
Chapter Five

Physical Database Design Methodology for Relational Database

We have established that there are three levels of database design:

- Conceptual design: producing a data model which accounts for the relevant entities and relationships within the target application domain;
- Logical design: ensuring, via normalization procedures and the definition of integrity rules, that the stored database will be non-redundant and properly connected;
- Physical design: specifying how database records are stored, accessed and related to ensure adequate performance.

It is considered desirable to keep these three levels quite separate -- one of Codd's requirements for an RDBMS is that it should maintain logical-physical data independence. The generality of the relational model means that RDBMSs are potentially less efficient than those based on one of the older data models where access paths were specified once and for all at the design stage. However the relational data model does not preclude the use of traditional techniques for accessing data - it is still essential to exploit them to achieve adequate performance with a database of any size.

We can consider the topic of physical database design from three aspects:

- What techniques for storing and finding data exist
- Which are implemented within a particular DBMS
- Which might be selected by the designer for a given application knowing the properties of the data

Thus the purpose of physical database design is:

1. How to map the logical database design to a physical database design.
 2. How to design base relations for target DBMS.
 3. How to design enterprise constraints for target DBMS.
 4. How to select appropriate file organizations based on analysis of transactions.
 5. When to use secondary indexes to improve performance.
 6. How to estimate the size of the database
 7. How to design user views
-

-
8. How to design security mechanisms to satisfy user requirements.
 9. How to design procedures and triggers.

Physical database design is the process of producing a description of the implementation of the database on secondary storage.

Physical design describes the base relation, file organization, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

- Sources of information for the physical design process include global logical data model and documentation that describes model. Set of normalized relation.
- Logical database design is concerned with the *what*; physical database design is concerned with the *how*.
- The process of producing a description of the implementation of the database on secondary storage.
- Describes the storage structures and access methods used to achieve efficient access to the data.

Steps in physical database design

1. *Translate logical data model for target DBMS*
 - 1.1. *Design base relation*
 - 1.2. *Design representation of derived data*
 - 1.3. *Design enterprise constraint*
 2. *Design physical representation*
 - 2.1. *Analyze transactions*
 - 2.2. *Choose file organization*
 - 2.3. *Choose indexes*
 - 2.4. *Estimate disk space and system requirement*
 3. *Design user view*
 4. *Design security mechanisms*
 5. *Consider controlled redundancy*
 6. *Monitor and tune the operational system*
-

1. Translate logical data model for target DBMS

This phase is the translation of the global logical data model to produce a relational database schema in the target DBMS. This includes creating the data dictionary based on the logical model and information gathered. After the creation of the data dictionary, the next activity is to understand the functionality of the target DBMS so that all necessary requirements are fulfilled for the database intended to be developed.

Knowledge of the DBMS includes:

- how to create base relations
- whether the system supports:
 - definition of Primary key
 - definition of Foreign key
 - definition of Alternate key(Unique keys)
 - definition of Domains
 - Referential integrity constraints
 - definition of enterprise level constraints

1.1.Design base relation

To decide how to represent base relations identified in global logical model in target DBMS.

Designing base relation involves identification of all necessary requirements about a relation starting from the name up to the referential integrity constraints.

For each relation, need to define:

- The name of the relation;
- A list of simple attributes in brackets;
- The PK and, where appropriate, AKs and FKs.
- A list of any derived attributes and how they should be computed;
- Referential integrity constraints for any FKs identified.

For each attribute, need to define:

- Its domain, consisting of a data type, length, and any constraints on the domain;
 - An optional default value for the attribute;
 - Whether the attribute can hold nulls.
-

-
- Whether the attribute can be derived , if do how it should be computed

The implementation of the physical model is dependent on the target DBMS since some has more facilities than the other in defining database definitions.

The base relation design along with every justifiable reason should be fully documented.

1.2.Design representation of derived data

While analyzing the requirement of users, we may encounter that there are some attributes holding data that will be derived from existing or other attributes. A decision on how to represent any derived data present in the global logical data model in the target DBMS should be devised.

Examine logical data model and data dictionary, and produce list of all derived attributes. Most of the time derived attributes are not expressed in the logical model but will be included in the data dictionary. *Whether to store derived attributes in a base relation or calculate them when required* is a decision to be made by the designer considering the performance impact.

Option selected is based on:

- Additional cost to store the derived data and keep it consistent with operational data from which it is derived;
- Cost to calculate it each time it is required.

Less expensive option is chosen subject to performance constraints.

The representation of derived attributes should be fully documented.

1.3.Design enterprise constraint

Data in the database is not only subjected to constraints on the database and the data model used but also with some enterprise dependent constraints. These constraint definitions are also dependent on the DBMS selected and enterprise level requirements.

One need to know the functionalities of the DBMS since in designing the enterprise constraints for the target DBMS some DBMS provide more facilities than others.

All the enterprise level constraints and the definition method in the target DBMS should be fully documented.

2. Design physical representation

This phase is the level for determining the optimal file organizations to store the base relations and the indexes that are required to achieve acceptable performance; that is, the way in which relations and tuples will be held on secondary storage.

Number of factors that may be used to measure efficiency:

- Transaction throughput: number of transactions processed in given time interval.
- Response time: elapsed time for completion of a single transaction.
- Disk storage: amount of disk space required to store database files.

However, no one factor is always correct.

Typically, have to trade one factor off against another to achieve a reasonable balance.

