

Dynamic Monitoring

To briefly complete what was said about the catalog, I should add that all big product provide some monitoring facilities that are views actually mapped to memory structures, which allow to see what is going on on a server right now. I'm giving here whta Postgres as but you should find something more or less equivalent everywhere.

`pg_locks`

When an application looks stuck, it's sometimes because of locks. Locks are taken on a row when you modify it in a transaction, and are only released when you commit or rollback. This may cause waits.

`pg_stat_...`

Many `pg_stat_` views. One shows index usage. Good way to check whether an index is useful (if it doesn't enforce a constraint)

`pg_stat_statements`

DATABASE?

But let's turn our attention to physical storage. To start with, the word "database" has a different meaning depending on the DBMS (and sometimes DBMS version) that you are talking about.



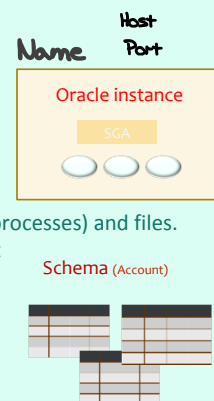
User's View

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.

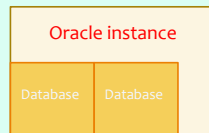
```
select ...
from schema.table_name
synonym_name
```



User's View

ORACLE

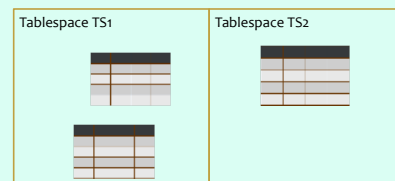
> 12C



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

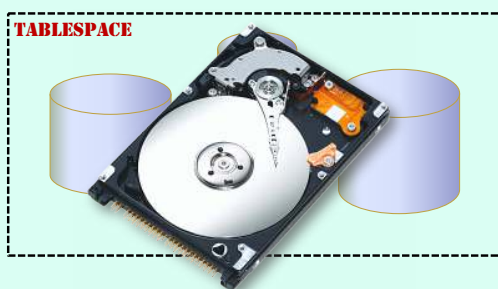


A developer who creates tables or indexes in Oracle create them in a tablespace, which can be specified at the end of the CREATE statement (there is a default tablespace for every account)

DBA's View

ORACLE

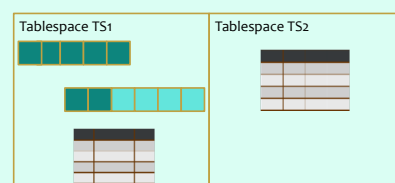
For a DBA, a tablespace is a set of files. It can also be a partition without a filesystem on a disk (raw device)



