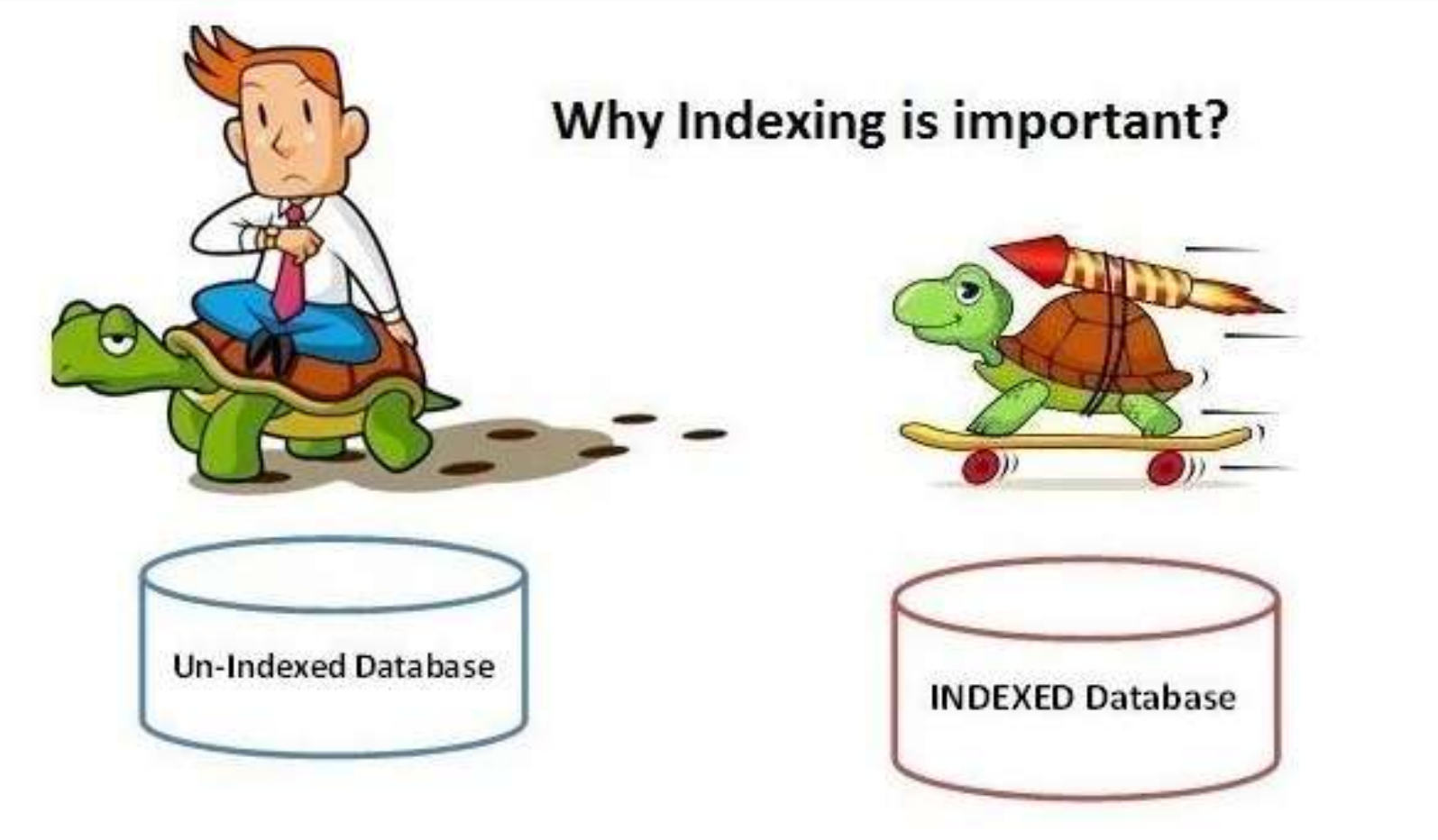
Index

NGUYEN Hong Phuong
Email: phuongnh@soict.hust.edu.vn
Site: https://users.soict.hust.edu.vn/phuongnh

Content

- What is Index?
- What is the index database used for?
- The structure of the index
- Types of indexes
- How to use index database effectively?


Why Indexing is important?
Un-Indexed Database
INDEXED Database

What is Index?

- A data structure is used to locate and fast access data in tables or views.
- One way to increase database query performance, by reducing the amount of access to memory during query execution
- SQL Server provides two types of indexes
  - Clustered
  - Non-clustered

What is the index database used for?

- Query: SELECT * FROM student WHERE last_name = 'May'
- If there is no index for the last_name column, the system will scan all the rows of the 'student' table to compare and retrieve the row that satisfies

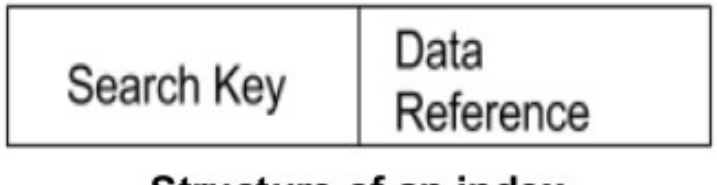
student

student_id	first_name	last_name	dob	gender	address	note	clazz_id
1234	David	Beckham	12/21/1997	Male	London, UK		1
1238	Theresa	May	08/06/1998	Female	London, UK		1
1452	David	Cameron	07/06/1997	Male	Bangor, UK		1
1497	Tony	Blair	03/01/1999	Male	Bath, UK		2
1516	John	Major	03/01/1998	Male	Bradford		2
1542	Margaret	Thatcher	05/08/1997	Female	Cambridge		2

- An index points to the address of data in a table, similar to a book's table of contents, making queries fast
- Index can be created for one or more columns in a table. Indexes are usually created by default for primary keys, foreign keys. In addition, it is also possible to create additional indexes for columns if needed.

