# DDL.
# Data Definition Language

Unit 9

# Introduction

- The DDL (Data Definition Language), is part of the SQL. It covers the statements orientated to define the database structure.
- That is why it is usually used by the database administrator, and sometimes by programmers.
- Its verbs are:
  - CREATE
  - DROP
  - ALTER
- The objects involved in the statements that we will learn are:
  - Databases
  - Tables
  - Indexes
  - Views

# DDL Database

A database is stored in several files:

- The main data file  .mdf
- Secundary data files .ndf
- Log files .ldf

# CREATE DATABASE

```
CREATE DATABASE DBname
   [ ON [ PRIMARY ] [ <file_desc> [ ,...n ]
   [ LOG ON { < file_desc > [ ,...n ] } ] ]
   [ COLLATE CollationName]
   [ WITH <external_access_option> ]] [;]
```

# CREATE DATABASE

<file_desc> ::=

(NAME = *LogicalFileName* ,

FILENAME = '*OSFileName*'

[ , SIZE = *size* [ KB | MB | GB | TB ] ]

[ , MAXSIZE = { *max_size* [ KB | MB | GB | TB ]

|UNLIMITED } ]

[ , FILEGROWTH = *growth_inc*

[ KB | MB | GB | TB | % ] ]

)

# CREATE DATABASE

COLLATE <CollationName>

< CollationName >:: =
CollationDesignator_ CaseSensitivity_AccentSensitivity

- CollationDesignator : Windows collation name, specifies the base collation rules.
- CaseSensitivity: CS specifies Case-Sensitive, CI specifies Case-Insensitive
- AccentSensitivity: AS specifies Accent-Sensitive, AI specifies Accent-Insensitive

Example:
COLLATE Modern_Spanish_CS_AI

# DROP DATABASE

Eliminate a database (data and structure).

DROP DATABASE { *DBname* } [ ,...n ] [;]

# ALTER DATABASE

Modify the properties of the database.

```
ALTER DATABASE DBname
{
        <FileChanges>
        | <GroupChanges>
        | <options>
        | MODIFY NAME = new_DBname
        | COLLATE collationName
}
[;]
```

# ALTER DATABASE

<FileChanges>::=
{ ADD FILE < file_desc > [ ,...*n* ]
[ TO FILEGROUP { *groupName*| *DEFAULT* } ]
| ADD LOG FILE < file_desc > [ ,...*n* ]
| REMOVE FILE *FileName*
| MODIFY FILE <file_desc>
<file_desc>::=
( NAME = *FileName [ , NEWNAME = new_FileName ]*
[ , FILENAME = 'OS_FileName' ]
[ , SIZE = *size [ KB | MB | GB | TB ] ]*
[ , MAXSIZE = {*max_size [K|MB|GB|TB]|UNLIMITED* } ]
[ , FILEGROWTH = *increment [ KB | MB | GB | TB| %]]*
[ , OFFLINE ])

# CREATE TABLE

CREATE TABLE [ DBName.][SchemaName.] TableName
( { <Column_def>
| < ComputedColumn_def> } [ ,...*n* ]
[ <table_constraint> ] [ ,...n ] ) [ ; ]

< Column_def > ::=
*ColName <Datatype>* [ NULL | NOT NULL *]*
[ COLLATE *CollationName* ]
[ [ CONSTRAINT *Constraint_name* ]
DEFAULT *constant_expr* ]
[ IDENTITY [( *seed, increment* )] ]
[ ROWGUIDCOL ]
[ <Col_constraint> [ ...*n* ] ]

# CREATE TABLE

< Col_constraint > ::= [ CONSTRAINT *Constraint_name]*
{ {PRIMARY KEY|UNIQUE }[CLUSTERED | NONCLUSTERED]
| [FOREIGN KEY] REFERENCES
[*SchemaName.] referenced_tbl_name [ ( ref_col) ]*
[ *ON DELETE { NO ACTION* | *CASCADE* | *SET NULL* | SET DEFAULT } ]
[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ] [ NOT FOR REPLICATION ]
| CHECK (*logical_expression)*
}

# CREATE TABLE

< table_constraint > ::=
[ CONSTRAINT *Constraint_name] {*
{PRIMARY KEY|UNIQUE } [CLUSTERED|NONCLUSTERED ]
(*ColName[ASC|DESC][ ,...n ] )*
| FOREIGN KEY(*ColName[,...n])REFERENCES*
*referenced_tbl_name*
*[(ref_col)[,...n] )]*
[ ON DELETE { NO ACTION | CASCADE | SET NULL |
SET DEFAULT } ]
[ ON UPDATE { NO ACTION | CASCADE | SET NULL |
SET DEFAULT } ]
| CHECK (*logical_expression ) }*

# CREATE TABLE

Example:

```
CREATE TABLE persons (
   id int,
   idcard int,
   mother int,
   father  int,
   tutor varchar(10),
   PRIMARY KEY (id),
   UNIQUE (idcard),
   FOREIGN KEY (mother) REFERENCES Persons,
   FOREIGN KEY (father) REFERENCES Persons (id),
   FOREIGN KEY (tutor) REFERENCES Persons (idcard));
```

