E-Book

4 Ways to Improve Player Engagement with a Real-Time Database

The 4-Part Series in One Volume

redis

Contents

Introduction

Every video game has features and services designed to improve player engagement. After all, the more players are engaged, the higher their lifetime value. Higher engagement also has a snowball effect as players talk up the game, create buzz, and bring more players in, leading to better player retention, more MAUs, a stronger community, and increased revenue.

But what happens when those features and services aren't responsive enough to meet player expectations? Inventory lag at the wrong time can be the difference between victory and defeat. And there's only so long a player's going to wait for multiplayer or personalization options to load before they quit. If you want to give your players a truly engaging experience, then you need real-time responsiveness for your features. They also need to be implemented correctly and easily so you don't spend all of your time optimizing current features instead of launching new ones.

And that's where a real-time database comes in. The four examples in this e-book cover some of the most common player engagement features. With the right real-time database, you can get these running with real-time performance and you give players a seamless experience that keeps them engaged.

What does real-time mean?

Research into human response indicates applications have roughly 100 milliseconds (ms)—one third of the time it takes to blink—before users feel like they're waiting for a response. To be considered real-time, a feature or service needs to send a request, have it processed, receive the response, and return it in less than 100ms.

While responsiveness is critical in any application, gamers have some of the fastest reflexes around, making it extra important for your game to respond in real-time.

01 Tap Into the Competitive Spirit with Leaderboards

Leaderboards are an essential part of any game with multiple players—and have been since the early days of video games. Take Space Invaders. It holds legendary status for any number of reasons related to gameplay, but a big part of its popularity (and staying power) came from its introduction of competitive high scores to arcades back in 1978. Suddenly given a way to gauge their skill against others, players came back again and again to take their shot at the high score crown.

That core competitive concept is still at the heart of leaderboards today. They're an essential part of creating an engaging multiplayer experience that motivates players to see just how high they can get. Players stay engaged and active, which, in turn, means they contribute more revenue to your game.

But we've come a long way from a simple list of high scores in each arcade cabinet. Today's games have thousands of leaderboards for different events, regions, teams, and more. Each player profile on those leaderboards incorporates a host of different data types and variables. And, most importantly, players expect those leaderboards to be instantaneously updated every time they check their ranking—no matter where they're logging in from.

It's a lot of data. And you need to deliver all of it at the performance players expect without spending so much time on this one feature that you neglect the rest of the game. The only way you're going to do that is with a high-speed, real-time database, one that natively supports data models ideally suited for leaderboards.

Games (and players!) ask more of leaderboards than ever

Leaderboards have gotten more sophisticated

Leaderboards tap into that competitive spirit and incentivize people to try that much harder and play that extra bit more. But they can also be demoralizing. Sure, you might spend an extra 30 minutes (or make an in-game purchase) to jump from #23 to #19 and get that sweet Top 20 reward, but you might just as easily give up if you're #15,375,892 out of 20 million.

Effective gamification requires attainable goals and the ability to have a sense of achievement and accomplishment. If the competitive field is too big, there's a high potential for player apathy, which usually results in players deleting your game. But if you focus the field to a manageable level, players are more engaged and stay an active user that much longer. In the 80s, each high score list was unique to a specific arcade game, physically dividing leaderboards by location to create segmentation. While this narrowed the field, it also meant high score lists dominated by the local exceptional players, which made it easy for more casual players to walk away. Today, leaderboards are much more sophisticated—and personalized.

Modern multiplayer games create leaderboards for different leagues, player rankings, locations, factions, events, and more. A racing game, for example, might have a separate leaderboard for each player level range (1-5, 6-10, 11-15, etc.), then break that leaderboard down by track, then further break that leaderboard down by car. As a result, a player can see how they rank against players with a similar level of upgrades on the same track with the same base car model. Instead of competing against a field of millions (some of whom they never have a chance of beating), they're only competing against a couple hundred people—and on a more equal playing field. That level playing field makes them more engaged and keeps them playing.

But maintaining that level playing field can be more complicated than just personalization. Cheating is, sadly, a very real concern with multiplayer games, and letting cheaters dominate the rankings is a good way to lose players fast. To prevent this, leaderboards often have additional cheating analytics engines or checks built into their ranking systems. These help keep the game fair, but also increase the complexity of the leaderboard services.

All that sophistication and personalization requires complex architectures of services and technologies. And through them all runs a massive amount of data as leaderboard applications process millions of factors for millions of users. It creates a huge demand on the database (and the database team) behind the leaderboards, but personalization complexity isn't the only demand on leaderboards.

Players expect fast, up-to-date leaderboards tailored to them

Sophisticated, personalized leaderboards are now the norm. Over the last decade or so, they've become a crucial part of more and more games. As a result, players are so used to seeing leaderboards tailored to their playing experience that it's gone from a nice-to-have to expected behavior.

During that same time, speeds in every part of games have increased. Players expect all of their gaming experiences to feel instantaneous and to be constantly updated—and that includes leaderboards. The idea of waiting a full second to see your ranking is laughable, especially when leaderboards have time limits, like with tournaments. Players will be competing up to the last second, and they expect the leaderboard to respond accurately in real time so they know whether to make that last push.

The result? Players expect real-time responsiveness of massively complicated leaderboards applications. This is a tall enough order, but there's one more wrinkle: player location. Games have millions of users logging in from around the globe, and all of them expect real-time leaderboards no matter where they are. They don't care that the person ranked right above them is located thousands of miles away. If they're going to spend in-game resources (or purchase in-game currency) to fight one more battle, play one more puzzle, or run one more race before the timer runs out, they expect to see the exact amount of points they need to get the higher ranking.