The structure of the index

- Index includes:
  - Search Key column: contains a copy of the indexed column's values
  - Data Reference column: contains the pointer to the address of the record with the corresponding index column value



Structure of an index

Types of indexes

- B-tree
- Hash

B-tree

- Usually, if you don't specify the index type, the default is to use B-Tree.
- Syntax:
  - Create index

```
CREATE INDEX id_index ON table_name
(column_name [, column_name…]) USING BTREE;
ALTER TABLE table_name ADD INDEX id_index
(column_name [, column_name…])
```

  - Delete the index

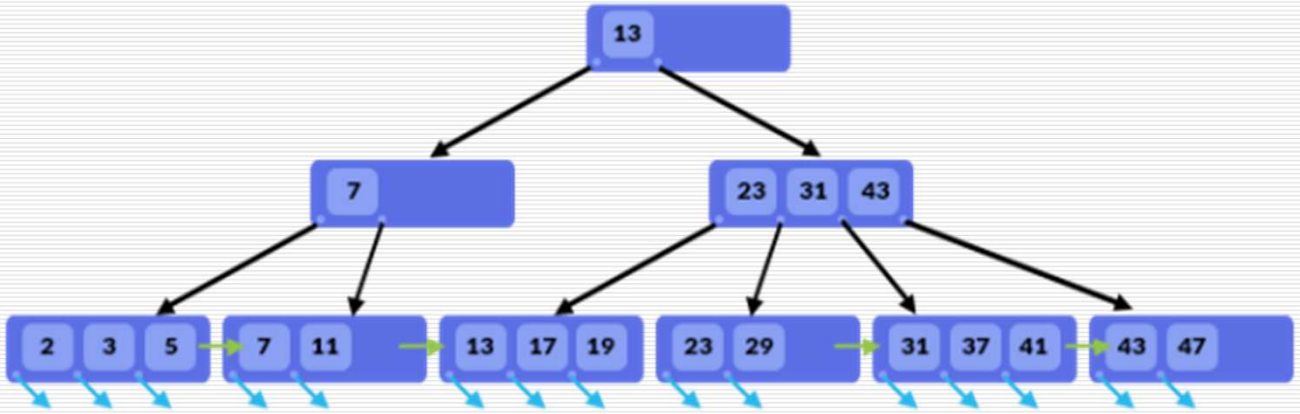
```
DROP INDEX index_name ON table_name
```

B-tree

- Features of B-Tree Index:
  - Index data is organized and stored in the form of tree, ie root, branch, leaf.
  - The values of the organized nodes increase from left to right.
  - The B-tree index is used in comparison expressions: =,>,> =, <, <=, BETWEEN, and LIKE. ⇒ Possible good for the ORDER BY statement

B-tree

- When searching for data, it will not scan the entire table. A search in B-Tree is a process that starts from the root node and searches for the branch and leaf, until finding all data satisfying the query condition.



Hash

- Hash index is based on Hash Function algorithm. Corresponding to each block of data, index will generate a bucket key (hash value) to distinguish.
- Syntax:
  - Create index

