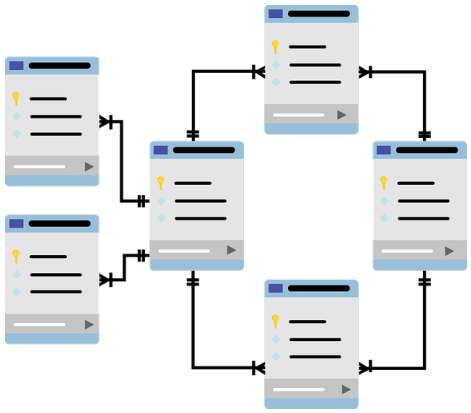
Data Architecture and Databases

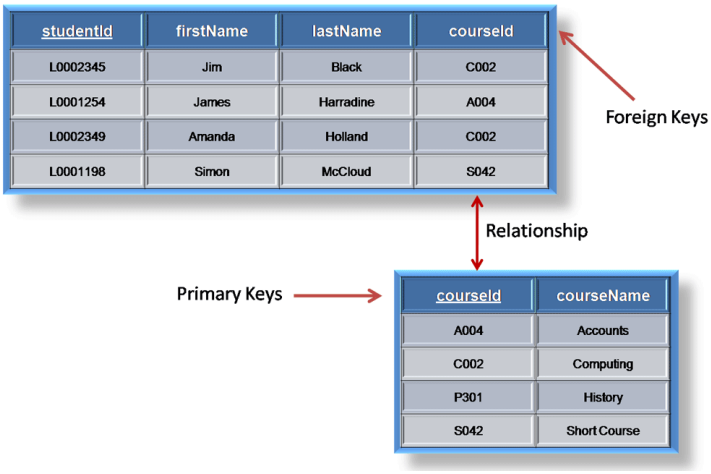
▼ Introduction to Database Systems

Relational Database (RDBMS)

> Digital database based on the relational model of data.
>
> A software system used to maintain relational databases is a **relational database management system (RDBMS)**.
>
> Many relational database systems have an option of using the **SQL (Structured Query Language)** for querying and maintaining the database.

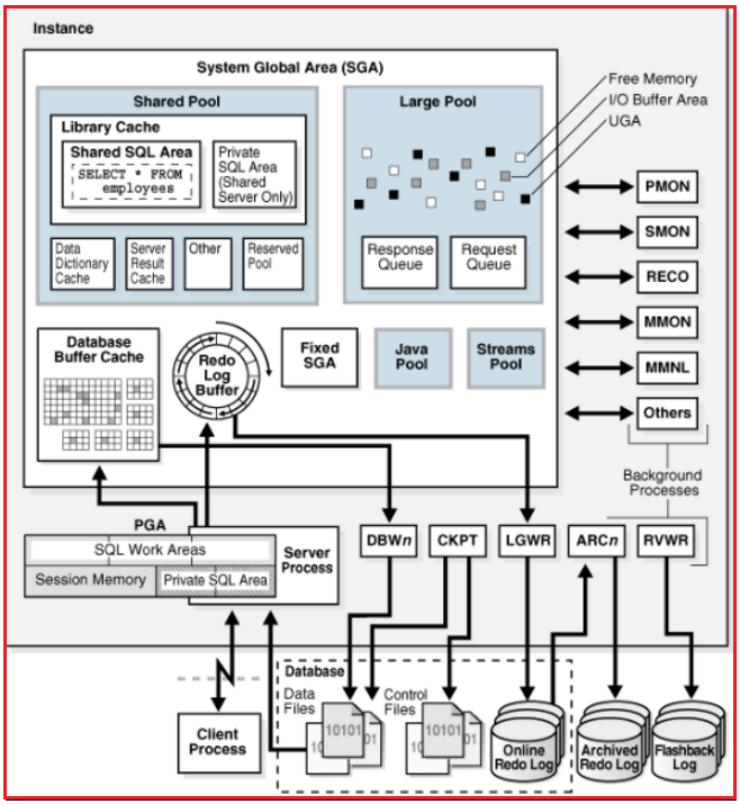
▼ **Graphic representation of Relational Database**





- **Primary key**: Guarantees there are no repetitions.
- **Secondary key**: Rest of the features.
- **Foreign key**: Makes reference to another table.

▼ **RDBMS simplified architecture**



▼ **Database vs Database Instance**

> **Database**
>
> data files + undo files + redo files + dictionary files + temporary files, **all on disk**

> **Database instance**
>
> background processes + memory structures. Many background processes are working together when running a relational database system.

▼ **Data access mechanism: Read, Write**

> ## Data Cache
>
> memory is used mainly to cache most accessed data

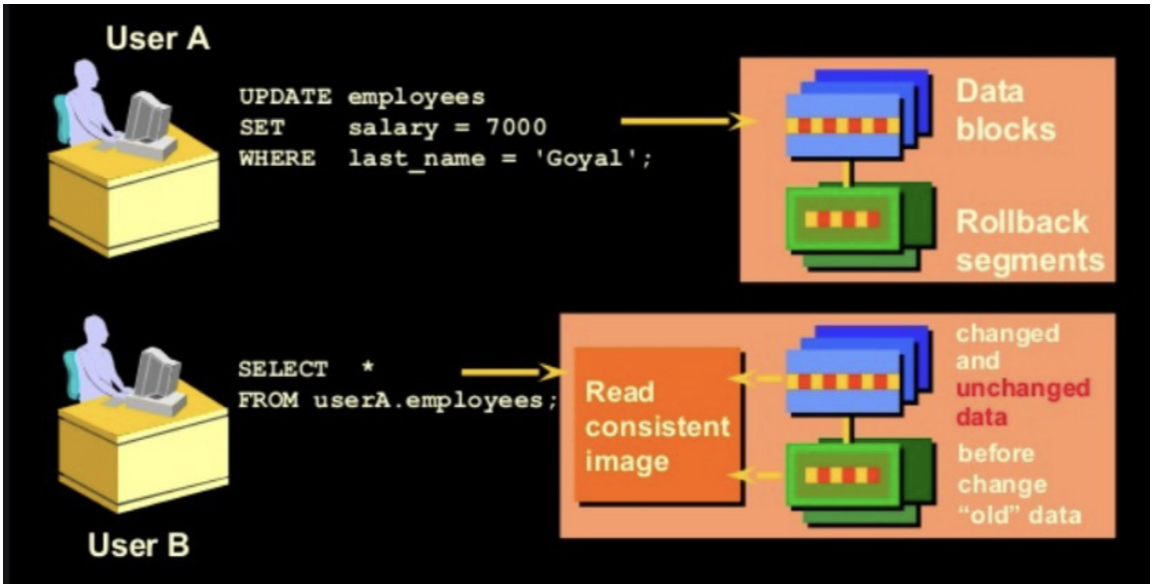
> **TEMP**
>
> temporary files used for temporary operations like sorting, hash joining, grouping.

