Physical Database Design and Tuning

1

Physical Database Design

- It is the process of transforming a logical data model into a physical model of a database.
- Unlike a logical design, a physical database design is optimized for data-access paths, performance requirements and other constraints of the target environment, i.e. hardware and software.

2

Physical Database Design

- Before you can begin the physical design, you must have:
  - (1) logical database design
    - minimally third normal form
  - (2) Transaction characterization, such as
    - most frequent transactions
    - most complex or resource-insensitive transactions
    - distributions of transactions, over time
    - mix of insert, update, delete and select statements
    - most critical transactions to the applications

3

Physical Database Design

- (3) Performance requirements

4

Physical dB design activities

- 1. Defining Tables and Columns –
  - The initial transformation of the logical model into a physical model, including naming objects, choosing data types and lengths, and handling null values.
- 2. Defining Keys –
  - Choosing primary and foreign keys, including the use of surrogate keys.

5

Physical dB design activities

- 3. Identifying Critical Transactions –
  - Identifying business transactions that are high-value, mission-critical, frequently performed, or costly in terms of computing resources.
- 4. Adding Redundant Columns –
  - The first of a series of denormalization techniques: adding columns to tables that exist in other tables.

6

Physical dB design activities

- 5. Adding Derived Columns –
  - Adding a column to a table based on the values or existence of values in other columns in any table.
- 6. Collapsing Tables –
  - Combining two or more tables into one table.
- 7. Splitting Tables –
  - Partitioning a table into two or more disjoint tables. Partitioning may be horizontal (row-wise) or vertical (column-wise)

7

Physical dB design activities

- 8. Handling Supertypes and Subtypes
  - Deciding how to implement tables that are involved in a supertype-subtype relationship in the logical model.
- 9. Duplicating Parts of Tables –
  - Duplicating data vertically and / or horizontally into new tables.
- 10. Adding Tables for Derived Data –
  - Creating new tables that hold data derived in columns from other tables

8

Physical dB design activities

- 11. Handling Vector Data –
  - Deciding how to implement tables that contain plural attributes or vector data. Row-wise and column-wise implementations are discussed.
- 12. Generating Sequence Numbers –
  - Choosing a strategy to generate sequence numbers, and the appropriate tables and columns to support the strategy.

9

Physical dB design activities

- 13. Specifying Indexes –
  - Specifying indexes to improve data access performance or to enforce uniqueness.
- 14. Maintaining Row Uniqueness –
  - Maintaining the uniqueness or primarykey values.
- 15. Handling Domain Restriction –
  - Defining SQL Server rules and defaults on the columns of a table to maintain valid data values in columns.

10

Physical dB design activities

- 16. Handling Referential Integrity –
  - Deciding how to handle primary-key updates and deletes, and foreign-key inserts and updates. Using triggers to ensure referential integrity.
- 17. Maintaining Derived and Redundant Data –
  - Specifying how data integrity will be maintained if the data model contains derived or redundant data.

11

Physical dB design activities

- 18. Handling Complex Integrity Constraints –
  - Deciding how to handle complex business rules such as sequence rules, cross-domain business rules, and complex data domain rules. Using triggers to implement complex business rules.
- 19. Controlling Access to Data –
  - Restricting access to commands and data.

12

Physical dB design activities

- 20. Managing Object Sizes –
  - Calculating the estimated size of a database and its objects.
- 21. Recommending Object Placement
  - Allocating databases and their objects on available hardware to achieve optimal performance.

13

Physical dB design goals

- improve system performance
  - reduce disk I/O
  - reduce joins
- embed business rules into the database design
  - through defaults, rules, constraints, stored procedures, or triggers
- make it understandable to users
  - use meaningful and indicative names for tables and columns

14

Defining Keys

- If there are more than one candidate key in a table, select the primary key as below:
  - select the key which transactions will know about most often. This will avoid additional lookups.
  - select the shortest length key when used in indexes
  - consider what other keys are available in other tables on which to join.

15

Identify Critical Transactions

- To understand the transactions and performance requirements, you need to know:
  - types of transactions (select, insert, update, delete)
  - tables and column affected by each transaction
  - select criteria – fixed or variable (i.e. pre-defined queries or ad-hoc queries)
  - frequency and volume of each transaction

16

Identify Critical Transactions

- how many rows (percentage) are typically affected (select or modified)
- size (no. of rows and total bytes) of tables involved
- when the transaction is executed
  - during the day or after office hours
- relative importance of each transaction
  - who use it, how often, how critical is it to the business process
- response time or throughput desired

17

Identify Critical Transactions

- security and integrity
- how many tables will be joined
- sort order

18

Adding Redundant Columns

- required when an unaccepted number of joins is needed to perform a critical transaction.
- add redundant columns in order to reduce the no. of joins. –
  - It is a de-normalization process. Tables will not be in 3NF.
- The concept of strong FD, weak FD, relax-replicated 3NF relation can be used as the theory for this process.

19

Adding Redundant Columns

- Benefits:
  - better response time
  - The chance to eliminate a foreign key –
  - The reduction of lock contention;
    - this cut down blocking or deadlock situations.

20

Adding Derived Columns

- Derived data may include:
  - column data aggregated with SQL aggregate function such as sum(), avg(), over N detail rows
  - column data which is calculated using formulas over N rows.
  - counts of details rows matching specific criteria
- **Example**: Total-sales in Titles table Titles (title-id, title, type, pub-id, price, **Total_sales**, pubdate, pubname)

21

Collapsing Tables

- Required when the application program must frequently access data in multiple tables in a single query.
  - **e.g.** Combining the publishers and Titles tables will improve the performance of the critical query
- de-normalization
- similar to adding redundant columns
- in order to get better performance

22

Splitting tables

- Required when it is more advantageous to access a subset of data.
- **Vertical table splits**:
  - e.g. Emp (Eno, name, salary, tax, mgr#, dept#) can be split to 2 tables: Emp_bio (Eno, name, mgr#, dept#) Emp_comp (Eno, salary, tax)
  - The rows are smaller. This allows more rows to be stored on each data page, therefore no. of I/Os is reduced.