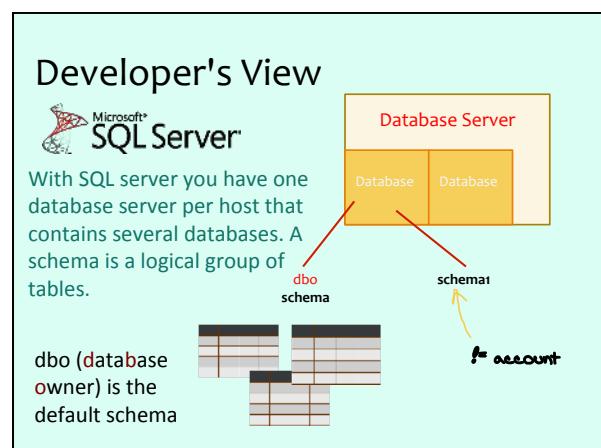
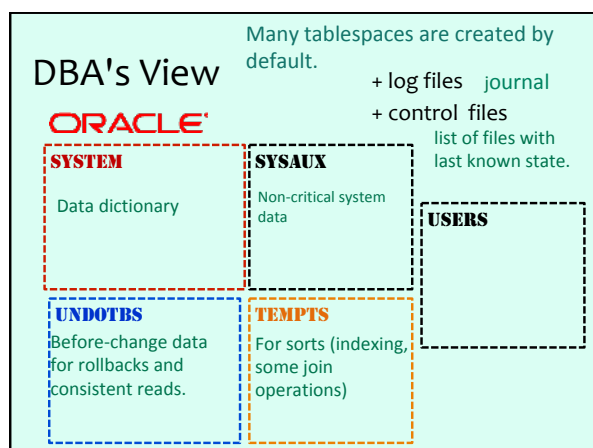
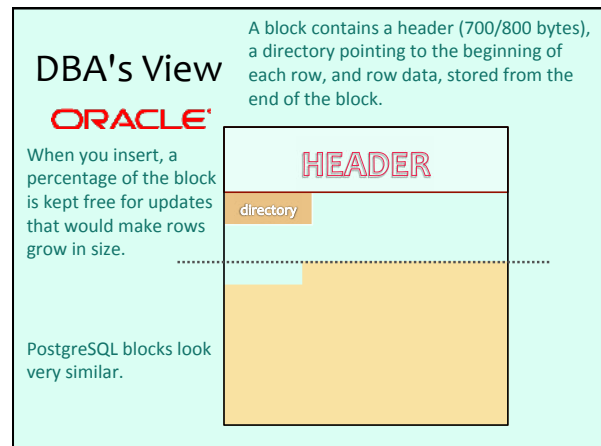
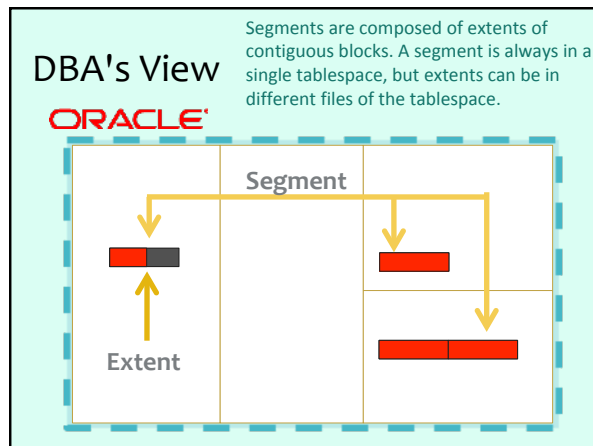
DBA's View

ORACLE


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



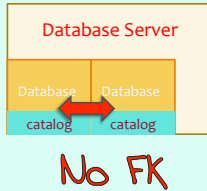
Segments grow as needed. They are made of BLOCKS, which is the transfer unit between disk and memory (size can vary, 8K is common)



Developer's View





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


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.