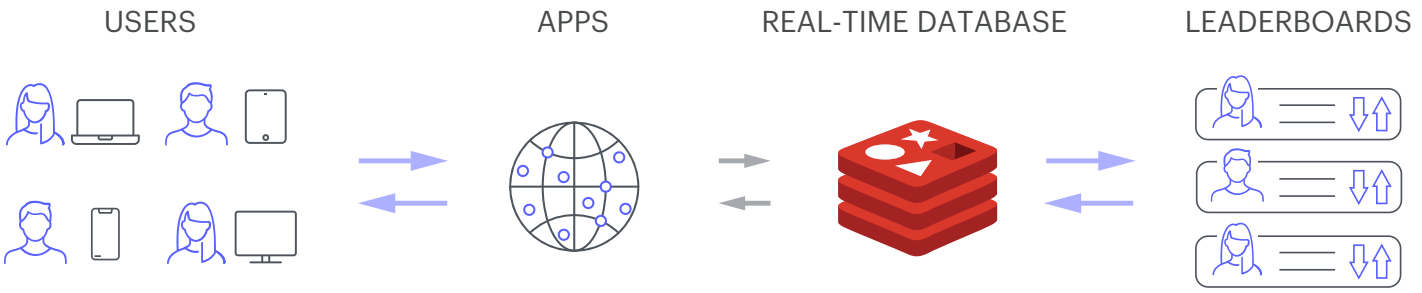
All this adds up to a pretty tall order for leaderboards. The application needs to be sophisticated enough for personalization, fast enough for real-time responsiveness, optimized for leaderboards data models, and distributed enough to be available everywhere your players are.

Data technology requirements to pull off real-time leaderboards

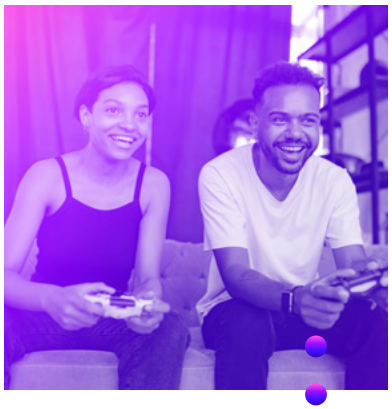
How do these demands translate into technical requirements for the database behind the leaderboard? After all, it's not enough to say the leaderboards have to be fast. If it's going to achieve real-time speed, a leaderboard service requires a database that meets these specific needs:

- **Ideal leaderboard data models** – One-size-fits-all data models can struggle to meet real-time demands, forcing database teams to spend a lot of time and effort optimizing the database. The right database should natively support data models ideally suited for real-time leaderboards.
- **High database concurrency** – Millions of scores, rankings, and records need to be updated and queried concurrently. The leaderboard databases have to be able to handle concurrent operations—and be able to intelligently handle concurrent write conflicts.
- **High throughput** – Read and write throughput must be high enough to handle millions of write and read requests. This throughput also needs to be able to respond to peaks caused by player surges, like during the last 5 minutes of a tournament.
- **Low latency** – A certain amount of time will always be taken up by data transfer and processing, which means your database needs to take up as little latency as possible for leaderboards to be served to players in real time.
- **Scalability with consistent performance** – Even if a database can deliver the data models, concurrency, throughput, and low latency required, it's not going to truly support leaderboards if it can't scale. The database needs to seamlessly scale up (and down) without any drop in performance.
- **Flexible deployability** – Your infrastructure is going to be determined by a large number of requirements—more than just your leaderboards. To integrate as seamlessly as possible, the database behind your leaderboards needs to be deployable in any environment.
- **Global synchronization** – Millions of players will access the same data across the globe, and they expect that data to be consistent. The local cached leaderboards need to seamlessly replicate and synchronize to provide an up-to-date global leaderboard with both local latency and consistent data.

Data is the heart of any leaderboard. Your leaderboards are only going to respond as fast as your data. If you want to meet the high expectations of your game and your players, then your leaderboard database needs to meet these technology requirements.



02 Build a Strong Community with Smart Matchmaking



Matchmaking is an essential part of building a strong, active community. It doesn't matter how good the gameplay and game design is. If you spend all your time staring at search screens while trying to find a game, waiting on matchmaking algorithms, or playing against wrong players, you never get to really experience the game.

Whether it's building teams, putting together battle royales, or bringing players together in social games, matchmaking services need to run their algorithms quickly and accurately. In fact, it should be so quick and accurate that it's a seamless experience for the players. This combination is what makes smart matchmaking stand out. Not only is it fast, but it makes matches based on fresh data that perfectly fit the player.

Delivering smart matchmaking is easier said than done, but with the right database, your algorithms can run at real-time speeds to give your players a low-latency, flawless matchmaking experience that keeps them coming back to play again and again.

Players expect real-time response—and accept nothing less

Unfortunately, players don't really notice when matchmaking works perfectly. It's rare for a 5-star game review to go on about how great the matchmaking algorithm is. But if something goes wrong, you're pretty much guaranteed to hear about it. And see its results in your key game metrics like DAUs and MAUs. After all, there's only so long players will wait in the lobby for the matchmaking service to run or stare at the spinning wheel in a search bar before they jump to another game. And speed by itself isn't enough. Even if the matchmaking runs quickly, it has to be accurate. If it matches them with opponents or teammates that are too strong (or too weak), then the game isn't fun. Once again, they'll jump, and probably leave damaging reviews in the process.

