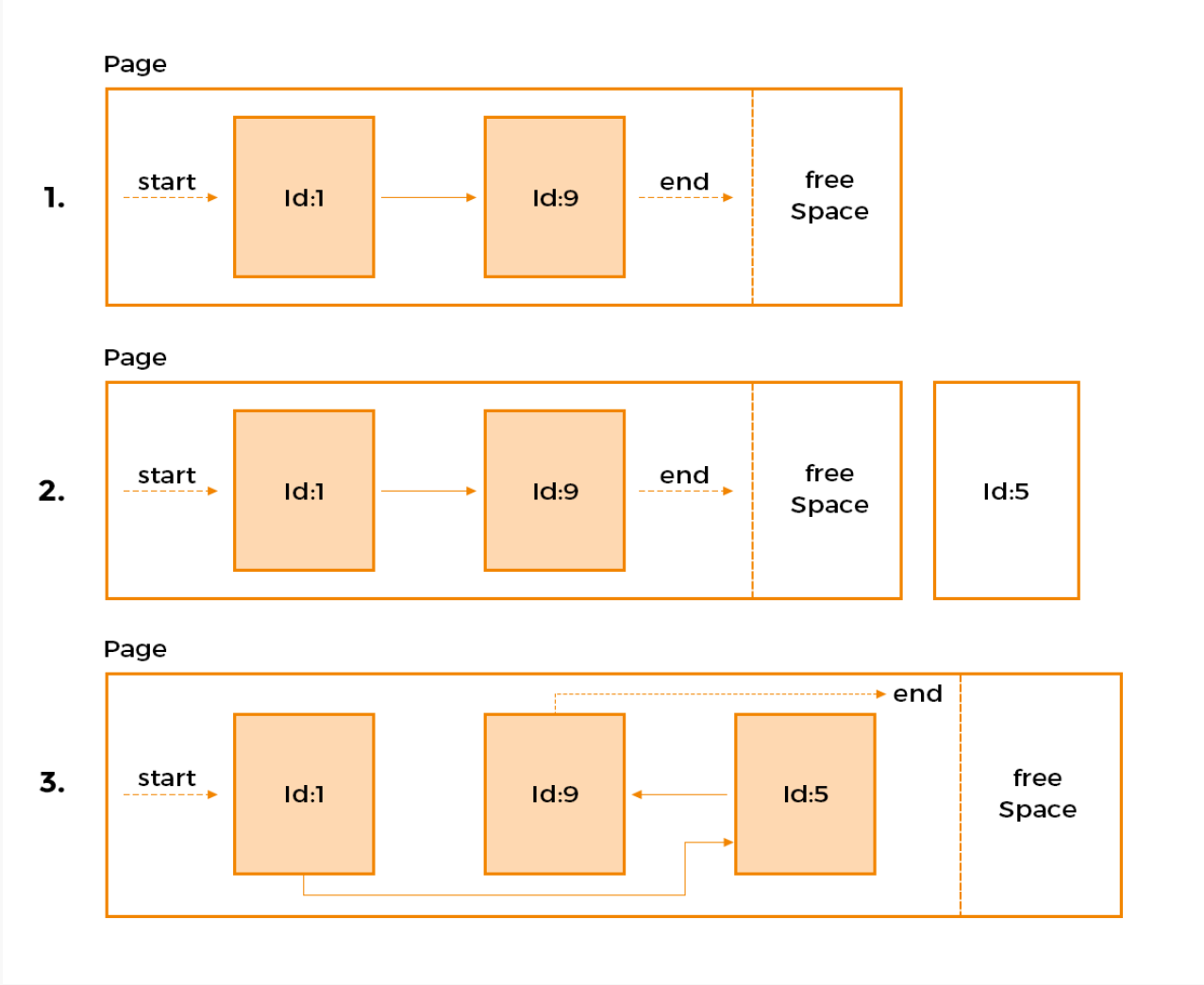
... continued

This lesson continues the discussion on indexes.

We have learned that pages form the leaf nodes of the tree stored on the disk. Now we'll see how rows are stored within each page.

Page

The pages of the B-tree split and merge as required. If a page is full and a new key is inserted, the existing page splits. Similarly, if enough rows are deleted from a page it may get merged with another. There are other intricate details around how pages are managed, but they aren't important for this introductory course on MySQL. Within a leaf-node page, the records or rows exist as a singly linked-list. The linked list enforces the index order on the rows. A new row is added to the available free space within the page. The existing records in the page aren't moved around in case the new row appears between existing rows in index order. Instead the new row is placed in the free space available within the page and the linked list pointers are manipulated to fix the order. This is depicted below:



A clustered index doesn't imply that the data rows are contiguously stored on the hard disk, or in other words, the pages aren't contiguous on the physical disk. The location on the disk where a row gets written to is the realm of the OS. A clustered index only ensures that the physical and logical order the rows appear in is the same.

