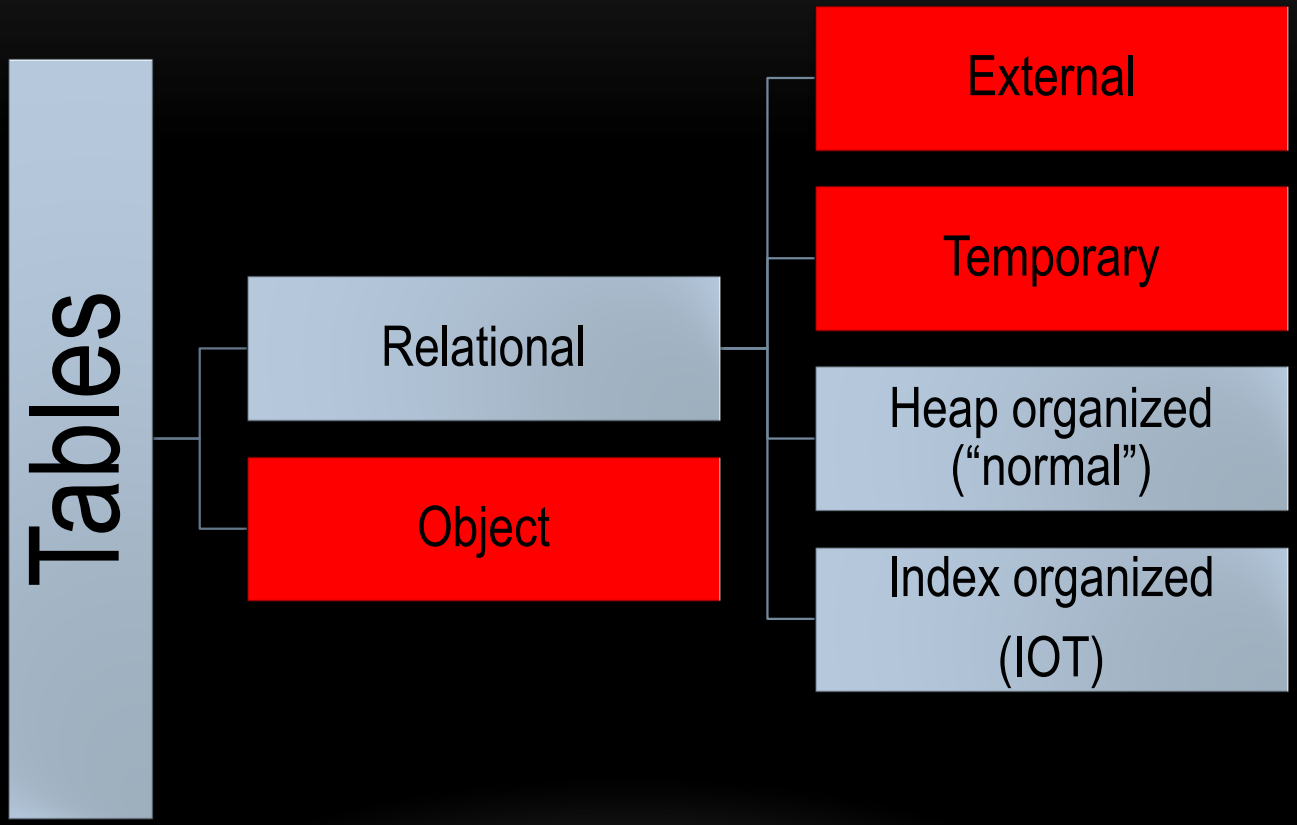
ORACLE TABLES

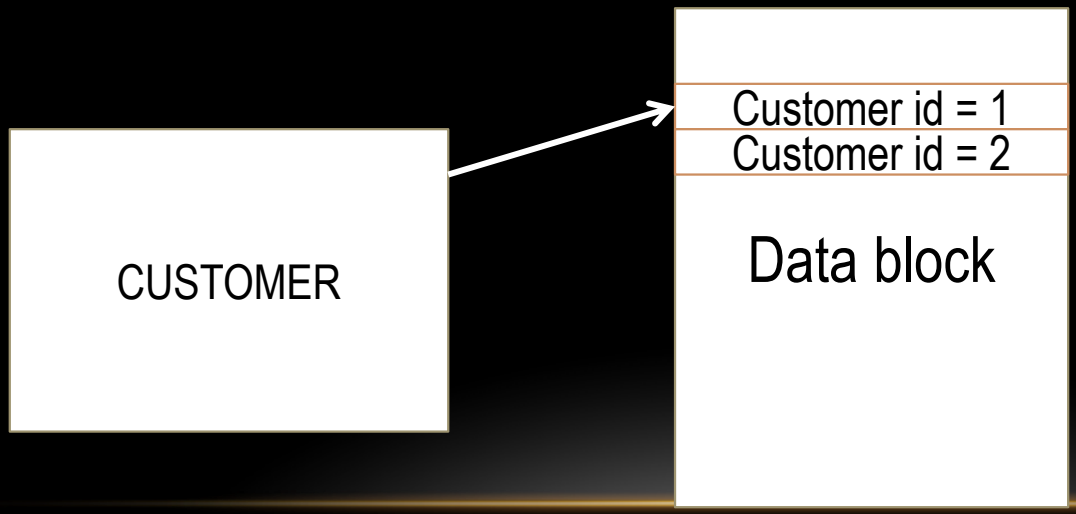
Juri Guljajev

WHAT WILL WE TALK ABOUT?



LITTLE THEORY - HOW DATABASE STORE DATA?

- Smallest unit of data in Oracle is data block.
- Default size of data block is 8 kilobytes.
- Normally only one table rows can be added into one data block
- Database tries to store the whole row in one block



LITTLE THEORY - ROWID

ROWID looks like **AAAAaoAATAAABrXAAA**

