Indexing Recap

Three possible index organizations:

- primary – ordered by same field(s) as file, and field(s) are a key
- clustering – ordered by same field(s) as file, but field(s) not a key
- secondary – ordered by different field(s) than file

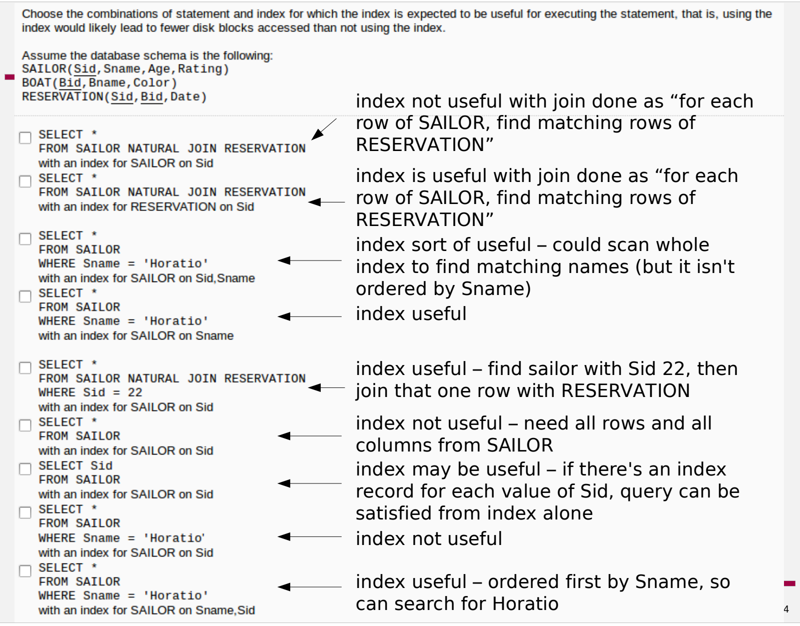
Performance:

- O($\log_2 b_i$ + s) for searching on indexing field(s)
  - s = number of matching records
- primary is better than clustering is better than secondary
  - but there can be at most one primary or clustering index per file
- multilevel index reduces search time from O($\log_2 b_i$) to O($\log_{bfr_i} b_i$)
- indexes are less beneficial for small tables and queries that match most of the rows

Uses for Indexes

MySQL uses indexes to speed up:

- finding rows matching a WHERE clause
- eliminating rows from consideration
- retrieving rows from other tables when performing joins
- sorting or grouping a table
- query evaluation
  - finding min or max of an indexed column
  - if all values needed are present in index



Choose the combinations of statement and index for which the index is expected to be useful for executing the statement, that is, using the index would likely lead to fewer disk blocks accessed than not using the index.

Assume the database schema is the following:

```
SAILOR(Sid,Sname,Age,Rating)
BOAT(Bid,Bname,Color)
RESERVATION(Sid,Bid,Date)
```

☐
```
SELECT *
FROM SAILOR NATURAL JOIN RESERVATION
```
with an index for SAILOR on Sid

index not useful with join done as “for each row of SAILOR, find matching rows of RESERVATION”

☐
```
SELECT *
FROM SAILOR NATURAL JOIN RESERVATION
```
with an index for RESERVATION on Sid

index is useful with join done as “for each row of SAILOR, find matching rows of RESERVATION”

☐
```
SELECT *
FROM SAILOR
WHERE Sname = 'Horatio'
```
with an index for SAILOR on Sid,Sname

index sort of useful – could scan whole index to find matching names (but it isn't ordered by Sname)

☐
```
SELECT *
FROM SAILOR
WHERE Sname = 'Horatio'
```
with an index for SAILOR on Sname

index useful

☐
```
SELECT *
FROM SAILOR NATURAL JOIN RESERVATION
WHERE Sid = 22
```
with an index for SAILOR on Sid

index useful – find sailor with Sid 22, then join that one row with RESERVATION

☐
```
SELECT *
FROM SAILOR
```
with an index for SAILOR on Sid

index not useful – need all rows and all columns from SAILOR

☐
```
SELECT Sid
FROM SAILOR
```
with an index for SAILOR on Sid

index may be useful – if there's an index record for each value of Sid, query can be satisfied from index alone

☐
```
SELECT *
FROM SAILOR
WHERE Sname = 'Horatio'
```
with an index for SAILOR on Sid

index not useful

☐
```
SELECT *
FROM SAILOR
WHERE Sname = 'Horatio'
```
with an index for SAILOR on Sname,Sid

index useful – ordered first by Sname, so can search for Horatio

Automatically-Created and Required Indexes

Indexes are automatically created in several cases:

- for primary key
  - InnoDB also orders file by primary key
- for UNIQUE fields
  - index used to check uniqueness constraints
- for foreign keys
  - index created for foreign key (referencing columns) if no other index has the columns first and in the right order
    - so FKs can be checked without reading whole table

Indexes are required:

- for referenced columns in foreign keys
  - must create manually if an index doesn’t already exist (e.g. not primary key)

