**Index**

An object that allows us to find specific data in a table faster using SELECT. If a column has few distinct values, index may not be useful. Creating an index on every column slows down INSERT, UPDATE, and DELETE, because the database needs to update the index for each these operations.

When we write CREATE INDEX, whether CLUSTERED or NONCLUSTERED, SQL Server then internally creates a **B+-tree** structure. It is the default type of index.

Clustered index is for numerical data or when the data is sortable. Non-clustered index is for unsortable data.

**B+-Tree Index**

The B+-tree is the most widely used variant of the B-tree in database systems. It has maximum of 4 levels. It stands for **Balanced-Tree**. We interact with indexes at a logical level using SQL DDL (Data Definition Language) statements like CREATE INDEX. The B+-tree is the specific data structure that SQL Server uses behind the scenes to implement these indexes efficiently on disk.

**Key Differences and Enhancements of a B+-Tree Compared to a B-Tree:**

1. **Data Records Only in Leaf Nodes:** In a B+-tree, the leaf nodes are the only nodes that store the actual data records (or pointers to them). Internal nodes only store keys and pointers to child nodes.
2. **Leaf Nodes are Linked:** The leaf nodes of a B+-tree are typically linked together in a sequential order (usually a doubly linked list). This linked list provides a significant advantage for range queries. Once the first key in the range is found in a leaf node, the rest of the records in the range can be efficiently retrieved by simply traversing the linked list of leaf nodes, without having to go back up and down the tree.
3. **Keys in Internal Nodes are Redundant:** The keys that appear in the internal nodes are also present in the leaf nodes. The internal nodes act purely as an index to guide the search to the correct leaf node.

**Advantages of B+-Trees for Database Indexes:**

* **Efficient Range Queries:** The linked list of leaf nodes makes retrieving records within a specific range very fast.
* **Stable Performance:** The balanced structure ensures consistent search performance regardless of the data distribution.
* **High Fan-Out:** Reduces the height of the tree, minimizing disk I/O operations.
* **Efficient for Equality Searches:** Finding a specific record requires traversing a small number of levels.

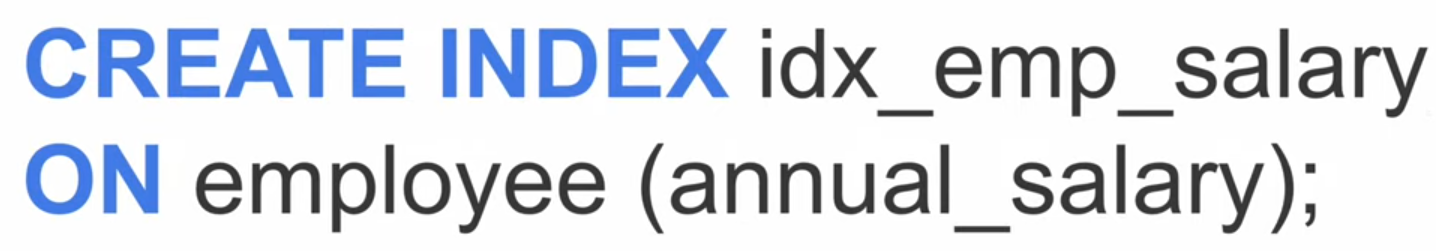
**Clustered Index:** The leaf level of the B+-tree *is* the sorted data of the table.

**Non-Clustered Index:** The leaf level of the B+-tree contains the index keys and row locators (pointers to the data rows).

A close up of a logo

AI-generated content may be incorrect.