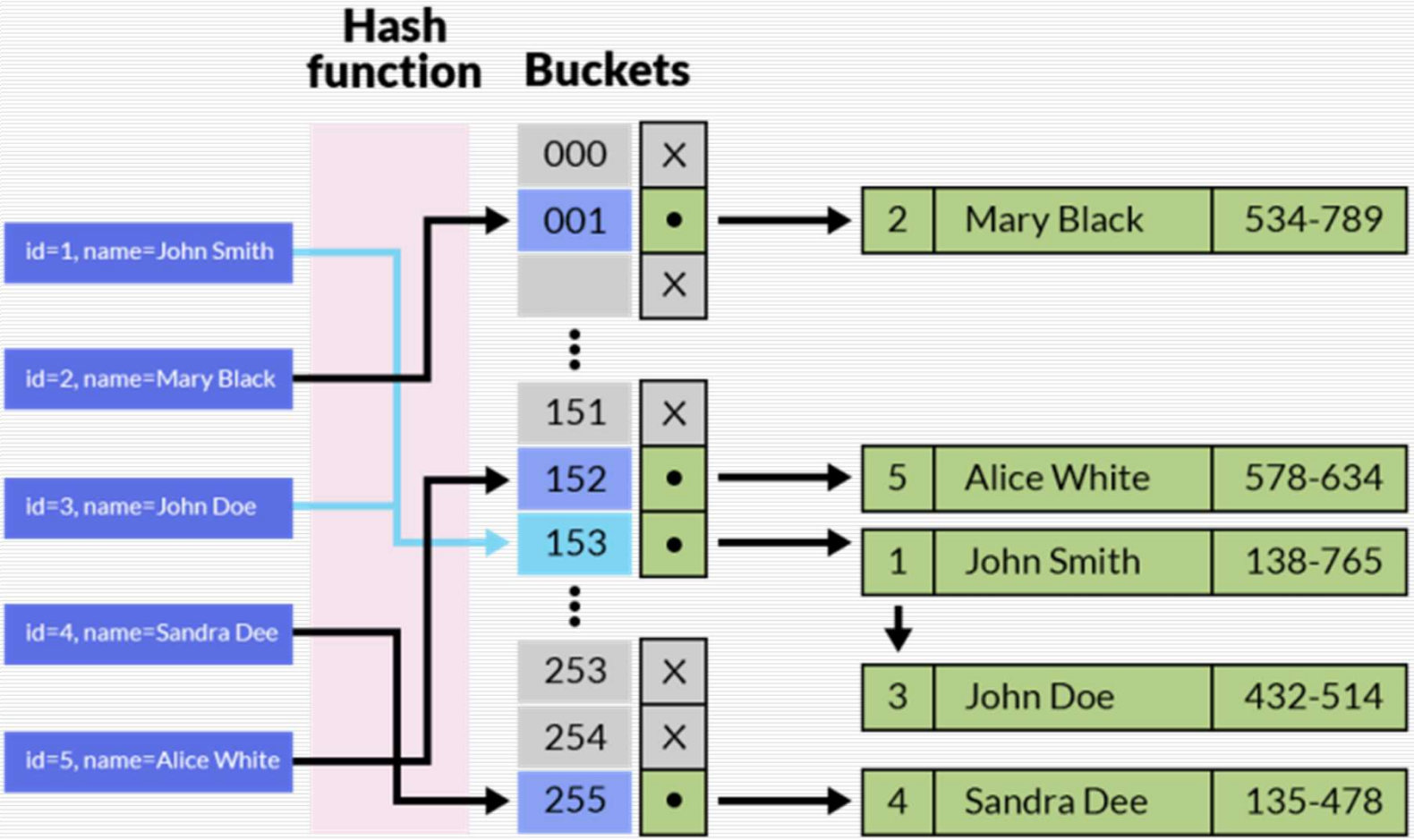
```
CREATE INDEX id_index
ON table_name(column_name [, column_name…]) USING HASH;

ALTER TABLE table_name
ADD INDEX id_index(column_name [, column_name…]) USING HASH;
```

Hash

- The features of Hash Index:
  - Hash index should be used only in operator '=' and '<>'. Do not use for operators to find a range of values such as> or <.
  - The ORDER BY operator cannot be optimized using the Hash index because it cannot find the next element in the Order.
  - Hash is faster than B-Tree type.

Hash


Hash function
Buckets
id=1, name=John Smith
id=2, name=Mary Black
id=3, name=John Doe
id=4, name=Sandra Dee
id=5, name=Alice White
000
001
151
152
153
253
254
255
2 Mary Black 534-789
5 Alice White 578-634
1 John Smith 138-765
3 John Doe 432-514
4 Sandra Dee 135-478

Storage Engine

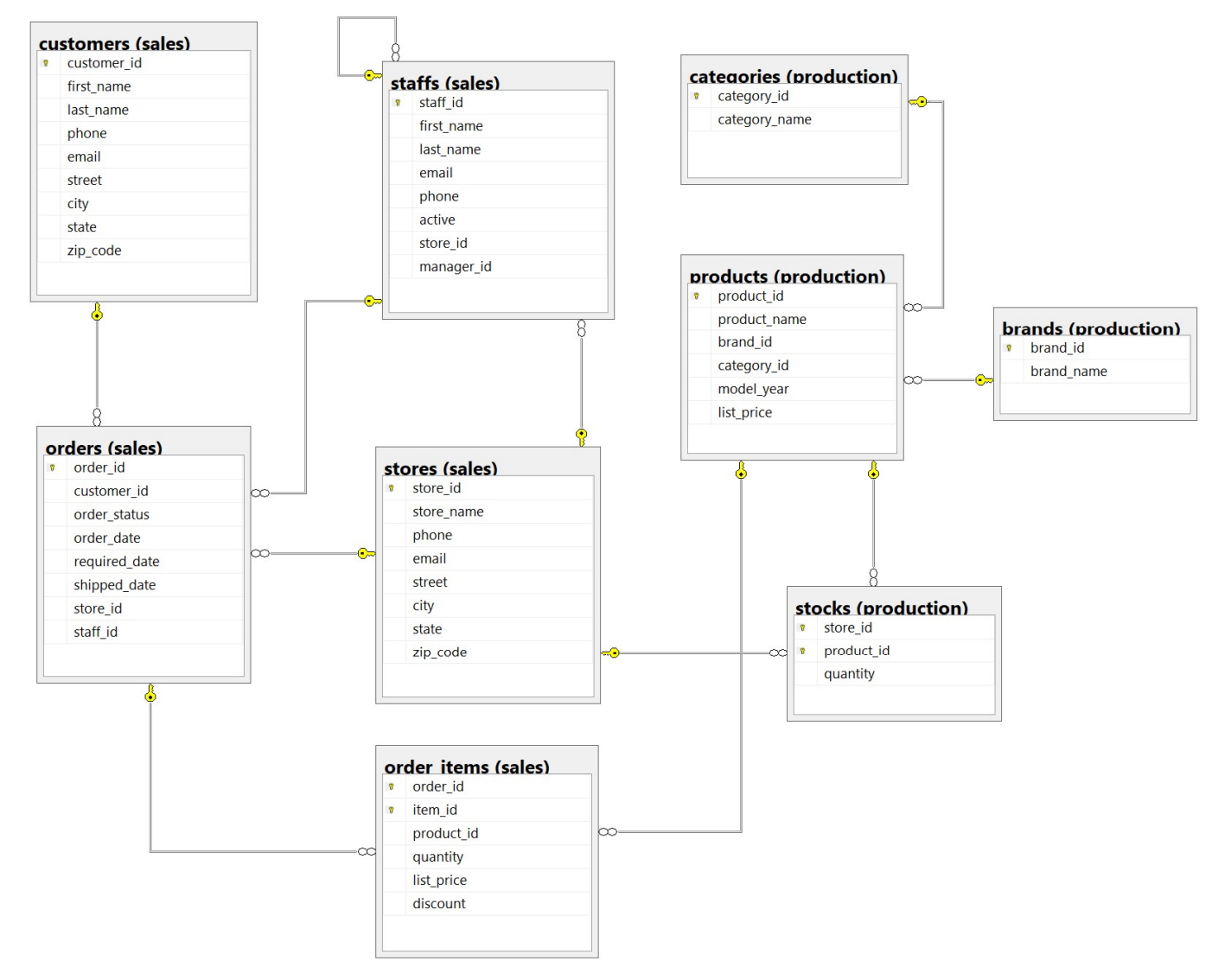
- Choosing the index of B-Tree or Hash type, apart from the purpose of use, also depends on whether or not the Storage Engine supports the type of index.
- Storage Engine and index types are supported
  - InnoDB BTREE
  - MyISAM BTREE
  - MEMORY/HEAP HASH, BTREE
  - NDB HASH, BTREE