DATA

master


dbname.mdf dbname.ndf



8 page extension

dbname_log.ldf

DBA's View




FILEGROUP

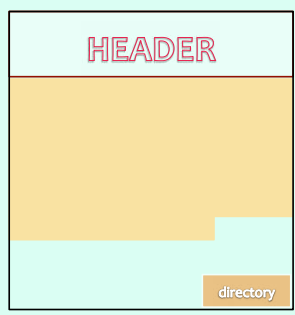
PRIMARY by default

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

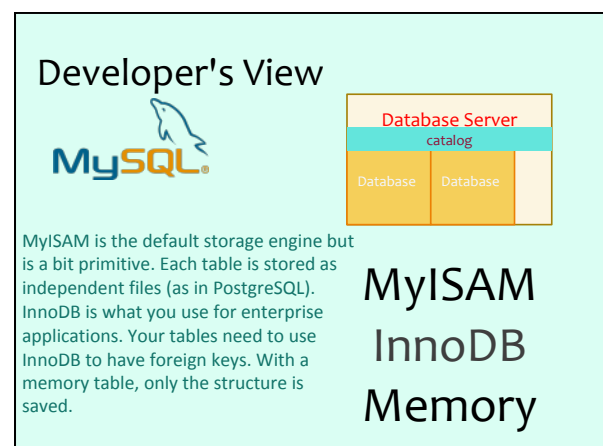
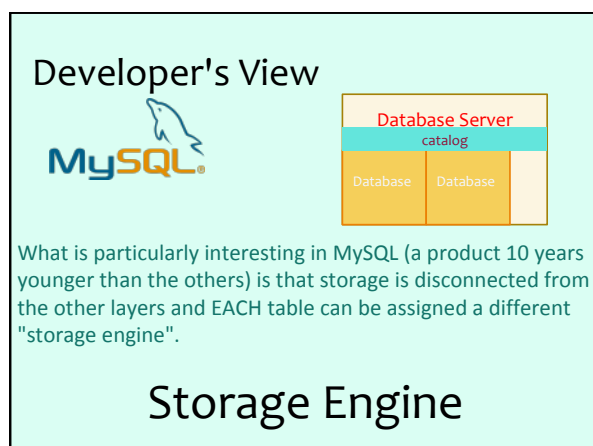
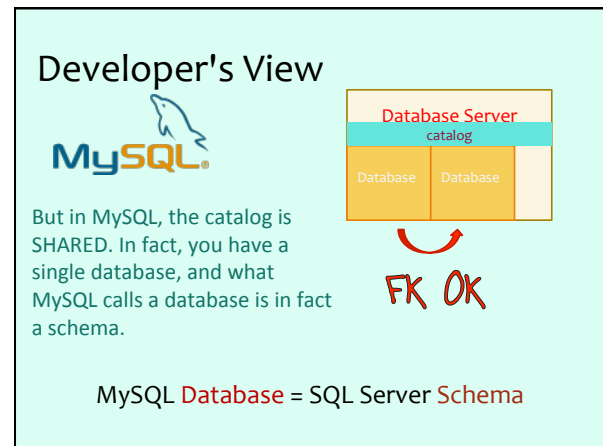
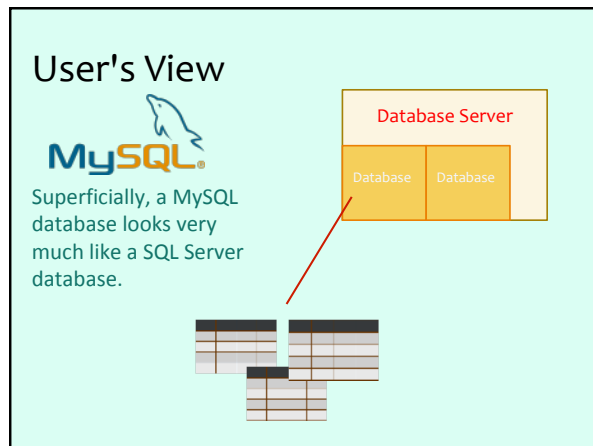
DBA's View

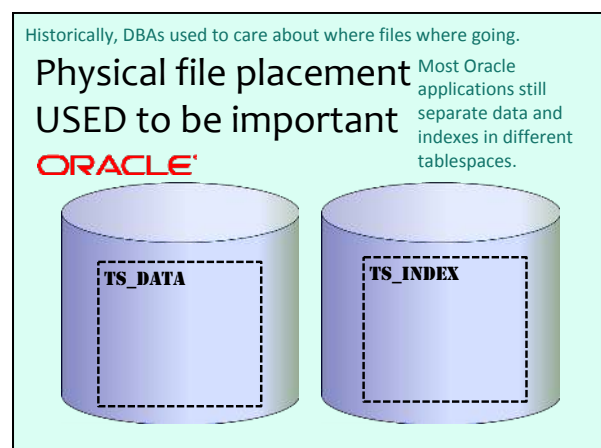
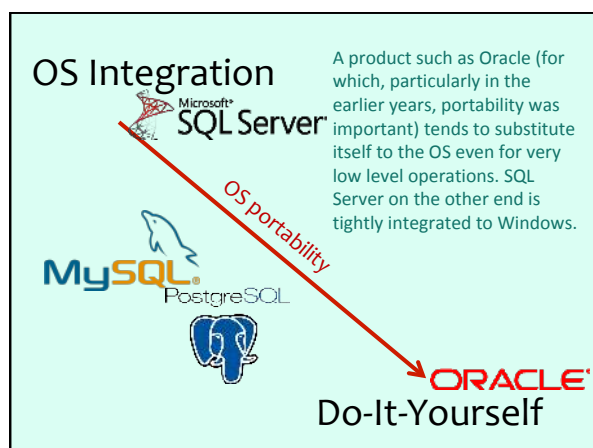
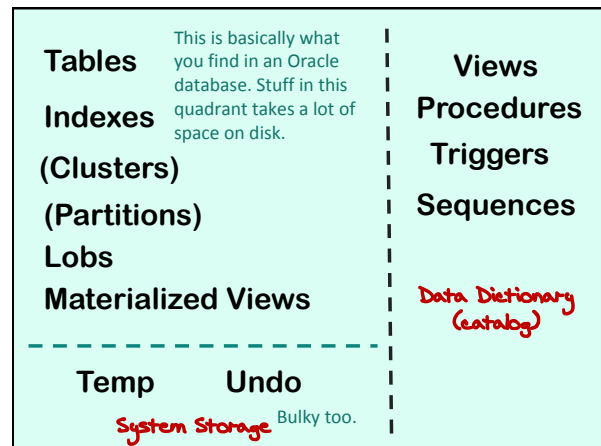
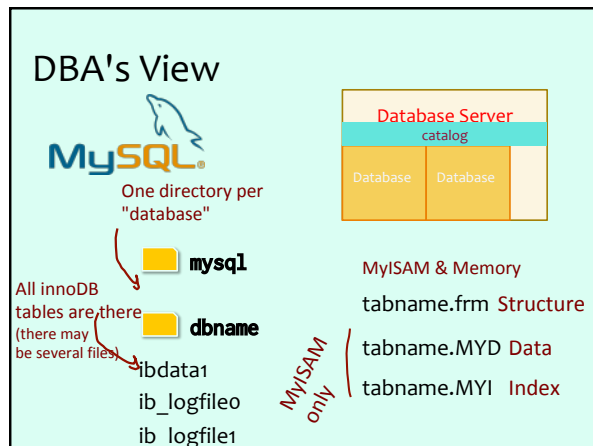