> **Redo Structures**
>
> any data written to the database *datafiles* is written first to *REDO*. REDO will be used to recover the database whenever a crash occurs.

> **Undo**
>
> before changing any data in the database, the “before” value is written to the *UNDO* structure. This enhances both read consistency and rollback capabilities

▼ **Read Consistency**

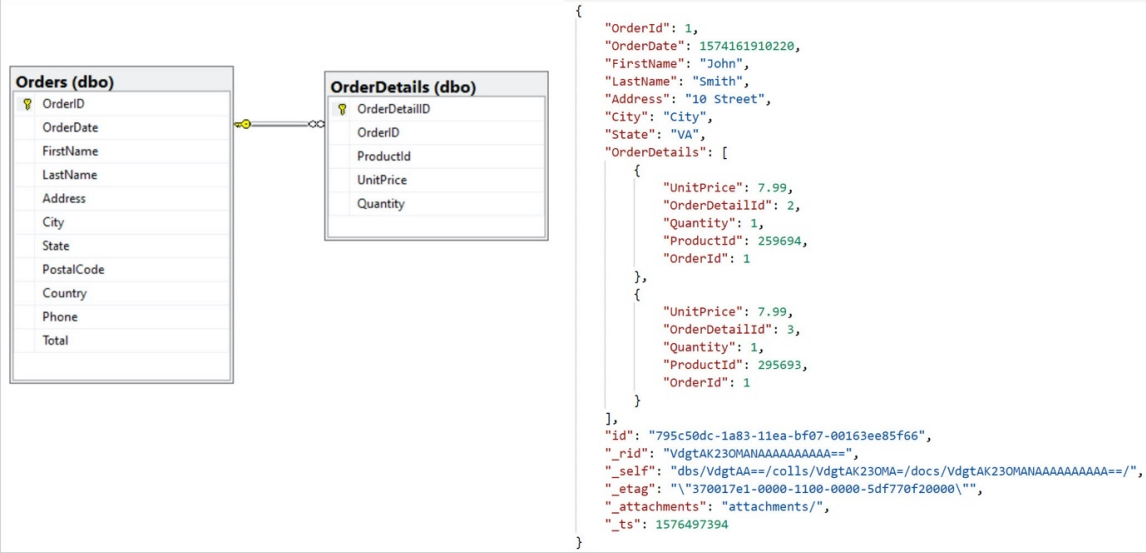
> 💡 To achieve **Read Consistency**, the RDBMS must provide the image of the data as it was when my query started.



NoSQL

> provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

▼ **Comparison Relational vs NoSQL**



> **Relational databases follow a *schema on write*:** Data structures and its relationships are well defined and known before writing on the database.
>
> - The main advantages of schema on write are precision and query speed

> **NoSQL databases follow a *schema on read:*** Application must be aware of the schema when reading the data.

▼ SQL Language

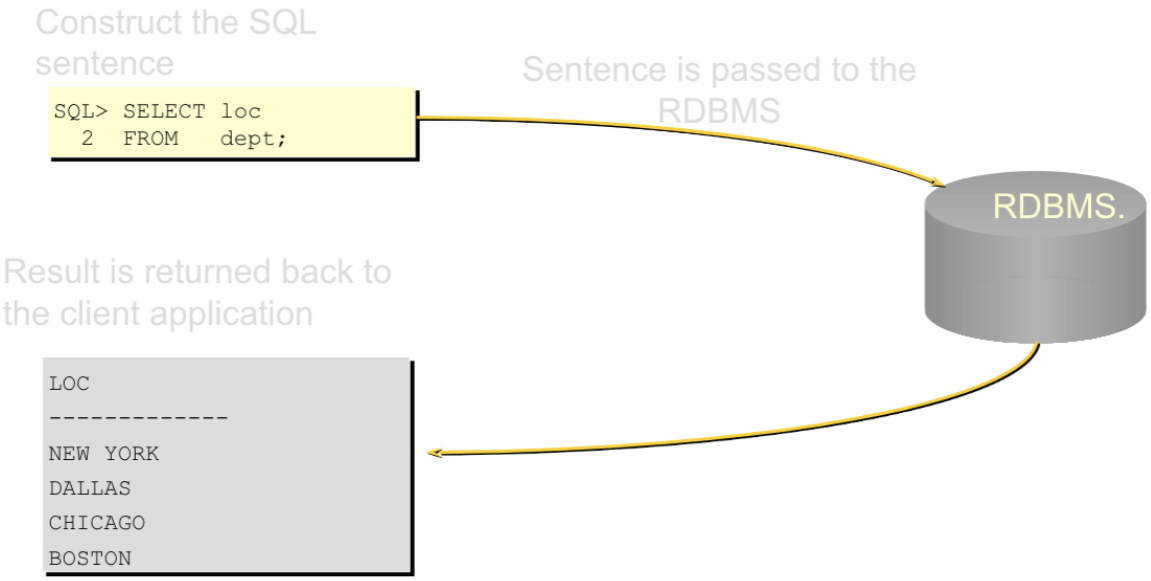
Structured Query Language (SQL) is a domain-specific language used in programming and designed for managing data held in a **relational database management system (RDBMS)**. It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL Command	Command Type	Purpose
SELECT	QUERY	Retrieve data
INSERT, UPDATE, DELETE	DML (Data Manipulation Language)	Create, modify or delete data
CREATE, ALTER, DROP, RENAME, TRUNCATE	DDL (data Definition Language)	Define, destroy, rename or modify structures
COMMIT, ROLLBACK, SAVEPOINT	Transaction control	Commit or rollback transactions
GRANT, REVOKE	Privileges control	Grant or revoke privileges

▼ Communication with RDBMS using SQL

A connection request is sent to the ***database listener***, using a connect string

- Database listener is a process running on the database machine, listening on a port(1521 by default for Oracle Databases, other port can be added) for incoming connections
- **Listener checks whether the request is correct** (host, port and service name)
- If the request is correct, a direct connection between the app and the database is started through a server process (process running on the database machine)



▼ Running SQL statement against the database

💡 Three steps in the database: ***parse***, ***execute*** and ***fetch*** (fetch only for SELECT statements returning rows to the app layer)

Parse: three sub-steps

1. **Syntactic analysis**: is the statement syntax correct ?
2. **Semantic analysis:** do the objects in the statement exist ? do you have the necessary privileges to access them ? Database dictionary is used during this phase.
3. **Execution plan construction**: find the best possible plan to solve the query.
    - Multiple plans are evaluated during parse, each plan is costed, and the Optimizer chooses the cheapest.
    - Optimizer highly relies on statistics collected on data structures (number of rows, distribution of the values in the

columns, ...).

