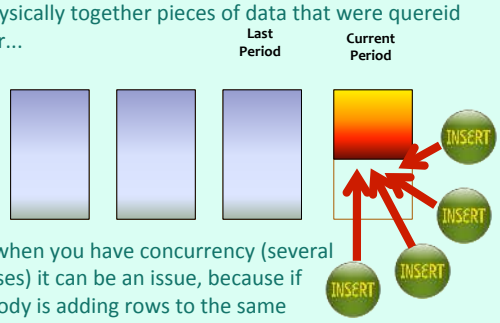
CS307

Database Principles

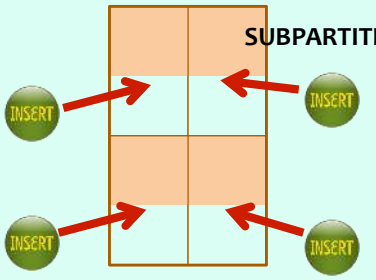
Stéphane Faroult
faroult@sustc.edu.cn

朱悦铭 Zhu Yueming zhuym@sustc.edu.cn

We have seen last time that partitioning was a great way to keep physically together pieces of data that were quereid together...



... but 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.



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.

Great for DBAs

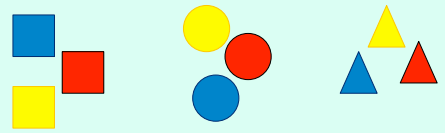
A product such as Oracle allows a lot of operations by just operating on metadata and not actually moving rows.

Table P1
Partition P2
Partition P3
Partition P4
Partition P5
Partition P6
Partition P7
Partition P8

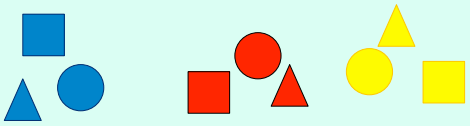
"Exchange partition"

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. Oracle also allows partitioning based on foreign keys.

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.

Partitioning ***almost*** mandatory with very big tables

HOW not always obvious

You can live without it below a few tens of million rows.

Picture by Andrew Fogg

If it were easy, where would be the fun?

Not default = issues?

Never forget that if an option isn't the default one, it mean that there may be issues in some cases. Software vendors aren't crazy, the default option is the one that works well in most cases. Especially if you haven't very clear ideas about the practical implications of a storage choice, tread carefully.

Keep simple

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.

Picture by Boston Public Library

INSERTs are inserts

many UPDATEs aren't updates they are inserts

many DELETEs aren't deletes they are updates

Remember what we have seen.

VOLUME

PERFORMANCE

Today everybody is happy with performance. Will it still be the case in five years' time?

Table scans ?

In database operations, a lot of tables are scanned. Sometimes because they are badly indexed or queries badly written (bad reason), sometimes because it's more efficient than index searches (good reason). Twice as big means twice the time.

Strategies

- **Parallelism**
- **Adding nodes**
- **Keeping scope constant**
- **Archiving**

If we want to keep the lid on performance, we may contemplate several strategies. Adding more CPUs, adding more computers to a shared database, only querying over a smiliar scope ... and archiving old data.

ARCHIVING

Too often, archiving old data (which is pure data management) comes as an afterthought, usually after the first big performance issues.

~~massive INSERT~~

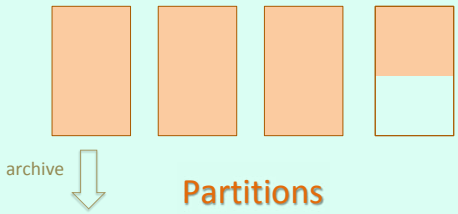
~~massive DELETE~~

can be slow

can generate LOTS of log

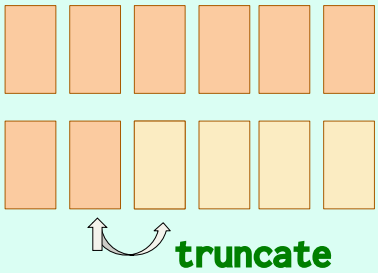
A proper, efficient archiving process can be tough to write.

ARCHIVING



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!)

ARCHIVING



Alternatively, you can use a fixed number of partitions (for instance one per month or week) in a circular way.

OS account | DB account

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. The exception is DB2 that directly uses OS accounts.

Exception: IBM DB2

22

You basically have three ways to authenticate. Authentication by the OS is often used in scripts that run on the server (no hardcoded password)

Authentication

By the OS (trusted connection)

By password

External (LDAP, Kerberos, etc.)

Uncommon

23

Default account

In all DBMS products, a default super-user account (often called "DBA account") is automatically created with the database. This account is the owner of the catalog (data dictionary) tables.
It's 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).

MySQL

In Oracle you can limit access from machines but elsewhere.

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

Can be changed by DBAs

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.

ORACLE

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.

NOT ENOUGH ...

