Lecture 11

Next Generation Databases - Guy Harrison

Slides

Distributed Transactions

One big problem is with transactions that **involve several servers**. Remember that transactions are meant to be atomic operations (a principle challenged by NoSQL databases).

TWO-PHASE COMMIT

- JUST before committing, you ask to all servers involved whether they are ready to commit. If you don't get an all clear, you can rollback.
- Otherwise, you can then (2nd phase) send the official "commit" signal.

IN-DOUBT transactions

You can still have cases when you don't know if one of the servers did or didn't commit in the end, and you end up with "in-doubt transactions" that you have to resolve (which means cancel) manually. It happens far more often than you may think, simply because the number of transactions in a big company is enormous, and a tiny percentage still means a few cases every week.

Latency

1 KM = 0.000005 s

Synchronous or Asynchronous

Programming with Databases

a PHP example

Object Relational Mapper

django

As soon as things become just a little bit complicated, it all goes downhill. You end up with a complex SPECIFIC syntax even for simple queries.

Hibernate

Escape route: HQL

In reality, among all the people who dream of database independence most of them won't migrate their applications to another DBMS. If they migrate, they'll also rewrite everything.

If you want to minimise DBMS impact, it's far better to isolate YOUR SQL code than rely on something that is generic and often inefficient.

- Isolate SQL code
- Use DBMS features

DBAs

Production DBA

Production DBAs have a LOT of databases to care about. There are usually many independent databases in a company, and each production database is shadowed by multiple databases.

- Performance
- Physical Storage
- User Management
- Backup / Recovery

Development DBA

Lab 12

Slides

Dynamic Monitoring

Physical Storage

- Automatic Storage Management

Redundant Array Independent Disks (RAID)

- Storage Area Network (SAN)
  - Seen as a disk
- Network-Attached Storage
  - Seen as a file-server
- Exadata ("Database Appliance")

Partitioning

Suppose that a table holds data for say over one year, and that we are interested in data for one month (say 10% of the table to simplify). If the table contains 500,000,000 rows, 10% are still 50,000,000 rows and fetching them one by one with an index will take ages. Scanning the full table may be more efficient, but perhaps that instead of discarding 90% of what we read, we could regroup data by month and only read what we want.

That's the idea behind partitioning. You see one table but physically data is regrouped in semi- independent tables called partitions. The DBMS know the organization and direct you to the right table and ignore the others.

When you create the table, you also create partitions (they can also be added and removed on the fly) and specify a set of columns (partition key) that determines in which partition a row should be inserted. When the partition key is a criterion in a query, only the right partitions are considered.

There are different ways to partition.

- By range (dates usually)
  - By list
- By hash

When your concern is to control data grouping, you usually partition by range (so as to have rows grouped by month, week, or whatever interval), or by list (If the row contains this value in the partition key, then it goes into this partition). When your concern is spreading inserts over a table, for instance, you may opt for hash partitioning, and the system will compute partition placement for you.

Partitioning may want to address very different problems. If you group your rows by period (good for querying) then you may have problems with insertions if many processes are inserting in parallel, because everybody will want to insert at the same place and you'll need some serialization.

For parallel insertions, you'd rather see each process inserting into separate partitions. But then if you want to retrieve all these rows at once in a query, you must query n partitions instead of just one, which isn't ideal.

Subpartitioning

You can sometimes have your cake and eat it by using subpartitioning, for

instance subpartitioning by hash partitions defined by range.

Exchange partition

A product such as Oracle allows a lot of operations by just operating on metadata and not actually moving rows.

- Turn partition into table and vice versa
- NO data move - just data dictionary updates.

One problem with grouping rows is that you often have several options for a partition key.

Most important is how tables are laid out relative to DATA

The right choice depends on the expected workload - will insertions be a problem? Do you want to improve some massive selects? Will there be many updates? As usual, it's rare that there is a single obvious way to organize your data.

Issues with normalization

partitioning films by continent?

Partitioning may also conflict with normalization. For instance, partitioning table MOVIES by continent might in some cases make sense. Except that CONTINENT belongs to COUNTRIES, not MOVIES.

Storing the continent as an attribute of the film goes against normalization but might help.

When partition becomes useless

Becomes pretty useless when one partition holds 90% of data

No need to bother with partitioning when the data that you want to partition on is heavily skewed. Indexes can take care of rare cases, and scanning a big partition or the table doesn't make much of a difference.

`update` = move row

Another point to keep in mind is that as data determines physical placement, updating the partition key isn't a plain update, but means physically moving the row from one partition to the other. If you do it very often, it can hurt.

Ordering destroys symmetry

The relational theory knows no order. It's worth repeating that the relational theory knows no order.

If you begin to physically order your rows, whether by really ordering them (cluster index) or more simply grouping them (partitioning) you are destroying symmetry by favoring one type of database operations (some queries, inserts) at the expense of other types of operations. You must make sure that what will suffer isn't, and won't become, important.

…