SQL Server pages are filled in the opposite way of Oracle blocks, but are otherwise very similar.



directory

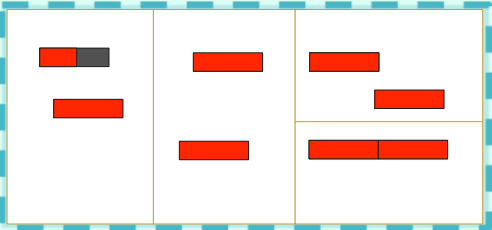




Physical file placement
USED to be important

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®

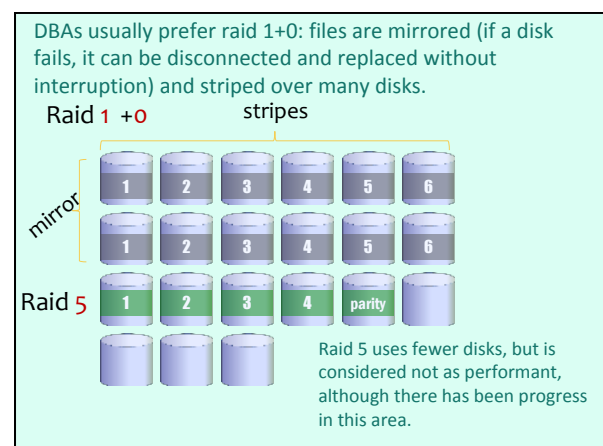
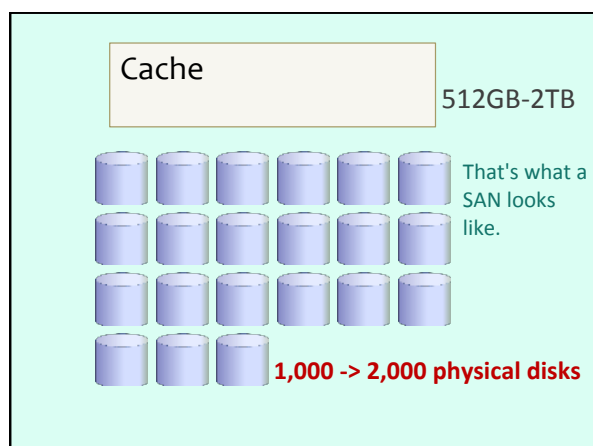
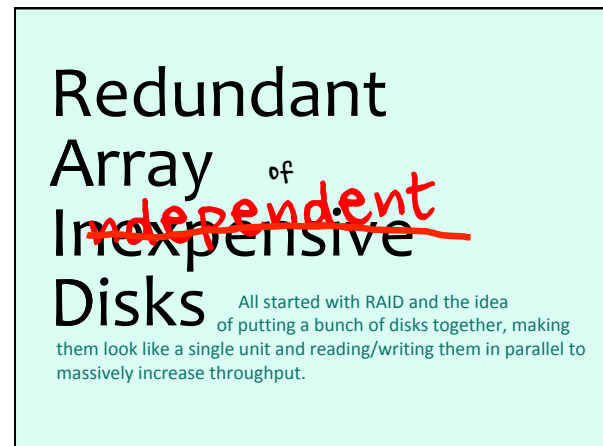
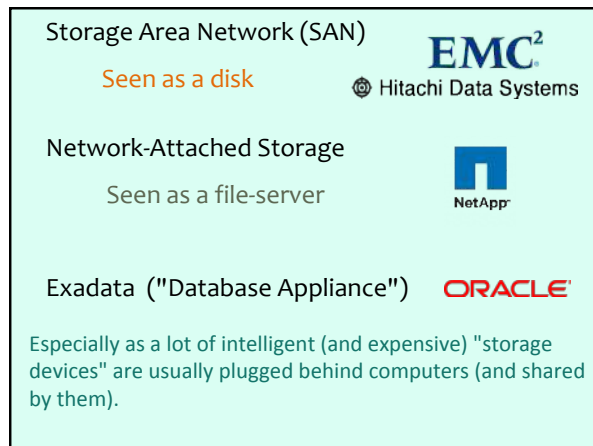
Today
Automatic Storage Management