**Example:**

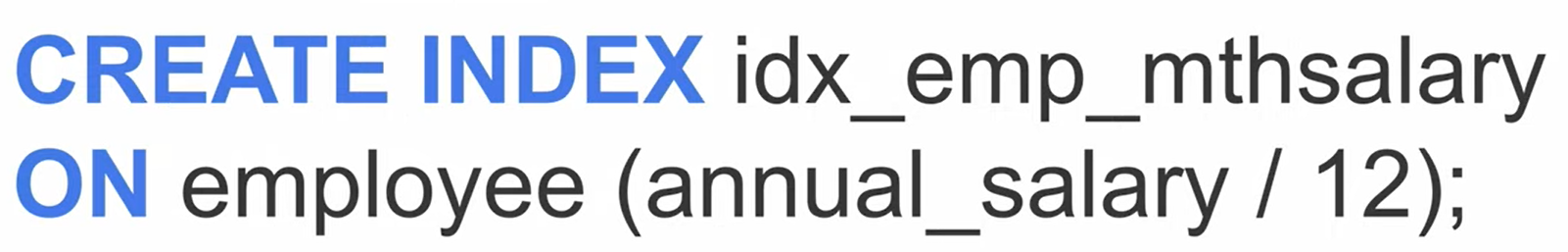


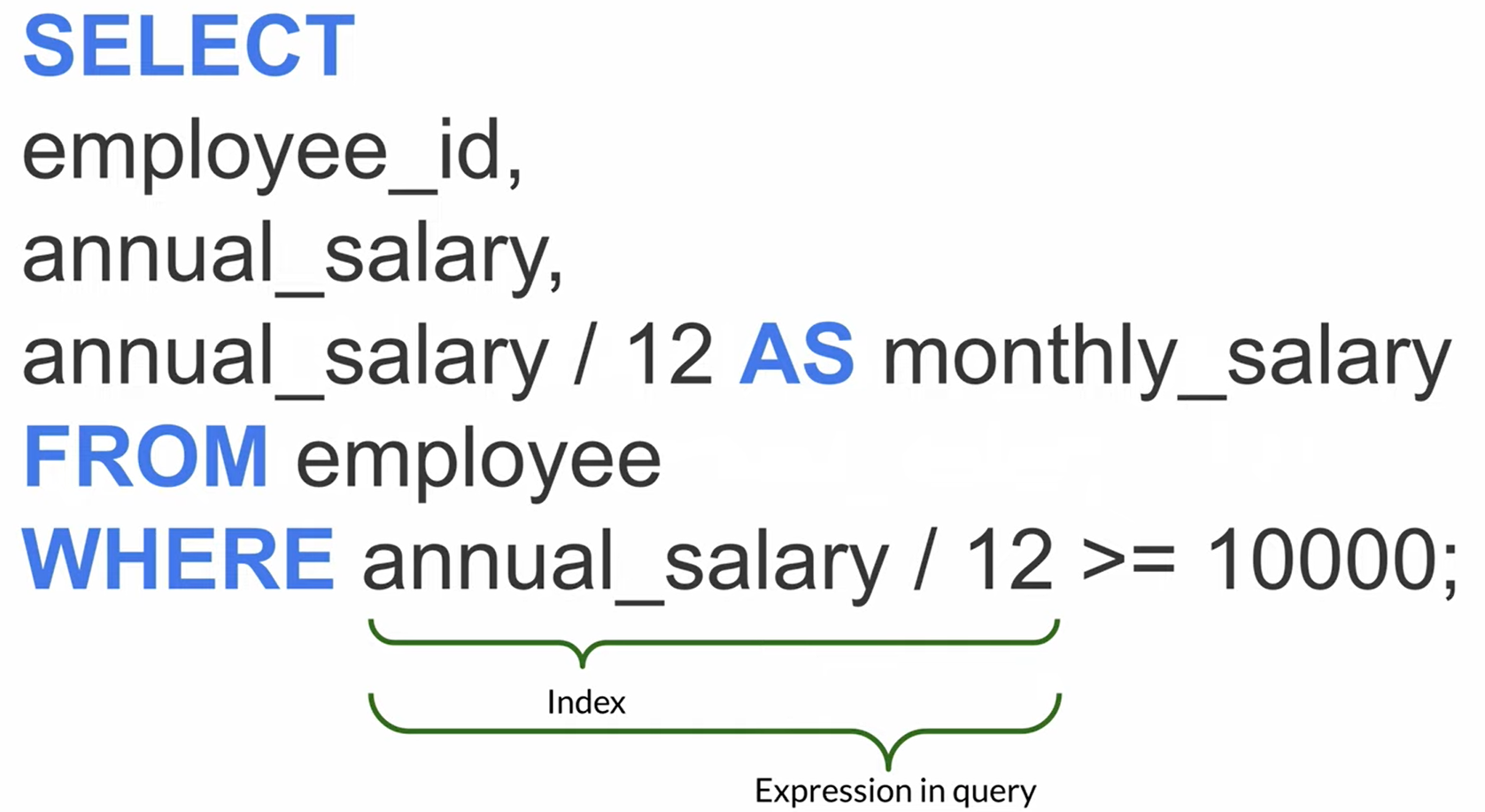
A white background with black text

AI-generated content may be incorrect.

