DATABASES

GS WEB APPLICATION DEVELOPMENT

Teacher: Vanesa Maldonado Guerrero
Curs 2025/26

Table index

Indexes are used to facilitate obtaining information from a table.

A **table index** performs the same function as a book index: it allows you to find data quickly; in the case of tables, it locates records.

- A table is indexed by one field (or several).
- The index is a type of file with 2 entries: a data (a value from some field in the table) and a pointer.
- An index enables direct and fast access, making searches more efficient. Without an index, the entire table must be traversed sequentially to find a record. (full scan table).

The purpose of an index is to speed up information retrieval.

The disadvantage is that it consumes disk space.

Indexing is a technique that optimizes data access and improves performance by speeding up queries and other operations. It is useful when the table contains thousands of records.

Indexes are used for various operations:
- to quickly search for records.
- to retrieve records from other tables using "join".

It is important to identify the field or fields for which it would be useful to create an index, those fields for which search operations are frequently performed.

Table index

There are different types of indices:

1) **"primary key"**: This is what we define as the primary key. Indexed values must be unique and cannot be null. MySQL names it "PRIMARY". A table can only have one primary key.
2) **"index"**: creates a common index; the values are not necessarily unique and accept "null" values. We can give it a name; if we don't, a default one is used. "key" is synonymous with "index". There can be several per table.
3) **"unique"**: creates an index for which values must be unique and different; an error message appears if you try to add a record with an existing value. It allows null values and multiple indexes can be defined per table. You can give it a name; if you don't, a default name is used.

All indexes can be **multi-column**, that is, they can be made up of more than 1 field.

A table can have up to **64 indexes**. Index names accept all characters and can have a maximum length of 64 characters. They can begin with a digit, but cannot consist entirely of digits.

A table can be indexed by **numeric or character fields**. It can also be indexed by a field containing NULL values, except for PRIMARY fields.

"show index" displays information about a table's indexes. For example:

```
show index from libros;
```

Primary type index.

The index called **"primary"** is created automatically when we set a field as the primary key; we cannot create it directly. The field used for indexing can be of numeric or character type.

- Indexed values must be unique and cannot be null. A table can only have one primary key; therefore, it only has one PRIMARY index.
- It can be multi-column, that is, it can be made up of more than 1 field.

Let's look at an example by defining the **"books"** table with a primary key:

```
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo)
 );
show index from libros;
```

Common Index.

A common index is created with **"index"**, the values are not necessarily unique and accept "null" values. There can be several per table.

Let's look at an example by defining the **"books"** table with a primary key:

One field we frequently use for queries is **"editorial"**; indexing the table by that field would be useful.

We **create an index** when we create the table:

```
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo),
  index i_editorial (editorial)
 );
```

After defining the fields, we place "index" followed by the name we give it and in parentheses the field or fields by which the index will be indexed.
If we do not assign a name to an index, by default it will take the name of the first field that is part of the index, with an optional suffix (_2,_3,...) to make it unique.

Common Index.

Indexes can be created for multiple fields:

```
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo),
  index i_tituloeditorial (titulo,editorial)
 );
```

To create indexes for multiple fields, list the fields within the parentheses, separated by commas. The index values are created by concatenating the values of the listed fields.
Remember that "index" is synonymous with "key".

Unique Index.

A **unique index** is created with **"unique"**. The values must be unique and distinct; an error message appears if you try to add a record with an existing value. It allows null values, and multiple indexes can be defined per table. You can give it a name; if you don't, a default name is assigned.

Let's work with our "books" table.

We will create two unique indexes, one for a single field and another for multiple columns:

```
create table libros(
 codigo int unsigned auto_increment,
 titulo varchar(40) not null,
 autor varchar(30),
 editorial varchar(15),
 unique i_codigo(codigo),
 unique i_tituloeditorial (titulo,editorial)
);
```

After defining the fields, we place **"unique"** followed by the name we give it and in parentheses the field or fields by which the index will be indexed.

Delete index (drop index).

