TOPIC- SQL - Indexes And View

1. Sql Index

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

The CREATE INDEX Command

The basic syntax of a CREATE INDEX is as follows.

CREATE INDEX index_name ON table_name;

Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

CREATE INDEX index_name

ON table_name (column_name);

Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

CREATE UNIQUE INDEX index_name

on table_name (column_name);

Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

```
CREATE INDEX index_name

on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

The DROP INDEX Command

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because the performance may either slow down or improve.

The basic syntax is as follows –

```
DROP INDEX index_name;
```

You can check the INDEX Constraint chapter to see some actual examples on Indexes.

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered.

Indexes should not be used on small tables.

Tables that have frequent, large batch updates or insert operations.

Indexes should not be used on columns that contain a high number of NULL values.

Columns that are frequently manipulated should not be indexed.

2. SQL Views

SQL CREATE VIEW Statement

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

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

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

CREATE VIEW Syntax

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name WHERE condition;

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

CREATE VIEW [Brazil Customers] AS

SELECT CustomerName, ContactName

FROM Customers WHERE Country = 'Brazil';

We can query the view above as follows:

Example

SELECT * FROM [Brazil Customers];

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

Example

```
CREATE VIEW [Products Above Average Price] AS

SELECT ProductName, Price

FROM Products WHERE Price > (SELECT AVG(Price) FROM Products);
```

We can query the view above as follows:

Example

```
SELECT * FROM [Products Above Average Price];
```

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW command.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name WHERE condition;
```

The following SQL adds the "City" column to the "Brazil Customers" view:

Example

```
CREATE OR REPLACE VIEW [Brazil Customers] AS

SELECT CustomerName, ContactName, City

FROM Customers WHERE Country = 'Brazil';
```

SQL Dropping a View

A view is deleted with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

The following SQL drops the "Brazil Customers" view:

Example

```
DROP VIEW [Brazil Customers];
```