Backend Question And MYSQL #backend

https://blog.daveallie.com/ulid-primary-keys

12 Microservices Design Patterns to Know Before the System Design Interview

Ref: **Grokking Microservices Design Patterns** - https://lnkd.in/dEVmrWfN

Here is the list of design patterns:

1. Strangler Fig Pattern
2. API Gateway Pattern
3. Backends for Frontends Pattern (BFF)
4. Service Discovery Pattern
5. Circuit Breaker Pattern
6. Bulkhead Pattern
7. Retry Pattern
8. Sidecar Pattern
9. Saga Pattern
10. Event-Driven Architecture Pattern
11. CQRS (Command Query Responsibility Segregation) Pattern
12. Configuration Externalization Pattern

If you need to fetch data from some resource and store it in cache and access same cached data how would you do ?
how would you design tables if user have multiple address in mongodb

What is P95 and P99 in terms of latency ?

P95 and P99, often referred to as percentiles, are metrics used to analyze and describe the distribution of latency (response times) in a set of data, especially in the context of performance monitoring and optimization. These percentiles are useful for understanding the behavior of a system under various conditions and identifying outliers or extreme cases.

1. **P95 (95th Percentile):**
   - The P95 latency represents the value below which 95% of the data falls. In other words, it indicates the latency that the majority (95%) of the requests or transactions experience.
   - For example, if the P95 latency of a system is 100 milliseconds, it means that 95% of the requests are processed in 100 milliseconds or less.

2. **P99 (99th Percentile):**
   - The P99 latency represents the value below which 99% of the data falls. It is a measure of the tail end of the latency distribution, capturing the worst-case scenarios or outliers.

- If the P99 latency is 200 milliseconds, it indicates that 99% of the requests are processed in 200 milliseconds or less, but 1% of the requests may experience longer latency.

In summary, P95 and P99 percentiles provide insights into the distribution of latency values, helping to identify the typical performance as well as the outliers or situations with higher latency. Monitoring and optimizing these percentiles are crucial for ensuring a reliable and responsive system, particularly in scenarios where consistent low-latency performance is essential, such as web applications, APIs, or real-time systems.

Difference between put and patch in RESTAPI ?

In RESTful API design, both PUT and PATCH HTTP methods are used to update resources, but there is a subtle difference between the two.

PUT method is used to update or replace the entire resource at the specified URL. When a client sends a PUT request, the server replaces the existing resource with the new one provided in the request payload. If the resource doesn't exist at the specified URL, a new resource is created with that URL.

For example, if we have a resource at `https://example.com/api/users/1` and we want to update it with new data, we can send a PUT request to that URL with the updated data in the request payload. The server will replace the existing user resource with the new data.

On the other hand, the PATCH method is used to make a partial update to a resource at the specified URL. The PATCH request only updates the specified fields in the resource, leaving the rest of the resource unchanged. If the specified fields don't exist, they are created.

For example, if we have a user resource at `https://example.com/api/users/1` and we want to update only the user's email address, we can send a PATCH request to that URL with the updated email address in the request payload. The server will update only the email address field and leave the rest of the user resource unchanged.

In summary, PUT is used to update or replace the entire resource, while PATCH is used to make a partial update to a resource. Which method to use depends on the requirements of the API and the resources being updated.

Difference between CROSS JOIN and UNION inMYSQL ?

1. CROSS JOIN:
A CROSS JOIN in MySQL (and in SQL in general) produces the Cartesian product of two tables, i.e., it combines each row from the first table with every row from the second table. The result set of a CROSS JOIN has the number of rows equal to the product of the number of rows in the two tables being joined.
Example:
sql

Copy code
SELECT * FROM table1 CROSS JOIN table2;
This will create a result set where each row from table1 is combined with every row from table2.
2. UNION:
UNION in MySQL is used to combine the result sets of two or more SELECT statements. The UNION operator removes duplicate rows from the result set.
Example:
sql

Copy code
SELECT column1 FROM table1 UNION SELECT column1 FROM table2;
This will create a result set that includes distinct values from column1 in both table1 and table2. Duplicate values are automatically removed.
Key Differences:

- **Purpose:**
  - CROSS JOIN is used to generate the Cartesian product of two tables, combining every row from the first table with every row from the second table.
  - UNION is used to combine the results of two or more SELECT statements, removing duplicates from the combined result set.
- **Result Set Structure:**
  - The result set of a CROSS JOIN is typically larger and includes all possible combinations of rows from the joined tables.
  - The result set of a UNION is a combination of distinct rows from the individual SELECT statements.
- **Syntax:**
  - CROSS JOIN is a clause in the FROM statement, specifying the tables to be crossed.
  - UNION is an operator used between two SELECT statements.
- **Duplicate Handling:**
  - CROSS JOIN does not handle duplicates because it simply combines every row with every other row.
  - UNION automatically removes duplicate rows from the combined result set.