- When a plan is agreed to be optimal, it is used to execute the SQL sentence.

4. **Parse result is cached in memory** for further executions of the same statement.

Execute:

Executes the statement using the calculated execution plan. Collects execution statistics and compare them with the estimation made during parse, to adjust the plan for the next executions.

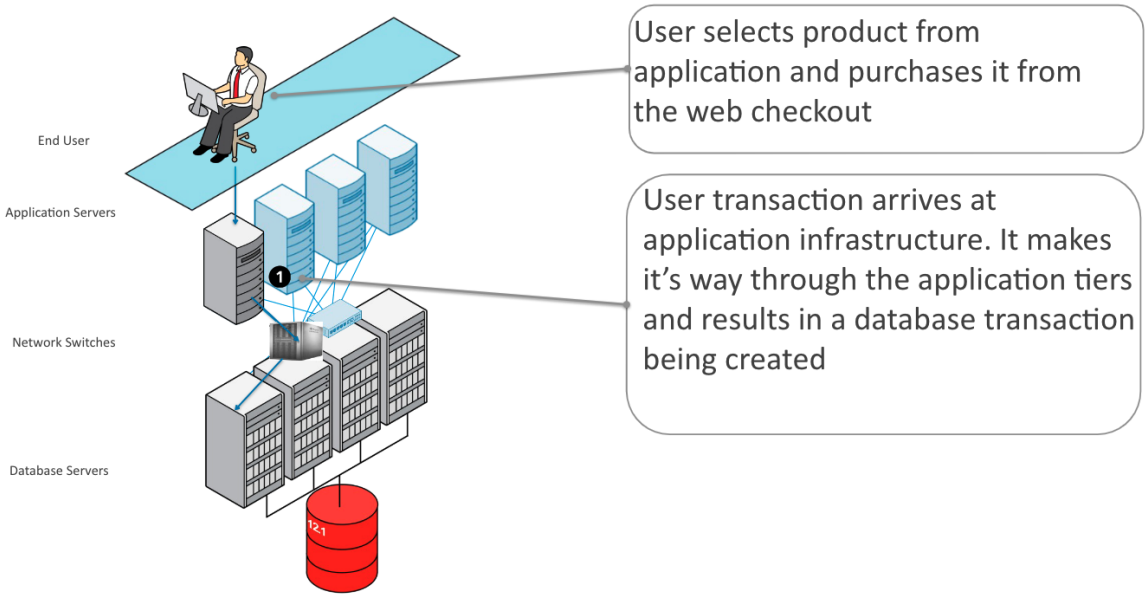
Fetch:

Fetches the rows into a result set, and return it to the application layer (**in the SELECT case only**).

▼ Common architectures

💡 **3-layer architecture**: end-user, application layer, database layer

- Application layer usually composed by several servers in cluster, to ensure high availability (HA)
- Database might be clustered as well to ensure HA



▼ Evolution of Data Management & JSON support in RDBMS

> **New tendencies eventually end integrated inside the RDBMS**: Object Oriented Models, XML, JSON, … until the next big thing.

> **JSON is very popular amongst developers** because of the simplicity of interaction.

> **JSON is fully supported in RDBMS**: store JSON documents into a table column. SQL has been enhanced with JSON specific constructs, to query JSON documents.

> **RDBMS can be queried using REST API**: URL are translated to SQL (simple cases) or mapped to SQL statements (more complex queries). This allows to have development teams specialized in SQL, that will provide only URL to non SQL specialized teams (**Abstract the application logic from SQL language**).

▼ Backup & Recovery

💡 **Backup & Recovery is the first step to MAA** (Maximum Availability Architecture). It’s required to get a protection against user errors and media failures.

▼ RTO and RPO

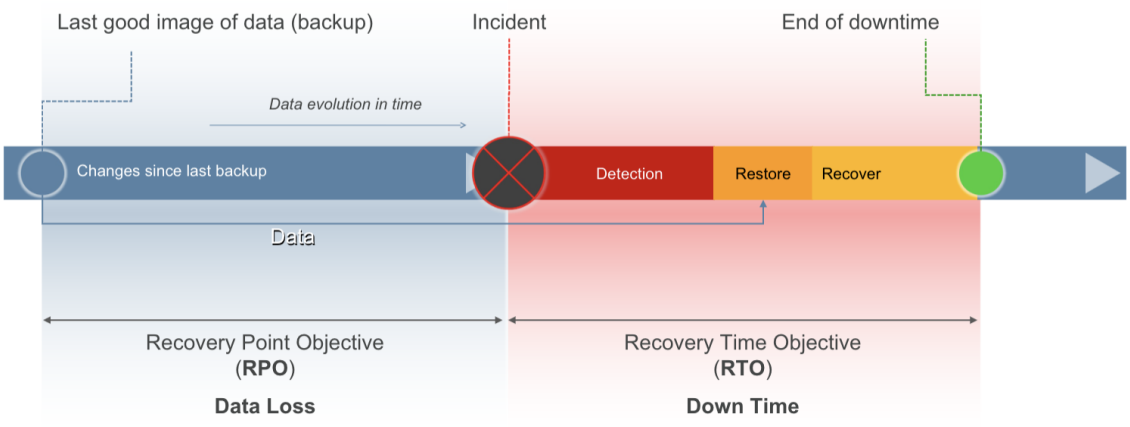
⏱️ **RTO = Recovery Time Objective**

The shorter the Recovery Time Objective (RTO) the quicker you get back to business

❤️‍🩹 **RPO = Recovery Point Objective.**

Tolerance for data loss (sec’s, hours, days); determines frequency of backups and replication approaches.

