storage costs estimation.
- Estimates of usage paths or patterns are used to select file organization and access methods, plans for the use of indexes, and plan a strategy for database distribution.

Data Distribution Strategy

- For organizing which uses distributed computing networks, there is a necessity to decide which nodes in the network to locate the data physically.
- Basically there are two main strategies:
  - Centralized Data Strategy ⇨
  - Distributed Data Strategy ⇨

Centralized Data Strategy

- All data are located at a single site.
- Data are not readily accessible to users at remote site and data communication costs may be high.
- If the central system fails, the whole DBMS fails totally.

Distributed Data Strategy

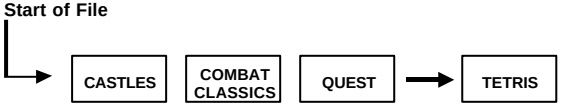
- A distributed database is a collection of data that logically belongs to the same system but physically spread over the sites of a computer network.
- This increased reliability and availability, as well as improved performance.

File Organizations

- A file organization is a technique for physically arranging the records of a file on secondary devices.
- Essential criteria in selecting file organization:
  1. Fast access for retrieval
  2. High throughout for processing transactions
  3. Efficient use of storage space
  4. Protection from failures or data loss
  5. Minimizing need for organization
  6. Accommodating growth
  7. Security from unauthorized use

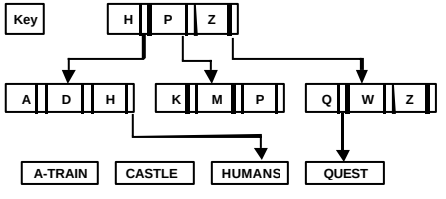
Sequential File Organization

- The records in this file are stored in sequence according to a primary key value.
- The location of a particular record in the file is not known by the storage organization.
- Scanning from the beginning to the desired record is necessary for every search.



Indexed File Organizations

- The records are either stored sequentially or nonsequentially.
- An index, which is a table of other data structure, is created to allow the user to locate individual records.

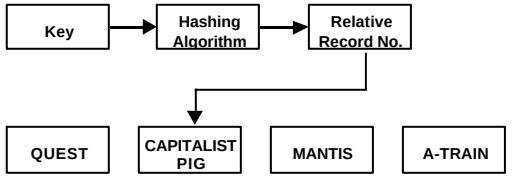


Indexed Sequential & Nonsequential

- Indexed Sequential
  - The records are stored sequentially by primary key value.
  - A simple index (called a block index) may be used, to help locate the particular page.
- Indexed Nonsequential
  - The records are stored sequentially by primary key value.
  - A full index (frequently called and inverted index) is required. Relations are often stored in this way.

Hashed File Organization

- The address for each record is determine using a hashing algorithm.
- A hashing algorithm is a routine that converts a primary key value into a relative record number.



Indexes

- Indexes may be defined on both primary and nonkey attribute values.
- Indexes are more compact than data records they reference.
- They can be kept in computer main memory to reduce secondary memory access.

When to use indexes?

- There is a tradeoff between improved performance for retrievals through the use of indexes, and degraded performance for inserting, deleting and updating the records in a file.
  - For database which are used primarily for data retrievals, example, decision support applications are encouraged to use more indexes.
  - For database that support heavy updating requirements, such as transaction processing, indexes are discouraged.

Rules for Choosing of Indexes for Relational Databases

- Specify a unique index for the primary key attribute of each table. This ensures the uniqueness of primary key values and speeds retrievals based on those values.
- Specify an index for foreign keys that are used in joining tables. This speeds up multiple-table queries.
- Specify an index for nonkey attributes that are referred to in sorting, and grouping commands in SQL.

Integrity Constraints

- The Database integrity are:
  - Entity Integrity
  - Referential Integrity
  - Domain Integrity

Domain Integrity

- The attribute values must come from the same domain and must satisfy specified class range or format restrictions.

Entity Integrity

- No attribute of a primary key can have a null value.