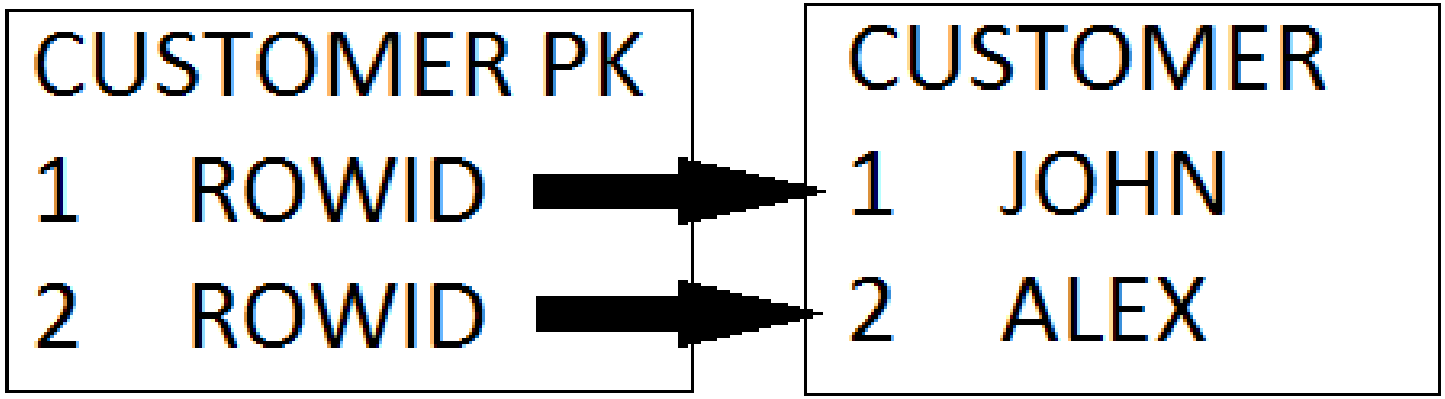
OOOOOOFFFBBBBBBRRR

- *OOOOOO: The data object number that identifies the database segment (AAAAao in the example).*
- *FFF: The tablespace-relative datafile number of the datafile that contains the row (file AAT in the example).*
- **BBBBBB: The data block that contains the row (block AAABrX in the example).**
- **RRR: The row in the block. (AAA in the example)**

LITTLE THEORY - INDEX IN HEAP TABLE

Index is using ROWID to point on data from table.