> RTO and RPO are technical concepts, but their value must be defined and constrained by **BUSINESS** requirements. (🔻 RTO or 🔻 RPO → 🔺 COST)



▼ Backup Concepts

Physical backups

Backups of the physical files used in storing and recovering your database, such as data files, control files, and archived redo logs.

Ultimately, every physical backup is a copy of files **storing database information to some other location**, whether on disk or some offline storage such as tape/cloud location.

These backups can be COLD or HOT.

COLD Backups

A copy is made when the database has been **cleanly shutdown**, hence the backup is **consistent** (doesn't need recovery after restoring). Implies a **business discontinuity**.

- RTO is typically the time needed to restore the file, from backup location to database location.
- RPO can be high, typically the time between the crash and the last available backup.

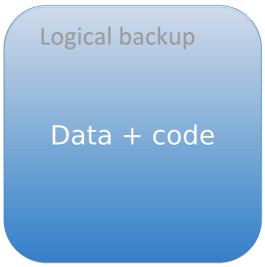
HOT Backups

A copy is made when the database is **up and running**, servicing client applications. Hence hot backup are always **inconsistent**, meaning that a recovery will be necessary after restoring.

- RTO is higher than with cold backup (need to recover)
- RPO can be minimized near zero, as we recover until the moment of the crash applying REDO from archived redo logs and UNDO.

Logical backups

Contains logical data (for example, tables or stored procedures) exported from a database and stored in a binary/flat file, for later re-importing into a database using the corresponding import utility. Is usually associated with huge values of RTO and RPO.



Errors and Failures Requiring Recovery from Backup

While there are several types of problem that can halt the normal operation of a database or affect database I/O operations, only two typically require DBA intervention and media recovery: **media failure**, and **user errors**.

▼ Backup Policies

💡 Apply **best practices to meet RTO and RPO**, depending of their expected values (business requirements).

Always test your policy, whatever simple it might be. Backup, restore and recover must be carefully tested, and RTO/RPO must be measured during the tests, to check they meet business requirements in any case.

A typical backup policy:

Hot full backup once a week (Sunday night), incremental backup and archived redo logs backup the rest of the days (by night).

With such a policy, if you have to recover from a complete database crash that occurred on Thursday, you will hopefully:

- Restore Sunday’s full backup

- Apply incremental backups until Wednesday
- Recover with archived redo logs until Thursday

▼ Restore vs. Recover

💡 **RESTORE:** Re-construct lost datafiles from a physical backup: datafiles might not be consistent with each other.

💡 **RECOVER:** After restoring lost datafiles, apply undo and redo until the whole database is in a consistent state.

▼ Performance Tuning

💡 ***DB Time*** is total time spent by the user processes either actively working or actively waiting in a database call.

***THE METRIC* we will use for performance tuning!**

Total time in database calls by foreground sessions:

$$DB_{time} = CPU_{time} + I/O_{time} + NonIdleWait_{time}$$

Average Active Sessions:

Number of active session, on average. An active session is a session consuming CPU or waiting for non idle wait events, like I/O. (DB time per second)

$$AAS = \frac{DB_{time}}{ElapsedTime}$$

Tuning:

Tuning database performance is all about reducing the DB Time measured during an interval of time, usually when the database is busy.

DB time distribution:

It will vary depending on the load profile of your database. Typically:

- OLTP database (short fast transactions), we expect the DB Time distribution to be around **90% CPU – 10% Wait (`CPU BOUND` = less than 20% idle Host CPU; The CPU capacity is considered the main bottleneck of the operations)**.
- On a Data Warehouse, we are more likely to see something like **40% CPU – 60% Wait**, as huge analytic queries usually wait on disk.

▼ **Performance tuning methodology**

The 7 Steps Iteration:

1. **Identify performance issue:** usually detected by end users, for example when performance SLAs are not met.
2. **Scope the issue:** problem might not be in the database. Review application servers, network possible issues, if the database machine is CPU bound, etc …
3. **Set goals**: goals must be defined by business
4. **Data capture:** provided out of the box by database instrumentation.
5. **Investigate DB Time distribution:** attack the component (Service Time or Wait Time) that offers the biggest potential improvement.
6. **Modify system to tune for largest gain:** one fix at a time.

7. **Evaluate against goals**: stop if goals are met, go to step 5 if not.

⚠️ When deciding where to focus performance tuning, focus on the **highest percentage of wait class** (not the weird ones even though they are abnormally high… fuck lab 4)

▼ Scalability

💡 **Scalability is a concern at any IT level**: hardware (including network), Operating systems, and applications. Developing scalable application is a **MUST**.

In the database world, scalability has to be tuned along with performance. It's about achieving the **best possible performance consuming as few resources as possible**.

An application can give good performance, but bad scalability: meaning that its performance will dramatically **drop under concurrency**.

It's about getting the best possible Response Time with the lowest DB Time!

Tuning a database system for **scalability is several orders of magnitude cheaper than scaling-up** the database server (cost of hardware, software licenses by cores, etc ...).

It might be even useless to scale-up the hardware. If the scalability problem is caused by a piece of software (application, OS, database RDBMS software), you won't see any benefit scaling the hardware up.

▼ Horizontal vs Vertical Scalability

Horizontal Scalability:

It's about adding **more servers**, working as a cluster: this is achieved by running a

Vertical scalability

It's about running your database on a

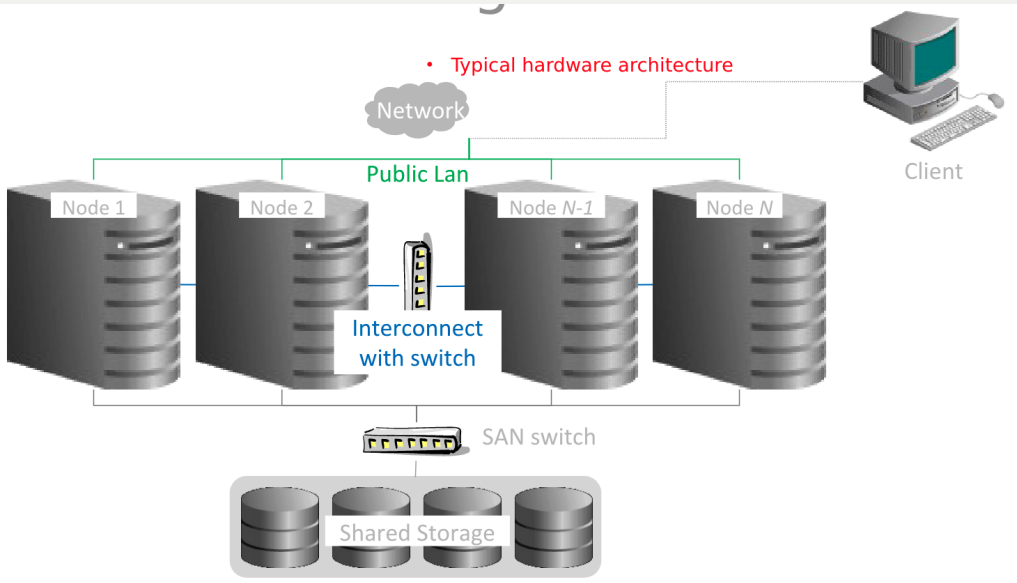
cluster database. Though not related to database, the "Seti at home" project was a great example of scaling up horizontally, with thousands of machines running in parallel.