In summary, CROSS JOIN is used to create all possible combinations of rows, while UNION is used to combine and deduplicate rows from multiple SELECT statements. They serve different purposes in SQL queries.

OPTIONS METHOD IN CONTEXT OF REST API ?

In the context of RESTful API design, the OPTIONS method is used to retrieve information about the communication options available for a resource at a particular URL.

When a client sends an OPTIONS request to a resource URL, the server should respond with a list of the HTTP methods that are supported for that resource, as well as any additional information that the server wants to provide. This information can include allowed headers, authentication requirements, and other options.

The response to an OPTIONS request should include the `Allow` header, which lists the HTTP methods that are supported for the requested resource. It can also include other headers such as `Access-Control-Allow-Origin` that specify any additional access control options for the resource.

Here's an example of an OPTIONS request and response:

Request:
```
OPTIONS /api/users/123 HTTP/1.1
Host: example.com
```

Response:
```
HTTP/1.1 200 OK
Allow: GET, PUT, PATCH, DELETE
Access-Control-Allow-Origin: *
```

In this example, the server is indicating that the resource at `/api/users/123` supports the GET, PUT, PATCH, and DELETE methods. The `Access-Control-Allow-Origin` header allows cross-origin requests from any origin.

The OPTIONS method is important in RESTful API design because it allows clients to discover the available methods for a resource and can help to prevent errors and improve security. By responding to OPTIONS requests with detailed information about a resource's communication options, servers can ensure that

clients are able to interact with the resource in a safe and effective manner.

Who is initiating the OPTIONS METHOD ?

When do this CORS thing occur ?

So when we have a server abc.com and another server xyz.com
and from abc.com we send a request to xyz.com then CORS origin happens in which the all the allowed header are returned

1. Cross-origin resource sharing (CORS): OPTIONS method is also used in CORS to determine whether a cross-origin request is allowed. When a client makes a cross-origin request, it first sends an OPTIONS request to the server to determine whether the actual request is allowed, based on the server's access control policy.

Follow up question :

Consider both the request is from abc.com to abc.com will CORS occur ?

1st request comes from port 3000 (abc.com)

2nd request comes from port 5000 (abc.com)

Ans:

If a request and response are from different ports on the same server, the web browser will see them as coming from different origins, and CORS will be triggered. In this case, the server needs to send the appropriate CORS headers to allow the web browser to access the resources.

DIFFERENCE BETWEEN CHAR AND VARCHAR ?

In SQL, `CHAR` and `VARCHAR` are both data types used for storing character strings.

The main difference between `CHAR` and `VARCHAR` is that `CHAR` has a fixed length, while `VARCHAR` has a variable length.

Here's a summary of the key differences:

- `CHAR`: Stores a fixed-length string of characters. When you define a

`CHAR` column, you specify the maximum number of characters it can store. For example, `CHAR(10)` would create a column that can store up to 10 characters. If you insert a shorter string, the remaining space is padded with spaces.
- `VARCHAR`: Stores a variable-length string of characters. When you define a `VARCHAR` column, you also specify the maximum number of characters it can store. However, if you insert a shorter string, the column only uses the necessary amount of space.

The advantage of using `CHAR` is that it can be faster for fixed-length data and can provide better performance for some operations, such as sorting and grouping. However, it can also waste storage space for shorter strings.

On the other hand, `VARCHAR` can be more space-efficient for variable-length data but may not perform as well in certain operations.

In general, it's best to use `CHAR` for columns where the length of the data is always the same (such as postal codes or phone numbers), and `VARCHAR` for columns where the length of the data can vary (such as names or descriptions).

DIFFERENCE BETWEEN DELETE AND TRUNCATE ?

In MySQL, both `DELETE` and `TRUNCATE` commands can release space, but they do it in different ways:

1. `DELETE`: When you use the `DELETE` command to remove rows from a table, it marks the space occupied by those rows as available for reuse. However, the space is not immediately reclaimed. Instead, it remains allocated to the table and can be reused by future `INSERT` operations or other modifications. The freed space is gradually reused as new data is added to the table, but the table's physical size on disk does not shrink immediately.

2. `TRUNCATE`: The `TRUNCATE` command removes all rows from a table, effectively deleting all data. Unlike `DELETE`, `TRUNCATE` also releases the space occupied by the table. It deallocates the data pages associated with the table, effectively resetting the table to its initial state. The space is released and made available for reuse immediately, resulting in a smaller physical size of the table on disk.

It's important to note that releasing space through `TRUNCATE` is more efficient compared to `DELETE` because `TRUNCATE` does not generate undo logs and does not write individual row deletions to the transaction logs. This makes it faster and helps to reclaim disk space more effectively.

However, be cautious when using `TRUNCATE` as it is a DDL statement and cannot be rolled back. Once a table is truncated, the data cannot be recovered, so make sure to have appropriate backups in place before performing a `TRUNCATE` operation.