23

Splitting tables

- Horizontal table splits
  - **e.g.** You can form horizontal fragments of the Supplier table.
- Benefits of splitting:
  - reduces the no. of index pages read in a query
  - The table split corresponds to an actual physical separation of the data rows, as in different geographical sites.
  - Table splitting achieves specific distribution of data on the available physical media

24

Adding Tables for Derived Data

- Many applications or reports call for data summaries, often at more than one level of grouping for the same source data.
- generating summaries with large tables, may become a performance bottleneck.
- **Example** Summary table
  - Titles (title-id, title, type, pub-id, price, pubdate) Summary-table (type, total-sales)

25

Specifying Indexes

- Indexes can be used to improve data access performance
- Indexes may be clustered or non clustered, unique or non unique, or concatenated.
- A table's indexes must be maintained with every insert, update, and delete operation performed on the table.
- Be careful not to over index.

26

Specifying Indexes

- Incorrect index selection can adversely affect the performance.
- The greatest problem will be deriving the best set of indexes for the database when conflicting applications exist (i.e. applications whose access needs and priorities are in conflict).

27

Specifying Indexes

- Tables that should be considered as candidates for indexes are:
  - tables that are used in critical transactions and that have a set of search criteria (or limit ranges)
  - tables involved in multi-table joins
  - tables with a large no. of rows
  - tables that require enforcement of uniqueness

28

Specifying Indexes

- Identifying Columns for Indexes
  - columns used to specify range in the *where* clause (clustered index)
  - columns used to join one or more tables, usually primary and foreign keys
  - columns likely to be used as search arguments
  - columns used to match an equi-join query
  - columns used in aggregate functions
  - columns used in a *group by* clause
  - columns used in an *order by* clause

29

Database tuning

- Database tuning is comprised of a group of activities used to optimize and regulate the performance of a database.
- It refers to configuration of the database files, the database management system (DBMS), as well as the hardware and operating system on which the database is hosted.

30

Database tuning

- The goal of database tuning is to maximize the application of system resources in an attempt to execute transactions as efficiently and quickly as possible.
- The large majority of DBMS are designed with efficiency in mind; however, it is possible to enhance a database's performance via custom settings and configurations.

31

Database tuning

- The tuning of a database management system centers around the configuration of memory and the processing resources of the computer running the DBMS.
  - This can involve setting the recovery interval of the DMBS, establishing the level of concurrency control, and assigning which network protocols are used to communicate throughout the database.

32

Database tuning

- Database performance can also be improved by using the cache to store execution procedures as they would not need to be recompiled with every transaction.
- By assigning processing resources to specific functions and activities, it is also possible to improve the concurrency of the system.

33

Database tuning

- "Database concurrency controls ensure that transactions occur in an ordered fashion.
  - The main job of these controls is to protect transactions issued by different users/applications from the effects of each other.
  - They must preserve the four characteristics of database transactions: atomicity, isolation, consistency and durability"

34

Database tuning

- Input/Output(I/O) tuning is another major component of database tuning.
  - I/O tuning mainly deals with database transaction logs.
  - Database transaction logs are files that are associated with temporary work spaces as well as both table and index file storage.
  - Transaction logs and temporary spaces are heavy consumers of I/O, and affect performance for all users of the database.

35

Database tuning

- Placing them appropriately is crucial.
- The main goal of I/O tuning a database is to optimize and balance the read and write transactions of the system in order to achieve an increased speed in database transactions and a decreased database access time.

36

Database tuning

- Another method of ensuring that a database is fast and reliable is the Use of RAID in the creation of the database.
- RAID stands for Redundant Array of Independent Disks.
- Here is an example as to why RAID is superior to a single disk.

37

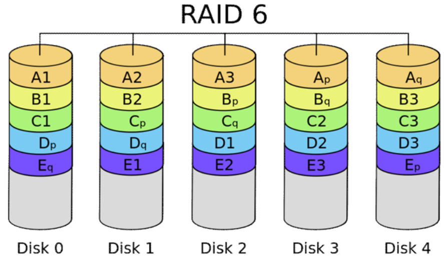
Database tuning

- If data are stored on one disk, the entire database is completely reliant on that one disk; if it were to fail, the database would not exist anymore.
- Another drawback to having it on a single disk is the read/write time.
- One hard disk can only be so fast. If there is a lot of I/O data being processed, it can be a lengthy process. One thing that RAID does is it divides and replicates the data onto several independent disks.

38

Database tuning

RAID 6



This shows the data layout for a RAID-6 array.

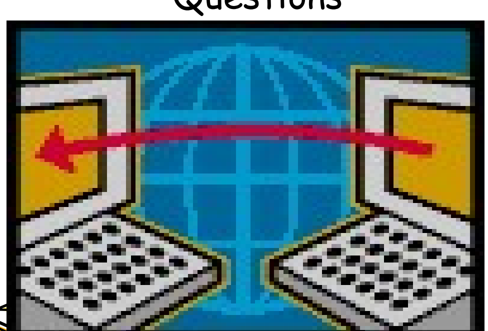
39

Database tuning

- Another common part of database tuning revolves around maintenance.
  - Database maintenance includes things such as backing up the database as well as the defragmentation of the data residing within the database.
  - When a database is under heavy use, transaction log entries must be removed in order to create space for future entries.

40

Questions



41