Here index won’t be used, because B-Tree does not work on Expressions. This is where **Function-based Index** comes in handy.

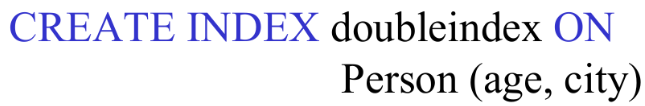
**Function-Based Index**





**Composite Index (Multi-Column Index)**

An index created on two or more columns. The order of columns in a composite index matters for its effectiveness.



A close-up of a sign

AI-generated content may be incorrect.

A close-up of a sign

AI-generated content may be incorrect.

**Views**

It is a virtual table. It is not physically stored on hard disk. It is the result set of a stored query.

**Pros:**

* **Security:** Restrict data access (hide specific columns). We don’t need to provide the original table with all information to everyone.
* **Simplicity:** Simplify complex queries.
* **Consistency:** Ensure uniform data presentation.
* **Insulation:** Buffer applications from schema changes.
* **Reusability:** Reuse stored query logic.
* **Readability:** Make SQL code easier to understand.

**Cons:**

* **Performance (Potential):** Overhead in resolving view definitions.
* **Updatability:** Many views are read-only.
* **Dependencies:** Changes in base tables can affect views.
* **Debugging:** Can complicate issue tracing.
* **Materialized Views (If Used):** Storage and refresh overhead.

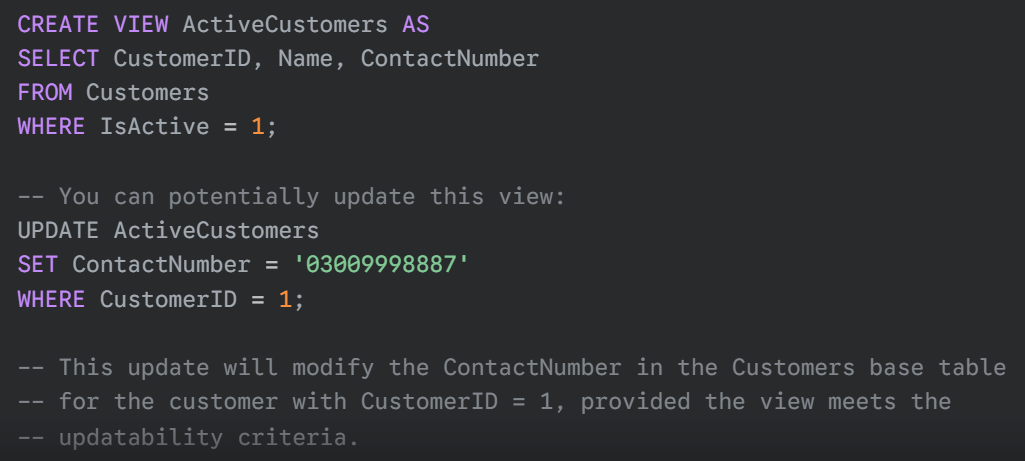
**Read-only**

Views that involve **aggregations** (e.g., COUNT, SUM, AVG), **GROUP BY** clauses, **DISTINCT**, or **joins** across multiple base tables are generally **not updatable**. The database engine often cannot determine which base table and which specific row should be modified when you try to update a row in such a view.

**Updatable**

Under certain conditions, you can insert, update, or delete data through a view, which then modifies the underlying base tables.