To delete an index, we use **"drop index"**. Example:

```
drop index i_editorial on libros;
drop index i_tituloeditorial on libros;
```

The index is removed with **"drop index"** followed by its name and **"on"** followed by the name of the table it belongs to.

We can delete indexes created with "index" and "unique", but not the one created when defining a primary key. A PRIMARY index is automatically deleted when the primary key is deleted.

Creating indexes on existing tables (create index)

We can add an index to an existing table.

To add a common index to the "books" table, we type:

```
create index i_editorial on libros (editorial);
```

So, to add a common index to an existing table we use **"create index"**, we specify the name, which table to index on and the field(s) by which it will be indexed, in parentheses.

To add a unique index to the "books" table, type:

```
create unique index i_tituloeditorial on libros (titulo,editorial);
```

To add a unique index to an existing table we use **"create unique index"**, specifying the name, the table to be indexed on and in parentheses, the field(s) by which it will be indexed.

A PRIMARY index cannot be added; it is created automatically when a primary key is defined.

Add indexes (alter table - add index)

We learned how to create indexes when creating a table. We also learned how to create them after creating the table, using "create index". We can also add them to a table using **"alter table"**.

We created the "books" table:

```
create table libros(
  codigo int unsigned,
  title varchar(40),
  author varchar(30),
  editorial varchar (20),
  precio decimal(5,2),
  quantity smallint
 );
```

To add a common index by the "editorial" field, we use the following statement:

```
alter table libros
  add index i_editorial (editorial);
```

We use "alter table" together with "add index" followed by the name we will give to the index and in parentheses the name of the field or fields by which it will be indexed.

Add indexes (alter table - add index)

To add a unique multi-field index, using the fields "title" and "publisher", we use the following statement:

```
alter table libros
  add unique index i_tituloeditorial (titulo,editorial);
```

We use "alter table" together with "add unique index" followed by the name we will give to the index and in parentheses the name of the field or fields by which it will be indexed.

In both cases, for common or unique indexes, if we do not specify an index name, a default one is assigned, as when we create them along with the table.

Deleting indexes (alter table - drop index)

Common and unique indexes are removed with **"alter table"**.

We work with the "books" table of a bookstore, which has the following fields and indexes:

```
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  author varchar(30),
  editorial varchar(15),
  primary key(code),
  index i_editorial (editorial),
  unique i_tituloeditorial (titulo,editorial)
 );
```

To delete an index, we use the following syntax:

```
alter table libros
  drop index i_editorial;
```

We use "alter table" and "drop index" followed by the name of the index to be deleted.
To delete a unique index, we use the same syntax:

```
alter table libros
  drop index i_tituloeditorial;
```

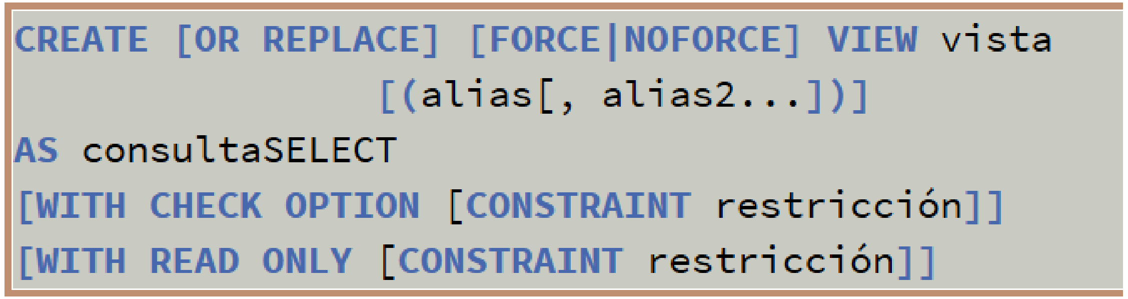
View

A **view** is a kind of "virtual" table created from an SQL query and stored in the database. The data displayed in the view is the result of the query that defines it and is stored in the actual database tables.