2.1. Analyze transactions

The objective here is to understand the functionality of the transactions that will run on the database and to analyze the important transactions.

Attempt to identify performance criteria, e.g.:

- Transactions that run frequently and will have a significant impact on performance;
- Transactions that are critical to the business;
- Times during the day/week when there will be a high demand made on the database (called the peak load).

Use this information to identify the parts of the database that may cause performance problems.

To select appropriate *file organizations* and *indexes*, also need to know high-level functionality of the transactions, such as:

- Attributes that are updated in an update transaction;
- Criteria used to restrict tuples that are retrieved in a query.

Often not possible to analyze all expected transactions, so investigate most 'important' ones.

To help identify which transactions to investigate, can use:

- Transaction/relation cross-reference matrix, showing relations that each transaction accesses, and/or
-

-
- Transaction usage map, indicating which relations are potentially heavily used.

To focus on areas that may be problematic:

1. Map all transaction paths to relations.
2. Determine which relations are most frequently accessed by transactions.
3. Analyze the data usage of selected transactions that involve these relations.

2.2. Choose file organization

The objective here is to determine an efficient file organization for each base relation

File organizations include Heap, Hash, Indexed Sequential Access Method (ISAM), B+-Tree, and Clusters.

Most DBMSs provide little or no option to select file organization. However, they provide the user with an option to select an *index* for every relation

2.3. Choose indexes

The objective here is to determine whether adding indexes will improve the performance of the system.

One approach is to keep tuples unordered and create as many secondary indexes as necessary.

Another approach is to order tuples in the relation by specifying a primary or clustering index.

In this case, choose the attribute for ordering or clustering the tuples as:

- Attribute that is used most often for *join operations* - this makes join operation more efficient, or
- Attribute that is used most often to access the tuples in a relation in order of that attribute.

If ordering attribute chosen is on the primary key of a relation, index will be a primary index; otherwise, index will be a clustering index.

Each relation can only have either a primary index or a clustering index. Secondary indexes provide a mechanism for specifying an additional key for a base relation that can be used to retrieve data more efficiently.

Overhead involved in maintenance and use of secondary indexes that has to be balanced against *performance improvement* gained when retrieving data.

This includes:

- Adding an index record to every secondary index whenever tuple is inserted;
- Updating a secondary index when corresponding tuple is updated;
- Increase in disk space needed to store the secondary index;
- Possible performance degradation during query optimization to consider all secondary indexes.

Guidelines for Choosing Indexes

- (1) Do not index small relations.
- (2) Index PK of a relation if it is not a key of the file organization.
- (3) Add secondary index to a FK if it is frequently accessed.
- (4) Add secondary index to any attribute that is heavily used as a secondary key.
- (5) Add secondary index on attributes that are involved in: selection or join criteria; ORDER BY; GROUP BY; and other operations involving sorting (such as UNION or DISTINCT).
- (6) Add secondary index on attributes involved in built-in functions.
- (7) Add secondary index on attributes that could result in an index-only plan.
- (8) Avoid indexing an attribute or relation that is frequently updated.
- (9) Avoid indexing an attribute if the query will retrieve a significant proportion of the tuples in the relation.
- (10) Avoid indexing attributes that consist of long character strings.

2.4. Estimate disk space and system requirement

The objective here is to estimate the amount of disk space that will be required by the database.

Purpose is to answer the following questions:

- If system already exists: is there adequate storage?
- If procuring new system: what storage will be required?

3. Design user view

To design the user views that was identified during the Requirements Collection and Analysis stage of the relational database application development lifecycle.

Define views in DDL to provide user views identified in data model

Map onto objects in physical data model

4. Design security mechanisms

To design the security measures for the database as specified by the users.

System security – Authentication

Data security-authorizations

5. Consider the Introduction of Controlled Redundancy

The objective here is to determine whether introducing redundancy in a controlled manner by relaxing the normalization rules will improve the performance of the system. This is sometimes known as denormalization

Informally speaking, denormalization is merging of relations

Result of normalization is a logical database design that is structurally consistent and has minimal redundancy.

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

It may be necessary to accept the loss of some of the benefits of a fully normalized design in favor of performance.

Also consider that denormalization:

- Makes implementation more complex;
- Often sacrifices flexibility;
- May speed up retrievals but it slows down updates.

Denormalization refers to a refinement to relational schema such that the degree of normalization for a modified relation is less than the degree of at least one of the original relations.

Also use term more loosely to refer to situations where two relations are combined into one new relation, which is still normalized but contains more nulls than original relations. No fixed rule when to denormalize but ,

Consider denormalization in following situations, *specifically* to speed up frequent or critical transactions:

- Step 1 Combining 1:1 relationships
 - Step 2 Duplicating non-key attributes in 1:* relationships to reduce joins
 - Step 3 Duplicating foreign key attributes in 1:* relationships to reduce joins
 - Step 4 Introducing repeating groups
 - Step 5 Merging lookup tables with base relations
 - Step 6 Creating extract tables.
-

6. Monitoring and Tuning the operational system

The objective here is to monitor operational system and improve performance of system to correct inappropriate design decisions or reflect changing requirements.

Importance of monitoring and tuning the operational system

- Avoids procurement of additional hardware
- Down size the hardware configuration → less and cheaper hardware → less expensive maintenance.
- Faster response time and high throughput → more productive
- Faster response time → good staff moral, customer satisfaction