bigger server (increasing the resources your server is using)

▼ Custer Databases

A cluster database is made of:

- **A single database**: datafiles reside on a shared storage
- **Several instances running that database**. Each instance runs on a single server, on which are running the background processes, and is allocated the memory for that instance.



A **cluster database software** ensures that instances are all synchronized with each other.

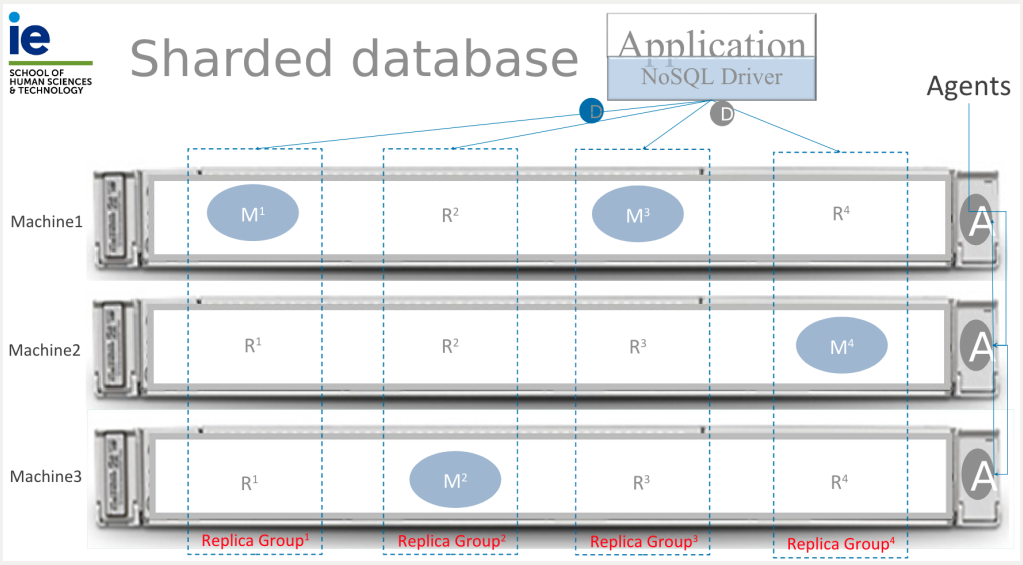
> The **workload** against the database **is balanced** on all the instances. This load balancing is done by a ***"less busy"*** mechanism, meaning that a new session will always be connected to the less busy instance (in terms of CPU consumption).

> Cluster database software must **avoid split brain**: split brain occurs when instances inside the cluster start to act as if they were alone in the cluster. This happens typically when the private connection is lost between instances, hence the cluster software is not able to synchronize them. To avoid split brain, the software cluster has to perform node eviction, removing one or more instances from the cluster configuration.

▼ Cluster DB VS Sharded DB

Sharded structures differs from cluster databases:

- *Shard* == **replication group**
- **Data is distributed evenly** among all the shards usually by a **hash algorithm**. Other distribution methods are also available, like list or range.
- Inside a shard, a Master does the write, and some replicas are receiving the data in an **asynchronous manner**. **Read operations can be performed against either the master or the replica**.
- Production Sharded databases should be deployed on a **number of servers equal to the number of shards**, as per best practices.



Component/feature	Cluster Database	Sharded Database
STORAGE	SHARED	REPLICATED
INCREASE SCALABILITY	LESS EASY	EASY
NUMBER OF INSTANCES	ONE PER NODE	REPLICA GROUPS
RELATIONAL DATA	YES	NO
FAULT TOLERANCE	MEDIUM	HIGH
DATA CONSISTENCY	FULL	EVENTUAL
DATA DURABILITY	COMMITTED ON DISK	CONFIGURABLE

Component/feature	Cluster Database	Sharded Database
BACKUP/RECOVERY	HOT (RPO=0)	SNAPSHOTS (RPO≠0)
PROTECTION AGAINST LOGICAL ERRORS	POSSIBLE	ALMOST IMPOSSIBLE
SPLIT BRAIN POSSIBLE	YES	NO
SINGLE READ PERFORMANCE	+++	+++++
SINGLE WRITE PERFORMANCE	+++	+++++
MULTIBLOCK READS/CROSS SHARDS READS	+++++	+

> Nowadays, relational databases can usually be sharded or clustered (these two options are not mutually exclusive!

▼ Disaster Recovery

> 💡 **A Disaster Recovery plan is ALWAYS needed**: Most of enterprises that suffer a disaster, not having implemented a disaster recovery plan, are doomed to disappear.
>
> The Disasters that are **most likely to occur are hardware faults, data corruptions, power outages**, etc …
>
> DR is **part of the Business Continuity plan**, that covers DR plans, business recovery plan and emergency response plans.

> When elaborating a DR plan, pay attention to:
>
> - **RTO/RPO**: these values have a direct impact on the costs of the DR plan, they should be carefully fixed in each case.
> - **Cost**
> - **Application Transparency**

- **Performance Impact**
- **Administrative Complexity**: This must be minimized, as it impacts costs as well, and as a high administrative complexity will make things difficult to test and implement.

▼ Technical Concerns

- Does this solution need separate integration with my applications?
- Will I be protected from data corruptions?
- Will my database at the DR data center be always kept consistent?
- How do I know that the data on the DR servers is always valid?
- How current is the data on the DR servers compared to production data?
- Can I leverage a secondary data center located 1000’s of miles away?
- Can I efficiently utilize my network bandwidth?
- How does synchronous replication affect my application response time?