# CREATE TABLE

Example:

```
CREATE TABLE LineOrder (
   order int,
   line int,
   productID int,
   suplierID   int,
   quantity int,
   PRIMARY KEY (order, line),
   FOREIGN KEY (productID ,suplierID) REFERENCES Products,
   UNIQUE (order, productID ,suplierID));
```

# CREATE TABLE

Computed columns:

< Column_def > ::=

*ColName AS expression* [ PERSISTED
[NOT NULL]
[ <Col_constraint> [… *n* ] ] ]

- A computed column can be used in the select list, in the WHERE clause , and in the ORDER BY clause.
- It cannot be used in a column constraint with the DEFAULT, FOREIGN KEY, NOT NULL clause.
- It cannot be the target of an INSERT or
- UPDATE.

# CREATE TABLE

TEMPORARY TABLES

- It is a table created in a process and when the process finishes the table is dropped.
- It can  be local or global.
- Local temporary tables are available only in the current session, global temporary tables are available in all the sessions.

*#TableName  for a local* temporary table

*##TableName for a global* temporary table

- Example:

```
CREATE TABLE #aux (col1 INT PRIMARY KEY);
```

# DROP TABLE

- Erase a table (data and structure)

DROP TABLE [[*DBName*].*SchemaName*.]*TableName*[ ,...n ] [;]

- Referential integrity is checked.
- Several tables can be dropped at the same time.
- Tables are dropped in the same order that they appear in the sentence. Useful when dropping referenced tables.

- Example:

```
DROP TABLE orders, products
```

# ALTER TABLE

- Modify its definition

ALTER TABLE [DBName.[SchName].| SchName.]TableName
{ ALTER COLUMN *ColName* <*columnChanges*>
| [WITH{ CHECK | NOCHECK}] ADD
{ <Column_def>
|<ComputedCol_Def>
| <table_constraint> } [ ,...n ]
| DROP {[CONSTRAINT] *constraintName*
|COLUMN *ColName}* [,...n ]
| {CHECK|NOCHECK} CONSTRAINT
{ALL|*constraint_name[*,...n ]}
| {ENABLE|DISABLE} TRIGGER {ALL | *TriggerNa* [ ,...n ] }
}[ ; ]