Every table is stored as a clustered index with the primary key as the sort key in MySQL **when the database engine is selected as InnoDB**. A database engine is the software module that a database management system uses to create, read, update, and delete data from a database. In case of MySQL, we have InnoDB and MyISAM as examples of two popular storage engines. You can view the available storage engines as follows:

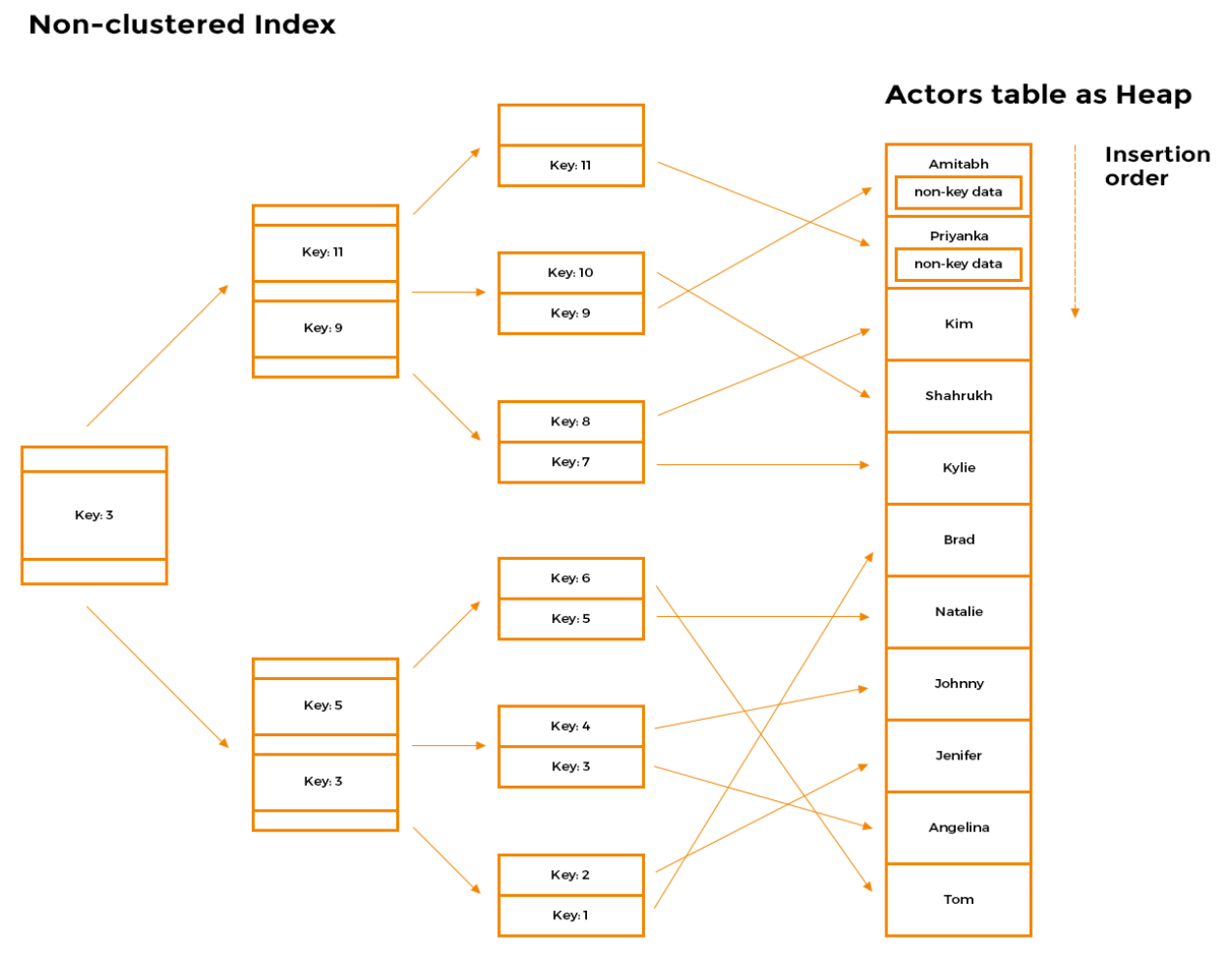
```
SHOW ENGINES;
```

MySQL creates a clustered index on the primary key. If no primary key is defined it looks for the first UNIQUE index with all the columns that form the key set as NOT NULL. If no UNIQUE index is available, MySQL

generates a hidden clustered index named **GEN_CLUST_INDEX** on a synthetic column containing row ID values. The rows are ordered by the ID that gets assigned to each row. This also implies that there can only be one clustered index per table as the table rows can only be arranged in one order on the disk. All other indexes are secondary indexes.

Non-clustered Index

In the case of the clustered index, we saw that leaf-nodes consisted of pages that contained the **actual** data. On the contrary, in a non-clustered index, the leaf-nodes don't hold the actual data, rather, a pointer to data stored elsewhere on the disk. If the selected database engine is **MyISAM** then the rows aren't stored in sorted order; rather they appear as a heap without any ordering. Such a table is called a **heap-table** since it contains an unordered pile of data. The MyISAM database engine is based on ISAM (Indexed Sequential Access Method), an indexing algorithm developed by IBM that allows retrieving information from large sets of data in a fast way. In the image below, the Actors table appears as a heap and a non-clustered index contains pointers to rows.



The rows appear in the insertion order when the engine is MyISAM. All indexes are secondary indexes when using MyISAM as the database

engine, even the primary index is a unique non-clustered index named

primary. However, in case of InnoDB, the secondary index's leaf nodes don't point to the rows as they do in MyISAM. Remember that pages can be split and merged, and the rows can be physically moved around on the disk.

This could be catastrophic if the leaf nodes of an InnoDB secondary index pointed to row locations on disk. To avoid the cost of rewriting, the secondary index's leaf nodes primary key values are stored on the leaf node rather than pointers to rows. This means we first seek the secondary index, get the primary key, and then use that primary key to navigate the primary index to locate the row. In contrast, a MyISAM secondary index can get the row data as soon as it reaches the leaf node of its index.

Cost of Indexing

An index doesn't come for free. For one, it takes up additional disk space, and two, it needs to be modified whenever an insert or an update is made to the table.