WhyUseSolr

Assuming the user has a relational DB, why use Solr? If your use case requires a person to type words into a search box, you want a text search engine like Solr.

Databases and Solr have complementary strengths and weaknesses. SQL supports very simple wildcard-based text search with some simple normalization like matching upper case to lower case. The problem is that these are full table scans. In Solr all searchable words are stored in an "inverse index", which searches orders of magnitude faster. For a more complete description of Solr's features, see http://lucene.apache.org/solr/features.html.

	Solr v.s. Relational Database	
Lucene	Solr	Relational DB
Text Search	Fast and sophisticated	Minimal and slow
Features	Few, targeted to text search	Many
Deployment Complexity	Medium	Medium
Administration Tools	Minimal open source projects	Many open source & commercial
Monitoring Tools	Weak	Very Strong
Scaling Tools	Automated, medium scale	Large scale
Support Availability	Weak	Strong
Schema Flexibility	Must in general rebuild	Changes immediately visible
Indexing Speed	Slow	Faster and adjustable
Query Speed	Text search is fast & predictable	Very dependent on design & use case
Row Addition/Extraction Speed	Slow	Fast
Partial Record Modification	No	Yes
Time to visibility after addition	Slow	Immediate
Access to internal data structures	High	None
Technical knowledge required	Java (minimal), web server deployment, IT	SQL, DB-specific factors, IT
Regular maintenance tasks		

From a database perspective, a Lucene index can be thought of as one DB table with very fast lookups and interesting enhancements for text search. This index is relatively expensive in space and creation time. Solr wraps this API with a full-featured front end, providing these additions:

- schema design and text processing features that match most Lucene deployments
- clean deployment as a web service for indexing and searching
- convenient scalability across multiple servers
- learning curve & adoption improvement of ~2 orders of magnitude