Lecture 12

(This lecture is quite challenging, at least for me. You may need to go through the original slides more carefully when reviewing.)

- `INSERT`s are inserts
- many `UPDATE`s aren't updates they are inserts
- many `DELETE`s aren't deletes they are updates

Performance

Strategies

- Parallelism
- Adding nodes
- Keeping scope constant
- Archiving

Archiving

Partitions

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!)

`truncate`

Alternatively, you can use a fixed number of partitions in a circular way.

User Management

In all products (other than SQLite, of course) you need to connect to a database a DB account which is different from (but can sometimes be tied to) an OS account.

Exception: IBM DB2

Authentication

- By the OS (trusted connection)
- By password
- External (LDAP, Kerberos, etc.) [uncommon]

Super-user account ("DBA account")

In all DBMS products, a default super-user account (often called "DBA account") is automatically created with the database. (Called `root` in MySQL.)

MySQL

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

From the root account you can create other user accounts, which can be restricted to access from specific machines (for instance the same machine, or a HTTP server).

Oracle

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

In Oracle you can limit access from machines but elsewhere. If you want to tie a database account to OS account `xxx`, you create an account called `<prefix>xxx`. The default prefix is `OPS$` and can be set to an empty string.

`grant create session to username`

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.

PostgreSQL

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

In PostgreSQL a "user" is a role that has the right to login. Both statements are synonym.

One default account, named `postgres`.

SQL Server

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

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).

The default user is called `sa` (Super Administrator) but is deactivated. You can create accounts by connecting without any fuss from the administrator account from which you have installed the DBMS.

Privileges

- System privileges

Privileges to change the structure

- Table privileges

Privileges to access the data

Three categories of users.

- Adminstrators

All system privileges

- Schema Owners

System privileges to create objects

- Regular Users

Minimum system privileges + Object privileges

Don't grant privileges "just in case"

JUST THE REQUIRED PRIVILEGES.

`grants on present objects`

`grants on future objects`

Developer Account

- `grant create table to ...`
  - `grant create index to ...` (MySQL)
  - `grant create sequence to ...` (Oracle)
  - `grant create trigger to ...` (Oracle)
- `grant create procedure to ...`
  - `grant create function to ...` (SQL Server)
- `grant create view to ...`
- `grant create type to ...`

Usually limited to one database

IMPORTANT

default schema/database/tablespace

You should also define (especially when inexperienced developers) where their tables will be created by default. You want to keep application tables separate from system tables.

Don't let developers "fix" things in production. Scripts that change structures in production should be tested and run by database administrators.

DEVELOPERS SHOULD NOT CONNECT TO PRODUCTION DATABASES.

Role

As you don't want to repeat a long series of `GRANT` statements everytime a new hire must be given access to the database, you create a role, give all the privileges to the role (system and tables) and grant the role to users.

`create role rank_and_file`

`grant create session to rank_and_file`

`grant rank_and_file to john_doe`

Alias / Synonym

Normally you are supposed to prefix table names with schema names. Don't do that.

Use aliases (synonyms): they can be dropped and changed at will to point to different tables while running exactly the same queries.

`create synonym employees for hr.employees`

`create synonym employees for training.employees`

`PUBLIC`

There is something special called `public` that means "everybody" (existing as well as future user accounts). Much used.

Web Applications

DON'T CONNECT AS THE SCHEMA OWNER.

As a general rule, don't give full DBA privileges to many people. There should only be a handful of DBA accounts at most. Beware of third-party software that requires unneeded privileges.

Big Management Issues

- Ethics are relative
- Not trusting people often makes people less trustable
- Cultural sensitivity
- Don't give unnecessary privileges
- Entrust the happy few with high privileges

Knowing WHO has high privileges, and holding them personally responsible is often better than paranoia.

Encryption

- One-way encryption (md5, sha1)
- Reversible encryption
  - sensitive data (credit cards, and so forth)

There may be issues with indexes (no range scan on an encrypted column).

SECURITY ISSUES

- Secure Server
- Secure Files and Directories

Even log files may contain a lot of information

- Check and remove unused defaults
- PASSWORDS
- Same security standards for Production and Development.
- Try to keep track of connections

BACKING UP A DATABASE

Logical backup

- Beware of consistency
- SLOW recovery
- Some data loss
- Smaller file (~30%) [No indexes, no bulky system areas]
- Partial restore [You can reload a single table]

Beware of consistency

If you dump database at a time when there is no update activity, either because there is a way to make the database temporarily read-only or because you prevented external connections, you won't have any problem. You may have problems if people are modifying tables while you are dumping them.