Important Note:

- Use TRUNCATE or DELETE when you want to remove data from a table while keeping the table structure.
- Use DROP when you want to remove an entire database, table, or index along with its structure.
- Be cautious when using DROP, as it permanently deletes the data and structure, and the operation cannot be undone. Always have a backup before using DROP, especially in a production environment.

What is on delete cascade constraints in mysql ?

In MySQL, the `ON DELETE CASCADE` option is used to define a referential action to be taken when a record in the parent table (referenced table) is deleted. Specifically, it indicates that when a record in the parent table is deleted, all corresponding records in the child table (referencing table) should also be automatically deleted. This is part of maintaining referential integrity between tables.

Here's an example to illustrate the usage of `ON DELETE CASCADE`:

Consider two tables: `employees` (parent table) and `employee_details` (child table). The `employee_details` table has a foreign key referencing the `employees` table.

```sql
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(255)
);

CREATE TABLE employee_details (
    detail_id INT PRIMARY KEY,
    employee_id INT,
    detail_info VARCHAR(255),
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id) ON
DELETE CASCADE
);
```

In this example:
- The `employees` table has a primary key `employee_id`.
- The `employee_details` table has a foreign key `employee_id` referencing the `employees` table.
- The `ON DELETE CASCADE` option is specified for the foreign key constraint in the `employee_details` table.

Now, let's say you have the following data:

```sql
INSERT INTO employees VALUES (1, 'John');
INSERT INTO employee_details VALUES (101, 1, 'Details for John');
```

If you attempt to delete the record in the `employees` table with `employee_id = 1`, the `ON DELETE CASCADE` action will automatically delete the corresponding record in the `employee_details` table:

```sql
DELETE FROM employees WHERE employee_id = 1;
```

After this delete operation, both the record in the `employees` table and the corresponding record in the `employee_details` table will be removed.

```sql
-- Query to check the result
SELECT * FROM employees;
SELECT * FROM employee_details;
```

This ensures that when a referenced record is deleted, all related records in the referencing table are also deleted, maintaining referential integrity.

Difference between REST and SOAP ?

REST (Representational State Transfer) and SOAP (Simple Object Access Protocol) are two different architectural styles used for designing web services. Here are the main differences between them:

	SOAP	RESR

Protocol	SOAP is a protocol	REST is an architectural style
	SOAP defines a specific set of rules for communication and data exchange	REST is a more flexible approach that leverages existing web standards
Message Format	SOAP uses XML (eXtensible Markup Language) as its message format. It typically involves wrapping data within an XML structure	REST can use various data formats such as XML, JSON (JavaScript Object Notation), or even plain text
Transport Protocol	SOAP can use various transport protocols, including HTTP, SMTP, and more. It is not limited to HTTP alone	REST, on the other hand, primarily uses HTTP as the transport protocol. It takes advantage of HTTP methods like GET, POST, PUT, DELETE, etc., for data operations
Ease of Use	SOAP, with its XML-based structure and strict rules, can be more complex and require additional tooling	REST is generally considered simpler and easier to use , It relies on standard HTTP methods and status codes, making it more intuitive and easier to understand.
Flexibility	SOAP, while still widely used in enterprise systems, may require specific toolkits and frameworks for implementation and integration	REST offers more flexibility in terms of data formats, scalability, and compatibility with different platforms and technologies. It can be easily consumed by a wide range of clients, including web browsers, mobile apps, and other web services.

Statelessness	SOAP, on the other hand, can maintain state through its use of session-based connections	REST follows a stateless model, meaning each request from a client to a server is independent and doesn't rely on the server's previous state
Caching	SOAP, being more complex, doesn't have built-in caching support and typically relies on custom caching techniques.	REST can take advantage of HTTP caching mechanisms, such as caching responses at the client or intermediate proxies

Overall, REST is widely adopted for building web services due to its simplicity, flexibility, and compatibility with modern web standards. SOAP, while still used in certain scenarios, is often associated with more complex enterprise systems and legacy applications. The choice between REST and SOAP depends on the specific requirements, constraints, and existing infrastructure of the system being developed.

Difference between Authentication and Authorisation ?

Authentication and Authorization are two distinct concepts in the field of security and access control. Here's the difference between them:

Authentication:
Authentication is the process of verifying the identity of a user or entity. It ensures that the user or entity claiming to be a particular identity is, in fact, who they say they are. Authentication is typically performed at the beginning of a session or when accessing a protected resource. The goal of authentication is to prevent unauthorized access and establish trust in the system. Common authentication mechanisms include usernames and passwords, biometrics (such as fingerprint or facial recognition), tokens, certificates, and multi-factor authentication (combining multiple authentication factors for enhanced security).

