**Database Design Course**

**week 4**

**⌨️ (6:01:12) Indexes (Clustered, Nonclustered, Composite Index)**

1. Clustered Index :

Clustered index is created only when both the following conditions satisfy –

1-The data or file, that you are moving into secondary memory should be in sequential or sorted order.

2-There should be a key value, meaning it can not have repeated values.

Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key. Clustered index is as same as dictionary where the data is arranged by alphabetical order.

In clustered index, index contains pointer to block but not direct data.

You can have only one clustered index in one table, but you can have one clustered index on multiple columns, and that type of index is called composite index.

2. Non-clustered Index :

Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

In non-clustered index, index contains the pointer to data.

3-Composite index

A composite key index is simply an index with multiple key columns. It may be clustered or non-clustered. Like all indexes, composite indexes are used to optimize queries. When all or most or all of the leftmost key columns are specified in query WHERE/JOIN clauses, the index facilitates reading only those rows needed by the query while avoiding others. Composite key indexes may additionally be used to guarantee uniqueness over multiple columns in order to guarantee data integrity, including indexes supporting composite primary key and unique constraints.

**⌨️ (6:14:36) Data Types**

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

In MySQL there are three main data types: string, numeric, and date and time.

data,numeric,string,char(8),varChar(8),Date,decimal,binary

**⌨️ (6:25:55) Introduction to Joins**

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

INNER JOIN:The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

LEFT JOIN:This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

RIGHT JOIN:RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

FULL JOIN:FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

**⌨️ (7:23:53) Introduction to Outer Joins**

But how do we get the full list of candidates along with the number of contributors for each, including those candidates who have no contributors in our data set? SQL provides the “OUTER JOIN” syntax for doing just that. Outer joins are typically defined by the table from which we want to include non-matching rows, and we do so by referring to where that table appears in the JOIN statement.

A LEFT OUTER JOIN includes all rows from the table on the left side of the statement and only matching rows from the table on the right side of the statement.

A RIGHT OUTER JOIN includes all rows from the table on the right side of the statement and only matching rows from the left side of the statement.

A FULL OUTER JOIN includes all rows from both tables.

**⌨️ (7:35:33) JOIN with NOT NULL Columns**

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255) NOT NULL,

Age int

);

**⌨️ (7:48:24) Alias**

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

SELECT column\_name AS alias\_name

FROM table\_name;

Alias Table Syntax

SELECT column\_name(s)

FROM table\_name AS alias\_name;

**⌨️ (7:52:13) Self Join**

A self join is a regular join, but the table is joined with itself.

Self Join Syntax

SELECT column\_name(s)

FROM table1 T1, table1 T2

WHERE condition;