CS307

Database Principles

Stéphane Faroult
faroult@sustc.edu.cn

朱悦铭 Zhu Yueming zhuym@sustc.edu.cn

Communication Manager
Query Processor
Transaction & Buffer Manager
Storage Manager
Process Manager

What we have seen last time explains why an enterprise-grade DBMS is a complicated beast, with a lot of subcomponents taking some special tasks in charge, embodied by several background processes that must be coordinated.

Loosely inspired by Hellerstein/Stonebraker/Hamilton
Architecture of a Database System

STARTUP

Something as "simple" as starting a database requires multiple steps, checking that the database was cleanly shut down, and starting a recovery process if it wasn't.

Read configuration file

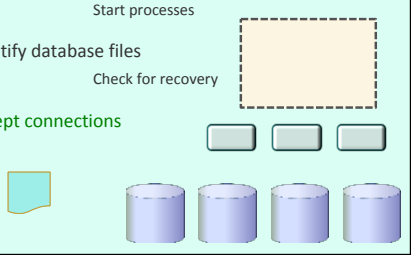
Allocate shared memory

Start processes

Identify database files

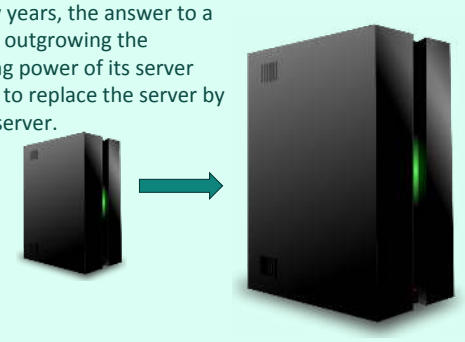
Check for recovery

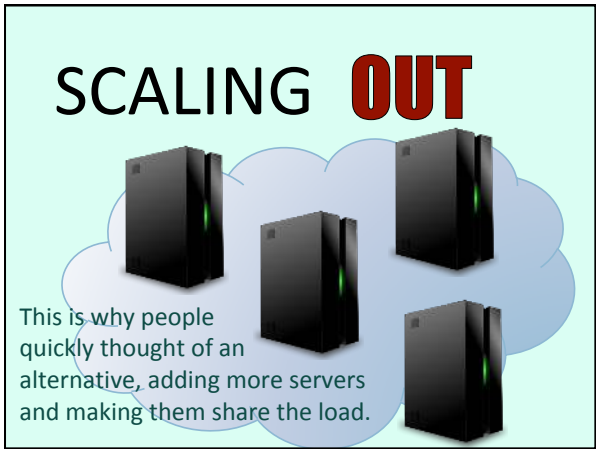
Accept connections



SCALING UP

For many years, the answer to a database outgrowing the processing power of its server has been to replace the server by a bigger server.



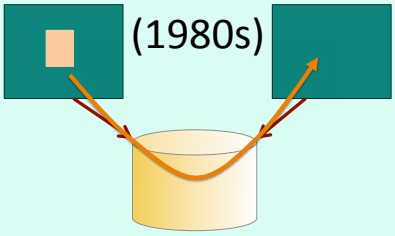


Distributed Systems

But here everything becomes more complicated all of a sudden.

In the 1980s, Oracle tried connecting multiple servers to a single database. Complete disaster when the two servers wanted to modify the same data (or simply when one wanted to see what had just been modified by the other), data had to transit via the files.

Oracle Parallel Server (1980s)



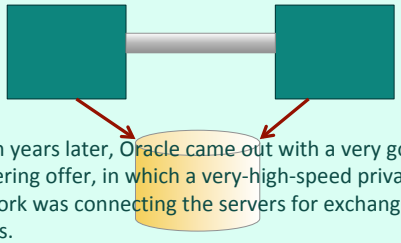
In 1994 Oracle bought from Digital Equipement (which they swallowed whole later), RDB, a respected relational database management system that only worked on Digital systems (SW/HW).

1994 RDB acquired by Oracle

One of the strengths of RDB was working with clusters of machine; it was the only product doing it well. All of a sudden, Oracle Development gained strong competencies in this area.

Oracle RAC (2001)

Real Application Clusters



Seven years later, Oracle came out with a very good clustering offer, in which a very-high-speed private network was connecting the servers for exchanging data blocks.