Authorization:
Authorization, on the other hand, is the process of granting or denying access rights and permissions to authenticated users or entities. Once a user's identity is established through authentication, authorization determines what actions or

resources they are allowed to access or perform. It is about defining and enforcing rules and policies that control what users can do within a system or application. Authorization can be based on roles, permissions, user groups, or other criteria. It ensures that users only have access to the resources they are authorized to use and helps enforce the principle of least privilege.

In summary, authentication verifies the identity of a user or entity, while authorization determines what actions or resources that authenticated user or entity is allowed to access. Authentication establishes trust in the system, while authorization controls and restricts access to maintain security and protect sensitive information. Both authentication and authorization are crucial components of a comprehensive security framework.

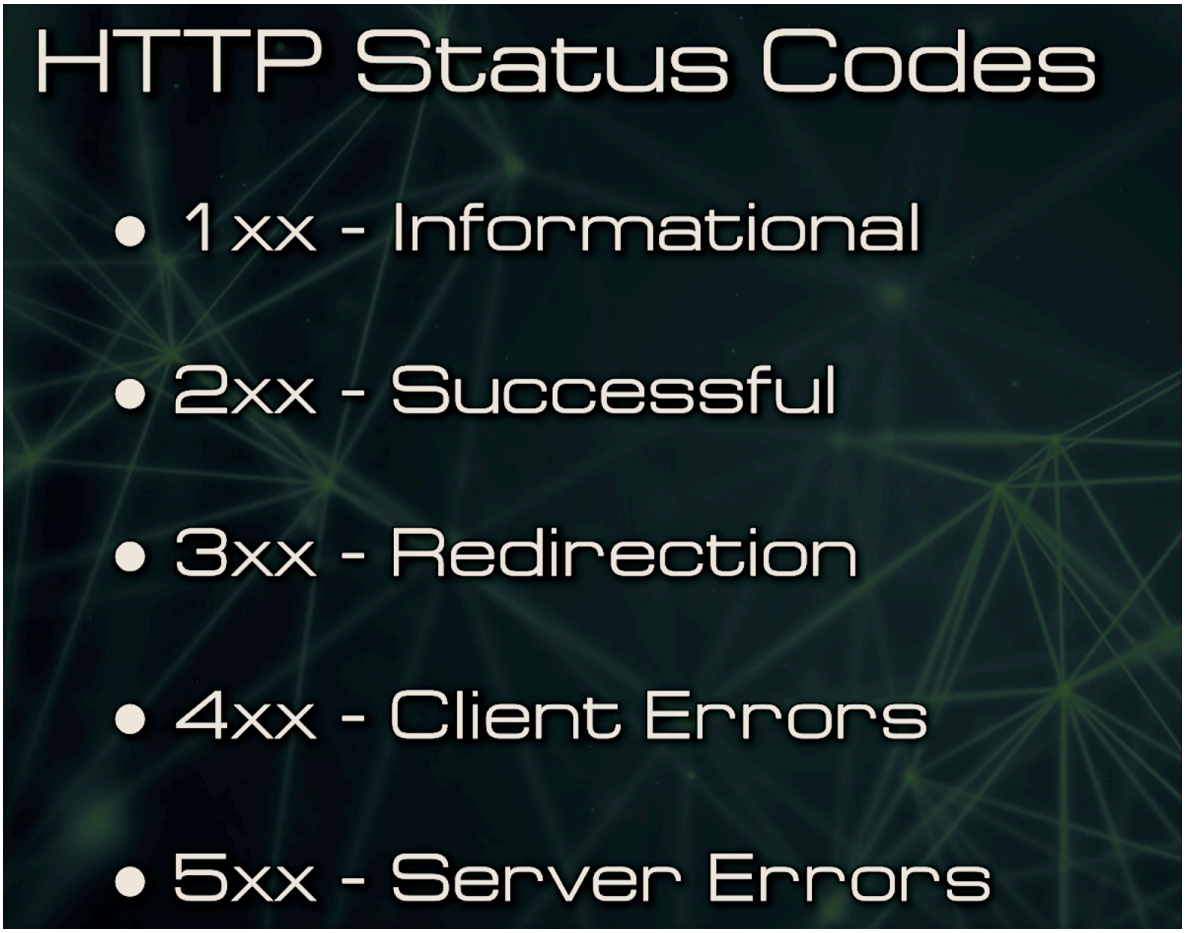
What is the diff between http and https ?

Key Differences Summary:

- **Encryption:**
  - HTTP is not encrypted, while HTTPS uses encryption to secure the data.
- **Security:**
  - HTTP is susceptible to security threats, whereas HTTPS provides a secure and encrypted connection.
- **Data Integrity:**
  - HTTPS ensures data integrity by detecting tampering, whereas HTTP does not provide mechanisms for this.
- **Port:**
  - HTTP uses port 80, and HTTPS uses port 443 by default.

In summary, HTTPS is the secure version of HTTP, and it adds an essential layer of security by encrypting the data exchanged between the user's browser and the server, making it significantly more resistant to eavesdropping and tampering. It is particularly important for websites that handle sensitive information, such as login credentials or payment details.