SLOW recovery

The big issue is that as data is reloaded, row addresses change. **Indexes must be completely rebuilt**, and on a big table it can be fairly slow.

When restoring from a logical backup, reloading data may not be too bad, but rebuilding indexes can be really slow, and you can't open the database before everything is recreated.

Therefore, logical backup is rarely popular with end-users (and even less with top management).

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.

You must carefully schedule regular backups, because what happened to changes between backup and crash?

Physical backup

= COPY FILES

but Everything happens in memory!

IMPORTANT

Get file names from the data dictionary.

As an aside, it's a good practice, just before a backup, to get the data location from the database itself (before shutting it down if you shut it down). Some data files may be outside usual directories and it's always unpleasant to be unable to open a restored database because some files are missing.

Cold Backup

Shutting down the database before copying the files

- Consitstency: Good
- Data loss: Whatever happened between backup and restore is lost.
- Availability: Good

Cloning

Additionally, there may be (operating system) tricks for shutting the database down a short time. Some advanced file systems support "cloning", which is very much like undo in DBMS products (saving disk blocks before they are modified) and allows the same as a "repeatable read". If all the data files are in a clonable file system, shut the database down, clone the filesystem (immediate), restart the database and copy the clone quietly while the database is active again. Your copy will ignore new changes.

With mirrored disks, you can shutdown the database, deactivate mirroring (immediate), restore the database with mirrorring deactivated on one disk, and quietly copy the disk that is no longer written. When you are done, you reactivate mirroring (no need to stop the database), there is a "resilvering" operation (the disk that you copied catches up and updates itself), et voilà.

Hot Backup

(Not supported by all products)

When a database is active, journal files track (synchronously) 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.

It works a bit like restarting after a crash, excpet that you reapply far more data.

INCREMENTAL BACKUP

If you archive log files AFTER the backup on a regular basis, you can even be able to apply them after restore and bring your database back to its state just before the problem. No data loss.

As you don't want to re apply 6-months worth of transactions before restarting your database, you must schedule regular full backups.

- Consitstency: Good
- Loss: No
- Availability: Good

In summary, Physical Backup

- Faster restart
- Can be incremental AND consistent
- No or little data loss
- Cannot recover ONE table
- More complicated (For hot backup only)
- Big files (Need twice the storage)

You may lose some data if you lose the last (unarchived) journal. However, journal files are usually mirrored, so the odds of losing them are very low.

Point-in-time Recovery

Hot and incremental backups allow fancy operations, such as a point-in-time recovery.

Remember that if you drop one table, you cannot restore it from a physical backup. It's all or nothing.

Data in backup = data in production database

On the chapter of security, perhaps it is worth reminding that backups contain the same data as the database and that making backup files publicly readable is a bad idea. Some recent big data breaches were simply stealing backup files. You can also encrypt them.

`RECYCLE BIN` - Oracle

You can "undrop" a table if this functionality is activated. Of course, as dropped tables are still physically there, you need to purge the "recycle bin" from time to time.

PRACTICE RECOVERY

Automate as much as you can.

- How long can you wait?
- What data loss can you bear?

Eliminating Single Point of Failure

High Avalaibility

If anything fails you want to be able to switch, as transparently as possible, to a backup system that may possibly be slightly less performant but can ensure the main functionalities.

A typical HA solution is a cluster of computers that may or not access the same database files.

Log Shipping

The idea is to periodically ship log files to a distant server that hosts a copy of the database and reapply them there.

Beware though that if all "ordinary" DML changes travel well from server to server, DDL operations are not always logged and may require special care.

With log shipping, you can reduce dramatically what can be lost to the content of logs that haven't yet been transferred.

Metrocluster

SAN systems manufacturers may offer metroclusters which may be disk-box-to-disk-box, physical replication that does only care about bytes, and not applications. It can be synchronous or asynchronous. In that case replication is transparent for the DBMS software.

Hardware or Software replication?

Cloud (for smaller companies)

There may be security issues if you don't encrypt your files.

Lab 13

Slides

…