In games with a PvP function, matchmaking can make or break the player experience. Whether it's team wars, battle royales, races, 1:1 duels, or social games, PvP needs to find players who are at similar power rankings and currently available to play (or similar activity levels if the PvP is asynchronous). Then they need to get rapidly sorted and slotted into matches based on these parameters. A good PvP experience quickly matches players against others who offer a challenge, but can be bested in a contest of skills or strategy. A poor experience takes a long time to load, then matches them against opponents who are too weak (removing the challenge) or too strong (removing the ability to win).

Matchmaking is also used for team recruitment. It could be called a clan, squad, house, country, or a hundred other names, it's all the same core idea: Players form up into teams of similar power and ability levels. Sometimes they're competing on leaderboards, sometimes against the computer, and sometimes against each other. In every case, the player is usually given a range of teams to choose from. They expect the team to include people with similar play styles, levels, time zones, and more. And they expect to be able to make a good team selection quickly. After all, they signed up to play, not spend all their time selecting a team.

Similar is a range, not an absolute.

Matchmaking algorithms can look at more than just power levels. Players that always lose can get discouraged (and players that always win can get bored), so keeping players engaged means finding a balance of challenge. This means looking at variables beyond power levels, such as the number of recent wins/losses. While players may not be aware of these additional variables, it does increase the amount of computations that have to fit within their low-latency expectations.

Player selection also adds additional frontend expectations to matchmaking. Players don't want to just be shuttled into whatever game is handy. They want to be able to search for just the right match. Adding search to matchmaking is a great way to increase engagement, but it also introduces another place where slow or faulty matchmaking could detract from the player experience. When they search for opponents (or allies), players want to instantly see an array of good matches to choose from.

No matter what kind of game, all matchmaking use cases have the same player expectation of seamless execution. Players don't mind waiting for other people (such as waiting in the lobby area for battle royale games while enough players join), but they don't want to wait for the game itself to load. In fact, players expect the matchmaking to be so seamless that it's almost invisible.

This is where real-time matchmaking comes in. Real-time matchmaking finds the right connections between the data, quickly gives players an array of choices, backfills the players to the right server spots based on their choice, and runs through the matchmaking queue so fast that players don't even notice.

Modern matchmaking is more complex than ever

Early online matchmaking was much simpler. Are people online? Great, put them on the same server, spit out a list of all available games for them to wade through and let them sort it out themselves. But this approach left way too much of the player experience up to chance—with potentially disastrous consequences. Modern matchmaking is sophisticated and complex, going that extra mile to make sure that as many players as possible have engaging experiences that make the game enjoyable.

To pull off that degree of personalized sophistication, matchmaking services have to process a massive amount of data. Player compatibility can be determined by variables like player rank, current inventory, location, recent win/loss ratio, and more. For potentially millions of players. It's a lot of data to run through the queue, sort, and then backfill into servers.

It gets further complicated when you add player choice into the mix. Take an online racing game. Players choose a racer, car, then track. These choices will have to also be included in the matchmaking, and processed fast enough to not slow down the players' experience. After all, how frustrating would it be if you choose your favorite racer, the perfect car, and the exact track you want, then lose your spot because matchmaking took too long to process your request? Similar player-led variables pop up in a wide variety of use cases, such as game modes for first-person shooters (like Capture the Flag or Free for All), player character types (for games that only allow 1 of each type), a selection of game maps, or tourneys where victory is a condition for advancement.

Showing players those choices presents its own technological demands. When a player goes to select a game, they're performing a specialized kind of search, even if it doesn't look like it on their end. But it's not enough for the search to just return an unsorted list of results. The results need to weigh all the factors above, then be sorted based on the best possibility of matches. For example, a player searching for a trivia game to join will need to be served options that best match their social graph, then sorted by things like number of available spots or time until game begins.

Incorporating all of these variables creates a richer, more personalized experience, one which can dramatically improve the player experience and increase engagement. But it comes with added complexity that has the potential to decrease performance. Since decreased performance can have a pronounced negative effect on player engagement, this has the potential to erase any gains from personalization.

Fortunately, there are many design and architecture elements to optimize a matchmaking service for this increased complexity. For example, storing data using a graph data model could make it simpler and faster to find relationships between records than it would be with a relational database. Streamlined search functionalities also help increase responsiveness. Instead of having to run results through a separate search database, integrated capabilities directly interact with the same core database.

