Prof. DI Dr. Erich Gams

Datenbanken

Indizes in MySQL

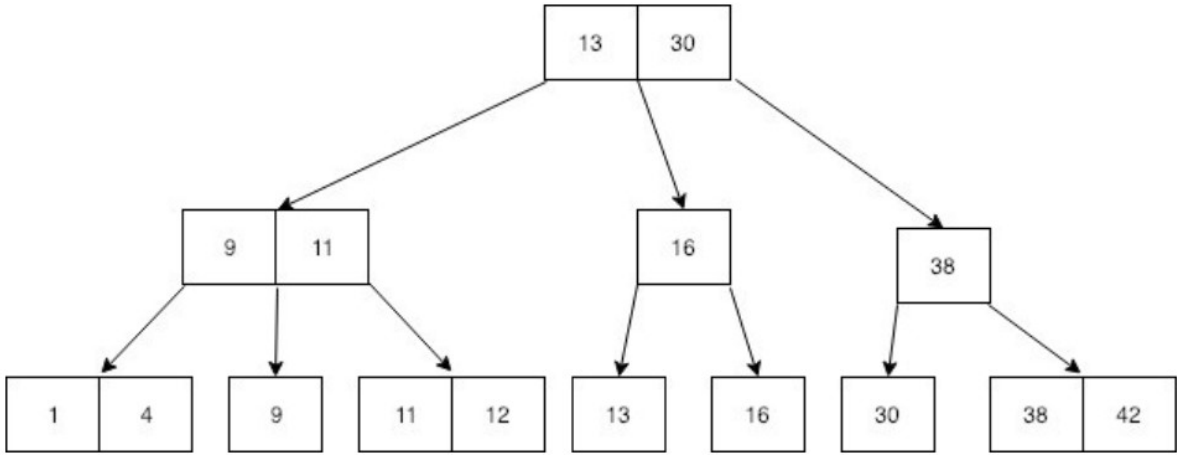
informationssysteme htl-wels

Was sind Indizes?

- Although a query can be fast, the database has to scan all the rows of the table until it finds the row.
- If the table has millions of rows, without an index, the data retrieval would take a lot of time to return the result.
- An index is a data structure such as B-Tree that improves the speed of data retrieval on a table at the cost of additional writes and storage to maintain it.
- The query optimizer may use indexes to quickly locate data without having to scan every row in a table for a given query.

B-tree – Search time

Type	Insertion	Deletion	Lookup
Unsorted Array	O(1)	O(n)	O(n)
Sorted Array	O(n)	O(n)	O(log(n))
B-tree	O(log(n))	O(log(n))	O(log(n))



https://dzone.com/articles/database-btree-indexing-in-sqlite

Create an Index

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 INT NOT NULL,
    c3 INT NOT NULL,
    c4 VARCHAR(10),
    INDEX (c2,c3)
);
```

```
CREATE INDEX idx_c4 ON t(c4);
```

Create an index

- Default: BTREE

Storage Engine	Allowed Index Types
InnoDB	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH, BTREE

Explain

- Example Query

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    jobTitle = 'Sales Rep';
```

Using Explain:

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ALL	NULL	NULL	NULL	NULL	23	10.00	Using where

Explain

```
CREATE INDEX jobTitle ON employees(jobTitle);
```

```
EXPLAIN SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    jobTitle = 'Sales Rep';
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ref	jobTitle	jobTitle	52	const	17	100.00	NULL

Indizes anzeigen/löschen

```
SHOW INDEXES FROM employees;
```

```
DROP INDEX index_name ON table_name
```

Algorithm

```
ALGORITHM [=] {DEFAULT|INPLACE|COPY}
```

- COPY
  - The table is copied to the new table row by row, the DROP INDEX is then performed on the copy of the original table.
  - The concurrent data manipulation statements such as INSERT and UPDATE are not permitted.
- INPLACE
  - The table is rebuilt in place instead of copied to the new one.
  - MySQL issues an exclusive metadata lock on the table during the preparation and execution phases of the index removal operation. This algorithm allows for concurrent data manipulation statements

LOCK Option