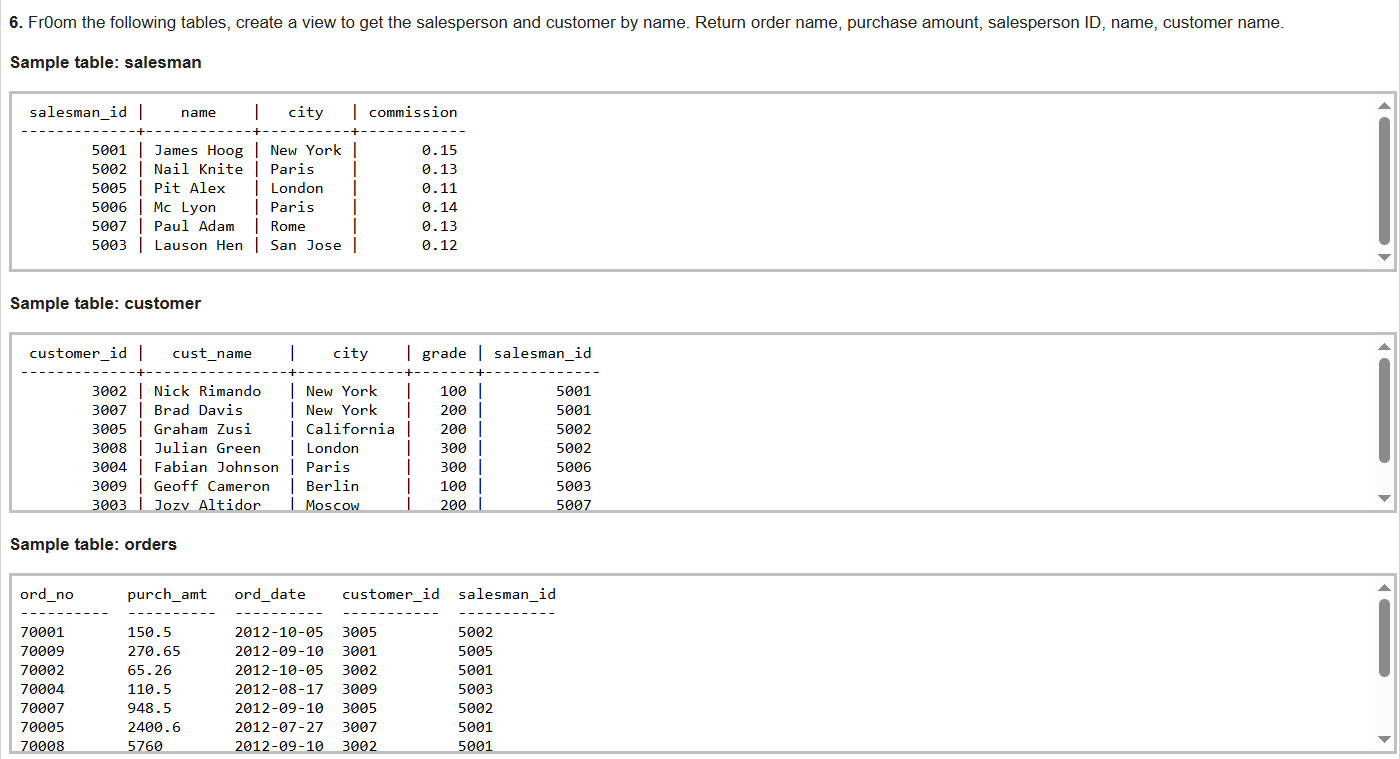
* The view must reference only one base table in the FROM clause.
* The view must include the primary key (or a key that uniquely identifies rows) of the base table.
* The view must not use aggregate functions, GROUP BY, HAVING, DISTINCT, or TOP.
* The view must not involve set operators like UNION, UNION ALL, INTERSECT, or EXCEPT.
* The columns being updated in the view must directly correspond to columns in the base table.



**Materialized Views**

These *do* physically store the result set on disk. They are like a snapshot of the data at a specific point in time. It is not always up-to-date. Any changes in the base table after the snapshot are not reflected in the materialized view. It is a separate copy of the base table. It is stored on hard disk. It is often used in data warehouses and data mining. It is physically store the result set of the view's query on disk, just like a regular table.

**--Example Queries--**



A screenshot of a computer program

AI-generated content may be incorrect.