▼ Business Concerns

- If a disaster were to occur, how soon can I get back to business (RTO)?
- Can I ensure that no data is lost whatsoever (RPO)?
- Can I make good use of the DR systems, or are they sitting idle most of the time – just accepting bits?
- How much does this solution cost?
- Will I have to pay more as I replicate more data (i.e. as my business grows)?
- Will this solution lock me into a single vendor’s storage system?
- Can I easily test our DR preparedness?

▼ Business Continuity

💼 **Business Continuity:** the umbrella program that encompasses DR plans, business recovery plan and emergency response plans.

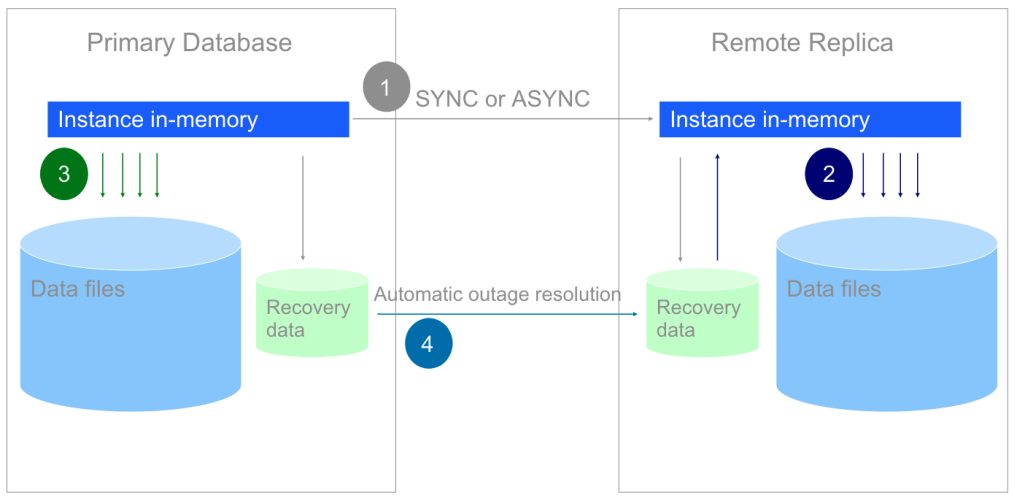
> **Business recovery Plan**: Departmental action plans for resuming critical business operations, generally at an alternate site, in the event of a disruption.

> **Disaster Recovery Plan**: The plan that deals with the recovery of a technology (systems and applications) in an agreed upon time frame, **driven by business requirements**.

> **Emergency Response Plan:** The site-specific plan or responding during the first minutes o hours of an event.

▼ Implementing a Standby Database

💡 Data changes are propagated from a primary to a standby database through *redologs*. Therefore the standby database is a **physical clone of the primary database**. The way we will propagate the changes will determine how synchronized will be the standby database at any moment.



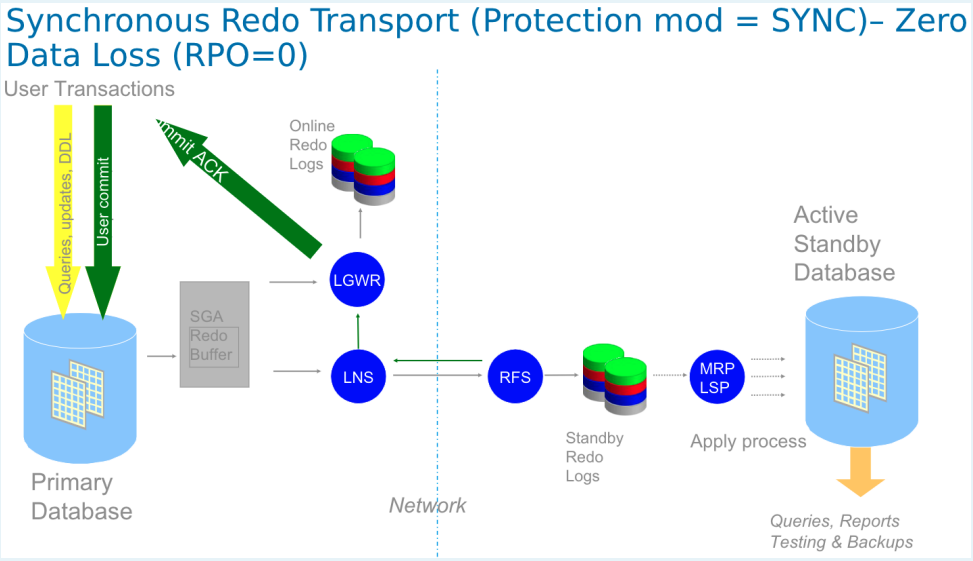
> To ensure a real DR plan, the standby database must be at a **different location** than the primary database, usually in a **second data center**. In Europe, companies usually use data centers in the same metropolitan area, sometimes in distinct metropolitan areas, distant of a few hundreds of kilometers.
>
> For companies that **cannot afford** a second Data Centers, a DR solution in the **Cloud might be a good solution**. It addresses the need of a second (or third) data center, without the huge infrastructure costs associated.

▼ 3 Protection Modes in this DR Architecture

SYNC

Data written on the primary database is <u>not committed until</u> it has been written to the remote (standby) *redologs*.