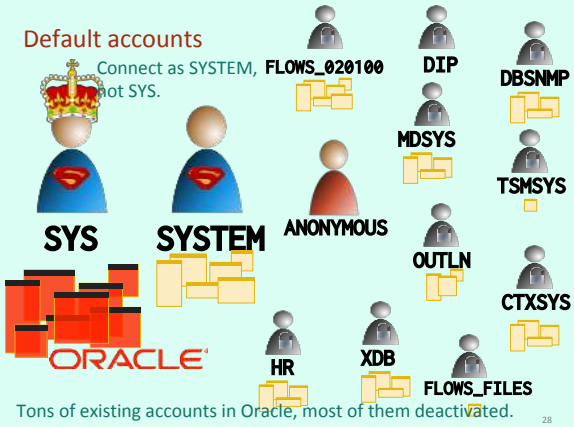
grant create session to username

System privilege

ORACLE That allows to some extent to distinguish pure schema (cannot create a session) from regular user

Default accounts

Connect as SYSTEM, not SYS.

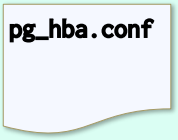


ORACLE

Tons of existing accounts in Oracle, most of them deactivated.

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

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



PostgreSQL

Server and network control is specified in a configuration file.

Default account

postgres

PostgreSQL

One default account, named "postgres".

```
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)

Microsoft SQL Server

Default login

sa

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.

Microsoft SQL Server

System privileges
Privileges
to change the structure

Remember that there are two broad categories of user privileges.

Table privileges
Privileges
to access the data

33

You usually have three categories of users.

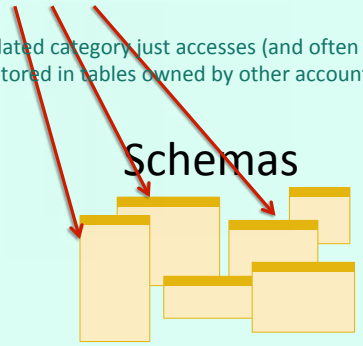
Administrators	All system privileges
Schema Owners	System privileges to create objects
Regular Users	Minimum system privileges Object privileges

Basic users

Data Administrators

Regular user accounts

The most populated category just accesses (and often changes) data stored in tables owned by other accounts.



35

User privilege is an area where you should be paranoid.

JUST THE REQUIRED PRIVILEGES !

Don't grant privileges "just in case"

36

For grants on objects, you can grant (by naming them) access to current objects, but there are also "blanket" privileges covering objects that WILL be created.

grants on present objects

grants on future objects

ORACLE ~~grant select any table to ...~~

MySQL `grant select on dbname.* to ...`

Only give it to performance consultants with Oracle ... Acceptable when limited to one schema.

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 ... SQLServer

grant create view to ...

grant create type to ...

Usually limited to one database

IMPORTANT

default schema/database/tablespace

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

User Account

grant select on *tablename* to ...
grant insert, update on *tablename* to ...
grant execute on *procname* to ...

Data Administrator

+ insert update delete — Reference tables (some of them)

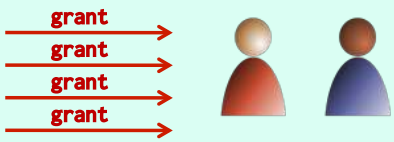
Don't let developers "fix" things in production. Scripts that change structures in production should be tested and run by database administrators.

DEVELOPERS
SHOULD NOT
CONNECT TO
PRODUCTION
DATABASES

in an ideal world

41

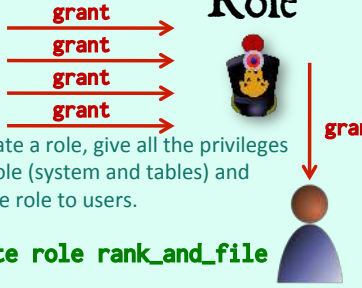
As you don't want to repeat a long series of "GRANT" statements everytime a new hire must be given access to the database ...




```
grant create session to john_doe
```

42

Role



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
```

43

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

Addressing Objects in another schema

Theory

```
select ...
from schemaname.tablename
where ...
```

You may want to use the same program against different sets of tables in different schemas.

Addressing Objects in another schema

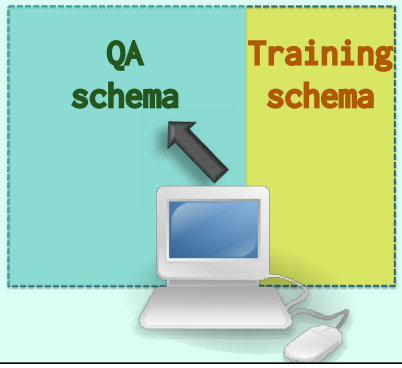
Practice

Reorganization?

Development, test, training?

For instance the same test database can harbor a schema for training end-users on a new application.

Test Database



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

Alias / Synonym

```
create synonym employees for hr.employees

create synonym employees for training.employees
```

ORACLE

Microsoft SQL Server

IBM DB2

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

PUBLIC

ORACLE `create public synonym ...`

role `grant ... to public`

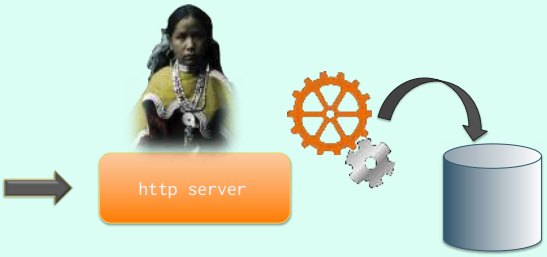
PostgreSQL

schema + role

IBM DB2

role

Web Applications



User management is of course a secondary concern in web applications, because connections to the database are issued by the HTTP server.