- Coordinator node
- Node "owns" blocks
- Owner can change

RAC implements a complicated "cache-fusion" algorithm in which a coordinator nodes always knows who is working on what , and blocks can be exchanged quickly. Of course it works better when the workload is fairly different for all servers, but when different servers can work in parallel on disjoint data, it can be extremely efficient and it handles conflicts well.

Neil Gunther's

Universal Law of Computational Scalability

Relative capacity C(*N*) of a computational platform:

$$C_{(N)} = \frac{N}{1 + \alpha\,(N-1) + \beta\,N(N-1)}$$

$0 <= \alpha, \beta < 1$

www.perfdynamics.com

α Level of contention

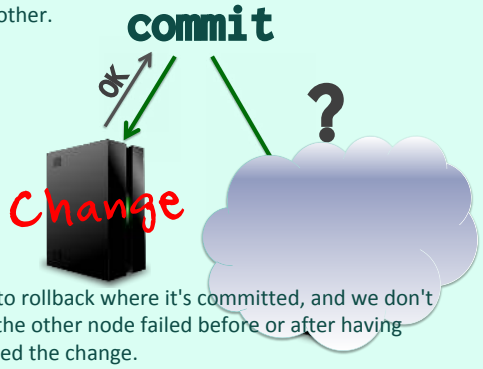
β Coherency delay (latency for data to become consistent)

The problem with clustering is that it follows a law of diminishing returns: adding a second server will less than double your capacity, and in practice people have clusters of 2, 3 or 4 machines at most (Neil Gunther is a famous Australian consultant/academic specializing on performance)

One big problem is with transactions that involve several servers. Remember that transactions are meant to be atomic operations.

Distributed Transactions

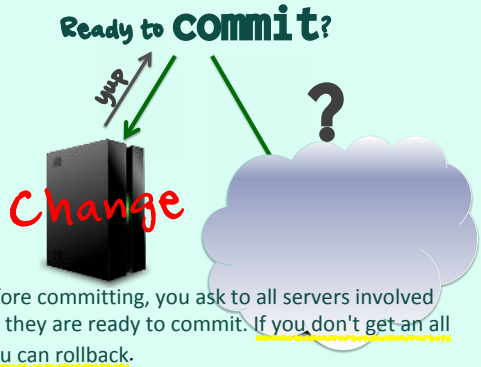
It may happen that when we commit we know for sure it worked on one server, and we get no acknowledgment from the other.



No way to rollback where it's committed, and we don't know if the other node failed before or after having committed the change.

TWO-PHASE COMMIT

One algorithm was devised (a long time ago), called a "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

Odds that something will fail are far lower than with a single-phase commit, but not zero.

There still may be 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 often 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

Additionally, you have latency issues. All machines in a cluster may not be sitting next to each other, they may be a few miles apart in different data centers for security reasons (fire, flood ...)

1 KM = 0.000005s

Even if information travels fast, multiplying exchanges (two-phase commit) may become a sensitive issue.

Synchronous? Asynchronous?

At which point you have to choose between a synchronous mode in which you always wait for an acknowledgment that everything went fine, or an asynchronous mode in which you cross fingers and switch to something else. The type of hardware you have may influence your decision: if all your disks have a big buffer and a reliable battery for instance, you may be more likely to trust them.

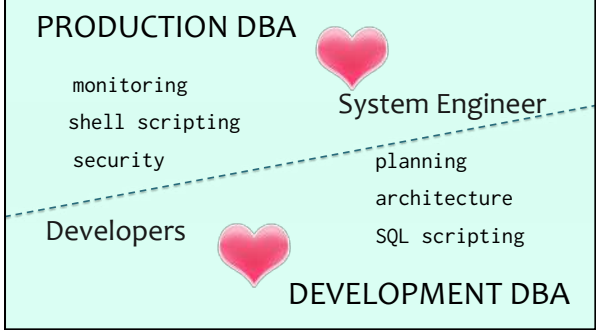
Senior Database Administrator

-/-

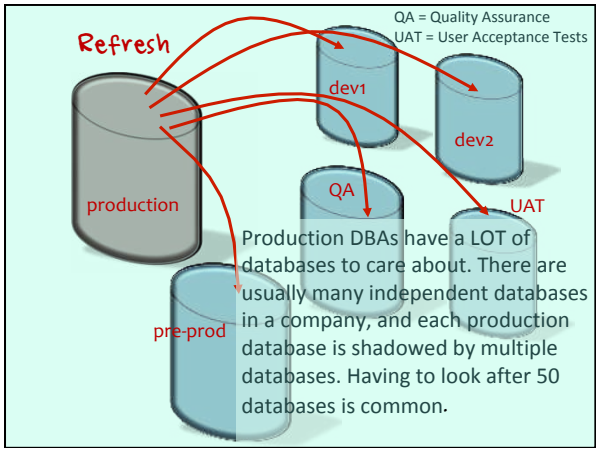
What you will do: Job posting (US)