How to use Index Database effectively?

- Should index the columns which are used in WHERE, JOIN and ORDER BY
- Do not use index in the following cases:
  - Small tables, containing little data
  - Tables are updated and data inserted regularly
  - Columns that contain so many NULL values
  - Columns are regularly updated
- Although index plays an important role in query optimization and speeding up in searching in the database, its downside is that it takes up more memory to store. Therefore, indexing of columns should be carefully considered

Practice with SQL Server

- BikeStores sample database



Create a new 'production.parts' table

```
CREATE TABLE production.parts(
    part_id   INT NOT NULL,
    part_name VARCHAR(100)
);

INSERT INTO
    production.parts(part_id, part_name)
VALUES
    (1,'Frame'),
    (2,'Head Tube'),
    (3,'Handlebar Grip'),
    (4,'Shock Absorber'),
    (5,'Fork');
```

- The 'parts' table doesn't have a PK, so the records are stored in an ordered structure called a heap.
- The statement finds records with id 5
- See execution plan estimates in SQL Server Management Studio
  - Select Display Estimated Execution Plan (press Ctrl + L)

```
SELECT
    part_id, part_name
FROM
    production.parts
WHERE
    part_id = 5;
```

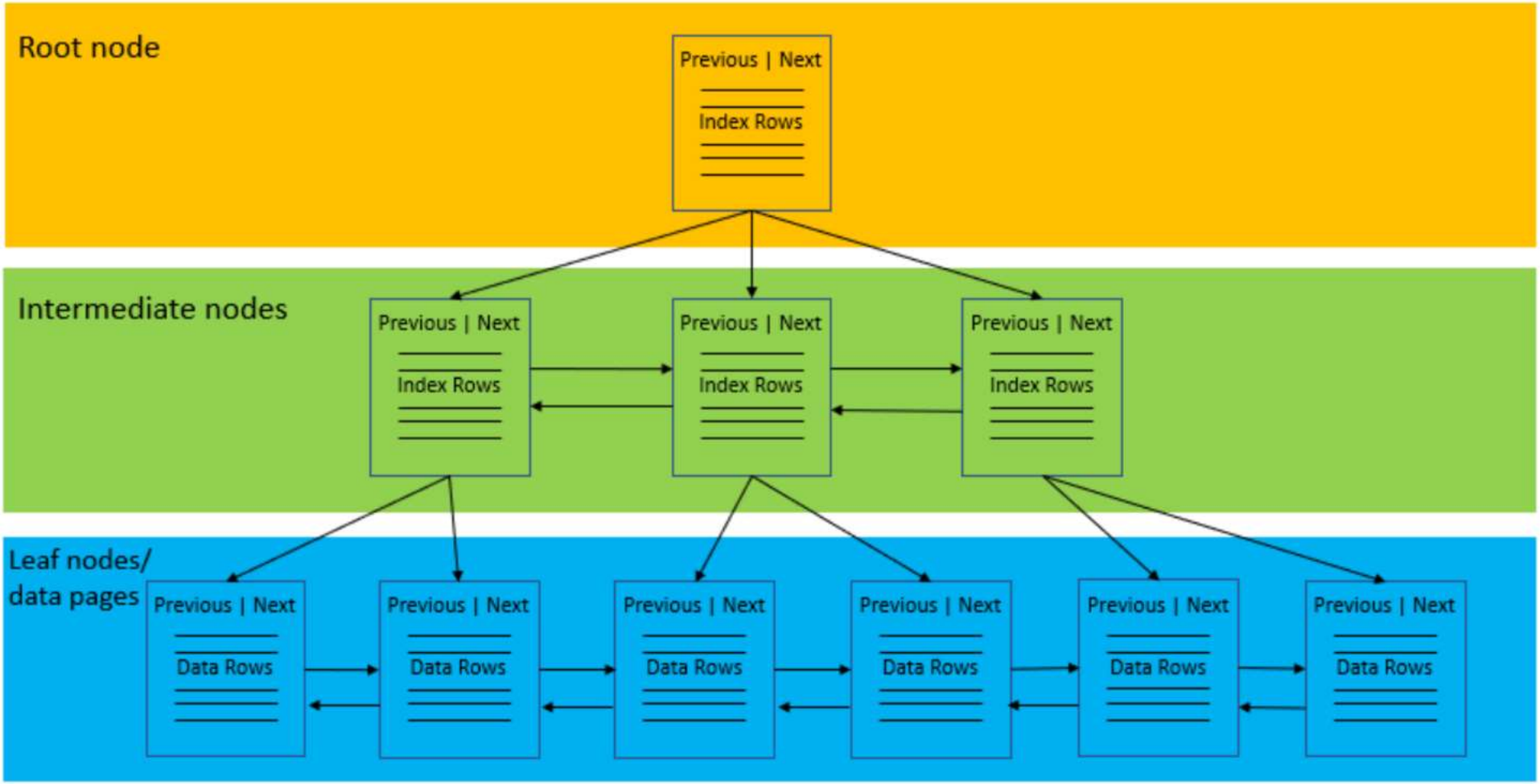
Two types of indexes in SQL Server

- Clustered index
- Non-clustered index

Clustered index

- Stores the records in an ordered structure based on its key value
- Each table has only one clustered index because the data records can only be arranged in one order
- A table that has a clustered index is called a clustered table
- Data in clustered index are organized under the form of B-tree

Clustered index

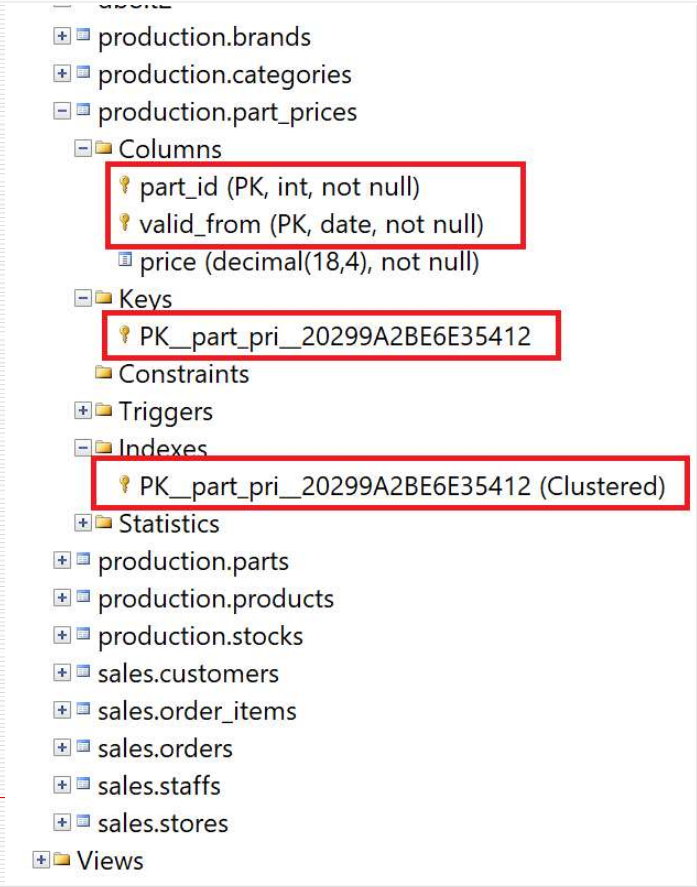


Clustered index

- The root node and the intermediate node contain index pages for storing the indexes of the records
- The leaf node contains the data pages of the table.
- Pages within each level of the index are linked in a double linked list structure

Clustered index

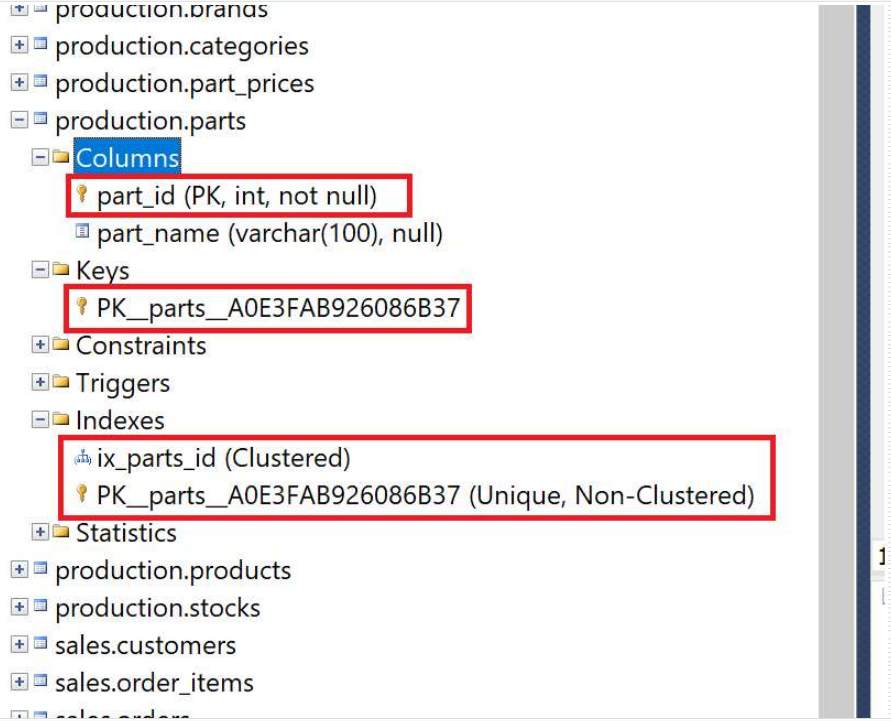
- When creating a table with the primary key PK, SQL Server automatically creates a clustered index on the PK columns.
- The statement creates a 'part_prices' table with a PK consisting of 2 columns:




```
CREATE TABLE production.part_prices(
    part_id int,
    valid_from date,
    price decimal(18,4) not null,
    PRIMARY KEY(part_id, valid_from)
);
```