Creating Indexes

Indexes are specified as part of CREATE TABLE.

```
CREATE TABLE RESERVATION (
  Sid smallint(5) unsigned NOT NULL,
  Bid smallint(5) unsigned NOT NULL,
  Day date DEFAULT NULL,
  PRIMARY KEY (Sid,Bid),          ← index on primary key
  KEY fk_RESERVATION_1 (Bid),     ↘ indexes on foreign keys
  KEY fk_RESERVATION_2 (Sid),     ↗
  INDEX RESERVATION_day (Day),    ← another index, on Day
  CONSTRAINT fk_RESERVATION_1
    FOREIGN KEY (Bid) REFERENCES BOAT (Bid)
    ON UPDATE CASCADE,
  CONSTRAINT fk_RESERVATION_2
    FOREIGN KEY (Sid) REFERENCES SAILOR (Sid)
    ON UPDATE CASCADE
) ENGINE=InnoDB;
```

KEY is synonym for INDEX

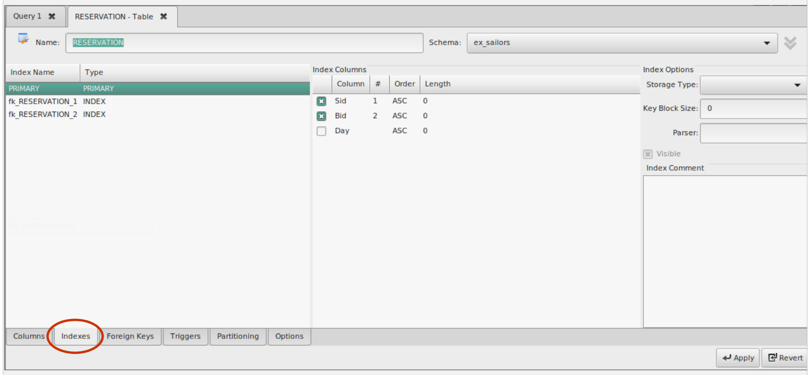
Viewing Indexes

Viewing indexes on a table:

```
SHOW INDEX FROM tbl_name;
```

Indexes in MySQL Workbench

- Indexes tab in table editor (create/alter table)



Name: RESERVATION   Schema: ex_sailors

Index Name	Type
PRIMARY	PRIMARY
fk_RESERVATION_1	INDEX
fk_RESERVATION_2	INDEX

Index Columns

	Column	#	Order	Length
☒	Sid	1	ASC	0
☒	Bid	2	ASC	0
☐	Day		ASC	0

Index Options

Storage Type:
Key Block Size: 0
Parser:
☒ Visible
Index Comment

Insert / Delete / Modify

Indexes have a cost.

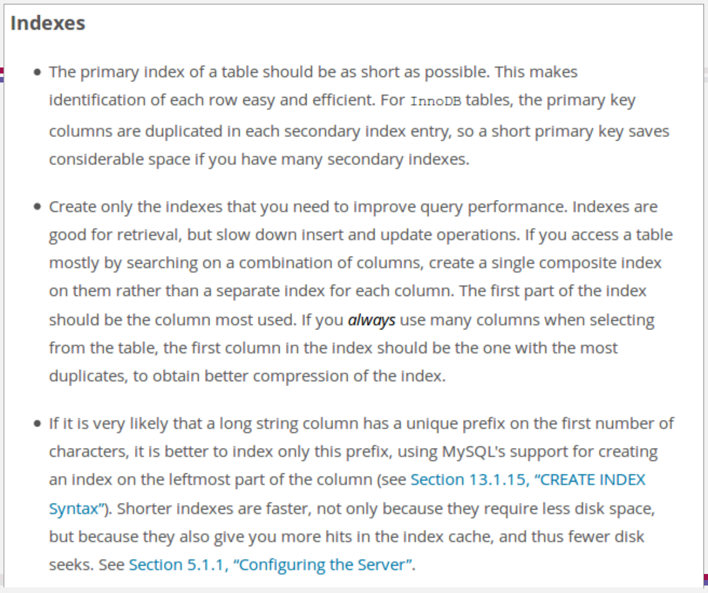
- space
- time (updates)

Index is sorted by indexing field, so insert / delete / modify potentially involves lots of shifting.

  – may also have to update block pointers if there is shifting in the data file

Can mitigate update costs (at the expense of space):

- deletion markers
- leaving extra space for insertion



Indexes

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient. For `InnoDB` tables, the primary key columns are duplicated in each secondary index entry, so a short primary key saves considerable space if you have many secondary indexes.

- Create only the indexes that you need to improve query performance. Indexes are good for retrieval, but slow down insert and update operations. If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used. If you ***always*** use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.

- If it is very likely that a long string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see Section 13.1.15, “CREATE INDEX Syntax”). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See Section 5.1.1, “Configuring the Server”.

https://dev.mysql.com/doc/refman/8.0/en/data-size.html#data-size-indexes