HTTP STATUS CODES:



DIFFERENCE BETWEEN DATETIME AND TIMESTAMP IN MYSQL ?

In MySQL, both DATETIME and TIMESTAMP are data types used to store date and time information, but they have some key differences in terms of range, storage, and behavior. Here's a summary of the main differences:

- **Range of Values:**
  - DATETIME: The DATETIME type has a broader range and can store dates from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
  - TIMESTAMP: The TIMESTAMP type has a more limited range, allowing dates from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC. This limitation is due to the use of a 32-bit integer to store the timestamp.
- **Storage Size:**
  - DATETIME: The DATETIME type requires 8 bytes of storage.
  - TIMESTAMP: The TIMESTAMP type also requires 4 bytes of storage.
- **Behavior with Time Zones:**
  - DATETIME: The values stored in a DATETIME column are not associated with any time zone information. When you retrieve the data, it is assumed to be in the time zone set for the MySQL

server.
    - TIMESTAMP: The values stored in a TIMESTAMP column are stored in UTC (Coordinated Universal Time) format and are converted to the session time zone when retrieved.
- **Automatic Initialization and Updating:**
    - DATETIME: Does not support automatic initialization and updating. You need to set the values explicitly when inserting or updating records.
    - TIMESTAMP: Supports automatic initialization and updating. You can set the column to automatically update to the current timestamp on insert or update using the DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP options.

Example:

```
-- DATETIME example
CREATE TABLE example_datetime (
  id INT PRIMARY KEY,
  event_time DATETIME
);

-- TIMESTAMP example
CREATE TABLE example_timestamp (
  id INT PRIMARY KEY,
  event_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

-- DATETIME example CREATE TABLE example_datetime (id INT PRIMARY KEY, event_time DATETIME); -- TIMESTAMP example CREATE TABLE example_timestamp (id INT PRIMARY KEY, event_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);
In summary, choose between DATETIME and TIMESTAMP based on your specific requirements. If you need a broad range of dates and times and do not require time zone conversion, DATETIME might be suitable. If you want automatic handling of time zones and support for future automatic updates, TIMESTAMP is a good choice.

A multiple column index is created over firstName, lastName city columns of a Customer table. Will this index be used for SELECT queries based on only first_name, only last_name or only city values?

The order of the columns in a multiple-column index matters when it comes to query optimization in MySQL. In the context of the provided scenario, where the multiple-column index is created over firstName, lastName, and city columns of a Customer table, the index can be effectively used for certain types of queries.

- **Prefix Queries:**
  - The index can be used for queries that involve a leftmost prefix of the index columns. For example, if you have a query that filters based on only the firstName or firstName and lastName, the index can be utilized.

```
-- Index can be used for these queries
SELECT * FROM Customer WHERE firstName = 'John';
SELECT * FROM Customer WHERE firstName = 'John' AND lastName = 'Doe';
```

Single-Column Queries:

- If your query involves only one of the indexed columns, the index can still be used.

```
-- Index can be used for these queries
SELECT * FROM Customer WHERE lastName = 'Doe';
SELECT * FROM Customer WHERE city = 'New York';
```

Composite Queries:

- If your query involves a combination of the indexed columns from left to right, the index can be used.

```
-- Index can be used for these queries
SELECT * FROM Customer WHERE firstName = 'John' AND lastName = 'Doe';
SELECT * FROM Customer WHERE firstName = 'John' AND city = 'New York';
SELECT * FROM Customer WHERE firstName = 'John' AND lastName = 'Doe'
AND city = 'New York';
```

However, keep in mind that the index won't be as effective for queries that don't match the leftmost prefix of the index. For example:

