Lecture 12

Partitioning

- the idea of 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.

You see one table but physically data is regrouped in semi- independent tables called partitions.

We also have **partition key** to index the partitions.

- Three Types
  1. By range (dates usually)
  2. By list (discrete value)
  3. By hash (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)
- Problems

  when you have **concurrency** (several processes) it can be an issue, because if everybody is adding rows to the **same block** other processes have to be kept on hold while one process is writing bytes.

  Use Subpartitioning

  Subpartitioning, supported by some database systems, is a possible solution. You can say that every row that contains a date in June 2018 should go to one table, but compute a hash value on some other column to determine a subpart.

  The relational theory knows no order Ordering destroys symmetry

  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.

What's behind the partitioning?

Exchange Partition

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

Deal with normalization

Oracle also allows partitioning based on foreign keys.

Deal with Upate

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.

Archiving

Old data can be archived. Use less expensive material to store them

- This is an area where partitioning by date can help a lot. You archive and drop the oldest partition, and creae a new one (beware of foreign keys!)
- Alternatively, you can use a fixed number of partitions (for instance one per month or week) in a circular way.

Great explanation about oracle archive mode on the YouTube

User Management

1. Mysql

   ```
   create user 'username'@'hostname' identified by 'a_password'
   create user 'username'@'%' identified by 'a_password'
   create user 'username'@'localhost'
   ```

2. Oracle

   In Oracle you can limit access from machines but elsewhere.

   ```
   create user username identified by a_password
   create user ops$username identified externally
   ```

   Note that with Oracle you won't be able to do anything with your account if you aren't ALSO given the right to create a session.

   ```
   grant create session to username
   ```

3. Postgresql

   In PostgreSQL a "user" is a role that has the right to login. Both statements are synonym. Server and network control is specified in a configuration file.

   default account: **postgres**

   ```
   create user username with password 'a_password'
   create role username with login password 'a_password'
   ```

4. sql server

SQL Server leaves you the choice between pure Windows authentication, or classic username/ password authentication (you may want to access SQL Server through JDBC from a Linux machine)

```
create login 'domainname\loginname' from windows
create login username with password 'a_password'
```

default_account: **sa**

Use Role to avoid repetition

```
create role rank_and_file
grant create session to rank_and_file
grant rank_and_file to john_doe
```

Alias/Synonym

- make referencing easier
- Eliminate hard-coding

public synonym : can be accessed by anybody

Back Up

1. **Logic Backup**

   > Using software to export the data to a file, it is different from the original data files, so it cannot be used to physical backup

- Issues
  - Consistency
    - Even if you see a consistent view of one table, basically anything that is committed AFTER you started your export of data should be ignored, otherwise you may save rows and not the rows that they reference.
  - Slow Recovery
    - row addresses change.
    - Indexes must be completely rebuilt, and on a big table it can be fairly slow.
  - Some data loss
    - Another problem of course is that as what you get is a copy at a given time of your data, if anything happened to it between the backup and the time when you restore, you restore an out-of-date database.

2. **Physical Backup**

   > it's all about copying files, it is data-blind. Everything happens in memory

3. **Hot Backup**

   journal files can track all data changes

The idea is, WHILE THE DATABASE IS ACTIVE, to copy the **datafiles and all journal files** generated since you started the backup. *You cannot do it if journal files get overwritten on a regular basis.*

(Achieve mode on)

First you have a cold backup database, then use the hot log files, you can restore more information

> it works a little bit like restarting after a crash

Back up log files is known as **incremental backup**

Advantages

- faster restart
- can be **incremental** and **consistent**
- No or little data loss (hot backup)

Shortcomings

- Cannot recover one table
- More complicated(hot backup)
- Big files