Should we use uuid or auto increment id in a database ?

bigint unsigned : size = 8 bytes, max value = 2e64 - 1

UUID

Universally Unique Identifier

The UUID value is a hexadecimal string of 32 characters, separated by hyphens in a specific format 8-4-4-4-12. (total 36 characters)
It is a 128-bit long number in hex characters separated by “-“
example : `be5883c8-86fd-4e89-b5ef-8d5d07754b1c`

The choice between using UUIDs (Universally Unique Identifiers) or auto-incremented integers as primary keys in a database depends on various factors and considerations. Each approach has its own advantages and disadvantages. Here are some considerations for both: Auto-Incremented Integer (e.g., BIGINT):

Advantages:

- Sequential Order: Auto-incremented integers typically generate values in a sequential order, which can be beneficial for indexing and querying.
- Smaller Storage: Integers generally require less storage space compared to UUIDs, which can be important for large datasets.
- Faster Indexing: Searching and indexing on integer columns may be faster due to their sequential nature.

Disadvantages:

- Predictability: The sequential nature of auto-incremented integers can make it easier for someone to guess or infer the next value in the sequence.
- Not Suitable for Distributed Systems: Generating unique integers across distributed systems may require coordination and synchronization, which can be challenging.

UUID (Universally Unique Identifier):

Advantages:

- Uniqueness: UUIDs are designed to be universally unique, making them suitable for distributed systems where uniqueness across nodes is crucial.
- No Central Coordination: UUIDs can be generated independently on different nodes without requiring central coordination.

- Security: UUIDs do not reveal any information about the order or frequency of generation, providing a level of security through obscurity.

Disadvantages:

- Larger Storage: UUIDs typically require more storage space than integers, which can impact database size and query performance.
- Randomness: The randomness of UUIDs can lead to less efficient indexing and querying compared to sequential integers.

Considerations:

- Application Requirements: Consider the specific requirements of your application. If predictability and sequential ordering are important, auto-incremented integers may be more suitable. If global uniqueness is critical, UUIDs may be preferred.
- Database Size and Performance: Evaluate the impact of storage size and performance in your specific use case. For large datasets or performance-critical applications, the choice between integers and UUIDs can have implications.
- Compatibility: Consider the compatibility with existing systems or tools that may work more naturally with one type of identifier over the other.
- Use Cases: Certain use cases, such as distributed systems, may benefit more from UUIDs, while other applications with simpler requirements may find auto-incremented integers sufficient.

Ultimately, the choice between UUIDs and auto-incremented integers depends on the specific needs and characteristics of your application. It's also worth noting that hybrid approaches, such as using a combination of UUIDs and integers for different purposes within the same database, are also possible.

source : chatgpt

Performance test

What is the more performant solution between an auto-increment id and an uuid ?

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private UUID uuid;
@Column(length = 36)
private String uuidString;
```

In the MySQL database, corresponding are id : bigint uuid : binary(16) uuid_string : varchar(36)

query in MySQL console

select * from identification where id | uuid | uuid_string =

1 268 309 rows id : 0.4187 ms uuid : 0.4754 ms uuid_string : 0.4906 ms

```
mysql> show profiles;
+----------+------------+-----------------------------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                                         |
+----------+------------+-----------------------------------------------------------------------------------------------+
|      105 | 0.00060725 | select * from identification where id = 500000                                                |
|      106 | 0.00052025 | select * from identification where uuid_string = '40e4c9ee-6c71-4011-84aa-99762c072400'        |
|      107 | 0.00050400 | select * from identification where uuid = 0x40E4C9EE6C71401184AA99762C072400                   |
+----------+------------+-----------------------------------------------------------------------------------------------+
```

Spring boot code (service layer)

id : 3 ms
uuid : 20 ms (7x)
uuid_string : 15 ms (5x)

Conclusion

requests done directly in the MySQL console does not show big differences between the three types.
Using a Java Spring boot application shows a big difference of performance in favour of the bigint identifier.