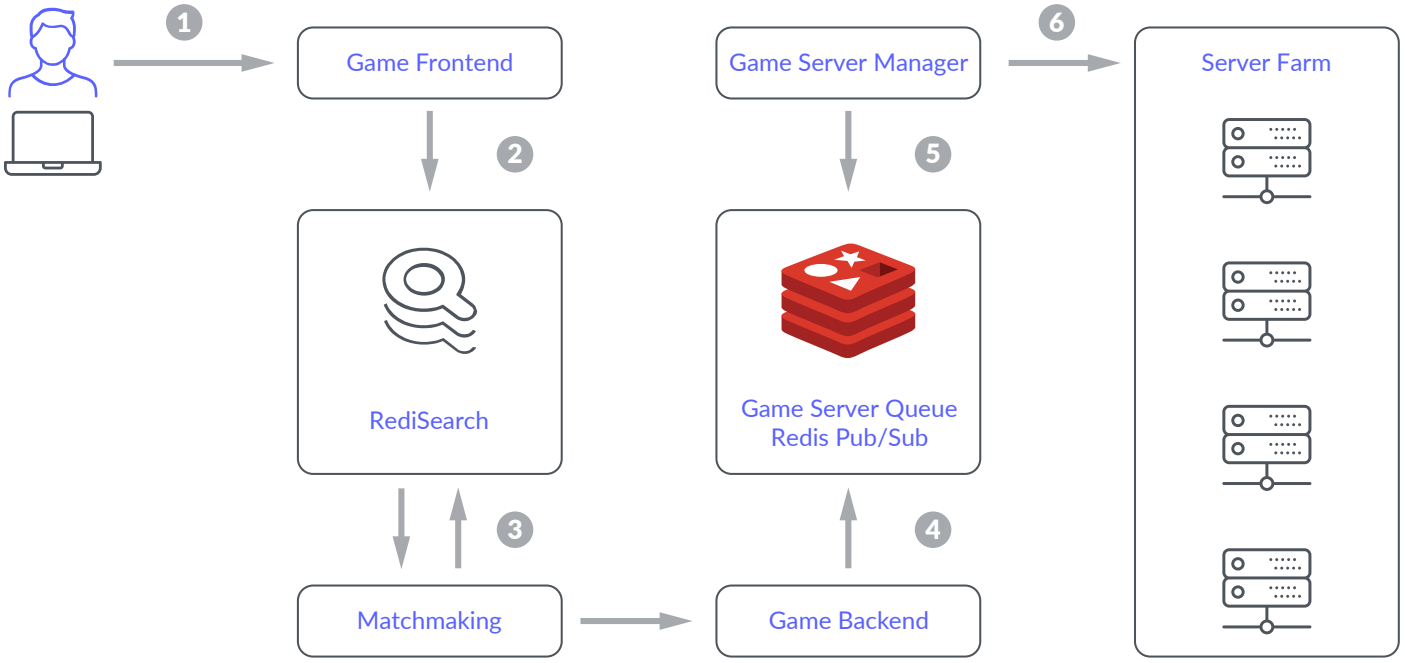
So yes, while the expectations from players demand faster and more complex matchmaking services than ever before, it's also possible to build services that complete the request within real-time speeds. But doing so creates specific technical requirements for the database and architecture behind the matchmaking service.

Technical requirements for a smart matchmaking database

Above, we defined real-time response as less than 100ms for receiving, processing, and returning the request. If a database is going to power the above complexity and have the latency for that 100ms speed, it needs to meet some very exacting technical requirements:

- **High database concurrency** – With potentially millions of match requests all hitting at the same time, the database needs to have the ability to handle concurrent operations without slowing down performance.
- **Low latency** – As a general rule of thumb when running applications in production, every millisecond spent in the database means 100 milliseconds felt at the application level, which means the database needs to be capable of <1ms latency to hit the 100ms goal.
- **Scalability with consistent performance** – The volume of matchmaking requests can surge or drop in massive spikes. The database needs to scale and deliver the same performance whether it's the top of the spike or the bottom of a trough.
- **Matchmaking data model compatibility** – Data models like graphs can drastically increase the speed of matchmaking, which means any real-time matchmaking database needs to include support and compatibility for those data models.
- **Integrated capabilities** – By integrating search or other capabilities into the database, you cut down on the number of transitions between services, which helps the matchmaking service run faster and the frontend experience more seamless.
- **Flexible deployability** – The matchmaking services, and its database, needs to be able to deploy wherever the games are run, whether it's in the cloud, on-prem, or in a hybrid cloud environment.
- **High global availability** – It's not enough for the matchmaking service to run fast. It needs to be reliably available whenever (and wherever) your players are online, which means your database also needs to have high availability while running on a global scale.

There's a lot that needs to happen between a matchmaking request and slotting the player into a specific game on a specific server. And it can only happen as fast and as accurately as your database can reliably respond to requests.



03 Give Players What They Need with Inventories

Real-time inventory response is a crucial component of gaming. Any lags or freezes will lead to a disconnect between the player and the game, creating friction between the two. This is even more relevant given that today's games have evolved to become tremendously detailed, with large numbers of items and recipes in a player's inventory.

And all of those need to be accessed instantaneously to provide a seamless playing experience, which means processing millions of queries simultaneously. After all, having instant access to specific items in gameplay can be the difference between winning and losing.

To compound this even further, we live in an era where we expect responses to be delivered at lightning speed, or to be more precise, in real time. Any failure to satisfy these expectations frustrates players and hampers engagement, pushing them to games that can meet these demands.

This is especially relevant given the prominence of online gaming, where databases must provide local latency everywhere to avoid any lags in inventory lookups. Achieving this is by no means an easy feat, but with the right database, you can provide real-time inventory responses with local latency that are both reliable and scalable.

Expectations are higher than ever for game inventory responses

Inventories have become a crucial part of modern games. As gaming has progressed through the years, players expect games to have the capacity to accommodate a great number of items in their inventories and catalogs.

Gameplay mechanics can be enhanced by providing infinite inventory, but these come at a cost: lag times that get worse as the inventory builds up. Even with finite inventory games, there are different types of inventory to be accessed quickly. That's a lot of data to be processed in real time. Yet, despite this, meeting these demands has become fundamental to maximizing the playing experience.

Currency inventories

Currencies are the lifeblood of any in-game economy. They're used to make important transactions for items of value that pull players deeper into the game. Gold coins, purple diamonds, red orbs, you name it—if you want to gloss up that car with your favorite vinyl, you're going to have to fork out.

Just like in the real world, there's a desire for ownership of belongings. Players want to get their hands on the best inventory to sharpen their competitive prowess or even just to enjoy the thrills of character development.