```
-- Index might not be used effectively for this query
SELECT * FROM Customer WHERE lastName = 'Doe' AND city = 'New York';
```

In this case, the index may still be used, but it won't be as efficient as when the query matches the leftmost prefix. To address such cases, you might consider creating additional indexes or reordering the columns in the index based on the common query patterns in your application.

How the NULL values are stored in SQL internal working in SQL ?

In SQL databases, the representation and handling of `NULL` values can vary depending on the database management system (DBMS) being used. However, I'll provide a general explanation of how `NULL` values are typically handled in SQL.

1. **Internal Representation:**
   - Internally, `NULL` values are usually represented differently from regular data values. Instead of storing a specific value like 0 or an empty string, databases often use a special bit pattern or a marker to indicate the absence of a value.

2. **Bitmaps or Flags:**
   - Some database systems use bitmaps or flags associated with each column to indicate whether the value is `NULL` or not. This approach allows for efficient storage and retrieval of `NULL` information.

3. **Storage Overhead:**
   - The storage overhead for `NULL` values is typically small. The database system needs to store an additional marker for each nullable column to indicate whether the value is `NULL` or not.

4. **Handling in Indexes:**
   - Index structures may have additional mechanisms to handle `NULL` values efficiently. For instance, some indexes may include a separate structure to track the presence or absence of `NULL` values.

5. **Query Processing:**
   - When querying data, SQL engines are designed to handle `NULL` values appropriately. SQL queries often include special constructs like `IS NULL` or `IS NOT NULL` to filter or identify `NULL` values.

6. **Comparisons:**
   - Comparisons involving `NULL` values (`NULL = NULL` or `NULL <> NULL`) usually result in an unknown or undefined outcome. This is because the actual value of `NULL` is unknown, and its purpose is to represent the absence of a value.

It's essential to note that the specifics of how `NULL` values are handled can vary between different SQL database systems (such as MySQL, PostgreSQL, SQL Server, etc.), and the information provided here represents a generalized explanation. Always refer to the documentation of the specific database system you are using for accurate details on how `NULL` values are handled internally.

Table : order

Order_no	customer_number

10100	363
10101	128
10102	181

Table : order_details

Order_no	prod_code	Quantity	price_each	
10100	S18_1749	30	136	
10100	S18_2248	50	55.09	
10100	S18_4409	22	75.46	
10100	S24_3969	49	35.29	
10101	S18_2325	25	108.06	
10101	S18_2795	26	167.06	
10101	S24_1937	45	32.53	
10101	S24_2022	46	44.35	
10102	S18_1342	39	95.55	
10102	S18_1367	41	43.13	

Q1 get order numbers, the number of items sold per order, and total sales for each Order?