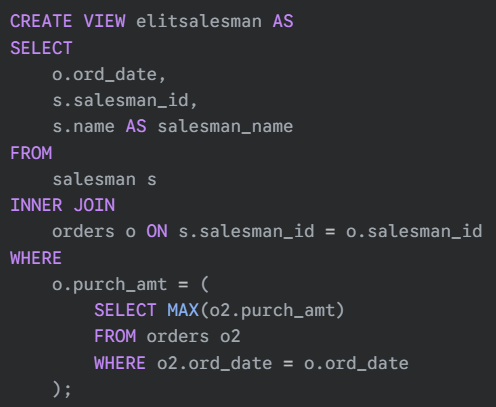
A screenshot of a computer program

AI-generated content may be incorrect.

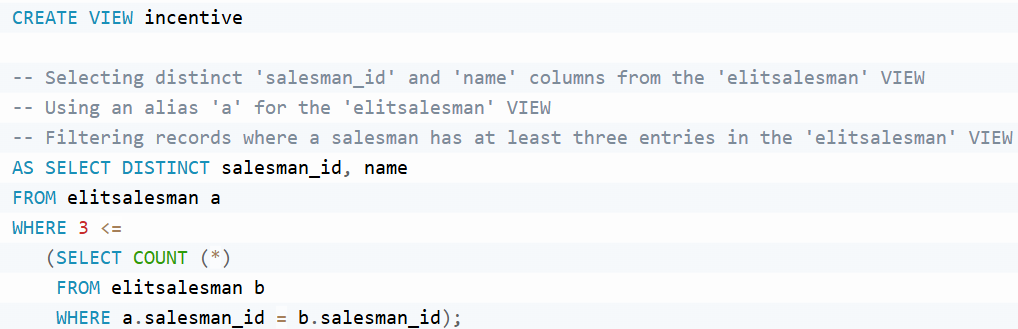
-------------------------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer

AI-generated content may be incorrect.

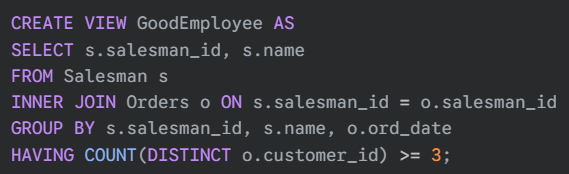


-------------------------------------------------------------------------------------------------------------------------------------------



-------------------------------------------------------------------------------------------------------------------------------------------

Salesmen who have dealt with at least 3 *customers* on *any single day*.



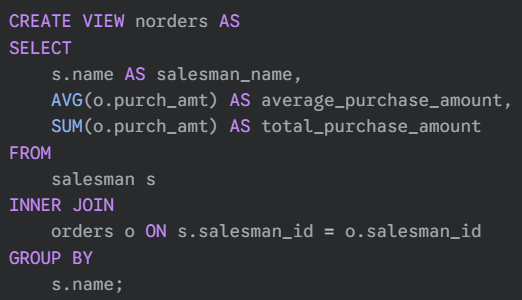
-------------------------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer code

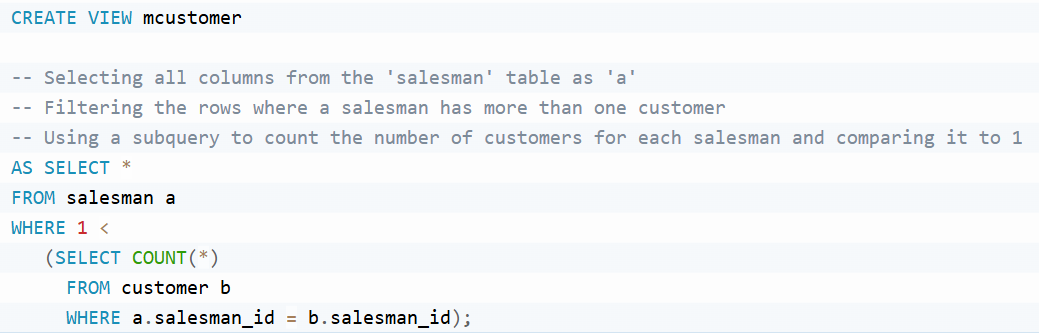
AI-generated content may be incorrect.

-------------------------------------------------------------------------------------------------------------------------------------------

Compute the average purchase amount and total purchase amount for each salesperson.



-------------------------------------------------------------------------------------------------------------------------------------------



-------------------------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

-------------------------------------------------------------------------------------------------------------------------------------------