USERS

Users are managed by the application, not by the DBMS.

USERID USERNAME USERPASS EMAIL …

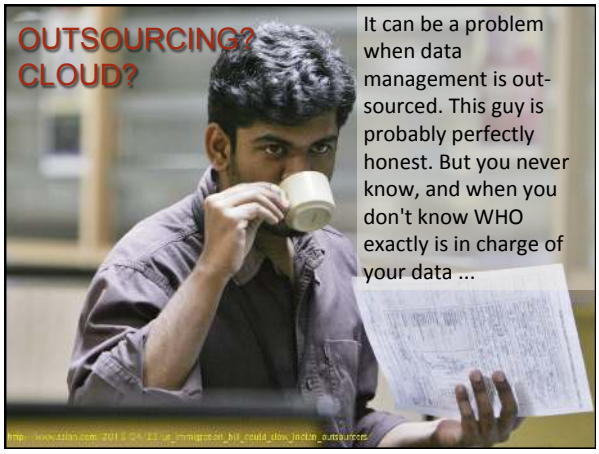
DON'T CONNECT AS THE SCHEMA OWNER

Nevertheless, try to have at least one account that owns the tables, and a DIFFERENT account that queries and modifies them, and which is the account used by the HTTP server.

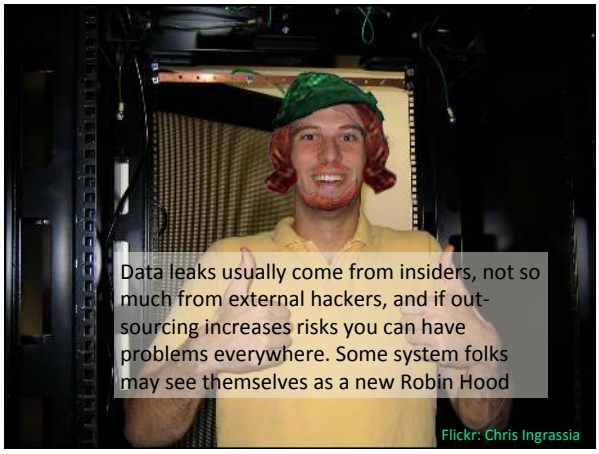
As a general rule, don't give full DBA privileges to many people. There should only be a handful of DBA accounts at most. Beware of third-party software that requires unneeded privileges.

DBAs can see it.

One problem with DBAs is that they can access EVERYTHING in the database

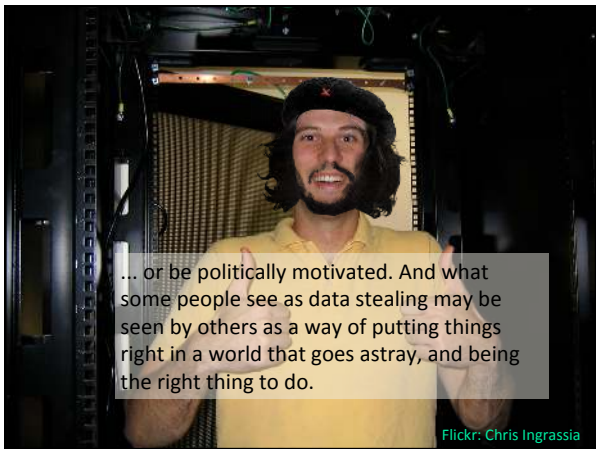


It can be a problem when data management is out-sourced. This guy is probably perfectly honest. But you never know, and when you don't know WHO exactly is in charge of your data ...



Data leaks usually come from insiders, not so much from external hackers, and if out-sourcing increases risks you can have problems everywhere. Some system folks may see themselves as a new Robin Hood

Flickr: Chris Ingrassia



... or be politically motivated. And what some people see as data stealing may be seen by others as a way of putting things right in a world that goes astray, and being the right thing to do.

Flickr: Chris Ingrassia

The truth is that it's often a management issue that wide-ranging encryption doesn't solve.

Big Management Issues

- Ethics are relative
- Not trusting people often makes people less trustable
- Cultural sensitivity

Big Management Issues

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 is OK when you don't need to decode, and just compare encrypted values.

One-way encryption (md5, sha1)

passwords

When you need to decrypt somebody must have a key somewhere …

Reversible encryption

sensitive data
(credit cards, and so forth)

Additionally, there may be issues with indexes (no range scan on an encrypted column)

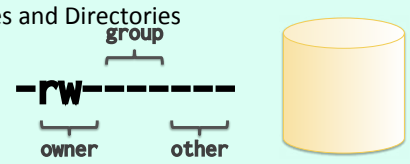
SECURITY ISSUES

The topic of security issues is of course a hot one, especially with databases. It doesn't get any better with distributed databases. There are many excellent specialized database security websites on the web. Very often, security is a matter of common sense - once you have a clear idea of how everything works, which some people lack.

Secure Server

Security of data starts with systems. Firewalls, and very importantly making the DBMS software account owner the only one able to access database files.

Secure Files and Directories



Even log files may contain a lot of information

Secure Server

Secure Files and Directories

Check and remove unused defaults

Another important step is to get rid of anything (sample database account, etc.) that exists, for the database AND EVERY THIRD PARTY APPLICATION USING IT. SAP has a lot of well-known default accounts, just like Oracle ...