SELECT * FROM customer WHERE id = 1;



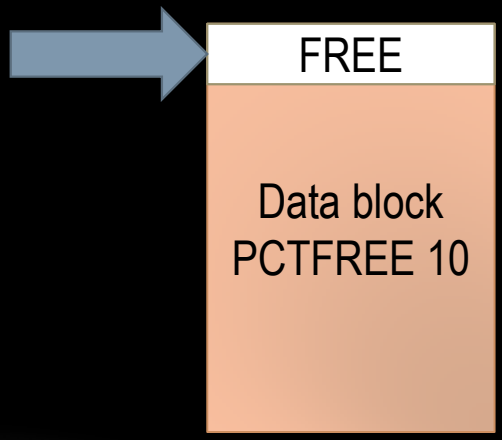
HEAP ORGANISED - *KEY FEATURES*

- Unordered collection of rows
- Retrieval order is not guaranteed
- Rows are inserted where they fit best
- Generally better DML performance (in compare to IOT)
- Worse query performance by PK (in compare to IOT)
- Can be stored in table cluster

HEAP ORGANIZED - *PCTFREE*

The PCTFREE parameter sets the minimum percentage of a data block to be reserved as free space for possible updates to rows that already exist in that block.

```
CREATE TABLE test
        ( id NUMBER)
PCTFREE 10;
```



HEAP ORGANISED - *PCTFREE*

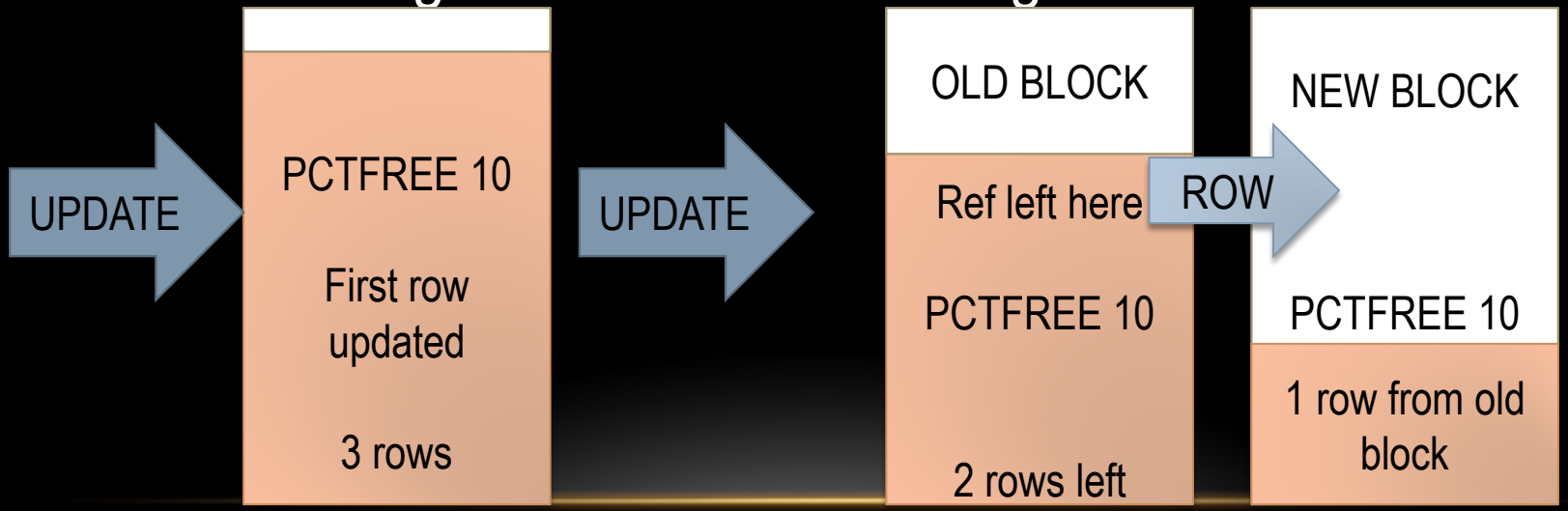
INSERT (example)

- 1 row size is: 5 bytes + 1900 bytes = 1905 bytes
- PCTFREE 10: (8192B – 10%) / 1905B = 3 rows

HEAP ORGANISED - *PCTFREE*

UPDATE (example)

