# UniqueKey

The Solr **uniqueKey** field encodes the *identity semantics* of a document. In database jargon, the *primary key*.



## Use cases

A Solr index does not need a unique key, but almost all indexes use one.

### Use cases which do not require a unique key

- Build an index from empty. Search for documents. When you have new documents to add, either add them or clear index and reindex from scratch. You know that you will never add the same document twice.
- Sort documents found in a search against a numerical field.

### Use cases which require a unique key

- Add documents incrementally. You do not rebuild the index from scratch but want to add new documents periodically. You may add the same document twice and it will only be stored once.
  - An example is an RSS feed from a blog: the RSS feed will be polled, therefore the same article may appear more than once.
- Do statistical analysis on indexed data.
  - If the index is large, you might want to pick a subset for your research. A wildcard search on the key field does this.
- Share document identity with other database systems in "vertical partition" style.
  - As an example, store only index data but not original fields for large documents. To fetch documents found in Solr, you will need to store the same unique key in both the index and the database.
- Change definition of document identity.
  - Use cases change, and you may want to change the identity of the documents. For example, an RSS feed for videos might change to give different entries for the same video in different sizes. You may decide that the different entries are really the same document. There is a saying in database design:*data sticks where it lands*. Once you store data in some format and container, it is very hard to change this decision. By adding a layer of indirection in the SOLR schema's identity, you give yourself the ability to change the innate identity of the document.
- Multiple queries about the same document, with document id saved for future reference.
- Delete documents. (Though you can also delete documents matching a query, rather than by unique key value.)
- If you use DistributedSearch, you need a unique key. As an added benefit, if the same document (determined by unique key) ends up indexed in multiple shards, then only one of the docs will get returned in user's query results.

### Use cases which require a unique key generated from data in the document

- Allow different database systems to create identity keys that work in other systems.
  - The documents may come from multiple sources, and be stored in multiple places. There may not be one convenient place in the indexing path to create a unique id. The different sources will need to separately implement the same algorithm. The key should be a short unique string (see UUID below).

## Unique key composition

There are different data sources available for the unique key:

- Text in the document
- UUID key generated from the time of insertion and a random number
- UUID data generated from data in the document

### Text field in the document

- In the blog RSS example above, the URL of each article. The field must be single-valued.
- It is **strongly** advised to use one of the un-analyzed types (e.g. string) for textual unique keys. While using a solr.TextField with analysis does not produce errors, it also won't do what you expect, namely use the output from the analysis chain as the unique key. The raw input before analysis is *still* used which leads to duplicate documents (e.g. docs with unique keys of 'id1' and 'ID1' will be two unique docs even if you have a Lowercase Filter in an analysis chain for the unique key). Any normalization of the unique key should be done on the client side before ingestion.

## UUID techniques

- UUID is short for Universal Unique IDentifier. The UUID standard RFC-4122 includes several types of UUID with different input formats. There is a UUID field type (called `UUIDField`) in Solr 1.4 which implements version 4. Fields are defined in the schema.xml file with:

```
<fieldType name="uuid" class="solr.UUIDField" indexed="true" />
```

  in Solr 4, this field must be populated via solr.UUIDUpdateProcessorFactory.

```
<field name="id" type="uuid" indexed="true" stored="true" required="true"/>
```

```
 <updateRequestProcessorChain name="uuid">
   <processor class="solr.UUIDUpdateProcessorFactory">
     <str name="fieldName">id</str>
   </processor>
   <processor class="solr.RunUpdateProcessorFactory" />
 </updateRequestProcessorChain>
```

- Due to low level changes to support SolrCloud, the uniqueKey field can no longer be populated via <copyField/> or <field default=...> in the schema.xml. Users wishing to have Solr automatically generate a uniqueKey value when adding documents should instead use an instance of solr.UUIDUpdateProcessorFactory in their update processor chain. See SOLR-2796 for more details.
- However, in Solr 3 it can be used as

```
<field name="id" type="uuid" indexed="true" stored="true" default="NEW"/>
```

  Also, the ExtractingRequestHandler automatically creates UUID version 4. You can also implement a UUID string from a cryptographic hash.

## Cryptographic hash

- A cryptographic hashing algorithm can be thought of as creating N very random bits from the input data. The MD5 algorithm create 128 bits. This means that 2 input data sets have a chance of 1 in 2^128 of creating the same MD5. There is a standard expression of this as 32 hexadecimal characters. RFC-1321. Several MD5 digest algorithm packages for various languages do not follow this standard. The UUID standard always includes the time at the creation of the UUID, which precludes some of the above use cases. You can cheat and ignore the clock requirement. It is best to use the UUID text format: *550e8400-e29b-41d4-a716-446655440000* instead of *550e8400e29b41d4a716446655440000*. (You will read many of these keys.) One advantage in using a crypto-generated unique key is that you can select a random subset of documents via wildcards. If the UUID data is saved as a string in the 32-character RFC format, 'd3adbe3fdeadb3e4deadbee4deadb3ef', the query "id:a*" will select a random 1/16 of the entire document set. "id:aa*" selects 1/256 of the document set, again very randomly. Statistical analysis and data extraction projects can use this to select small subsets instead of walking the entire index.