Clustered index

- If you add the primary key to a table that already has a clustered index, SQL Server forces the PK to use the non-clustered index




```
ALTER TABLE
production.parts
ADD PRIMARY KEY(part_id);
```

Clustered index

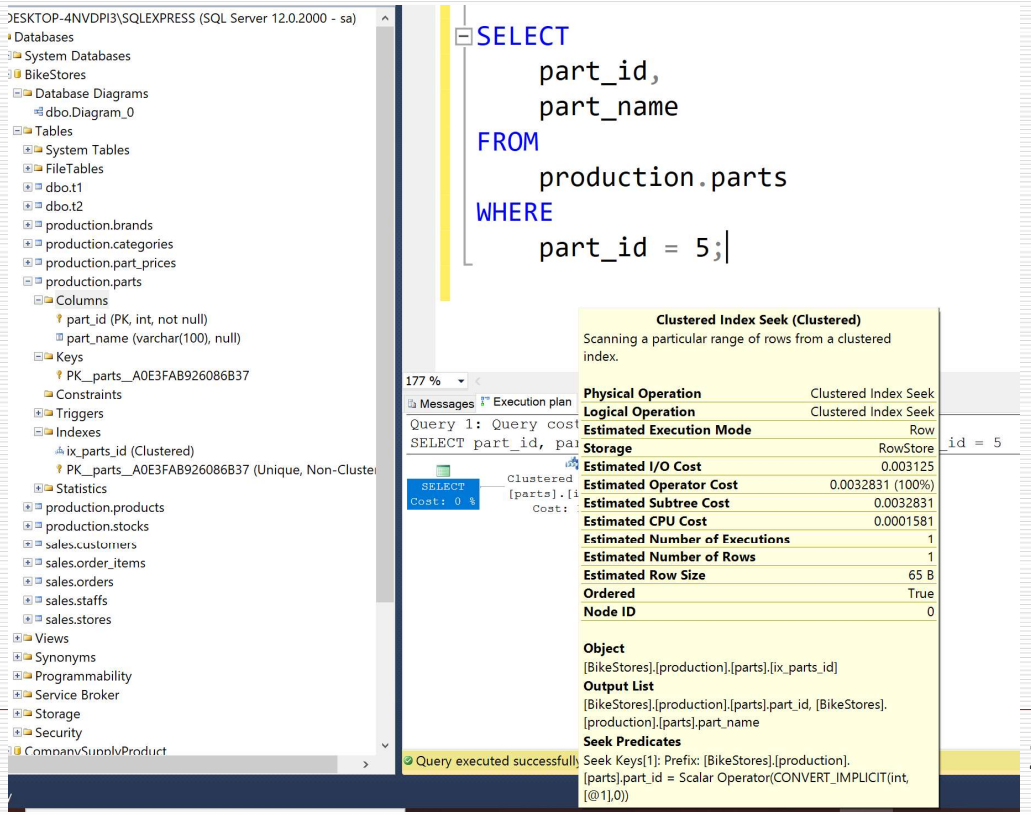
- Create clustered index
  - In the case the table does not have a PK

```
CREATE CLUSTERED INDEX index_name
ON schema_name.table_name (column_list);
```

```
CREATE CLUSTERED INDEX ix_parts_id
ON production.parts (part_id);
```