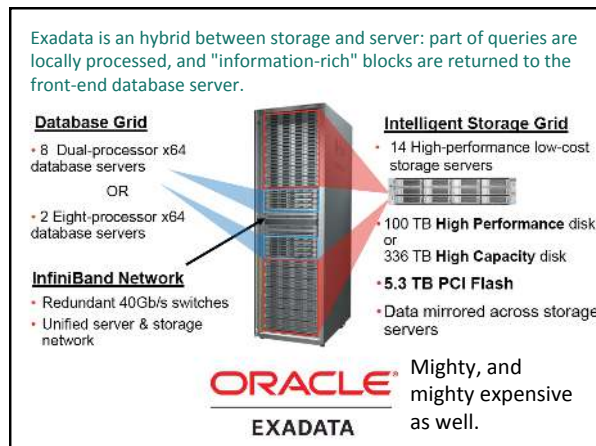
Oracle has for instance introduced ASM, which is a volume manager and file system specially designed for Oracle files, which automatically takes care of a lot of thorny issues. Oracle DBAs no longer care much about physical files.

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



Picture by Tony Austin





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?

A	
aborting transactions, 130	
ABS function, 48	
ACOS function, 50	
ADD_MONTHS function, 45	
aggregate functions, 56	
aliases	
column, 110-114	
UNION queries and, 133	
qualifying join columns, 82	
table, 35	
flashback queries and, 40	
in FROM clauses, 116	
ALL keyword, 56, 114	
comparison operators and, 97	
vs. FIRST keyword, 76	
ANSI/ISO CAST function, 9	
MySQL, 29, 34	
SQL Server, 23	
ANSI/ISO EXTRACT function, 10, 29	
ANSI/ISO WITH clause	
correlated subqueries, factoring out, 123-125	
noncorrelated subqueries, factoring out, 122	
recursive, 66	
ANSI_NULLS setting, 95	
ANY keyword and comparison operators, 97	
APPEND hint and direct-path inserts, 74	
AS OF keyword, 40	
ASCENDING keyword, 120	
ASIN function, 50	
asterisk (*) regular-expression metacharacter, 104, 106	
COUNT function and, 57	
qualifying with table names, 108	
returning all columns from a table, 107	
ATAN function, 50	
ATAN2 function, 50	
ATANH function, 50	
ATN2 function (SQL Server), 50	
autocommit mode, 35, 125	
AVG function, 57	

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

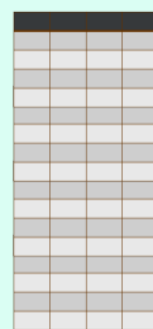
Or do we need this?