```
LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

- The lock_option controls the level of concurrent reads and writes on the table while the index is being removed.
- DEFAULT:
  - this allows you to have the maximum level of concurrency for a given algorithm. First, it allows concurrent reads and writes if supported. If not, allow concurrent reads if supported. If not, enforce exclusive access.
- NONE:
  - if the NONE is supported, you can have concurrent reads and writes. Otherwise, MySQL issues an error.
- SHARED:
  - if the SHARED is supported, you can have concurrent reads, but not writes. MySQL issues an error if the concurrent reads are not supported.
- EXCLUSIVE:
  - this enforces exclusive access

Example

```
CREATE TABLE leads(
    lead_id INT AUTO_INCREMENT,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    information_source VARCHAR(255),
    INDEX name(first_name,last_name),
    UNIQUE email(email),
    PRIMARY KEY(lead_id)
);
```

Examples

```
DROP INDEX name ON leads;
```

```
DROP INDEX email ON leads
ALGORITHM = INPLACE
LOCK = DEFAULT;
```

Show indices

- SHOW INDEXES FROM table_name;
- SHOW INDEXES FROM table_name IN database_name;
- SHOW INDEXES FROM database_name.table_name;
- SHOW INDEX IN table_name FROM database_name;

MySQL Prefix Index

- In case the columns are the string columns, the index will consume a lot of disk space and potentially slow down the INSERT operations.
- To address this issue, MySQL allows you to create an index for the leading part of the column values of the string columns using the following syntax:

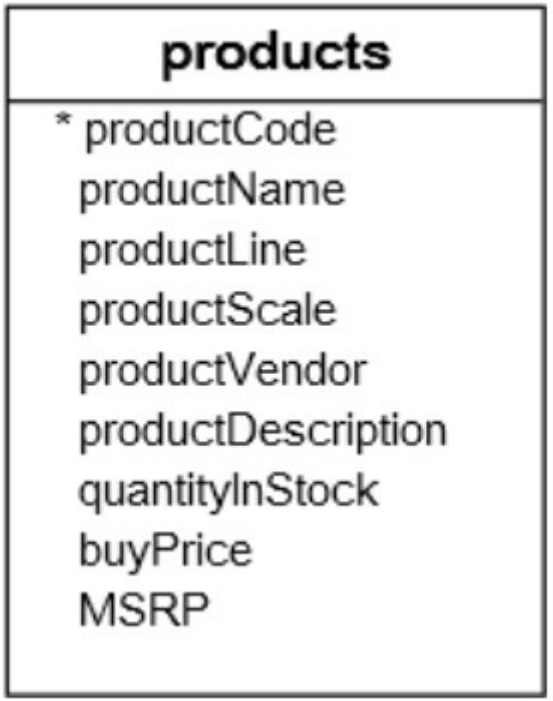
MySQL Prefix Index

```
CREATE TABLE table_name(
    column_list,
    INDEX(column_name(length))
);
```

```
CREATE INDEX index_name
ON table_name(column_name(length));
```

Example

```
SELECT
    productName,
    buyPrice,
    msrp
FROM
    products
WHERE
    productName LIKE '1970%';
```



Example

- Investigate the existing data

```
SELECT
    COUNT(*)
FROM
    products;
```

If we use the first 20 characters of the product name for the index, all product names are unique

```
SELECT
   COUNT(DISTINCT LEFT(productName, 20)) unique_rows
FROM
   products;
```

Example: Index creation

```
EXPLAIN SELECT
    productName,
    buyPrice,
    msrp
FROM
    products
WHERE
    productName LIKE '1970%';
```

```
CREATE INDEX idx_productname
ON products(productName(20));
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	products	NULL	range	idx_productname	idx_productname	22	NULL	4	100.00	Using where

MySQL composite index

- MySQL allows you to create a composite or multi-column index that consists of up to 16 columns.
- The query optimizer uses the composite indexes for queries that test all columns in the index, or queries that test the first columns, the first two columns, and so on.

MySQL composite index