The more inventory there is, the more there is to chase, hence games are now stacked full of inventories. To obtain these items, players need to generate currency. There are two types of currency in gaming:

- **Soft** – Earned via game play
- **Hard** – Earned by using physical money to buy virtual currency

Soft currency can be obtained by either playing the game or just waiting. Examples of this would be in Dynasty Warriors (players are rewarded with XP for completing battles), or coins in Idle Miner Tycoon (which are automatically given over time).

And then you have hard currency, which involves players spending real money to generate more virtual currency. Often this provides instant access to inventories as well as access to premium content (think of gems on Brawl Stars and gold bars in Candy Crush).

Having to accommodate both these types of currency proliferates the amount of inventories games have to process in real time. This is intensified even further when you take into account the number of players in a game and their currency requirements.

Player and character inventories

Character inventories refer to all of the items a character owns. In some games, players have multiple characters to build into teams. And as you'd expect, each of those characters will have an inventory of equipment, power ups, or anything else that makes up that inventory system.

But players also have their own inventory that can be distributed between their characters, creating a separate inventory system between the two. For example, in Fortnite a player can have an inventory of skins that they can equip their characters with. What's more is that an inventory can hold thousands of separate items, each with their own item count and current state.

When you add each player and their characters' inventories into the equation, databases end up with a colossal amount of data to process and update in real time. But here's another hurdle: characters are constantly shopping and changing their inventory as they progress deeper into the game.

Some games have a limited inventory system where characters are only allowed to carry a certain amount of inventory, forcing them to prioritize between items. To be able to juggle between different items and exchange old for new, the session stores behind the inventories need to respond in real time to create a seamless experience.

This is especially important for games that have extensive inventory systems. Escape from Tarkov, for example, allows players to store up to nearly 2,000 items in their inventory. The challenge this brings to the table is accessibility.

In many games, inventories are so vast that players have to dedicate time to sorting their catalogs. This is especially relevant for games similar to Escape from Tarkov, where their granular inventory systems make sorting almost mandatory.

And from a data perspective, these demands are astronomical. Games have millions of inventories and players are interacting with them all the time during gameplay, expecting responses to be in real time to meet their expectations.

Weapon inventories

First-person shooter (FPS) games like Call of Duty (COD) are fast-paced in nature and demand players react quickly and precisely to on-screen events. To put everything into context, the average FPS player has a reaction time of between 300 to 500 milliseconds, whereas for professional FPS players it's anywhere in the 100 to 250 millisecond range.

Now, remember our definition of real time? If a player needs to juggle between different inventory items, such as a pistol or an Mk47, then the whole process from request to processing to response to serving the players has to be completed within 100ms.

A slight delay in these data processing response times can be the difference between winning and losing, potentially robbing players of victory while also damaging the playing experience. And it won't take many losses due to lags before players start complaining, which can really hurt a game's community.

Across many games, being able to access and use inventory items at hyperspeed is a core aspect of gaming, making it almost a skill in itself. Real-time deficiencies in inventory responses will only frustrate players. In these scenarios, their inability to win might stem more from the deficiencies of the database rather than a lack of skill.

This creates a frustrating situation for the players due to being unable to carry out inventory commands in real time, hindering their ability to compete. Players have a low tolerance for jittery gameplay and so they'll simply jump to another game if frustrated.

Successful games require immersive and engaging experiences for players around the globe. Latency is the new outage and every millisecond counts. Success comes to those who offer performance and experience to encourage deeper, longer playing time.

The data technology requirements to pull off real-time game inventory responses

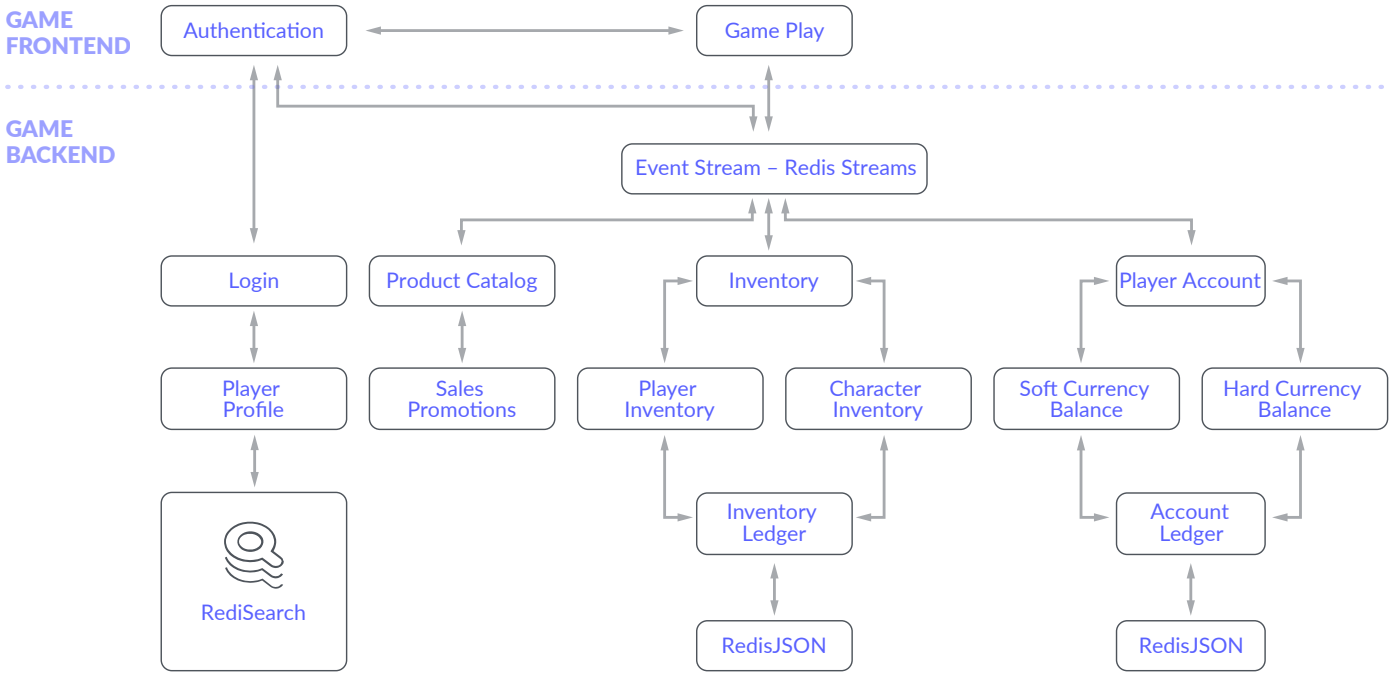
Achieving real-time game inventory response can be complicated. There could be millions of players who'll be requesting their unique account balances or querying for their unique player inventory. In such a case, developers need to integrate search into their database to cut down the number of transitions between services. Doing so will diminish the inventory response time as well as make the frontend more seamless.