- Manage MySQL in production/QA/dev environments including installation, configuration, upgrades, schema changes, etc.
- Troubleshoot database issues, maintain database systems availability and scalability within production environments
- Perform capacity planning exercises to properly identify required hardware, software, database configuration/architecture necessary to support application needs
- Monitor database performance, identify performance problems and make adjustments to database parameters as needed.
- Monitor key performance indicators and make enhancements to improve/maintain performance/productivity at acceptable levels
- Enforce best practices for improving performance, scalability and operational manageability of production databases
- Part of on-call rotation to respond to and resolve application issues to ensure production applications are online

DBAs come in two flavors (in smaller companies, it's the same guy). They often have completely different backgrounds, and don't necessarily agree on everything.



In many ways, a development DBA is a developer who knows more than the average about databases. We'll focus here a little more on the job of a production DBA, because architects who design information systems aren't always well aware of what their job is, and some "solutions" that look great on the paper are sometimes a hell to maintain in a daily production, with the constraints of keeping systems available and running as much as possible. Keeping everything simple and manageable should be the first concern of every architect.



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. Having to look after 50 databases is common.

If you aren't square, you can't survive in this job. You must automate as many tasks as possible.

STANDARDS

ORGANIZATION

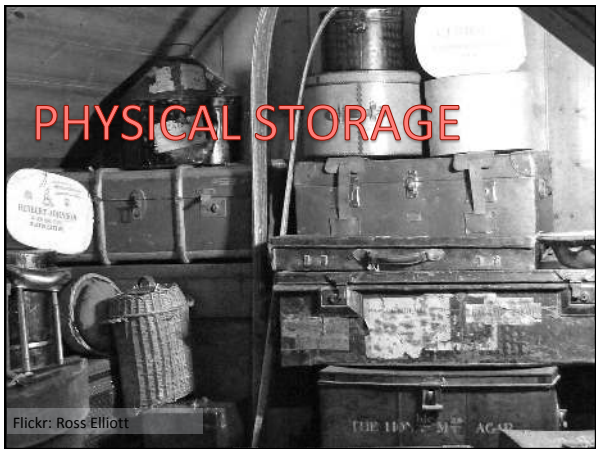
AUTOMATION

Other than performance issues (than we have partly seen) these are the big topics for a DBA. We have talked a bit about privileges in labs, let's talk about storage.

Physical Storage

User Management

Backup / Recovery



DATABASE?

To start with, the word "database" has a different meaning depending on the DBMS (and sometimes DBMS version) that you are talking about.

ORACLE

Microsoft SQL Server

?

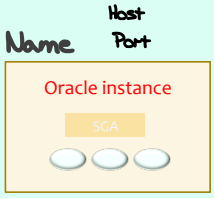
PostgreSQL

MySQL

User's View

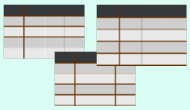
ORACLE

< 12C



For instance with Oracle up to 12c a database is basically an instance (shared memory and processes) and files.
A schema is a user account that owns tables.

Schema (Account)

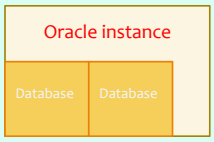


```
select ...
from schema.table_name
     synonym_name
```

User's View

ORACLE

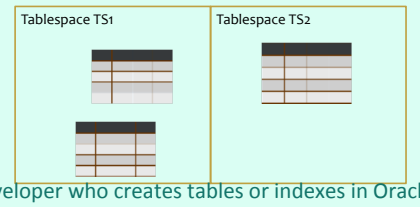
> 12C



From Oracle 12 the vision is altered, and an instance becomes a container that may hold several independent databases. Prior to Oracle 12 it was common to have multiple instances, each with their (big) shared memory and processes on the same server. Oracle 12 pools resources.