PASSWORDS

Lists of default passwords are available on the web

Lists of common passwords are available on the web

~~test/test~~

You can search ... The problem is that when you are connected to a database even with a low-privilege account you can see all the other account names, and try to break into them.

One thing that is strange is that often people seem to think that when a database is classified as "development" they can relax security and have an all-powerful TEST user (password TEST). Hey folks, how did you build your dev database? Copy of prod? Same data, then.

SAME SECURITY STANDARDS FOR PRODUCTION AND DEVELOPMENT

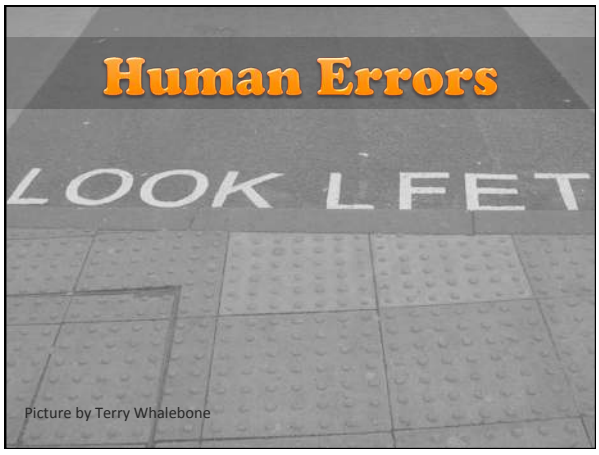
63

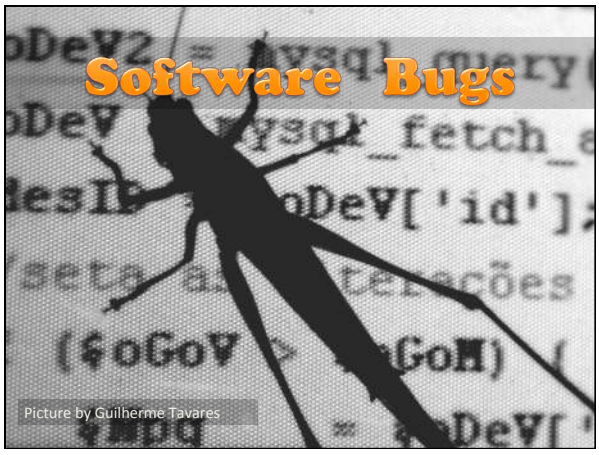
Try to keep track of connections

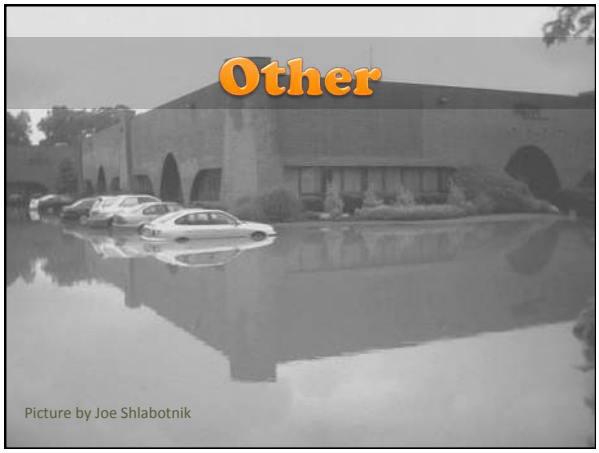
Many DBMS products allow you to audit connections. It may not be enough, it's more useful for forensics analysis and won't prevent people from breaking into the database, but it can be useful information to collect, just in case.

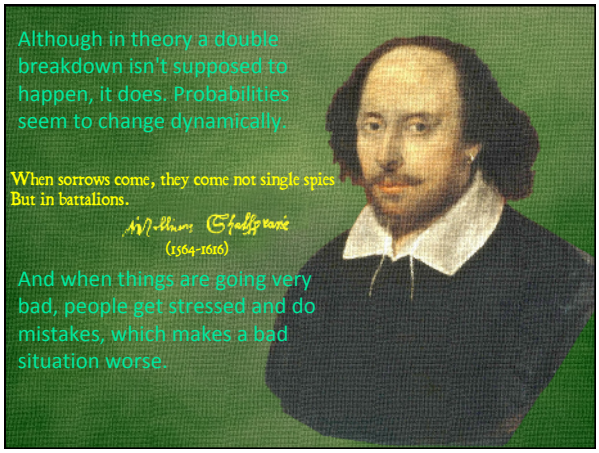
Things never go smoothly in a big company. Hardware breaks. People goof. Software is buggy. Some companies can have problems with end-of-month processes every month (it may be difficult to diagnose). I have known a big company that crashed its systems the Saturday before Xmas three successive years (I don't know after, I left them).