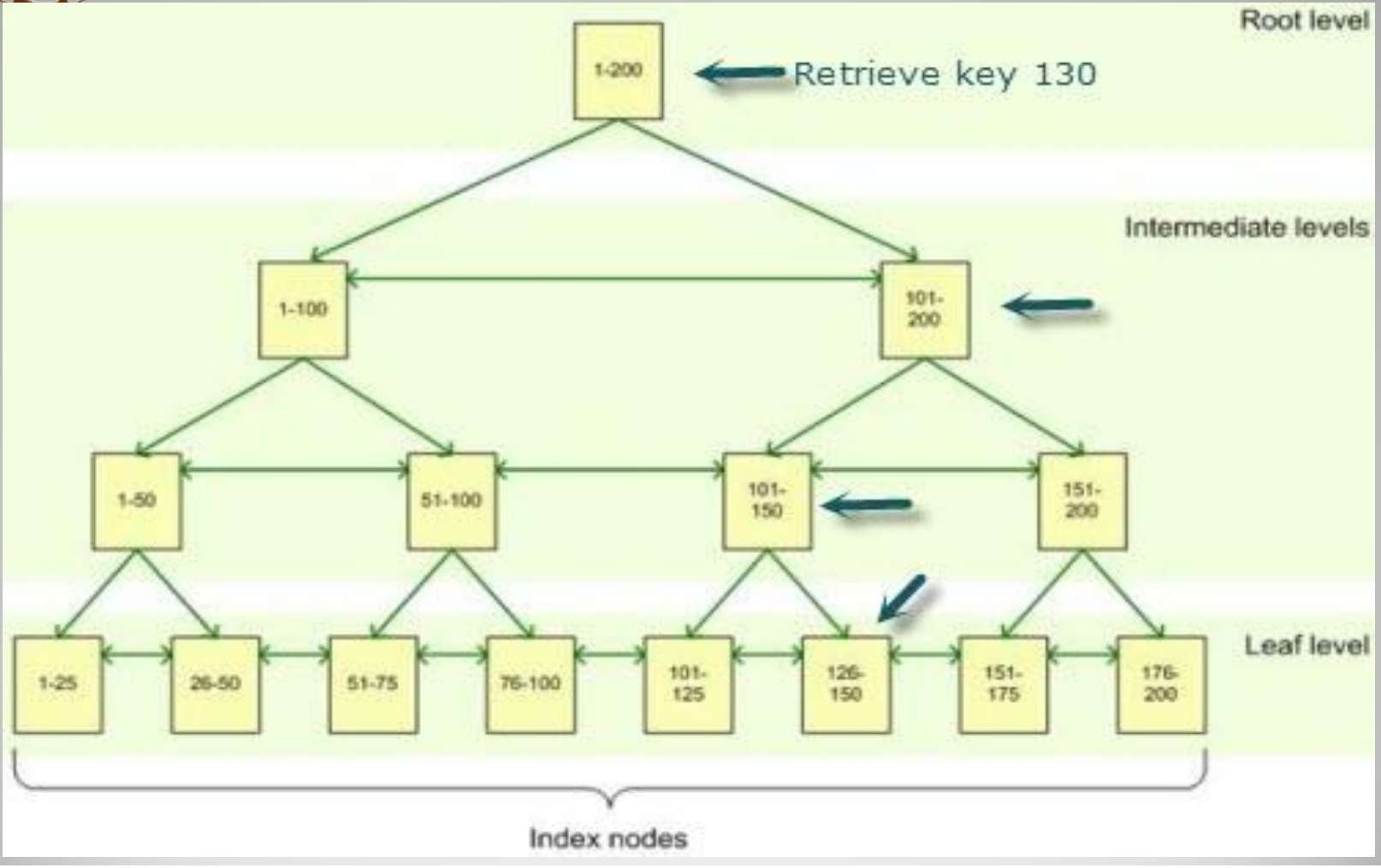
# ALTER TABLE

< *columnChanges* >::=
{
    <Datatype>
        [ NULL | NOT NULL ]
        [ COLLATE *CollationName*
    | {ADD | DROP } { ROWGUIDCOL |PERSISTED }
}

# INDEXES

- An index is an on-disk structure associated with a table or view that speeds up the retrieval of rows.
- An index contains keys built from one or more columns in the table. These keys are stored in a structure (B-tree) that enables SQL Server to find the row or rows associated with the key values quickly and efficiently.
- The selection of the right indexes for a database and its workload is a complex balancing act between query speed and update cost, usually done by the DB Administrator.

# INDEXES


Root level
1-200
Retrieve key 130
Intermediate levels
1-100
101-200
1-50
51-100
101-150
151-200
Leaf level
1-25
26-50
51-75
76-100
101-125
126-150
151-175
176-200
Index nodes

# INDEXES

- Instead of walking through *the whole table* and checking every row to see if it matches, the search is performed in the B-tree loaded in the memory.
- The leaf node contains either the entire row of data or a pointer to that row, depending on whether the index is clustered or nonclustered.
- A table that has a clustered index is referred to as a *clustered table (tabla agrupada)*. A table that has no clustered index is referred to as a *heap (montón)*.

# INDEXES

## INDEX TYPES

- Single column and composite index
  - Single column index →based on one column.
  - Composite index → based on several columns.
- Clustered and nonclustered index.
  - Clustered index (índice agrupado) → A clustered index stores the actual data rows at the leaf level of the index. So, the data rows are stored in order based on the index key.
  - Unclustered index → At the leaf level we have a pointer to the data row. The data rows are not guaranteed to be in any particular order.
- Unique index
  - An index that ensures the uniqueness of each value in the indexed column.

# INDEXES

PROS

- Indexes can enhance performance because they can provide a quick way for the query engine to find data.
- Indexes speed up the ordering process.

CONS

- They can take up significant disk space.
- Indexes are automatically updated when the data rows themselves are updated, which can lead to additional overhead and can affect performance.

# INDEXES

CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX

index_name

ON <object> (colum_na [ ASC | DESC ] [ ,...n ] )

[ ; ]

<object> ::=

{

[*DBName*.[SchemaName].| SchemaName.]TableViewName

}

# INDEXES

Examples:

CREATE INDEX Iclients_name ON Clients (name)

CREATE UNIQUE INDEX Iproducts_des ON products (description)

CREATE UNIQUE CLUSTERED INDEX Iorders_pk ON orders (id)

CREATE INDEX Ioffices_sal ON Offices (sales DESC)

CREATE INDEX Ioffices _2 ON Offices (region,city)

# INDEXES

DROP INDEX <index> [ ,...n ] [ ; ]

<indice>::=
{
indexname ON
[*DBName.[SchemaName].|SchemaName.]TableViewName*
}

Example: DROP INDEX Iclients_name ON Clients

# INDEXES

- ALTER INDEX

Modifies an existing index by:

- Disabling,
- Rebuilding,
- Reorganising (leaf level),
- setting options on it.

# VIEWS

CREATE VIEW [*SchemaName.] ViewName*
[ (*colum_na* [ ,...n ] ) ]
AS *select_sentence* [ ; ]

*select_sentence:*

- can use several tables and views.
- Can use functions, UNION or UNION ALL.
- Maximum 1.024 columns
- Is used as a table, but a virtual table. Its definition and execution plan are stored, which enhances performance.

# VIEWS

DROP VIEW [*SchemaName.*]ViewName[ ,...n ] [ ; ]

- If we drop a table, before hand we will have to drop all views based on the table, because they will not be eliminated automatically.