```
CREATE TABLE table_name (
    c1 data_type PRIMARY KEY,
    c2 data_type,
    c3 data_type,
    c4 data_type,
    INDEX index_name (c2,c3,c4)
);
```

```
CREATE INDEX index_name
ON table_name(c2,c3,c4);
```

(c2)

(c2,c3)

(c2,c3,c4)

MySQL composite index

```
SELECT
    *
FROM
    table_name
WHERE
    c2 = v2;
```

```
SELECT
    *
FROM
    table_name
WHERE
    c2 = v2 AND
    c3 = v3;
```

```
SELECT
    *
FROM
    table_name
WHERE
    c2 = v2 AND
    c3 = v3, AND
    c4 = v3;
```

Invisible Indices

- The invisible indexes allow you to mark indexes as unavailable=invisible for the query optimizer.
- MySQL maintains the invisible indexes and keeps them up to date when the data in the columns associated with the indexes changes.
- By default, indexes are visible.

Invisible Indices

```
CREATE INDEX index_name
ON table_name( c1, c2, ...) INVISIBLE;
```

```
ALTER TABLE table_name
ALTER INDEX index_name [VISIBLE | INVISIBLE];
```

Show invisible indices

```
SELECT
    index_name,
    is_visible
FROM
    information_schema.statistics
WHERE
    table_schema = ….
        AND table_name = ….;
```

Descending Index

- A descending index is an index that stores key values in the descending order.
- The query optimizer can take advantage of descending index when descending order is requested in the query.

```
CREATE TABLE t(
    a INT NOT NULL,
    b INT NOT NULL,
    INDEX a_asc_b_desc (a ASC, b DESC)
);
```

Example

```
CREATE TABLE t (
    a INT,
    b INT,
    INDEX a_asc_b_asc (a ASC , b ASC),
    INDEX a_asc_b_desc (a ASC , b DESC),
    INDEX a_desc_b_asc (a DESC , b ASC),
    INDEX a_desc_b_desc (a DESC , b DESC)
);
```

```
CREATE PROCEDURE insertSampleData(
    IN rowCount INT,
    IN low INT,
    IN high INT
)
BEGIN
    DECLARE counter INT DEFAULT 0;
    REPEAT
        SET counter := counter + 1;
        -- insert data
        INSERT INTO t(a,b)
        VALUES(
            ROUND((RAND() * (high-low))+high),
            ROUND((RAND() * (high-low))+high)
        );
    UNTIL counter >= rowCount
    END REPEAT;
END$$
```

Example

- The stored procedure inserts a number of rows (rowCount) with the values between low and high into the a and b columns of the t table.
- Let's insert 10,000 rows into the t table with the random values between 1 and 1000.

```
CALL insertSampleData(10000,1,1000);
```

Example

```
EXPLAIN SELECT
    *
FROM
    t
ORDER BY a , b; -- use index a_asc_b_asc
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	t	NULL	index	NULL	a_asc_b_asc	10	NULL	10157	100.00	Using index

MySQL USE INDEX hint

- In MySQL, when you submit an SQL query, the query optimizer will try to make an optimal query execution plan.
- One of the most important parameters for choosing which index to use is stored key distribution which is also known as cardinality.
- Run ANALYZE TABLE if the tables has been changed very often!

MySQL USE INDEX hint

- In addition, MySQL provides an alternative way that allows you to recommend the indexes that the query optimizer should by using an index hint called USE INDEX.

```
SELECT select_list
FROM table_name USE INDEX(index_list)
WHERE condition;
```

- The query optimizer may use it!

Force index

- In case the query optimizer ignores the index, you can use the FORCE INDEX hint to instruct it to use an index.
- For example, a query might request for products whose prices are between 10 and 80.
- If the statistics show that 80% of products have these price ranges, then it may decide that a full table scan is the most efficient.
- However, if statistics show that very few products have these price ranges, then reading an index followed by a table access could be faster and more efficient than a full table scan

Force index syntax

```
SELECT *
FROM table_name
FORCE INDEX (index_list)
WHERE condition;
```

Aufgabe

- Übungen in Moodle

Auf los geht's los ;-)