DATABASE DESIGN

Indexing

Indexing

- An index is a special lookup table that can improve query performance on a target database table
- Index contains values from selected columns of target table
  - Always includes the primary key of target table
- Index is sorted by the values in the selected columns

Customer

CustID	FirstName	LastName	Address
1	Fred	Jones	104 Anyplace Ave
2	Greta	Anderson	201 Rambling Rd
3	Amy	Peters	33 Farming St

Index on Customer.FirstName

FirstName	CustID
Amy	3
Fred	1
Greta	2

Creating an Index

Specify:

- Unique or non-unique
- Table and Columns to index
- Index sort order

Example:

```
CREATE UNIQUE INDEX cust_name
   ON Customer(FirstName ASC, LastName ASC)
```

Unique Index

- A unique index prevents duplicate values from occurring in the column(s) specified for the index
  - Example: A unique index on Employee(Emp_No) prevents employees from being assigned duplicate Emp_No's

Employee

Emp_ID	FirstName	LastName	Emp_No
1	Fred	Jones	014523
2	Greta	Anderson	521341
3	Amy	Peters	132112
4	Fred	Hayes	623132

Multi-Column Unique Index

- A multi-column unique index prevents duplicate values from occurring in the combination of columns specified for the index
  - Example: A unique index on Customer(FirstName, LastName) prevents two customers with the same first and last name

Customer

CustID	FirstName	LastName	Address
1	Fred	Jones	104 Anyplace Ave
2	Greta	Anderson	201 Rambling Rd
3	Amy	Peters	33 Farming St
4	Fred	Hayes	23 Winding Way

Indexes and Query Processing

How Queries Use Indexes

- Indexes can speed up queries containing
  - Joins
  - WHERE criteria
  - ORDER BY sort specifications

Sample Table and Indexes

Customer

CustID	FirstName	LastName	Address
1	Fred	Jones	104 Anyplace Ave
2	Greta	Anderson	201 Rambling Rd
3	Amy	Peters	33 Farming St

Cust_FirstName_Inx

FirstName	CustID
Amy	3
Fred	1
Greta	2

Cust_FullName_Inx

FirstName	LastName	CustID
Amy	Peters	3
Fred	Jones	1
Greta	Anderson	2

Cust_LastName_Inx

LastName	CustID
Anderson	2
Jones	1
Peters	3

Simple WHERE Criteria


```
SELECT * FROM Customer
WHERE LastName = 'Peters'
```

- Without index, a sequential scan of entire table is required
- Using Cust_LastName_Inx, query can quickly locate correct records

Simple WHERE Criteria

Indexes can help with:

- Value match
  - LastName = 'Example'
- Value range
  - LastName > *value*
  - 'B' <= LastName < 'C'
  - LastName LIKE 'B%'

Indexes cannot help with:

- Match middle or end of field
  - LastName LIKE '%s'

Compound WHERE Criteria


```
SELECT * FROM Customer
WHERE FirstName = 'Amy' AND LastName = 'Peters'
```

- Here, Cust_FullName_Inx would be used to improve performance

Covering the Query


```
SELECT CustID FROM Customer
WHERE FirstName = 'Amy' AND LastName = 'Peters'
```

- Cust_FullName_Inx can be used to satisfy the query
- When all columns referenced by a query occur in the index, we say the index "covers" the query

Which Index Will Be Used?


```
SELECT CustID FROM Customer
WHERE LastName = 'Peters'
```

- Cust_FullName_Inx and Cust_LastName_Inx cover the query
- Cust_FullName_Inx is ordered by FirstName
  - Can't use it to quickly locate records where LastName = 'Peters'
- Query processor will probably choose Cust_LastName_Inx
- Takeaway: Order of columns matters in an index
  - When leading column of index is not referenced in WHERE clause, index is less likely to be used

Sorting using Indexes


```
SELECT * FROM Customer
WHERE FirstName = 'Amy'
ORDER BY LastName
```

- Using Cust_FullName_Inx, index scan retrieves records in the desired order
- Query processor doesn't have to perform separate sorting step
  - ORDER BY is a NO-OP!

Index Sort Order

- Index values are sorted in ascending order by default
- You can set the sort order for each column in the index

```
CREATE UNIQUE INDEX cust_name
   ON Customer(FirstName ASC, LastName DESC)
```

So, when would you want to use descending sort order?

Sorting using Indexes


```
SELECT * FROM Customer
WHERE FirstName = 'Amy'
ORDER BY LastName DESC
```

- With this index:

```
CREATE UNIQUE INDEX cust_name
   ON Customer(FirstName ASC, LastName DESC)
```

- Query processor doesn't have to perform separate sorting step

Determining Index Usage

- Query processor generates a **query plan** to execute a given query
- Most databases provide a query analyzer tool that displays the query plan for a given query
- Query plan shows which indexes database will use