Developer's View

ORACLE



A developer who creates tables or indexes in Oracle create them in a tablespace, which can be specified at the end of the CREATE statement (there is a default tablespace for every account)

DBA's View

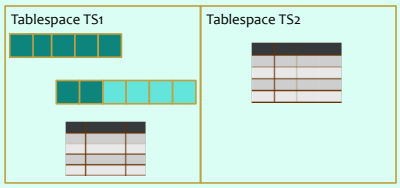
ORACLE

For a DBA, a tablespace is a set of files. It can also be a partition without a filesystem on a disk (raw device)

DBA's View

ORACLE

Where a developer sees tables in indexes, a DBA sees SEGMENTS of different types.

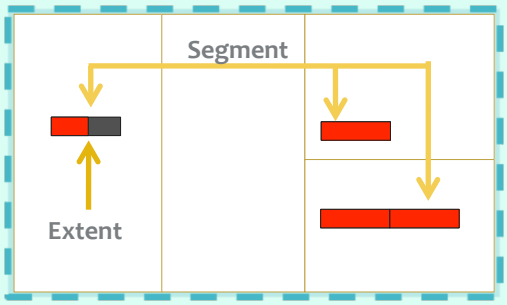


Segments grow as needed. They are made of BLOCKS, which is the transfer unit between disk and memory (size can vary, 8K is common)

DBA's View

ORACLE

Segments are composed of extents of contiguous blocks. A segment is always in a single tablespace, but extents can be in different files of the tablespace.

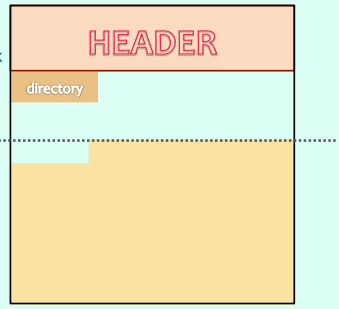


DBA's View

ORACLE

A block contains a header (700/800 bytes), a directory pointing to the beginning of each row, and row data, stored from the end of the block.

When you insert, a percentage of the block is kept free for updates that would make rows grow in size.



PostgreSQL blocks look very similar.

DBA's View

ORACLE

Many tablespaces are created by default.

+ log files journal
+ control files

list of files with last known state.

SYSTEM
Data dictionary

SYSAUX
Non-critical system data

USERS

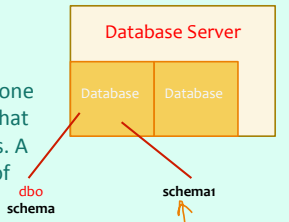
UNDOTBS
Before-change data for rollbacks and consistent reads.

TEMPTS
For sorts (indexing, some join operations)

Developer's View

Microsoft SQL Server

With SQL server you have one database server per host that contains several databases. A schema is a logical group of tables.

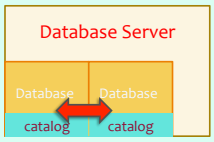


dbo (database owner) is the default schema

Developer's View

Microsoft SQL Server

IMPORTANT



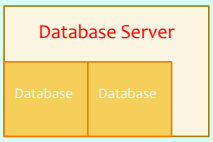
No FK

What is important is that each database comes with its own catalog. In other words, databases are independent and you can't have foreign keys between different databases. PostgreSQL works similarly.

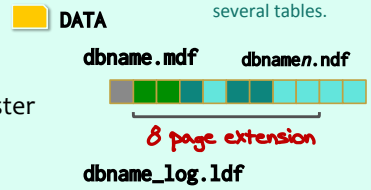
DBA's View

Microsoft SQL Server

Tables are stored in files managed by the DBMS, with pages in 8 page extents.



In SQL Server, pages in one extent can belong to one or several tables.



DBA's View

Microsoft SQL Server

FILEGROUP

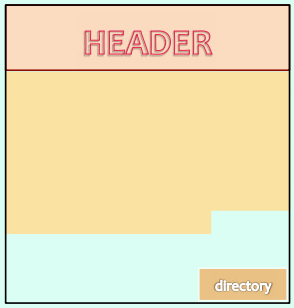
PRIMARY by default

There is no tablespace in SQL Server but "filegroups" serve a similar purpose and allow to store data elsewhere than in the default directories.