- Query

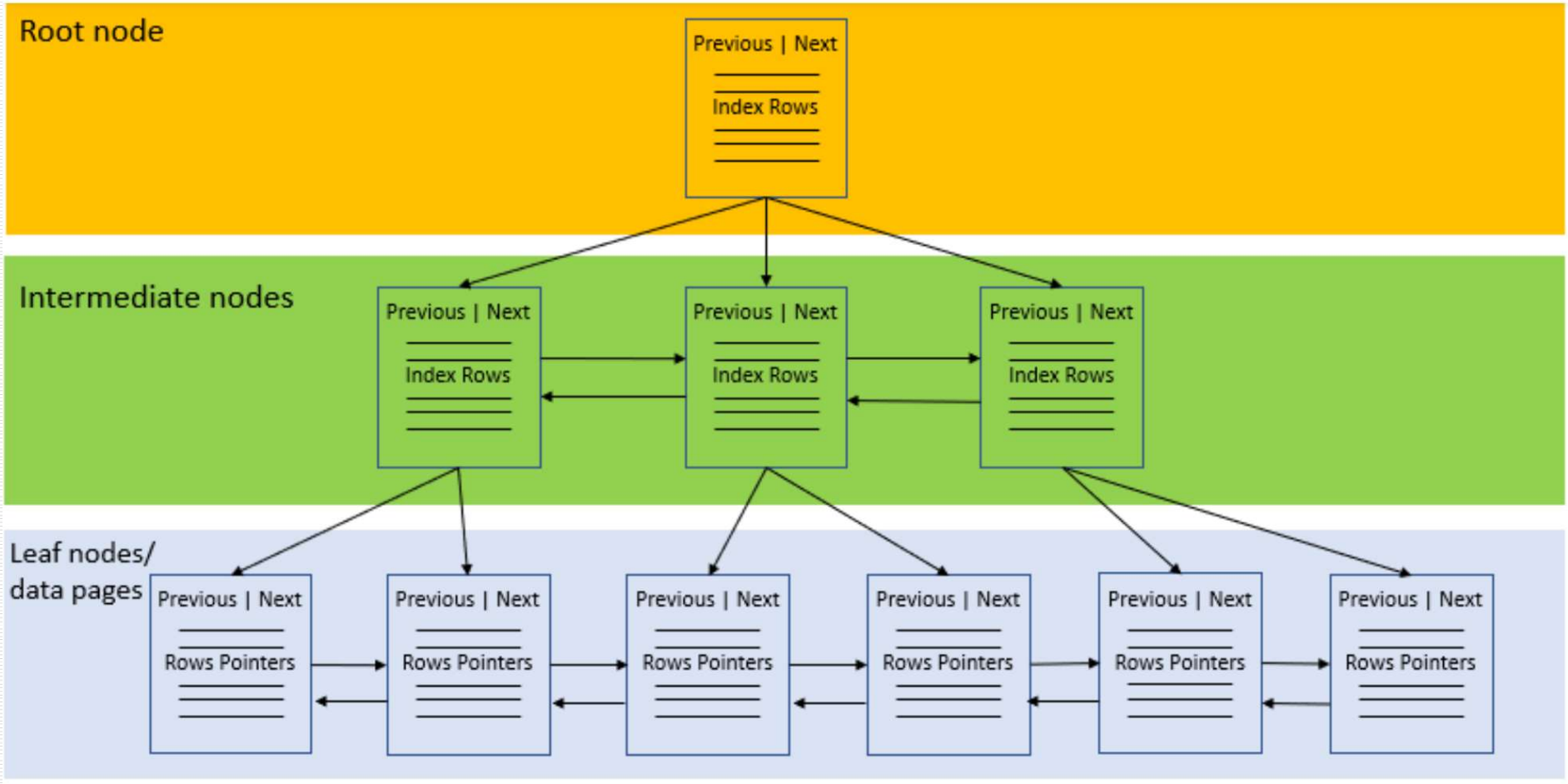
```
SELECT part_id,
part_name
FROM production.parts
WHERE part_id = 5;
```



Non-clustered index

- Data structure that improves the speed of retrieving data from tables
- Different from clustered index: Sorts and stores data separately from records in the table.
- Is the data copy of selected columns from a linked table.
- Use a B-tree structure to organize data
- A table can have one or more non-clustered indexes. Each non-clustered index can consist of one or more table columns.

Non-clustered index



Non-clustered index

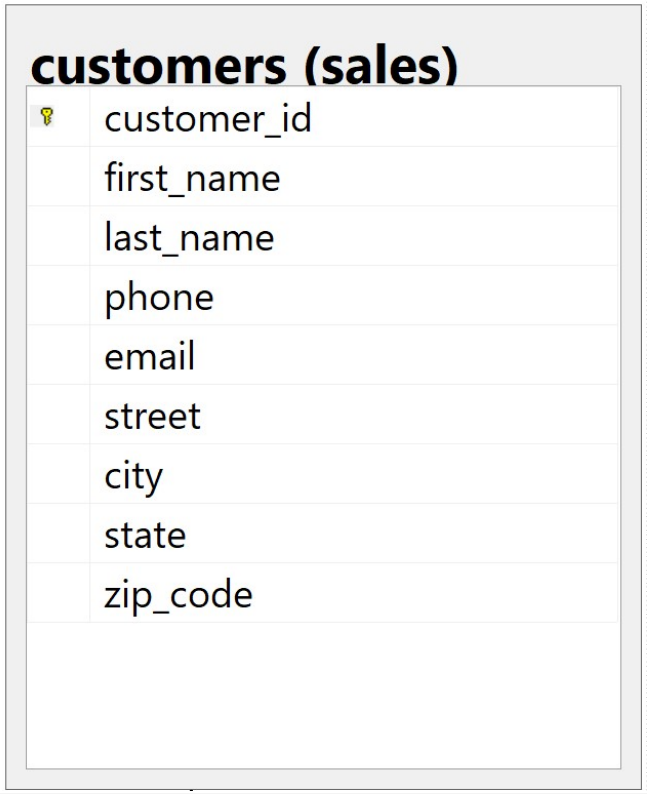
- In addition to storing the index key values, the leaf nodes also store pointers to the records containing the key values.
- These record pointers are also known as row locators.