```
SELECT
   Order_no,
   SUM(Quantity * price_each) AS total_sales
FROM
   order_details
GROUP BY
   Order_no;
```

Apache Kafka:

Traditional message queues, like RabbitMQ, are not the same as Kafka. RabbitMQ eliminates messages immediately after the consumer confirms them, whereas Kafka keeps them for a period of time (default is 7 days) after they've been received.

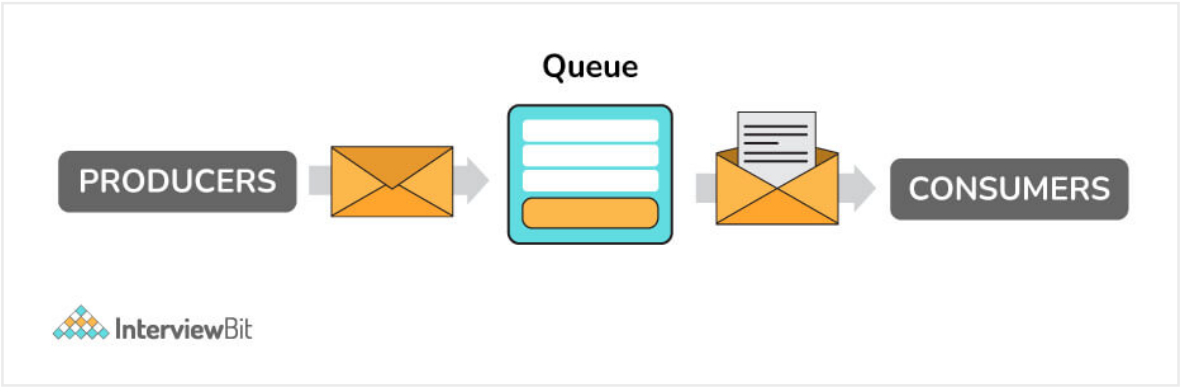
- Kafka is a messaging system built for high throughput and fault tolerance.
- Kafka has a built-in patriation system known as a Topic.
- Kafka Includes a replication feature as well.
- Kafka provides a queue that can handle large amounts of data and move messages from one sender to another.

- Kafka can also save the messages to storage and replicate them across the cluster.
- For coordination and synchronization with other services, Kafka collaborates with Zookeeper.
- Apache Spark is well supported by Kafka.

What are the traditional methods of message transfer? How is Kafka better from them?

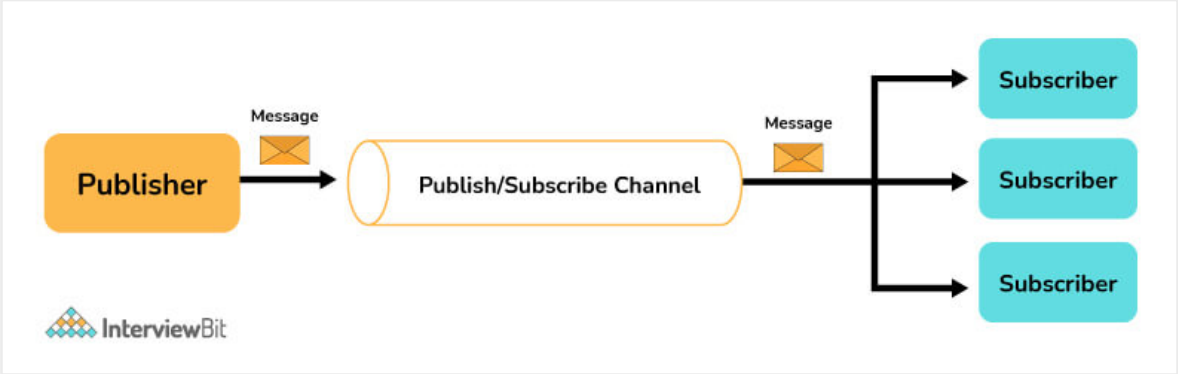
Following are the traditional methods of message transfer:-

- **Message Queuing:-**
  A point-to-point technique is used in the message queuing pattern. A message in the queue will be destroyed once it has been consumed, similar to how a message is removed from the server once it has been delivered in the Post Office Protocol. Asynchronous messaging is possible with these queues.
  If a network problem delays a message's delivery, such as if a consumer is unavailable, the message will be held in the queue until it can be sent. This means that messages aren't always sent in the same order. Instead, they are given on a first-come, first-served basis, which can improve efficiency in some situations.



- **Publisher - Subscriber Model:-**
  The publish-subscribe pattern entails publishers producing ("publishing") messages in multiple categories and subscribers consuming published messages from the various categories to which they are subscribed. Unlike point-to-point texting, a message is only removed once it has been consumed by all category subscribers. Kafka caters to a single consumer abstraction that encompasses both of the aforementioned- the consumer group. Following are the benefits of using Kafka over the traditional messaging transfer techniques:
  - **Scalable:** A cluster of devices is used to partition and streamline the data thereby, scaling up the storage capacity.
  - **Faster:** Thousands of clients can be served by a single Kafka broker as it can manage megabytes of reads and writes per second.

- **Durability and Fault-Tolerant:** The data is kept persistent and tolerant to any hardware failures by copying the data in the clusters.



What is the maximum size of a message that Kafka can receive?

By default, the maximum size of a Kafka message is **1MB** (megabyte). The broker settings allow you to modify the size. Kafka, on the other hand, is designed to handle 1KB messages as well.

How do you start a Kafka server?

Firstly, we extract Kafka once we have downloaded the most recent version. We must make sure that our local environment has Java 8+ installed in order to run Kafka.

The following commands must be done in order to start the Kafka server and ensure that all services are started in the correct order:

- Start the ZooKeeper service by doing the following:

$bin/zookeeper-server-start.sh config/zookeeper.properties

- To start the Kafka broker service, open a new terminal and type the following commands:

$ bin/kafka-server-start.sh config/server.properties

What do you mean by geo-replication in Kafka?

Geo-Replication is a Kafka feature that allows messages in one cluster to be copied across many data centers or cloud regions. Geo-replication entails replicating all of the files and storing them throughout the globe if necessary. Geo-replication can be accomplished with Kafka's MirrorMaker Tool. Geo-replication is a technique for ensuring data backup.

What are some of the disadvantages of Kafka?

Following are the disadvantages of Kafka :

- Kafka performance degrades if there is message tweaking. When the message does not need to be updated, Kafka works well.
- Wildcard topic selection is not supported by Kafka. It is necessary to match the exact topic name.
- Brokers and consumers reduce Kafka's performance when dealing

with huge messages by compressing and decompressing the messages. This has an impact on Kafka's throughput and performance.
- Certain message paradigms, including point-to-point queues and request/reply, are not supported by Kafka.
- Kafka does not have a complete set of monitoring tools.

How to improve API performance

The diagram below shows 5 common tricks to improve API performance.

1. Pagination
   This is a common optimization when the size of the result is large. The results are streaming back to the client to improve the service responsiveness.
2. Asynchronous Logging
   Synchronous logging deals with the disk for every call and can slow down the system. Asynchronous logging sends logs to a lock-free buffer first and immediately returns. The logs will be flushed to the disk periodically. This significantly reduces the I/O overhead.
3. Caching
   We can cache frequently accessed data into a cache. The client can query the cache first instead of visiting the database directly. If there is a cache miss, the client can query from the database. Caches like Redis store data in memory, so the data access is much faster than the database.
4. Payload Compression
   The requests and responses can be compressed using gzip etc so that the transmitted data size is much smaller. This speeds up the upload and download.
5. Connection Pool
   When accessing resources, we often need to load data from the database. Opening the closing db connections add significant overhead. So we should connect to the db via a pool of open connections. The connection pool is responsible for managing the connection lifecycle.

Other ways of improving API performance:
- Avoiding the thread-per-request model and using an event loop instead. Using non-blocking APIs whenever possible on the backend.
- Offer filtering and field projection capabilities in your API. This way, your clients will only fetch the data they're interested in, avoiding overfetching.

- For larger payloads, using HTTP chunked encoding or gRPC streaming to transfer data piece by piece.
- Be careful with pagination with SQL databases as limit and offset operations are costly. See: https://stackoverflow.com/questions/4481388/why-does-mysql-higher-limit-offset-slow-the-query-down
- Limit the number of requests. Use batching instead.
- Make sure HTTP/2 is enabled.
- JSON parsing is costly. Switch to binary formats like Protobuf.
- Use PATCH instead of PUT to update just a subset of the resource.

How to Optimize Paging in MySQL? 3 Best Ways

Paging or pagination is a common requirement for almost all web applications where all the documents are divided into discrete pages. This is the easiest way to select only what you need from the database.
However, optimizing paging operations is crucial to ensure efficient and fast retrieval of data in
MySQL
. So how do you optimize paging in MySQL? There is a technique named "deferred join" which is a great optimizing solution for more efficient pagination in MySQL.
In this article, we will explain the "deferred join" technique including the other 2 best ways to optimize pagination in MySQL. Let's get started below.

3 Ways to Optimize Paging in MySQL

There are several ways to optimize paging in MySQL and among them, using a deferred join, paging without discarding records, and maintaining a place column is the 3 most effective way to optimize paging. Read on to explore them in detail.

1. Use Deferred Join

The "Deferred join" in MySQL is a great technique to optimize paging. Instead of the entire table on the database, you can make the pagination on a subset of data using this.
For example, you have a customer page with a record of 500 customers where only 10 customers are displayed on each page. Here, the query will use **LIMIT** to get 10 records and **OFFSET** to skip all the previous page results.