DBA's View

Microsoft SQL Server

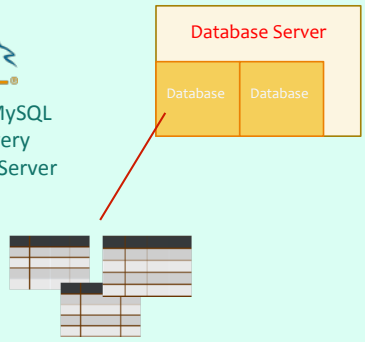
SQL Server pages are filled in the opposite way of Oracle blocks, but are otherwise very similar.



User's View

MySQL

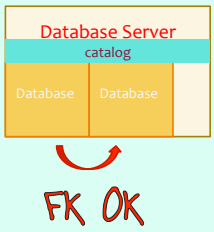
Superficially, a MySQL database looks very much like a SQL Server database.



Developer's View

MySQL

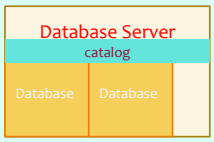
But in MySQL, the catalog is SHARED. In fact, you have a single database, and what MySQL calls a database is in fact a schema.



MySQL Database = SQL Server Schema

Developer's View

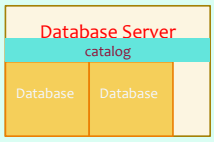
MySQL



What is particularly interesting in MySQL (a product 10 years younger than the others) is that storage is disconnected from the other layers and EACH table can be assigned a different "storage engine".

Storage Engine

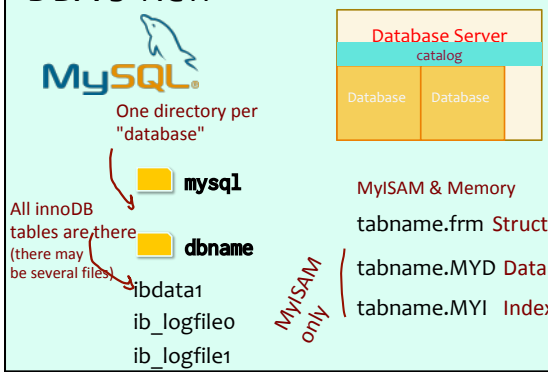
Developer's View



MyISAM is the default storage engine but is a bit primitive. Each table is stored as independent files (as in PostgreSQL). InnoDB is what you use for enterprise applications. Your tables need to use InnoDB to have foreign keys. With a memory table, only the structure is saved.

MyISAM
InnoDB
Memory

DBA's View



Tables
Indexes
(Clusters)
(Partitions)
Lobs
Materialized Views

This is basically what you find in an Oracle database. Stuff in this quadrant takes a lot of space on disk.

Views
Procedures
Triggers
Sequences

Data Dictionary (catalog)

Temp **Undo**

System Storage Bulky too.

OS Integration

A product such as Oracle (for which, particularly in the earlier years, portability was important) tends to substitute itself to the OS even for very low level operations. SQL Server on the other end is tightly integrated to Windows.

OS portability

ORACLE

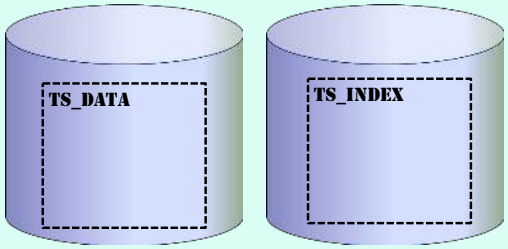
Do-It-Yourself

Historically, DBAs used to care about where files where going.

Physical file placement USED to be important

ORACLE

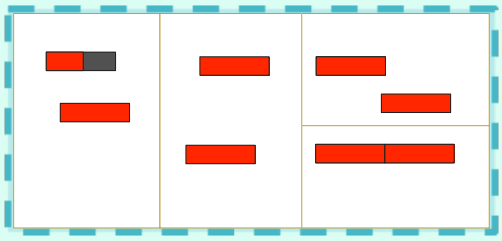
Most Oracle applications still separate data and indexes in different tablespaces.



Physical file placement USED to be important

ORACLE

Defragmentation used to be a concern. SQL Server DBAs periodically rebuild their indexes.