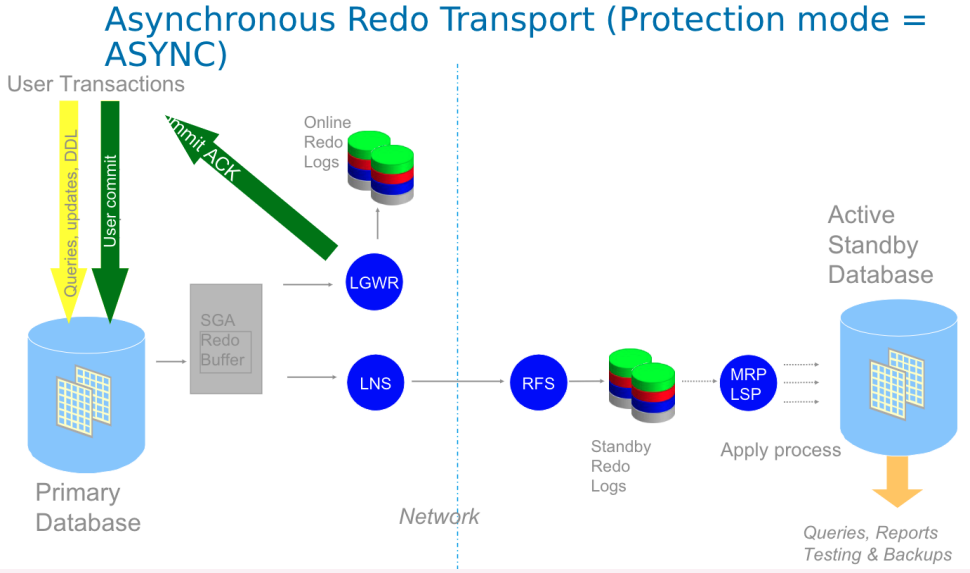
- This protection mode is the only one that **guarantees RPO=0**, as primary database will shutdown if it cannot propagate the changes to the standby site.
- This protection mode is very sensitive to network latency between primary and standby databases, hence can't be implemented if the two sites are not in the same metropolitan area (latency between the sites <= 5ms)



ASYNC

Data written on the primary is committed, and the data is propagated asynchronously to the standby *redologs*.

- If the changes can't be propagated for any reason, it will be re-intended later, and a lag appears between the primary and the standby database. This means that we **can't guarantee RPO=0**.
- This mode is not sensitive to network latency, but we must be aware that high latencies between the primary and the standby sites may cause a lag that will increase RPO.



HYBRID

SYNC by default, **ASYNC whenever SYNC is not possible** for any reason. If the protection mode has been degraded to ASYNC, the system tries to get back to SYNC mode.

PROTECTION MODE	RTO	RPO
SYNC	LESS THAN 1 min	0
ASYNC	5-10 min	SECONDS TO MINUTES

PROTECTION MODE	RTO	RPO
HYBRID	5-10 min	SECONDS TO MINUTES

▼ 3 possible actions DR+STANDBY

🔄 **SWITCHOVER**

This is a role permutation, where **primary and standby databases permute their respective roles**. It's an ordered maneuver, used for planned maintenance on the primary site. No data is lost during a switchover maneuver.

🚨 **FAILOVER**

Is the *emergency maneuver* done when primary database has been lost or is unusable. The **standby database becomes the new primary, and the former primary needs to be re-constructed**: that's the re-instate action, discussed below. Data might be lost during a failover operation, unless DR was configured in SYNC mode.

🚑 **RE-INSTATE**

This is the **reconstruction of the standby database from a primary database**. This must be done to re-create a standby database after a failover maneuver.

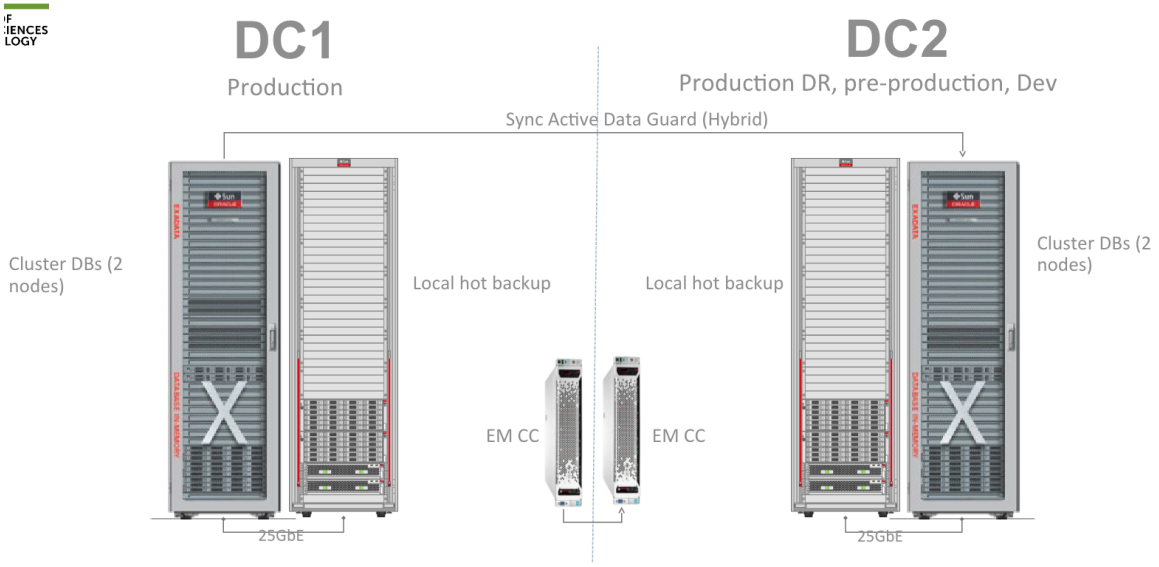
▼ DR - Best Practices

> **You need to think bigger than the database:** think about Business Continuity, and design a DR for your apps as well.

> **Design your DR strategy to meet RTO/RPO:** not all the databases need the same DR solution.

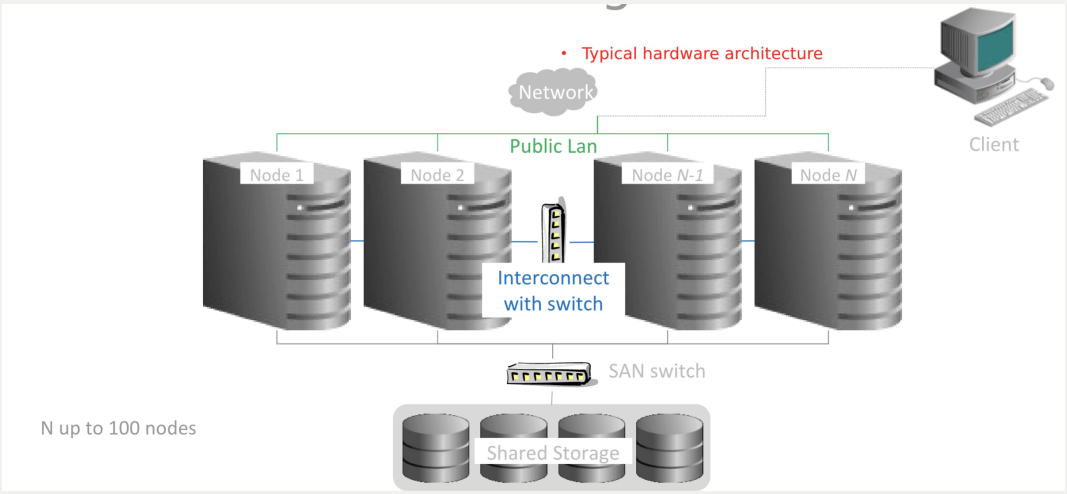
> **Automate as much as possible the maneuver steps**. Implement and test, ensure you'll meet RTO and RPO in any case.