Some of the main advantages of using views are the following:

- They allow us to convert a complex query into a "virtual" table to make it easier to work with.
- They allow us to hide columns from a real table that we don't want to be visible to certain users.:

Create or modify a view:



```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW vista
                [(alias[, alias2...])]
AS consultaSELECT
[WITH CHECK OPTION [CONSTRAINT restricción]]
[WITH READ ONLY [CONSTRAINT restricción]]
```

View

- **OR REPLACE**. If the view already existed, it replaces it with the current one (this way we can modify a previously created view).
- **FORCE**. Creates the view even if the data in the SELECT query does not exist.
- **View**. Name given to the view
- **Alias**. A list of aliases set for the columns returned by the SELECT query on which this view is based. The number of aliases must match the number of columns returned by SELECT. Aliases are used to ensure that all columns have a unique name and to prevent columns with calculations from being left unnamed.
- They are not necessary if a correct name and/or alias has already been put in each column of the SELECT statement.
- **WITH CHECK OPTION**. This ensures that only the rows displayed in the view can be added (INSERT) or modified (UPDATE). The restriction that follows this section is the name given to this CHECK OPTION restriction .
- **READ ONLY**. Makes the view read-only. Allows you to save a name for this restriction.

View

Example:

Create a view that displays, for each order, the order code, date, name of the customer who placed the order, and total order amount.

```sql
CREATE OR REPLACE VIEW resumen_pedidos AS
SELECT
  pedido.codigo_pedido,
    pedido.fecha_pedido,
    cliente.nombre_cliente,
    SUM(detalle_pedido.cantidad * detalle_pedido.precio_unidad) AS total
FROM
  cliente INNER JOIN pedido
    ON cliente.codigo_cliente = pedido.codigo_cliente
    INNER JOIN detalle_pedido
    ON pedido.codigo_pedido = detalle_pedido.codigo_pedido
GROUP BY pedido.codigo_pedido
```

View

Example:

When creating a view, you can create an alias for each column. The following example shows the column names enclosed in parentheses.

```
CREATE OR REPLACE VIEW resumen_pedidos (codigo_pedido, fecha_pedido, nombre_cliente, total) AS
SELECT
  pedido.codigo_pedido,
    pedido.fecha_pedido,
    cliente.nombre_cliente,
    SUM(detalle_pedido.cantidad * detalle_pedido.precio_unidad)
FROM
  cliente INNER JOIN pedido
    ON cliente.codigo_cliente = pedido.codigo_cliente
    INNER JOIN detalle_pedido
    ON pedido.codigo_pedido = detalle_pedido.codigo_pedido
GROUP BY pedido.codigo_pedido
```

Delete a view

DROP VIEW removes one or more views. You must have the DROP privilege for each view.

If any views named in the argument list do not exist, the statement fails with an error indicating by name which nonexisting views it was unable to drop, and no changes are made.

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

Example:

```
DROP VIEW resumen_pedidos;
```

Renaming a view

RENAME TABLE renames one or more tables. You must have ALTER and DROP privileges for the original table, and CREATE and INSERT privileges for the new table.

```
RENAME TABLE
    tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

Example:

```
RENAME TABLE old_table TO new_table;
```

Consult the statement that was used to create a view

This statement shows the CREATE VIEW statement that creates the named view.

```
SHOW CREATE VIEW view_name
```

```
mysql> SHOW CREATE VIEW v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE ALGORITHM=UNDEFINED
                      DEFINER=`bob`@`localhost`
                      SQL SECURITY DEFINER VIEW
                      `v` AS select 1 AS `a`,2 AS `b`
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
```

Views with updatable content using INSERT, UPDATE and DELETE

In MySQL, you can use the view to perform inserts, modifications, and deletions on the view's base table.

It is important to note that if the data in a view is modified, the base table is also modified.

Data can be inserted, updated, or deleted from a table through a view, taking into account the following modifications made to the views:

- They cannot affect more than one queried table.
- The fields resulting from a calculation cannot be changed.
- The view cannot have grouping functions (count - max - min - sum - avg)
- It cannot have the clauses distinct, union, group by, having.
- You cannot have subqueries in the select clause.

The view should be simple enough so that it can be updated later.