ECON485 Introduction to Database Systems

Lecture 07 – Views and Indexes

Views

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view acts like a table. It contains rows and columns,

The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

Note that a view does not physically store the data.

When you issue the SELECT statement against the view, your DBMS (ie. MySQL) executes the query specified in the view's definition, and returns the result set which appears as a table.

This is why we define a view as a virtual table.

Views

How do we create a view?

CREATE VIEW view_name **AS**

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Then we can query the view as if it's a table.

SELECT column1 **FROM** view_name;

Note that, we may have to set permissions to access the view. Our default behavior in our database course is that there is a single user which has access to the entire database. In practical applications there are many users which have individually crafted access rights.

Views

Why do we create views?

We might want to hide the underlying table structure from a computer program.

Any SQL query within a program has to be designed with knowledge about the structure of the tables.

If the structure changes, then the SQL queries have to be updated and the program has to be re-built.

If the SQL queries within the program are designed to work with the views, then the program will be more independent from changes in the database structure. When the database structure changes, we will simply update the view, and not even tell the program.

CREATE OR REPLACE VIEW view_name **AS**
SELECT column1, column2, ...
FROM table_name
WHERE condition;

In IT industry making one part of a large system independent from another part is a very important goal. It decreases total cost of ownership over the entire life-cycle of the system significantly.

Views

Why do we create views?

We might want to limit access to certain parts of tables to certain users, but our DBMS does not have a very refined access control mechanism.

We create a view to show only the parts (selected rows and columns, based on the query). Then we set read-only access to the view only.

GRANT SELECT ON database.view1 **TO** 'someuser'@'somehost';

If some views are read-only that also helps optimize database performance.

Recall that one of the core problems solved by databases is coordinate / synchronize reads and writes to a shared data structure.

If some users and their access patterns are flagged as read-only, then it becomes an easier problem to solve.

How to drop a view?

DROP VIEW view_name;

Indexes

Indexes help us access data within a table more efficiently.

Without an index, we search through the database looking for the exact match to our query.

- This works fine, but might be inefficient if we have too many rows.
- Why too many rows?
  - Data from way back in the same table.
  - Real intensive insert behavior.
  - Simply a large application.

Database users cannot see the indexes. They cannot use the indexes explicitly in their queries.

- Indexes are used for a single purpose: Speed up searches/queries.

Indexes

Creating an index is easy.

We focus on the columns, for which we query a lot.

An index value will be created for each row, using the values in the columns we focus on.

The index will act like an additional column, and queries will act much faster with this column.

The SQL command syntax is as follows.

CREATE INDEX index_name

ON table_name (column1, column2, ...);

Indexes

Index values can be duplicate.

There is nothing wrong with duplicate values. This is because there could be duplicate values in the rows themselves.

If you want to have unique indexes, then the SQL syntax differs slightly.

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

A table might be queried in many ways.

If these queries involve different column sets, then it may be wise to create separate indexes for each of these column sets.

This allows for optimization on each type of query.

Indexes

The indexes created with CREATE INDEX statement are called simple indexes because they work in the most simple sense.

Many database systems allow for more intelligent indexes, however their creation requires DBMS-specific SQL commands.

In MySQL, we use the ALTER TABLE statement to add specific types of indexes which may or may not involve primary keys.

ALTER TABLE table_name **ADD PRIMARY KEY** (column_list) – This statement adds a PRIMARY KEY, therefore the indexed values must be unique and cannot be NULL.

ALTER TABLE table_name **ADD UNIQUE** index_name (column_list) – This statement creates an index where the non-NULL values must be unique, but allows NULL values.

Indexes

In MySQL, we use the ALTER TABLE statement to add specific types of indexes which may or may not involve primary keys. (cnt'd)

ALTER TABLE table_name **ADD INDEX** index_name (column_list) — This adds a simple index in which any value may appear more than once.

Indexes

In MySQL, we use the ALTER TABLE statement to add specific types of indexes which may or may not involve primary keys. (cnt'd)

ALTER TABLE table_name **ADD FULLTEXT** index_name (column_list) – This creates a special FULLTEXT index that is used for text-searching purposes.

A full text index is required if you want to run full text searches based on pattern matching on a text column. MySQL has natural language processing capabilities.

The SQL query for a natural language query is something like this:

SELECT * **FROM** table_name **WHERE MATCH**(col1, col2)
AGAINST('search terms' **IN NATURAL LANGUAGE MODE**);

When MATCH() is used in a WHERE clause, rows returned in the result set are **automatically sorted with the highest relevance first.**

Indexes

The way indexes are implemented may be very different from database to database.

Therefore, the only common expectation is an improvement in the performance of SELECT queries.

In the case of MySQL, indexing is a native part of the storage engine, that is how the data is actually written on the disks.

MySQL has two primary storage engines, InnoDB and MyISAM.

Whenever a row is updated in InnoDB, the index has to be updated as well. This is because InnoDB indexes have to be **sequential**. This requires sorting the index altogether every time an update is made, slowing down the UPDATE statements.

MyISAM indexes are implemented differently, and they **do not have to be sequential**. Therefore using MyISAM, there is no performance penalty in updates.