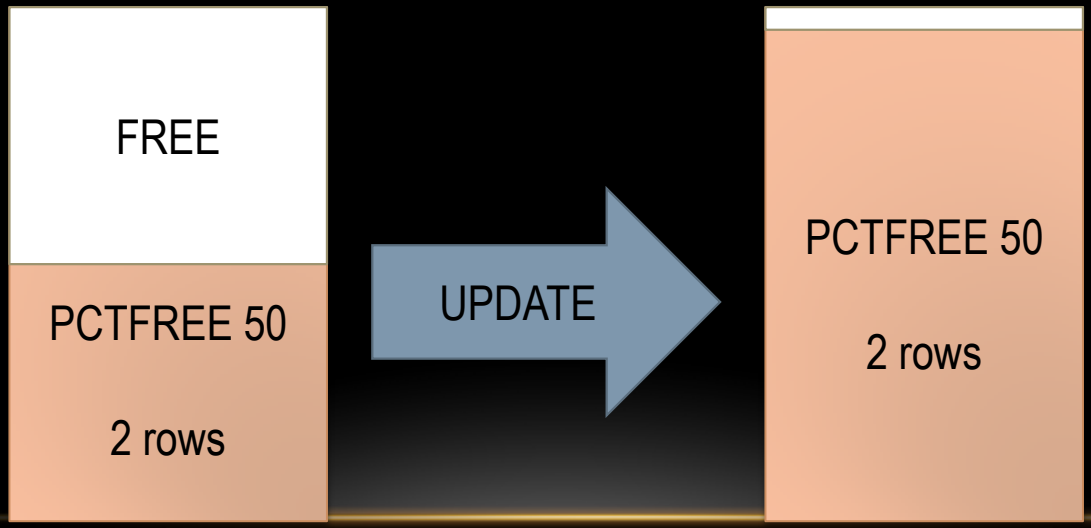
Rows, that can't fit into block after update were moved to new data block and references were left in old data block. ROWID of migrated row is not changed.



HEAP ORGANISED - *PCTFREE*

UPDATE

Table, that have PCTFREE 50 have enough storage room to fit added data into the same data block.

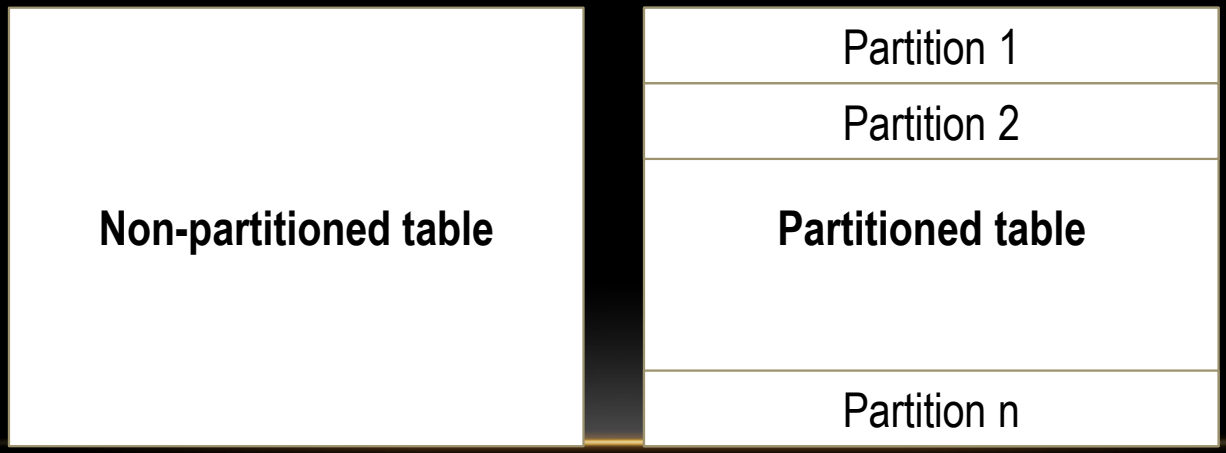


HEAP ORGANISED - *PCTFREE*

- Better query performance
- Better storage usage
- Use with caution (set PCTFREE 50, but rows are filled by 80% with insert)

HEAP/INDEX ORGANISED - *PARTITIONING*

Partitioning helps to improve performance with high data volumes and allows DBAs to manage data in more convenient way.