Non-clustered index

- Create non-clustered index

```
CREATE [NONCLUSTERED] INDEX index_name
ON table_name(column_list);
```

- The 'customers' table is a clustered table because it has the customer_id PK



Non-clustered index

- Search for customers whose address is at 'Atwater'

```
SELECT customer_id, city
FROM sales.customers
WHERE city = 'Atwater';
```

- See execution plan estimation, the query optimizer scans the clustered index for records, since the 'customers' table doesn't have an index for the 'city' column.
- Type the following command, and then see the estimation again

```
CREATE INDEX ix_customers_city
ON sales.customers(city);
```

Non-clustered index

- Create non-clustered indexes for multiple columns
  - Find a customer with the last name 'Berg' and first name 'Monika'

```
SELECT customer_id, first_name, last_name
FROM sales.customers
WHERE last_name = 'Berg' AND first_name = 'Monika';
```

  - See execution plan estimation, type the following command, and run the above step again

```
CREATE INDEX ix_customers_name
ON sales.customers(last_name, first_name);
```

Rename index in SQL Server

- The statement uses the sp_rename stored procedure

```
EXEC sp_rename  index_name, new_index_name, N'INDEX';

EXEC sp_rename @objname = N'index_name', @newname =
N'new_index_name',  @objtype = N'INDEX';
```

  - Ví dụ:

```
EXEC sp_rename
    @objname = N'sales.customers.ix_customers_city',
    @newname = N'ix_cust_city' ,
    @objtype = N'INDEX';

EXEC sp_rename
    N'sales.customers.ix_customers_city',
    N'ix_cust_city' ,
    N'INDEX';
```

Rename index in SQL Server

- Or use SQL Server Management Studio, right click,….

Unique index in SQL Server

- Unique indexes can be one or more columns.
  - If a column, the values in the column are unique
  - If more than one column, the combination of values in these columns is unique
- Unique indexes can be clustered or non-clustered indexes
- Syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name(column_list);
```

Unique index

- For example, create a unique index for an email column
  - First of all, check to make sure there are no duplicate email addresses

```
SELECT email, COUNT(email)
FROM sales.customers
GROUP BY email
HAVING COUNT(email) > 1;
```

```
CREATE UNIQUE INDEX ix_cust_email
ON sales.customers(email);
```

Unique index

- Try creating a table with 2 columns, then create a unique index on both of those columns
- Then, insert the data
- Is it okay if applying a unique index on a column with multiple NULL values?
- Unique index vs. unique constraint

Disable indexing in SQL Server

- Before updating the table, disabling the index speed up this process

```
ALTER INDEX index_name
ON table_name
DISABLE;
```

- Disable all indexes

```
ALTER INDEX ALL ON table_name
DISABLE;
```

Disable indexing in SQL Server

- If an index is disabled, the optimizer will not use that index to plan the query execution
- Disabling indexing on a table, SQL Server retains index definition in metadata and index statistics in non-clustered indexes.
- Disable index on view, SQL Server will delete all index data
- If a clustered index of a table is disabled, the data of the table cannot be accessed using SELECT, INSERT, UPDATE, and DELETE until the clustered index is rebuilt / deleted.

Disable indexing in SQL Server

- Example:

```
ALTER INDEX ix_cust_city
ON sales.customers
DISABLE;

SELECT
    first_name,
    last_name,
    city
FROM
    sales.customers
WHERE
    city = 'San Jose';
```

```
ALTER INDEX ALL
ON sales.customers
DISABLE;

SELECT * FROM sales.customers;
```

Enable indexes in SQL Server

- After disabling the index for UPDATE, it is necessary to re-enable the index
  - The index needs to be rebuilt to reflect the new data in the table
- Use one of the following two commands
  - ALTER INDEX
  - DBCC DBREINDEX

Enable indexes in SQL Server

- ALTER INDEX and CREATE INDEX

```
ALTER INDEX index_name
ON table_name
REBUILD;

CREATE INDEX index_name
ON table_name(column_list)
WITH(DROP_EXISTING=ON)

ALTER INDEX ALL ON table_name
REBUILD;
```

Enable indexes in SQL Server

- DBCC DBREINDEX

```
DBCC DBREINDEX (table_name, index_name);

ALTER INDEX ALL ON sales.customers
REBUILD;
```

Delete indexes in SQL Server

- DROP INDEX

```
DROP INDEX [IF EXISTS] index_name
ON table_name;
```

- The DROP INDEX statement cannot delete indexes created by PK or a unique constraint
  - To remove indexes associated with these constraints, use the ALTER TABLE DROP CONSTRAINT command
- Remove multiple indexes from one/multiple tables, use the following command:

```
DROP INDEX [IF EXISTS]
    index_name1 ON table_name1,
    index_name2 ON table_name2,
    ...;
```

Filtered index in SQL Server

- Sometimes, it is inefficient to index all the records by a certain column, as it is only partially queried for a few of records of the whole table.
- A filtered index is a non-clustered index with an expression that specifies which records should be added to the index.
- Syntax:

```
CREATE INDEX index_name
ON table_name(column_list)
WHERE predicate;
```

Filtered index in SQL Server

- For example, using the 'customers' table, the phone column has so many NULL values

```
SELECT
    SUM(CASE
        WHEN phone IS NULL
        THEN 1
        ELSE 0
    END) AS [Has Phone],
    SUM(CASE
        WHEN phone IS NULL
        THEN 0
        ELSE 1
    END) AS [No Phone]
FROM sales.customers;
```

```
CREATE INDEX ix_cust_phone
ON sales.customers(phone)
WHERE phone IS NOT NULL;
```

```
SELECT
    first_name,
    last_name,
    phone
FROM sales.customers
WHERE phone = '(281) 363-3309';
```

Filtered index in SQL Server

- INCLUDE

```
CREATE INDEX ix_cust_phone
ON sales.customers(phone)
INCLUDE (first_name, last_name)
WHERE phone IS NOT NULL;
```