Yet despite this, other factors come into play to generate inventory responses in real time. Data models must be compatible enough to provide tailored experiences for individual profiles through stored session states with unique IDs. Having numerous models of high availability and geographic distribution will give players local latencies.

Sorting and managing large streams of data is equally important. When events are created during gameplay or authentication, these events need to be allocated to the places where they'll be processed correctly.

To provide real-time inventory responses, a database needs to meet these technical requirements:

- **High database concurrency** – Databases will have potentially millions of players requesting their unique balances or flipping through inventories. Requests need to be processed simultaneously without any lags in performance.
- **Low latency** – Game inventory commands must be sent, processed, and then received in less than 100ms, requiring the database to be able to achieve <1ms latency to reach the 100ms goal.
- **Scalability with consistent performance** – The amount of inventory response requests can rise and drop in sudden surges. The database must be able to scale and deliver the same performance whether it's at the peak of the surge or the bottom of a trough.
- **Inventory response data model compatibility** – The right data models can bolster the speed of inventory responses, requiring any real-time inventory response database to include support and compatibility for those data models.
- **Integrated capabilities** – Incorporating search or other capabilities into the database will reduce the number of transitions between services, enabling the inventory response service to run faster as well as making the frontend experience more seamless.
- **Flexible deployability** – The inventory database needs to be deployed wherever the games are run, regardless of whether that's in the cloud, on-premises, or in a hybrid cloud environment.
- **Geo-distribution** – Reliability is just as crucial as speed. Millions of players across the globe will access their inventory and they expect to do so in real time, irrespective of their location.


GAME FRONTEND
GAME BACKEND
Authentication
Game Play
Event Stream – Redis Streams
Login
Product Catalog
Inventory
Player Account
Player Profile
Sales Promotions
Player Inventory
Character Inventory
Soft Currency Balance
Hard Currency Balance
RediSearch
Inventory Ledger
Account Ledger
RedisJSON
RedisJSON

04 Mold the Game to Players with Personalization

We all love personalization. Even if it's just a smiley face the barista draws on the cup at our local coffee shop, it's the personal touches that make something truly special. The same is true for your video game players. Every player out there is looking for a great game, one they can sink themselves into and truly make theirs. It's the holy grail for game developers, one where you have a loyal, actively engaged player base with high retention.

Unfortunately, players are also extremely picky about which games are worth their time, which makes the bar you have to clear to get their loyalty and hit those crucial 7-day and 30-day metrics even higher. That's where personalization comes in.

With personalization, you can customize the game experience to give players that personal touch. It includes a wide range of activities, covering everything from ad serving to gameplay customization to adaptive AI to character/avatar skins—even community interactions. In short, almost every part of your game is a candidate for personalization.

Creating that personalization is a huge lift requiring a robust, complex backend data structure. It's also necessary in the modern environment. Competing games are working just as hard to add exceptional personalization, and even the smallest hiccup can be enough to send players to another game within those crucial first few weeks.

In short, your personalization has to be complex enough to tailor the game perfectly to your players, while the database has to have the performance to make the personalization seamless.

It sounds impossible, but when it's powered by the right database, real-time personalization can be a reality for all of your players—no matter where they log in from.

How modern personalization improves the player experience

Personalization comes in thousands of different flavors. While it's not a fully inclusive categorization, for the purpose of data use they can be sorted into player-led personalization and game-led personalization.

Player-led personalization

Players love having the option to personalize their gaming experience and customize their gameplay. And when they do, they expect games to respond seamlessly, without any lag, loading delays, or dropped responses. Any hiccup or noticeable latency is a potential source of frustration. The system behind these personalizations has to have real-time responsiveness, with each personalization having its own specific use case demands:

- **Avatar/character personalization** – Whether it's tailoring every aspect of a character's appearance or showing off the latest skin, players love the creation (and bragging rights) of a personalized appearance. But this can also mean rapidly cycling through hundreds, even thousands, of options as players search for just the right look. And for a real-time experience, every last one needs to be pulled from the database and rendered in less than 100ms.
- **Gameplay customization** – Whether it's switching out the tires on a car, the rifle you use in a FPS, or a puzzle powerup, games thrive on giving players ways to customize their gameplay. Unlike character personalization, customization has a direct impact on gameplay, such as making cars faster, characters stronger, or puzzles easier. Many customizations even have mods or customizations of their own, allowing players millions of combinations to try out. And you can be sure the players will definitely try them, requiring systems to rapidly load, render graphics, modify stats, and then log the new stats for gameplay.
- **Interactive leaderboards** – Modern games can easily contain dozens of leaderboards for each player. Racing games might have different leaderboards for each track, fantasy RPGs for arena battles, or puzzle games for each level. Not only are these parsed by variables like leagues, server location, guilds, and more, but many of them have interactive elements that require rapid sorting, search, and so on. Players don't care that the leaderboard can have thousands of entries with multiple data dimensions—they still expect them to load and respond to commands instantly.
- **Smart matchmaking** – Matchmaking can make or break multiplayer games. When done right, smart matchmaking responds in real time to perfectly match players together, whether it's in battle royales, 1-on-1 matches, or when making teams. Matchmaking is further personalized by giving players choices of opponents, races, modes, and more. And no matter how many times the player hits that "refresh options" button, they still expect the choices to load seamlessly.

These are only a few of the many ways you can personalize your game with player-led interactions. But throughout all of them, there's a single thread: Players don't just demand real-time responsiveness, they expect it. And you have to deliver that responsiveness if you want to keep them engaged and in the game.

Game-led algorithmic personalization

Game-led personalization takes a more proactive approach to improving the player experience. Instead of providing options for players to respond to, game-led personalization analyzes and learns from the player to create a more personalized experience. Rather than seeing the personalization actions, such as when modifying gear and seeing the stats change, players instead see the results of the personalization. Personalizations like these add an extra layer of services, but the end result can have a much more profound effect on the player experience:

- **Adaptive gameplay** – By using machine learning to create player models, games can personalize their gameplay to fit individual players. This includes anything from adjusting the difficulty of the AI to serving different puzzles types. When done right, players might not even notice the difference. All they notice is that they're engaged and having fun. But doing it right requires the performance and data models to seamlessly analyze player behavior and respond in real time.
- **Offer serving** – The right offer at the right time not only makes the player more likely to make a purchase, it can also make the player feel like the purchase has more of an impact. For example, a special discount after a player has failed a level multiple times can turn around a negative experience. Losing is frustrating, and odds are good the player might have been about to quit. But when you make the right offer, they'll make the purchase, then feel triumphant when they pass the level. Whether the offer uses real currency or in-game currency, finding that combination of the right offer at the right time requires a flexible, high-performance database.
- **UI personalization** – Services like searchable/sortable inventories, notifications, or community chats improve the player experience. Each service also builds off personalized player profiles. However, any player engagement gain from these services can be lost if there's too much of a lag or the services are inconsistent. You'll need real-time responsiveness to reap the benefits of these personalizations.

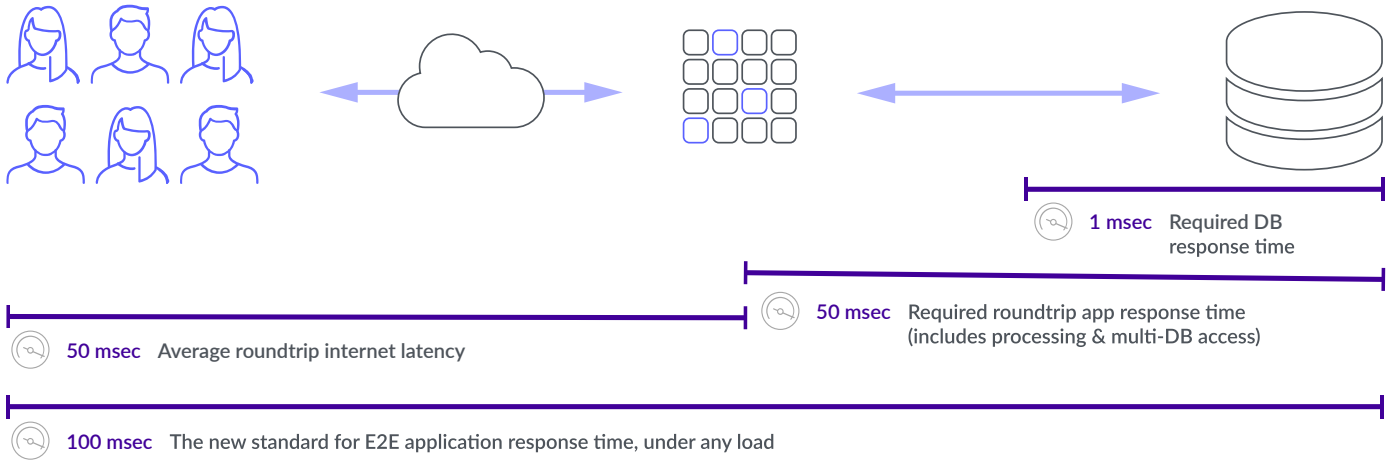
In these and other game-led personalizations, the common thread lies in using backend analysis and data processing to provide real-time personalization services. It's part of the growing standard of **360° player information views**. Rather than break up player data for later batch analysis or piecemeal processing, this approach analyzes a complete view of the player. It's a more data-centric approach that increases personalization and monetization.