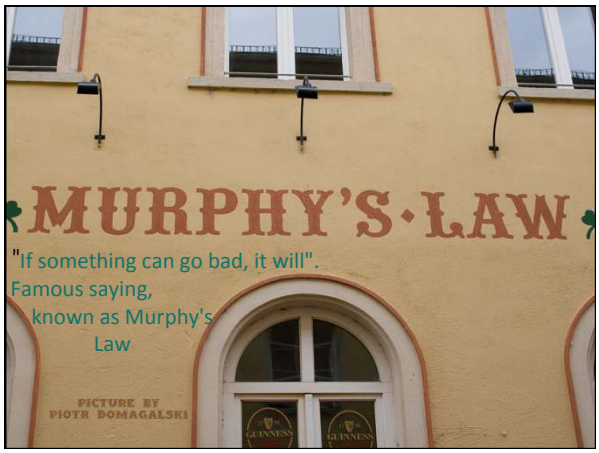
What can go wrong?

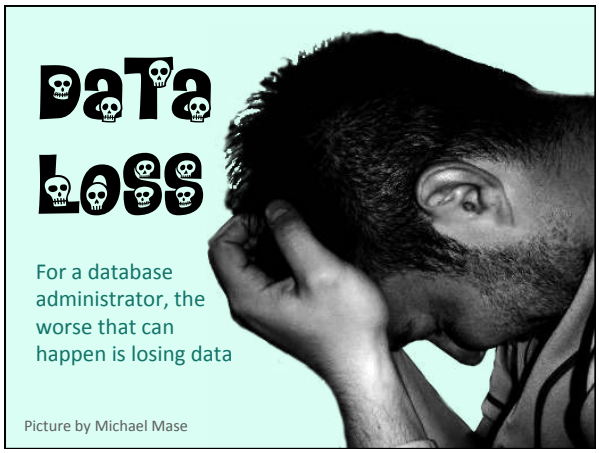





Other
Picture by Joe Shlabotnik


Although in theory a double breakdown isn't supposed to happen, it does. Probabilities seem to change dynamically.
When sorrows come, they come not single spies
But in battalions.
(1564-1616)
And when things are going very bad, people get stressed and do mistakes, which makes a bad situation worse.


MURPHY'S·LAW
"If something can go bad, it will".
Famous saying,
known as Murphy's
Law
PICTURE BY
PIOTR DOMAGALSKI


DaTa
LoSS
For a database administrator, the worse that can happen is losing data
Picture by Michael Mase

The first duty of a database administrator should be to know how to backup and restore a database.

BACKING UP A DATABASE

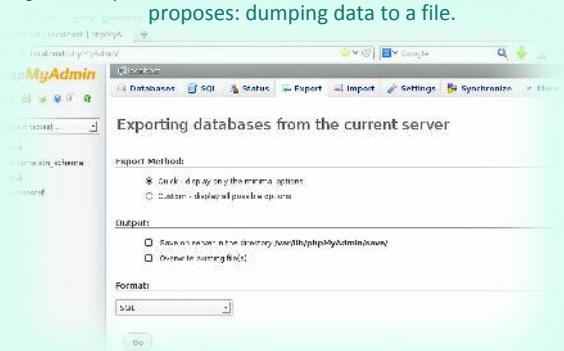
Logical backup

Physical backup

There are several ways to do it, and there are very significant differences between the different ways.

Logical backup

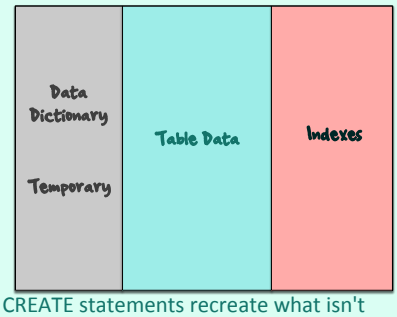
A logical backup is what phpMyAdmin proposes: dumping data to a file.



That's what many MySQL users only know.

Logical backup

A logical backup with mostly save CREATE statements and table data



CREATE statements recreate what isn't saved.

Logical backup

+

Database up — You are querying from it

Smaller file (~30%) — No indexes (just CREATE), no bulky system areas.

Reloading re-organizes

Partial restore — You can reload a single table

Logical backup

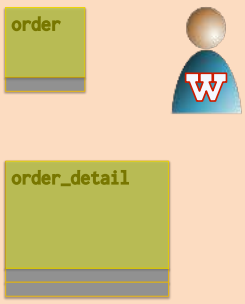
–

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.

Logical backup

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, COMMIT otherwise you may save rows and not the rows that they reference.



It's a "repeatable read" problem.

Logical backup

The big issue is that as data is reloaded, row addresses change.

Beware of consistency

SLOW recovery

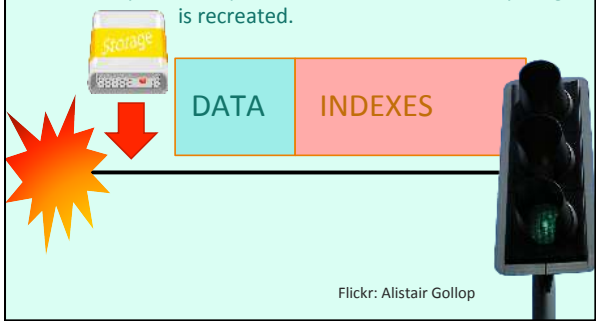
Indexes must be completely rebuilt, and on a big table it can be fairly slow.

Can be a problem

Always a problem

Logical backup

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.