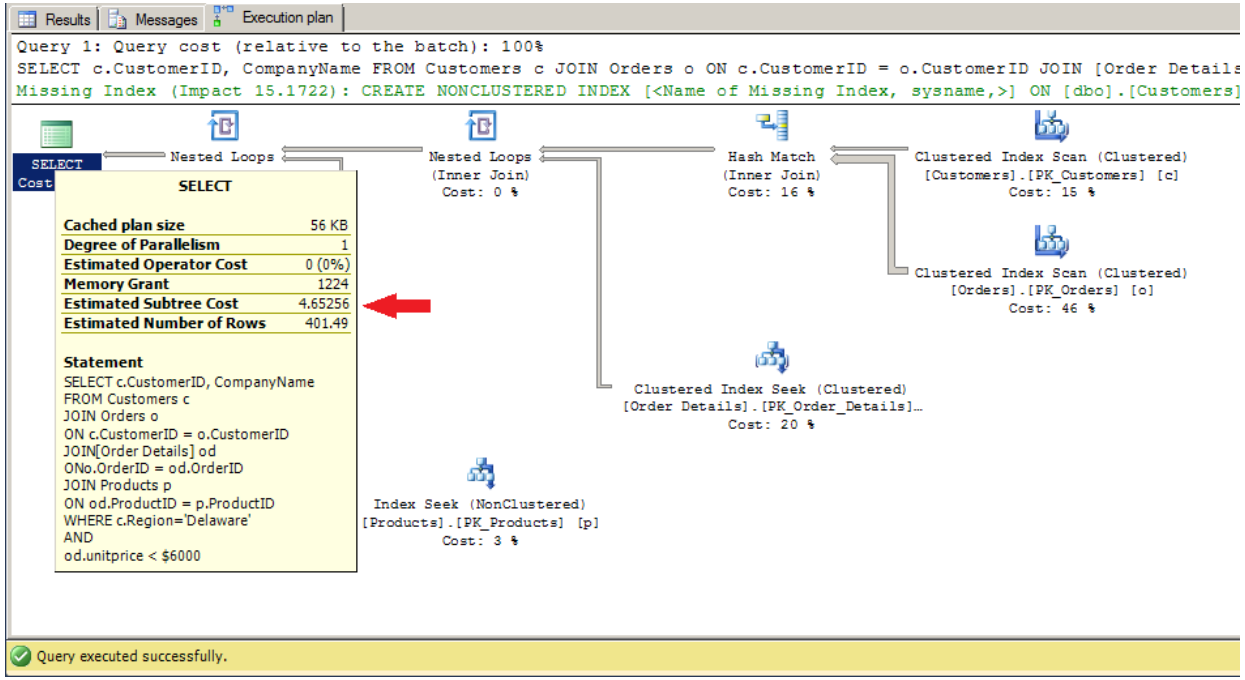

Table Statistics

- Query processor uses table statistics to determine query plan
- Common statistics:
  - Number of rows in the table
  - Size of row
  - Number of distinct values in each column

Why Is My Index Not Used?

- Updating Table Statistics
  - If table statistics are out of date, query processor will make poor choices
  - DBMS may automatically update table statistics
  - DBMS provides tools to force table statistics to be updated
- Query processor will use index only if it will improve query performance
  - Important to check query plan to see if an index will be used
  - Note that the query plan may change for a given query over time, as table statistics change

Indexing Adds Overhead

- Indexes speed queries but slow updates
- Each insert/delete/update to a table incurs cost of updating all of the table's indexes
- Caution: Adding an index that is never used by the query processor will not speed up any queries and will **slow down** overall database performance

Indexing Best Practices

- Indexing small tables is pointless
  - Databases cache small tables in memory during query processing
- Queries that return a large portion of a table are not likely to use an index
  - Unless the index covers the query
- When possible, add a column to an existing index rather than creating a new index
  - More indexes slow update performance

Non-Unique Indexes

- If a column contains duplicate values, an index on that column must be defined as **non-unique**
- In general, indexes are more likely to be used by the query processor when they contain a large percentage of distinct values
  - Example: A status code column with a handful of distinct values would not be a good candidate for an index
- Selectivity of a column = # distinct values / total # of values
- Best Practice: Avoid creating indexes on columns with low selectivity

Designing Indexes

Designing Indexes

- Wrong Question
  - "What indexes should be created for this schema?"
- Right Question
  - "What indexes should be created for these queries?"
- Index design is driven by queries, not schemas

The Three-Star Index Rating System

Lahdenmaki and Leach:

- First Star
  - Columns used in Where clause appear leftmost in index
- Second Star
  - Index contains columns used in Order By clause
  - These columns follow the First Star columns
- Third Star
  - Index contains all columns referenced by query
  - These columns are at the end of the index

Using the Three-Star System to Design an Index


```
SELECT LastName, SalesRep
FROM Customer
WHERE LastName = 'Smith' AND City = 'Washington'
ORDER BY CustBalance
```

One-Star Index:

```
CREATE INDEX Cust_Info_Inx(LastName, City)
```

Using the Three-Star System to Design an Index


```
SELECT LastName, SalesRep
FROM Customer
WHERE LastName = 'Smith' AND City = 'Washington'
ORDER BY CustBalance
```

Two-Star Index:

```
CREATE INDEX Cust_Info_Inx(LastName, City, CustBalance)
```

Using the Three-Star System to Design an Index


```
SELECT LastName, SalesRep
FROM Customer
WHERE LastName = 'Smith' AND City = 'Washington'
ORDER BY CustBalance
```

Three-Star Index:

```
CREATE INDEX Cust_Info_Inx(LastName, City, CustBalance, SalesRep)
```

Three Star System Comments

- Works best for queries on a single table that combine WHERE criteria with AND (not OR)

Diagnosing Query Performance Problems

- Analyze query logs
  - Database query log
  - Application query log
- Need to know
  - How long each query takes
  - How frequently each query is executed
  - Which indexes are being used
  - Which indexes are never used

Resources

- Lahdenmaki and Leach. *Relational Database Index Design and the Optimizers*. Wiley, 2005.
- https://youtu.be/ELR7-RdU9XU
  How to Design Indexes, Really (Focus on MySQL)
- https://youtu.be/qd0RcBXpDI8
  MySQL Indexing: Best Practices
- https://www.percona.com/blog/
  Percona Database Performance Blog (Focus on MySQL)