```
SELECT id, name, address, phone FROM customers ORDER BY name LIMIT 10 OFFSET 490;
```

With this, while you're at the 50th page, it is doing **LIMIT 10 OFFSET 490**. In this way, the MySQL server has to go and read all the five hundred records, scan an index, retrieve rows by primary key id, and then discard them.
This is a lengthy process, and you can make it faster with the "deferred join" technique. It will only use the primary key to fetch data and it can be done by "using index" which is loved by most programmers.

```
SELECT idFROM customersORDER BY nameLIMIT 10 OFFSET 490;
```

Now, combine this using an **INNER JOIN** to get the ten rows and data you want:
SELECT id, name, address, phoneFROM customersINNER JOIN (SELECT idFROM customersORDER BY nameLIMIT 10 OFFSET 490)AS my_results USING(id);
Paging queries in SQL can be slow due to the frequent use of the OFFSET keyword. It simply makes the server scan, collect and discard unnecessary rows while fetching only a subset. By implementing deferred join or maintaining a designated position column, you can enhance the performance of your database significantly.

2. Try Paging Without Discarding Records

The main thing we have to do is avoid discarding records. The main target of optimizing for paging in MySQL is to prevent the server from fetching all the data. For this, you can try calling by ID.

```
select id, name, address, phoneFROM customersWHERE id > 490ORDER BY id LIMIT 10;
```

However, this solution will only work if you were paging by ID. If you're paging by name, it might get messier as there may be multiple people with the same name. So, if **ID** doesn't work for your application, you can try the **USERNAME**.

```
SELECT id, usernameFROM customersWHERE username > 'Justin'ORDER BY username LIMIT 10;
```

Isn't this cool? Now, let's try another trick!

3. Maintain A Column

Another way to optimize paging in MySQL is to maintain a column for the page. In this case, you have to update that column whenever you –

1. INSERT a row
2. DELETE a row
3. move a row with UPDATE.

Although this may not be suitable for a page, it can still make paging easier with a straight place or position.

```
SELECT id, name, address, phoneFROM customersWHERE page = 50ORDER BY name;
```

Or you can try with a place column like this:

```
SELECT id, name, address, phoneFROM customersWHERE place BETWEEN 490 AND 499ORDER BY name;
```

Frequently Asked Questions (FAQs)

What Is the Best Way to Do Pagination in MySQL?

The best way you can achieve pagination in MySQL is by implementing the LIMIT clause. While using the SELECT statement, the LIMIT clause will simply select a limited amount of data.

What Are The Different Ways To Optimize A MySQL Query?

You can optimize a MySQL query in different ways. For example, you can use the "ANALYZE TABLE table_name" to update the key distributions of the scanned table. You can use the Global and table-level STRAIGHT_JOIN, or you can turn global or thread-specific system variables.

How Do I Optimize My MySQL Database?

You can do several things to optimize the database in MySQL. For this, you

have to understand your workload and optimize queries. You should never use MySQL as a Queue. Monitoring the fundamental resources and recognizing the scalability problems is the key to optimizing the MySQL database.

Conclusion

When you are dealing with large datasets, optimizing paging in MySQL is crucial for improving the performance of your web application. The three best ways to optimize for paging in MySQL are described in this article and we hope, now you can easily optimize paging in MySQL after implementing any of these. For further queries, our comment box is always open for you. Thanks for reading!