Flickr: Alistair Gollop

Logical backup

Rarely popular with end-users (and even less with top management)

SORRY FOR ANY INCONVENIENCE

Flickr: Bev Goodwin

Logical backup

Beware of consistency

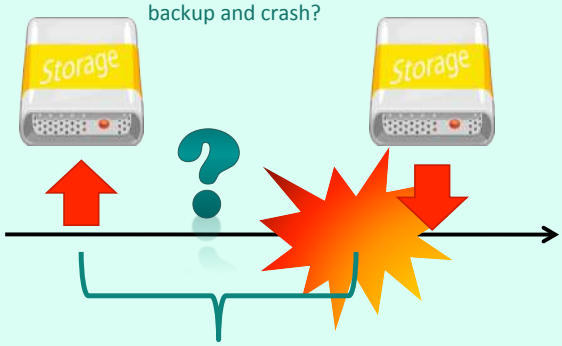
SLOW recovery

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.

Logical backup

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



BACKING UP A DATABASE

Logical backup

Physical backup

The other main way of backing up a database is the physical backup, which is completely different.

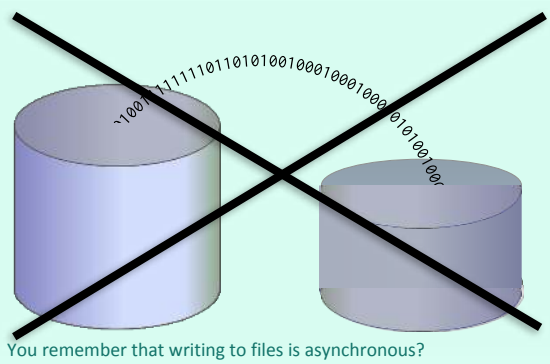
Physical backup

COPY FILES

A logical backup is like running a select * on all tables in a succession. A physical backup doesn't care about tables and data. It's just copying the files that contain them. It's data-blind.

Physical backup

Except that you can't just do a copy of a database file without any precaution.



You remember that writing to files is asynchronous?

Physical backup

Because the "true image" of the database is in memory, datafiles alone don't reflect it - unless the database is shut down and everything has been flushed to files in the process.

COPY FILES

but ...

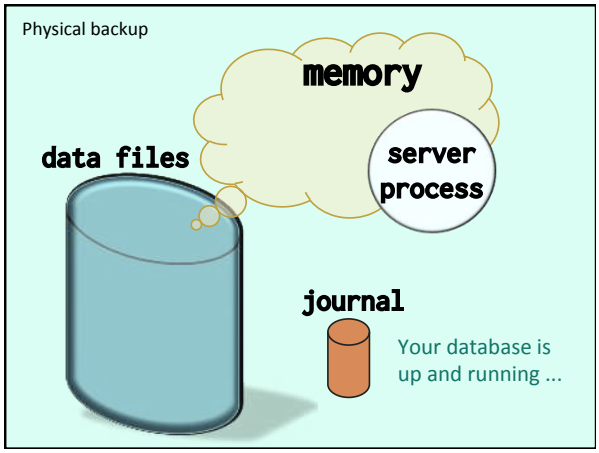
Everything happens in memory!