Technical requirements for a real-time personalization database

Whether it's player-led or game-led, personalization requires processing an incredible amount of data—and the amount of data involved is increasing exponentially. In order to meet the seamless standards required by players, you need both specialized approaches to processing data and a database capable of real-time performance. On top of that, your database needs to be able to reliably serve that personalization anywhere and everywhere your players are. All that translates into specific technical requirements for your database:

- **High write throughput** – Methods like batch processing create periods of stale data which can hinder your personalization efforts. The latest player data (identity, credentials, inventory items, recent actions, etc.) need to be stored as they happen, which requires a high level of write throughput.
- **Persistent player stores** – Player data needs to be stored in persistent session stores that can scale with servers so they're always accessible and actionable for personalization—because your players demand the same personalization experience no matter how many people are playing.
- **Low latency** – A rule of thumb when running applications in production states that every millisecond spent in the database means 100 milliseconds felt at the application level. That means the database needs to be capable of <1ms latency to hit the 100ms goal.
- **Analytics capabilities** – Every time you have to run data through a separate service, it adds another link in the chain that can fail or introduce latency. For reliability and performance, the database should have integrated analytics capabilities so it can apply smart analytics and surface the insights necessary for personalization.
- **Flexible deployability** – Personalization needs to happen unilaterally across your game no matter what environment it's deployed in. That means your database also needs to be able to deploy in any environment, whether it's in the cloud, on-premises, or hybrid.
- **High global availability** – You need to be able to count on your personalization services—and the database behind them. High availability is essential for any personalization database, as well as the ability to provide consistent performance anywhere around the globe.
- **Scalability** – It doesn't matter if there's thousands of players online or millions. Every player needs to have the same experience, which means your database needs to be able to scale while still maintaining the performance necessary for seamless personalization.

The driving power behind personalization is data. The more data you collect and the faster (and more accurately) you process it, the better your personalization is going to be. And in an age where personalization can be the deciding factor between lasting player loyalty and game deletion, you can't afford to take chances with your personalization database.



Select a Real-Time Database for Victory

Remember that definition of real-time responsiveness from the beginning?

If your services are going to respond fast enough for players to feel no latency, then the whole process from request to processing to response to serving the data to players has to happen in less than 100ms.

Unfortunately, the data still has to travel to the servers, then be processed by the servers. Network time can easily take up 50ms. Server and infrastructure time can take up another 50ms. Which leaves somewhere between 0ms and 1ms for the database to respond if the leaderboard is going to have a real-time response. Traditional disk-based databases simply aren't capable of latency that low, which is why you'll want to look at in-memory NoSQL databases. Redis Enterprise, for example, is capable of 50 million operations per second at <1ms latency.

But speed isn't the only requirement for a real-time database. It also needs the data capabilities crucial for success in each use case:

- **Leaderboards** – Every time the data is pulled from one data structure and converted to another, it takes time—and creates a potential breakdown point. When you use the right data model, you remove unnecessary conversions, increasing speed and reliability. Sorted Sets, a data type natively supported by Redis, is a perfect example. Like the name suggests, it stores data in sets where data is pulled in order instead of being ordered afterwards. Not only does this make it faster to get ranges of data already sorted by scores, but it also cuts down on developer effort by making it simple to integrate the leaderboard data with the database.
- **Matchmaking** – On the backend, matchmaking needs to be able to find the right relationships quickly and reliably, while on the frontend players need search capability that's responsive and accurate. The right database is expandable and customizable to fit your data model and functionality needs. With Redis, this is done through modules. The RedisGraph module provides graph database functionality, while the RediSearch module provides frontend search capability.
- **Inventories** – The database must also include features like native support for data models for inventory responses. Once again, modules are your friend here. The Redis Streams module can improve frontend functionality, while the RediSearch module provides search and analytics capability. And RedisJSON can bolster performance by storing in a dynamic schema and retrieving documents at lightning speed.
- **Personalization** – Beyond the obvious need for speed are the additional requirements for analytics capabilities. Running analytics through additional services creates more links in the chain that can break, so you need modules like RedisBloom. By supporting compact, probabilistic data structures, RedisBloom allows you to create filters that reduce the compute and memory needs to search massive data sets, like personalization analysis.

Finally, your database needs to deliver the same real-time responsiveness to every player, every time—no matter where they log in from. That's why your real-time database needs to provide real-time responsiveness at any scale and high availability on a global level. And that demands geo-distribution, automated resharding, single-digit automated failover, 99.999% availability, and the ability to deploy to any environment you need it.

Conclusion

Player engagement is more than just having the right features in your game. It's about delivering them in a seamless way which pulls the player into the world of your game. It doesn't matter whether it's a cartoonish battle royale, a complicated strategy game, a gritty FPS, or a charming puzzler. You and your team have created a world where players want to spend their time, one where they can really dig in and have fun. Whether it's customized leaderboards, smart matchmaking, rich inventories, or focused personalization, all those features are built with one goal—keep players immersed in the game and having fun.

When you get down to it, that immersion is really what real-time player engagement is all about. And when you can seamlessly deliver the right features and services to your players, they'll get sucked into the game, stay in the zone, and keep playing. That engagement, in turn, leads to better player retention, higher DAUs/MAUs, and increased game revenue.

And, most importantly, a community of players who love your game and keep it going strong.

Learn more ways a real-time database can level up your game and increase player engagement. Go to redis.com/gaming or download the *Level Up Your Gametech with a Real-time Database* white paper.

About Redis

Data is the lifeline of every business, and Redis helps organizations reimagine how fast they can process, analyze, make predictions, and take action on the data they generate. Redis provides a competitive edge to any business by delivering open source and enterprise-grade data platforms to power applications that drive real-time experiences at any scale. Developers rely on Redis to build performance, scalability, reliability, and security into their applications.

Born in the cloud-native era, Redis uniquely enables users to unify data across multi-cloud, hybrid, and global applications to maximize business potential. Learn how Redis can give you this edge at redis.com