TOO BIG TO REORG

Came one day when bases became so big and availability requirements so strong that defragmenting a database simply was out of the question. Today focus is mostly on finding a way to store data that doesn't degrade too much over time. DBMS vendor also introduced tools for reorganizing an active database with not too much impact.

Physical file placement USED to be important

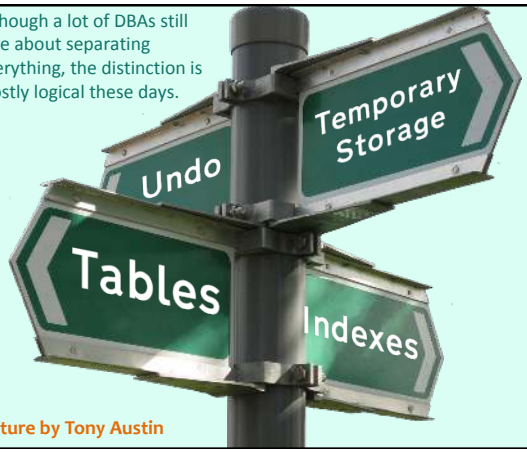
ORACLE

Today

Automatic Storage Management

Oracle has for instance introduced ASM, which is a volume manager and file system specially designed for Oracle files, which automatically takes care of a lot of thorny issues. Oracle DBAs no longer care much about physical files.

Although a lot of DBAs still care about separating everything, the distinction is mostly logical these days.



Picture by Tony Austin

Storage Area Network (SAN)

Seen as a disk

EMC² Hitachi Data Systems

Network-Attached Storage

Seen as a file-server

NetApp

Exadata ("Database Appliance") ORACLE

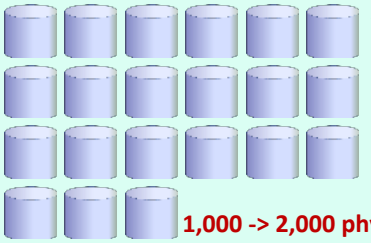
Especially as a lot of intelligent (and expensive) "storage devices" are usually plugged behind computers (and shared by them).

Redundant Array of ~~Inexpensive~~ independent Disks

All started with RAID and the idea of putting a bunch of disks together, making them look like a single unit and reading/writing them in parallel to massively increase throughput. 'I' in RAID used to mean inexpensive (because each disk IS inexpensive), but as you have a LOT of inexpensive disks the total isn't cheap ... It was conveying the wrong idea and 'I' now stands for "independent".

Cache

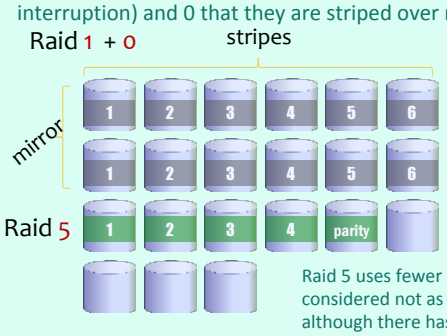
512GB-2TB



That's what a SAN looks like.

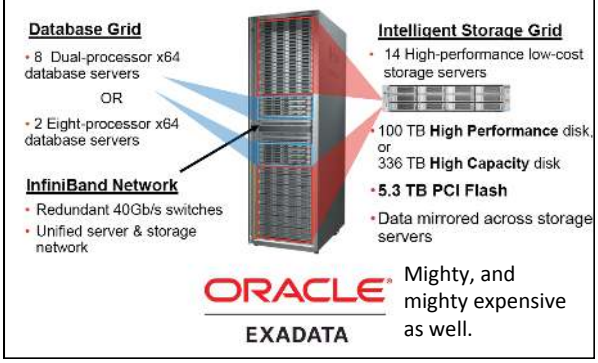
1,000 -> 2,000 physical disks

DBAs usually prefer raid 1+0: 1 means that files are mirrored (if a disk fails, it can be disconnected and replaced without interruption) and 0 that they are striped over many disks.



Raid 5 uses fewer disks, but is considered not as performant, although there has been progress in this area.

Exadata is an hybrid between storage and server: part of queries are locally processed, and "information-rich" blocks are returned to the front-end database server.



Mighty, and mighty expensive as well.