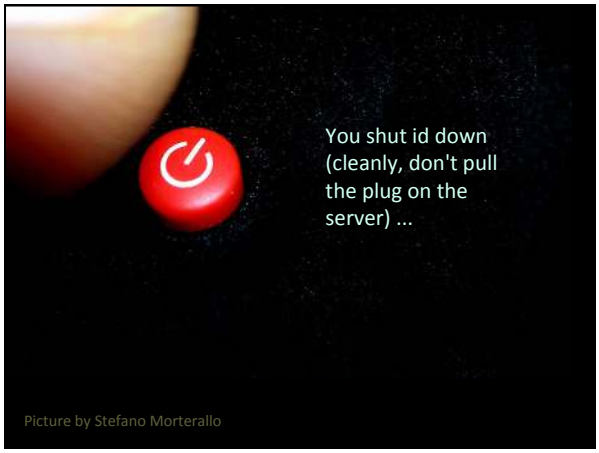
IMPORTANT

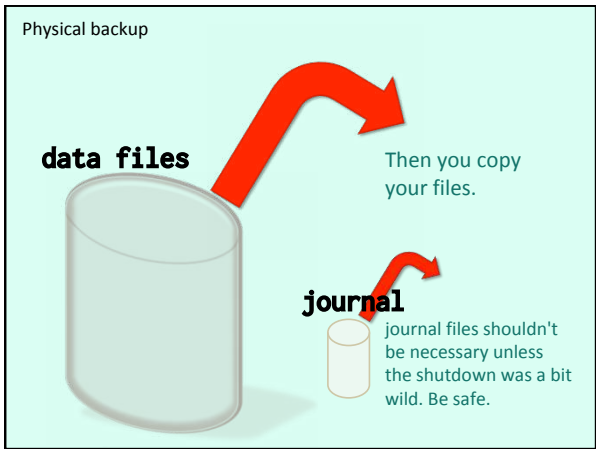
Get file names from the data dictionary

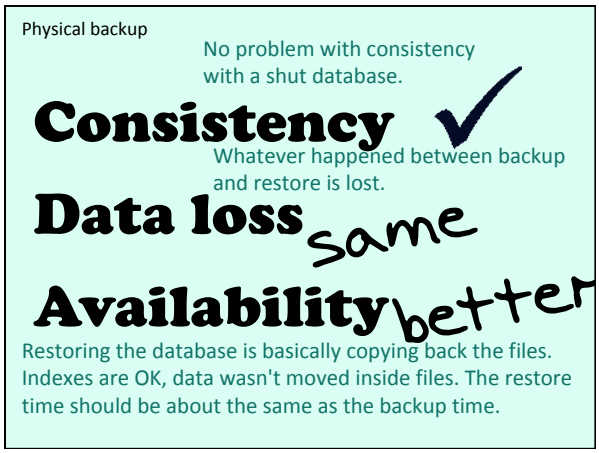
As an aside, it's a good practice, just before a backup, to get the data location from the database itself (before shutting it down if you shut it down). Some data files may be outside usual directories and it's always unpleasant to be unable to open a restored database because some files are missing.

Shutting down the database before copying the files is called a cold backup. You can do it with any DBMS.


Physical backup
memory
data files
server process
journal
Your database is up and running ...

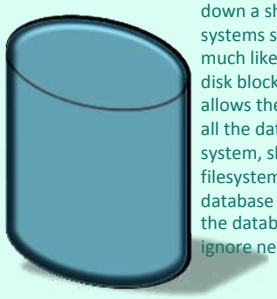

You shut id down (cleanly, don't pull the plug on the server) ...
Picture by Stefano Morterallo


Physical backup
data files
Then you copy your files.
journal
journal files shouldn't be necessary unless the shutdown was a bit wild. Be safe.


Physical backup
No problem with consistency with a shut database.
Consistency ✓
Whatever happened between backup and restore is lost.
Data loss same
Availability better
Restoring the database is basically copying back the files. Indexes are OK, data wasn't moved inside files. The restore time should be about the same as the backup time.

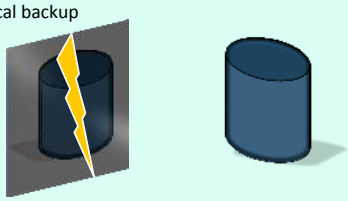
Copying big files takes time (even with fast disks) but a scheduled database shutdown is acceptable in many companies that don't operate on a 24x7 mode

Physical backup

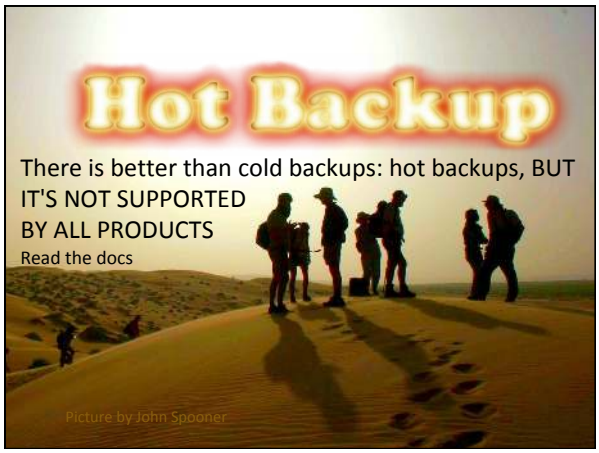


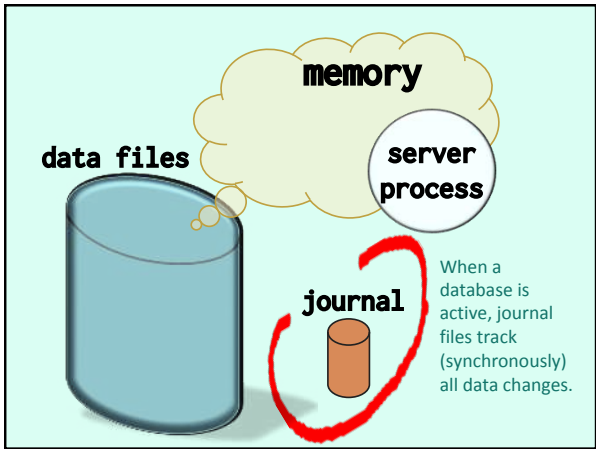
Additionally, there may be (operating system) tricks for shutting the database down a short time. Some advanced file systems support "cloning", which is very much like undo in DBMS products (saving disk blocks before they are modified) and allows the same as a "repeatable read". If all the data files 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.