> In production, plan and **execute switchover maneuvers** from time to time to ensure your Business Continuity Plan still works.



▼ High Availability (HA)

General Architecture - Cluster Databases:



▼ Application continuity

Unknown transactions state:

→ leads to poor customer experience

→ Database and infrastructure outages can cause in-flight work to be lost, leaving users and applications in-doubt

→ Often leads to:

- User pains
- Duplicate submissions
- Rebooting mid-tiers
- Developer pains

> **Current situation**
>
> 1. User selects product from application and purchases ir form the web checkout

2. User transaction arrives at application infrastructure. It makes its way through the application tiers and results in a database transaction being created
3. A failure in the infrastructure on its journey means the application server never receives an acknowledgement.
4. The transaction commits inside of the database and an acknowledgement begins it return journey
5. The application is left in an unknown state. It returns an error to the user who may order the product for a second time resulting in them potentially being charged twice.

A reliable replay of in flight work

1. User select product from application and purchases it from the web checkout
2. User transaction arrives at application infrastructure. It makes its way through the application tiers and results in a database transaction being created
3. The infrastructure hosting the database fails just before the transaction is committed to the database.
4. If the transaction need to be replayed. “Application Continuity” will submit all of the inflight work to a surviving node in the cluster and perform a commit. This all happens transparently to the application.
5. The jdbc driver detects the failure and checks with an available node in the cluster, using “Transaction Guard”, whether the transaction committed or needs to be replayed

> 6. The user receives confirmation that his order has been successfully completed.

▼ Avoid SPOF → Golden rule for high availability

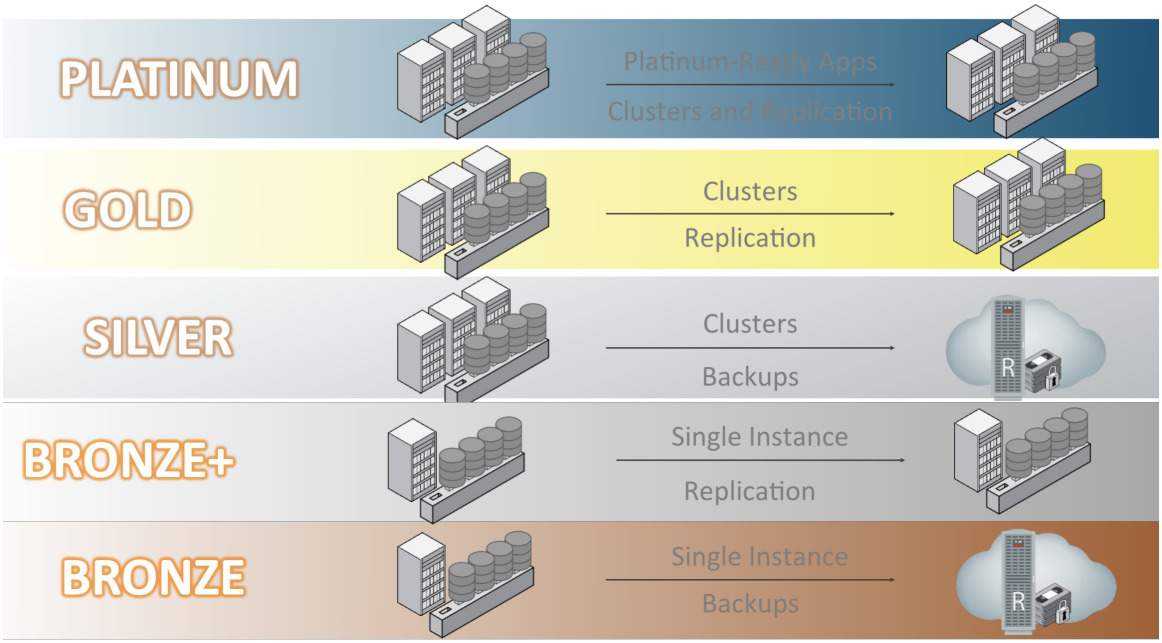
> 💡 **Track SPOF (Single Point Of Failure) in the whole architecture:**
>
> - At hardware level: Power supply, disks, network devices, etc.
> - At server level: use clusters for databases, application servers
> - At software level: use cluster software.

Track and avoid SPOF

> Channel bonding on network devices
>
> App servers in cluster
>
> Channel bonding on network devices
>
> Redundant switches
>
> Database servers in clusters
>
> Channel bonding on network devices
>
> Stripped and mirrored Disk arrays

▼ Maximum Availability Architecture

▼ Oracle MAA Availability tiers



▼ Oracle Cloud Topology and Terminology

> **Fault domains**
>
> - Isolated power and networks

> **Availability Domains**
>
> - Independent data centers

> **Regions**
>
> - Geographical separation

▼ Oracle MAA Availability tiers

	RTO (on prem)	RTO (Cloud)	RPO (on prem)	RPO (Cloud)
Platinum	0	0	0	0
Gold	0 for local failure (Minutes)	0 for local failure (Minutes)	0 for local failures (0 if SYNC)	0 for local failure (0 can choose SYNC)

	RTO (on prem)	RTO (Cloud)	RPO (on prem)	RPO (Cloud)
Silver	0 for local failure (Hours)	0 for local failure (Minutes to hours)	0 for local failure 0 (if hot backup)	0 for local failure 0 (if hot backup)
Bronce +	Minutes	Minutes	0 if SYNC	0 (we can choose SYNC)
Bronce	Hours	Minutes to hours (8 TB/hour)	0 if hot backup	0

▼ Conclusions: Cloud Pros and Cons

Pros:

- Costs: Capex, Opex
- Flexibility: no need to size based on Peaks. Size based on average and scale-up when needed.
- HA: SPOF avoided in the Cloud
- DR: SYNC between AD (Regional DR), ASYNC between region
- Always modern Hardware
- Possibility of hybrid DR
- Security by design

Cons:

- Regulatory constraints: sovereignty of data
- Vendor caging