Functions	120
Date Functions	120
Numeric and Math Functions	122
Trigonometric Functions	124
String Functions	125
Miscellaneous Functions (Oracle)	130
Grouping and Summarizing	132
Aggregate Functions	132
GROUP BY	133
Useful GROUP BY Techniques	135
HAVING	137
GROUP BY Extensions (Oracle)	139
GROUP BY Extensions (SQL Server)	141
Hierarchical Queries	143
ANSI/ISO Recursive WITH (DB2)	143
CONNECT BY Syntax (Oracle)	146

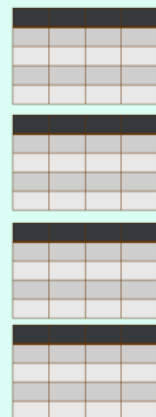
When there are many entries in the index for what you are looking for, you rather turn to the table of contents.

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.

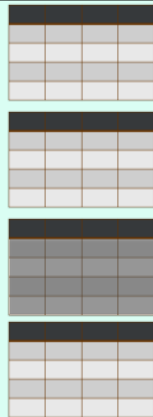


That's the idea behind partitioning. You see one table but physically data is regrouped in semi-independent tables called partitions. The DBMS know the organization and direct you to the right table and ignore the others.



Partition key

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.



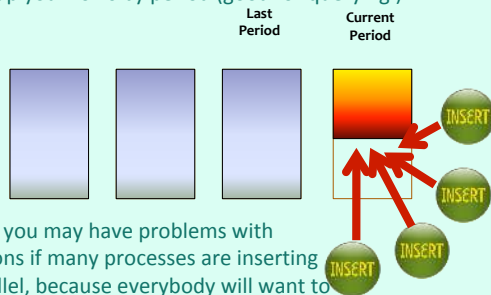
There are different ways to partition. When your concern is to control data grouping, you usually partition by range (so as to have rows grouped by month, week, or whatever interval), or by list (If the row contains this value in the partition key, then it goes into this partition). When your concern 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

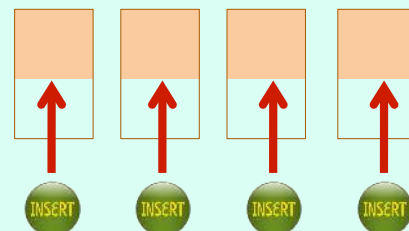
By hashing

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

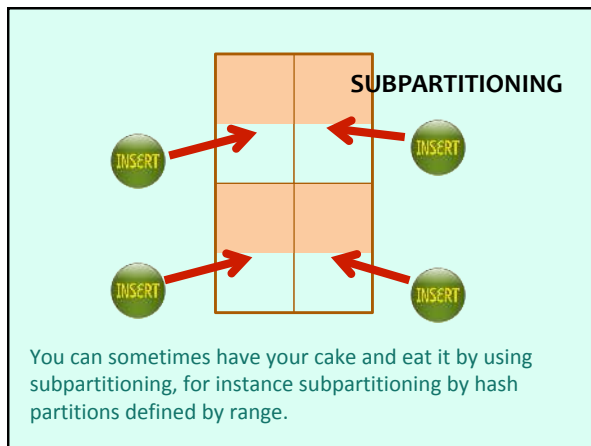


... then you may have problems with insertions if many processes are inserting in parallel, because everybody will want to insert at the same place and you'll need some serialization.

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.



Great for DBAs

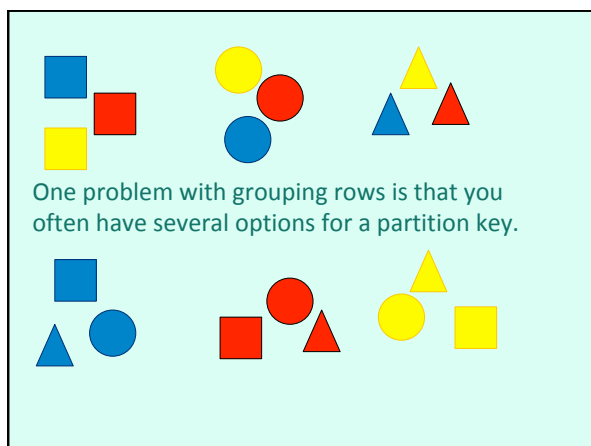
A product such as Oracle allows a lot of operations by just operating on metadata and not actually moving rows.

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.



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.

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 means 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