**Assignment 4**

**Part I. Indexes**

**Q1**. **Creating Indexes on Foreign Keys:**

**Query:**

CREATE INDEX casting\_movieid\_idx ON casting (movieid);

CREATE INDEX casting\_actorid\_idx ON casting (actorid);

Result:

A close-up of a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

**Q2**. **Creating Indexes on Non-key Columns:** (1) Construct SQL commands to create indexes on actor’s name, movie’s title, and movie’s votes. (2) Show the executions of these commands.

Query:

CREATE INDEX actor\_name\_idx ON actor (name);

CREATE INDEX movie\_title\_idx ON movie (title);

CREATE INDEX movie\_votes\_idx ON movie (votes);

Result:

A black background with white text

Description automatically generated

**Q3**. **Retrieving Index Information:** (1) Develop a SQL query to gather detailed information about the indexed on the actor, casting and movie tables. Utilize both USER\_INDEXES and USER\_IND\_COLUMNS dictionary views to include details such as the index name, index type, associated table name, uniqueness status, indexed column name, and column position within the index. Order the query output by table name, followed by index name and column position within the index. (2) Present the results of your query.

Query:

SELECT

idx.indexname AS index\_name,

idx.indexdef AS index\_type,

idx.tablename AS table\_name,

CASE WHEN idx.indexdef LIKE '%UNIQUE%' THEN 'UNIQUE' ELSE 'NON-UNIQUE' END AS uniqueness,

cols.column\_name AS column\_name,

cols.ordinal\_position AS column\_position

FROM

pg\_indexes AS idx

JOIN

information\_schema.columns AS cols

ON

idx.tablename = cols.table\_name

WHERE

idx.tablename IN ('actor', 'casting', 'movie')

ORDER BY

idx.tablename,

idx.indexname,

cols.ordinal\_position;

Result:

A screen shot of a computer

Description automatically generated

**Q4**. **Analyzing Index Types**: Based on the index information retrieved in Q3. **(1)** Identify which indexes serve as primary indexes and which ones are secondary. **(2)** Discuss the fundamental difference between primary and secondary indexes, particularly in terms of their roles in physical file organization.

Based on the provided output from Q3, we can identify primary and secondary indexes:

1. **Primary Indexes:**
   * For the actor table:
     + actor\_pk: This index is defined as a unique index (UNIQUE) on the id column of the actor table, which indicates it serves as the primary key. Primary keys are unique and not null, ensuring each row in the table is uniquely identifiable.
   * For the casting table:
     + casting\_pk: Similar to the actor table, this index is defined as a unique index (UNIQUE) on the combination of movieid and actorid, indicating it serves as the primary key for the casting table.
   * For the movie table:
     + movie\_pk: This index is defined as a unique index (UNIQUE) on the id column of the movie table, indicating it serves as the primary key.
2. **Secondary Indexes:**
   * Secondary indexes are those indexes that are not primary keys.
   * Examples of secondary indexes include:
     + Indexes on columns such as name, title, yr, score, votes, director, and ord.
     + These indexes provide alternate access paths to the data stored in the tables, improving query performance for queries that don't directly use the primary key for retrieval.

In terms of their roles in physical file organization:

* **Primary Indexes:**
  + Dictate the physical order of the rows in the table.
  + Ensure uniqueness and facilitate quick access to individual rows using the primary key.
  + Serve as the main organizational structure for the table's data storage.
* **Secondary Indexes:**
  + Do not dictate the physical order of the rows.
  + Provide additional access paths to the data based on columns other than the primary key.
  + Improve query performance for specific search criteria but do not impact the overall organization of the table's data.

These distinctions highlight the importance of primary keys for uniquely identifying rows and the role of secondary indexes in enhancing query performance for various search criteria.

**Q5**. **Identification and Application of Clustered Indexes:** Given the schemas for ‘action’ and ‘move’ tables, discuss whether these tables are ideal candidates for clustered indexes.

Response:

**Actor Table (actor):**

* Ideal Candidate: Yes
* Reasoning: id column serves as the primary key, likely used for efficient retrieval. Clustered index on id would align with common usage pattern.

**Movie Table (movie):**

* Ideal Candidate: Possibly
* Reasoning: id column serves as primary key, but queries may often involve other criteria. More analysis needed to determine suitable columns for clustering.

**Conclusion:**

* actor table is a good candidate for clustered index on id.
* For movie table, consider query patterns before deciding on clustering strategy.

**Oracle RDBMS:**

* Data Organization: Index-Organized Tables (IOTs) are used for tables with primary key constraints. Data is physically organized based on primary key values within the index structure itself.
* Retrieval: Efficient retrieval is facilitated through direct access via the primary key index.

**Comparison with SQL Server:**

* Oracle (IOTs):
  + Terminology: Index-Organized Tables (IOTs).
  + Concept: Data storage and index organization are tightly integrated, with the primary key index serving as the storage structure.
* SQL Server (Clustered Indexes):
  + Terminology: Clustered Indexes.
  + Concept: Physical order of data on disk is determined by the clustered index. Data retrieval is efficient due to direct access via the clustered index.

**Differences:**

* Oracle emphasizes Index-Organized Tables (IOTs), where data and index storage are merged.
* SQL Server employs Clustered Indexes, dictating the physical order of data on disk.

In essence, both Oracle and SQL Server achieve efficient data retrieval, but they differ in terminology and approach to physical data organization.