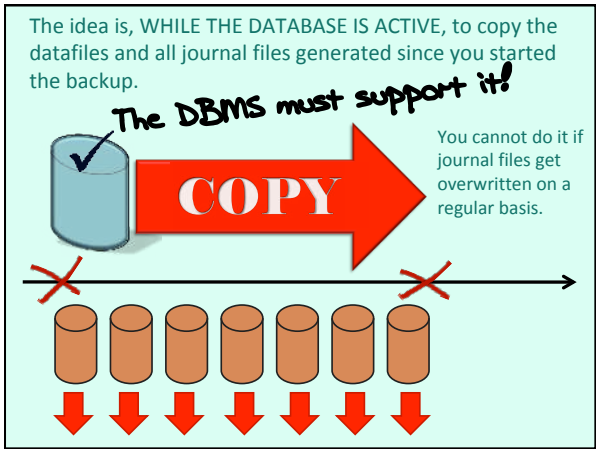
Physical backup

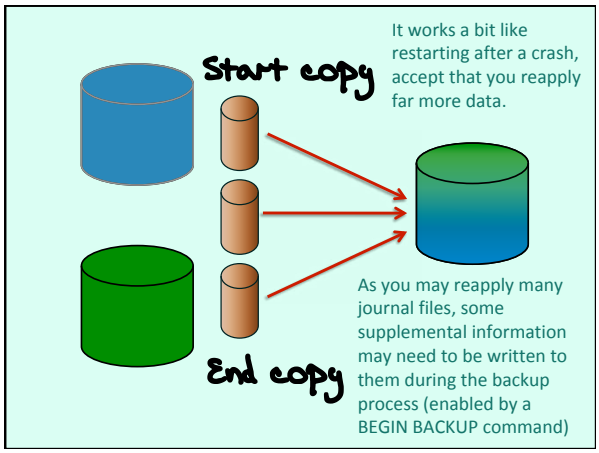


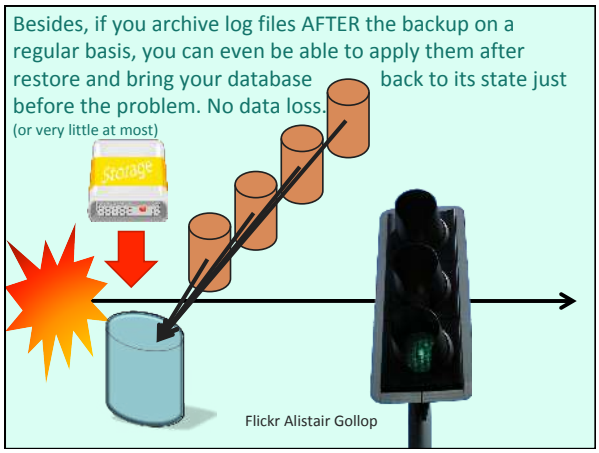
With mirrored disks, you can shutdown the database, deactivate mirroring (immediate), restore the database with mirrorring deactivated on one disk, and quietly copy the disk that is no longer written. When you are done, you reactivate mirroring (no need to stop the database), there is a "resilvering"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




memory
data files
server process
journal
When a database is active, journal files track (synchronously) all data changes.

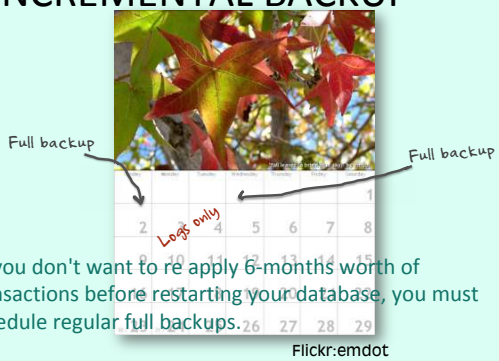

The idea is, WHILE THE DATABASE IS ACTIVE, to copy the datafiles and all journal files generated since you started the backup.
The DBMS must support it!
COPY
You cannot do it if journal files get overwritten on a regular basis.


Start copy
It works a bit like restarting after a crash, accept that you reapply far more data.
End copy
As you may reapply many journal files, some supplemental information may need to be written to them during the backup process (enabled by a BEGIN BACKUP command)


Besides, 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.
(or very little at most)
Storage
Flickr Alistair Gollop

Backing up log files is known as an incremental backup.

INCREMENTAL BACKUP



Flickr:emdot

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

Consistency is perfect, in many cases you won't lose any data at all.

Consistency ✓

Data loss ✓

Availability

You need to restore your last full backup, then reapply all logs generated since then.

good

Physical backup

+

Faster restart

Can be incremental AND consistent

No or little data loss

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.

Physical backup

−

Cannot recover ONE table

Unless you restore your backup as a different database, then dump the data from that table (long and painful).

More complicated

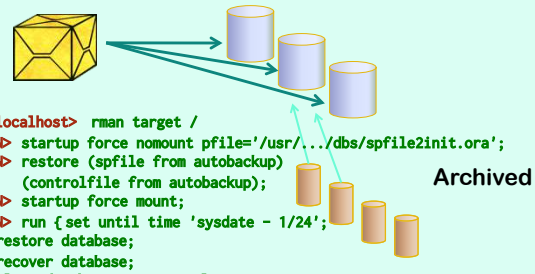
For the hot backup only

Big files

You need twice the storage used by the database. Can be VERY big.

POINT-IN-TIME RECOVERY

Hot and incremental backups allow fancy operations, such as a point-in-time recovery. Scenario: you start the brand new application at 2:05pm and notice after 50 minutes that because of a bug it has created a lot of inconsistencies in the database. You can restore the last backup and ask for logs to be reapplied up to one hour ago, when the database was still pristine.




```
me@localhost>  rman target /
RMAN> startup force nomount pfile='/usr/.../dbs/spfile2init.ora';
RMAN> restore (spfile from autobackup)
2>    (controlfile from autobackup);
RMAN> startup force mount;
RMAN> run {set until time 'sysdate - 1/24';
2> restore database;
3> recover database;
4> alter database open resetlogs;}
```

Oracle's product Recovery Manager (RMAN) does it very well, albeit with a syntax that borders on the esoteric.

ORACLE The commands make sense, but only to an Oracle DBA.