Referential Integrity

- A referential integrity constraints addresses the validity of references by one object in a database to some other objects the database.
- Foreign key of relation R2, must match the primary key value of some tuple in relation R2 or the foreign key value must be completely null.

Example of Referential Integrity

- The ITEM table contains a list of items, and the SUPPLIER-NO is a foreign key in the ITEM relation.
- For each row in the ITEM relation SUPPLIER-NO references the supplier who supplies that item in the SUPPLIER relation.

Example of Referential Integrity

ITEM

ITEM NO	ITEM NAME	SUPPLIER-NO
100	Gizmo	10
200	Widget	40
300	Thingie	30
400	Whatsit	10

SUPPLIER

SUPPLIER NO	SUPPLIER-NAME
10	AAA
20	BBB
30	CCC
40	DDD

Referential Integrity Rules

- We will talk about the referential integrity rules in the context of insertions and deletions.
  - Insertion Rules
  - Deletion Rules

Insertion Rule

- This states that the row should not be inserted in the referencing table unless there already exists a matching entry in the referenced table.
  - Example, 500 Klunge 50 is not permitted to be inserted into the ITEM table, because SUPPLIER-NO 50 is not available in the SUPPLIER file.

Insertion Rule

- Some application software can allow the insertion of item information, by leaving the supplier number NULL in the referencing table.
  - Example, instead of inserting "500 Kluge - - " is inserted:

ITEM

ITEM NO	ITEM NAME	SUPPLIER-NO
100	Gizmo	10
200	Widget	40
300	Thumzer	30
400	Whatsit	10
500	Kluge	--

Deletion Rules

- This states that a row should not be deleted from the referenced table if there is a matching row in the referencing table.
  - Example, all the records in the SUPPLIER cannot be deleted since there is a matching row in the referencing table, ITEM.

Deletion Rules

- However, it is possible to use nulls to enable deletion. Three possible delete rules are:
  - Restrict - which disallows the delete request.
  - Nullify - which reset to null any SUPPLIER-NO 10 in the ITEM relation, then delete SUPPLIER-NO 10 from SUPPLIER.
  - Cascade - Delete any item in the ITEM table for supplier number 10, then delete vendor number 10 from SUPPLIER.

Example

ITEM NO	ITEM NAME	SUPPLIER-NO
100	Gizmo	10
200	Widget	40
300	Thumzer	30
400	Whatsit	10

SUPPLIER-NO	SUPPLIER-NAME
20	BBB
30	CCC
40	DDD

ITEM NO	ITEM NAME	SUPPLIER-NO
200	Widget	40
300	Thumzer	30

SUPPLIER-NO	SUPPLIER-NAME
20	BBB
30	CCC
40	DDD

Denormalization

- It is not necessary to normalize fully.
- There is a tradeoff between the simplicity of tables presentation and number of physical table access for every queries.
- Our design goal may be to reduce the number of physical database tables to be accessed, by reducing the number of joins needed to derive a query answer.

Denormalization

- Denormalized rows are larger, which means that fewer rows can fit into a given secondary memory block.
- This may cause slower processing time.
- The followings are situations for denormalization
  - Two entities with a one-to-one relationship.
  - A many-to-many relationship with non-key attributes.
  - Independent reference data that are of very few.

Two Entities with a One-to-One Relationship

- Even if one of the entities is an optional participant, if the matching entity exists most of the time, then it may be wise to combine these two entities into one table.

A Many-to-Many Relationship with Nonkey Attributes

- Instead of having three tables joined together to extract data from two entities in the relationship, it may be advisable to combine columns from one of the entities into table representing the many-to-many relationship.
- This avoids one join in many queries.

Reference Data

- Suppose there are a one-to-many relationship, the reference data exists in an entity on the one-side of a one-to-many relationships.
- We should then consider merging the two entities in this situation especially when there are very few instances of the entity on the many-side for each entity instance on the one-side.

Review Questions

- What is the purpose of data column and usage analysis?
- What are the important criteria in selecting file organization?
- Explain the two popular referential integrity rules that you have learned.
- When would you consider to implement denormalisation?