FILE PLACEMENT

⇨ Storage Specialist

With a SAN at the back of a computer, a DBA these days no longer really knows where everything is. A storage specialist usually knows, and a DBA may have to work with this specialist for files that are heavily accessed, including log files that are written sequentially and should ideally not be on disks that undergo other heavy I/O activity.

Things that STILL matter To DBAs

Directory structures (scripts!)

Table organization

The logical structure though still matters (having database files everywhere is a bad idea, especially in /tmp) and care must be taken of inside table organization.

Do we need this?

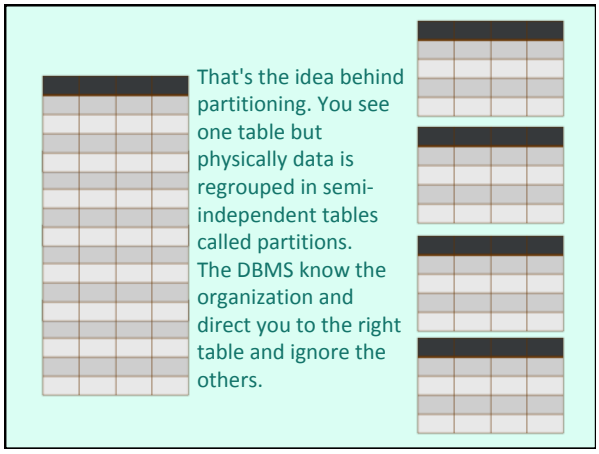
You are going to turn to a book index only when you are looking for very specific information.

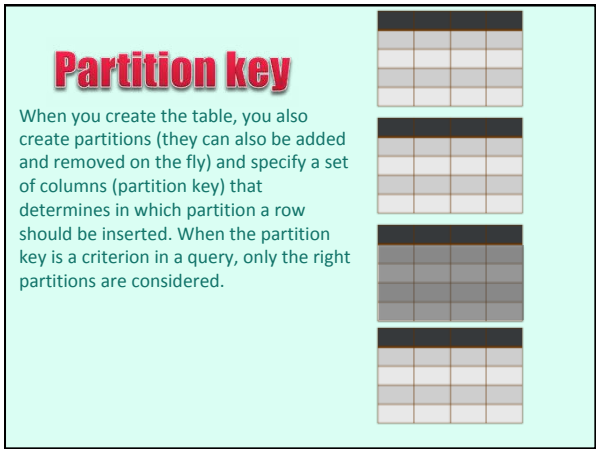
Or do we need this?

When there are many entries in the index for what you are looking for, you rather turn to the table of contents.

Partitioning

The idea of partitioning is very similar. 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.





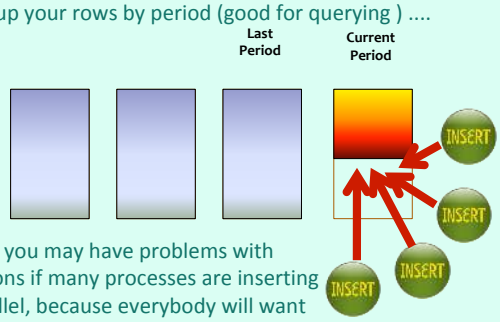
There are different ways to partition. When your concern is to control data grouping, you usually partition by range (so as to have rows grouped by month, week, or whatever interval), or by list (If the row contains this value in the partition key, then it goes into this partition). When your concern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

By range *(dates usually)*

By list

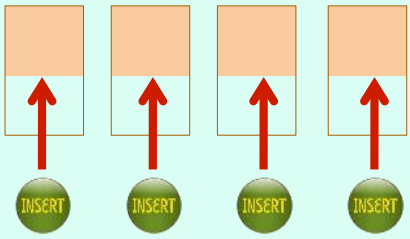
By hashing

Partitioning may want to address very different problems. If you group your rows by period (good for querying) ....



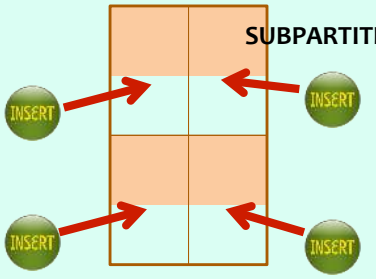
... then you may have problems with insertions if many processes are inserting in parallel, because everybody will want to insert at the same place and you'll need some serialization (two processes cannot write bytes in the same block at the same time).

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.