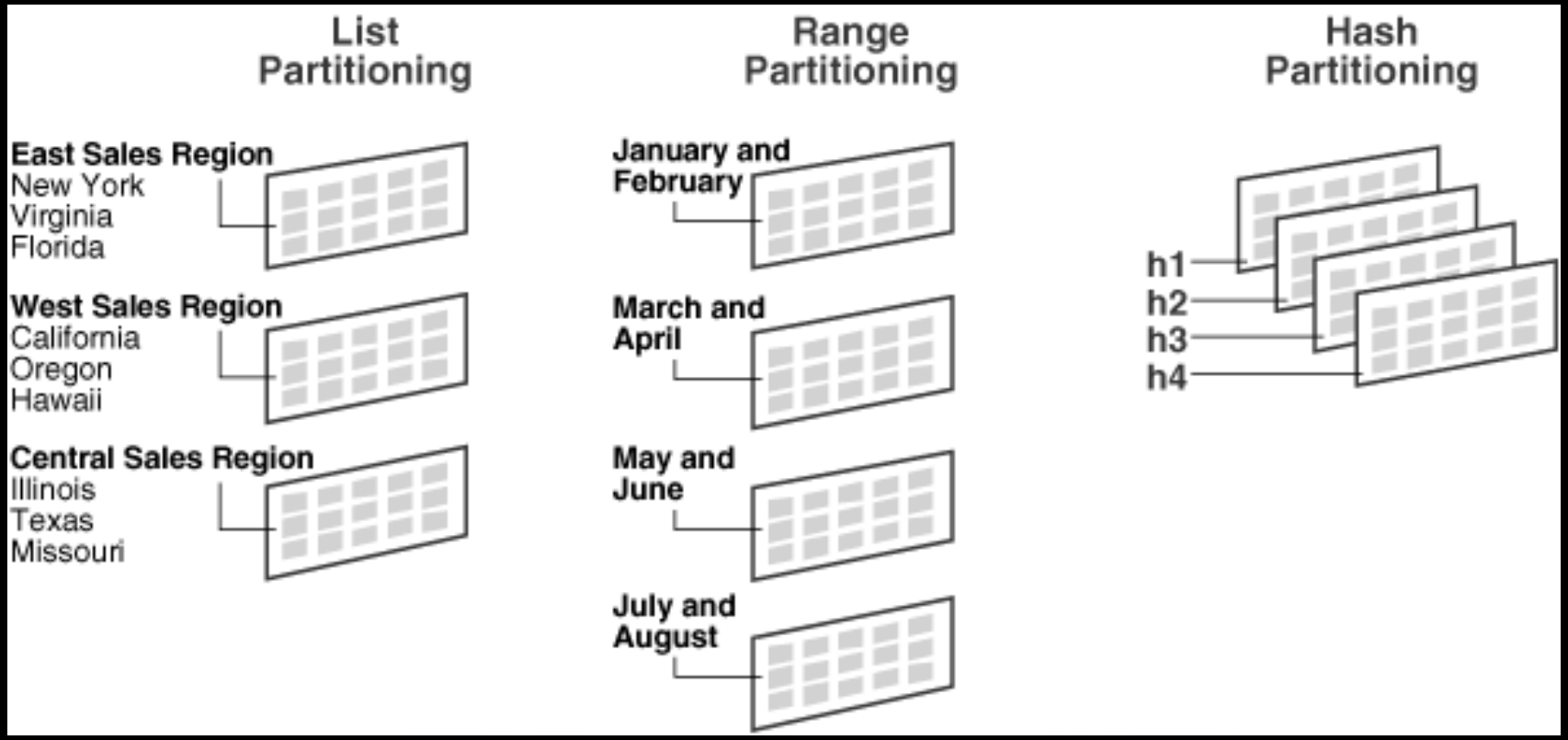
HEAP/INDEX ORGANISED - *PARTITIONING*

Table can be partitioned by

- Range (for historical data, often partitioned by date column)
- Hash (no obvious partitioning column)
- List (to group data specifically by some known values)

Range and List partitions may have subpartitions.

HEAP/INDEX ORGANISED - *PARTITIONING*



HEAP/INDEX ORGANISED - *PARTITIONING*

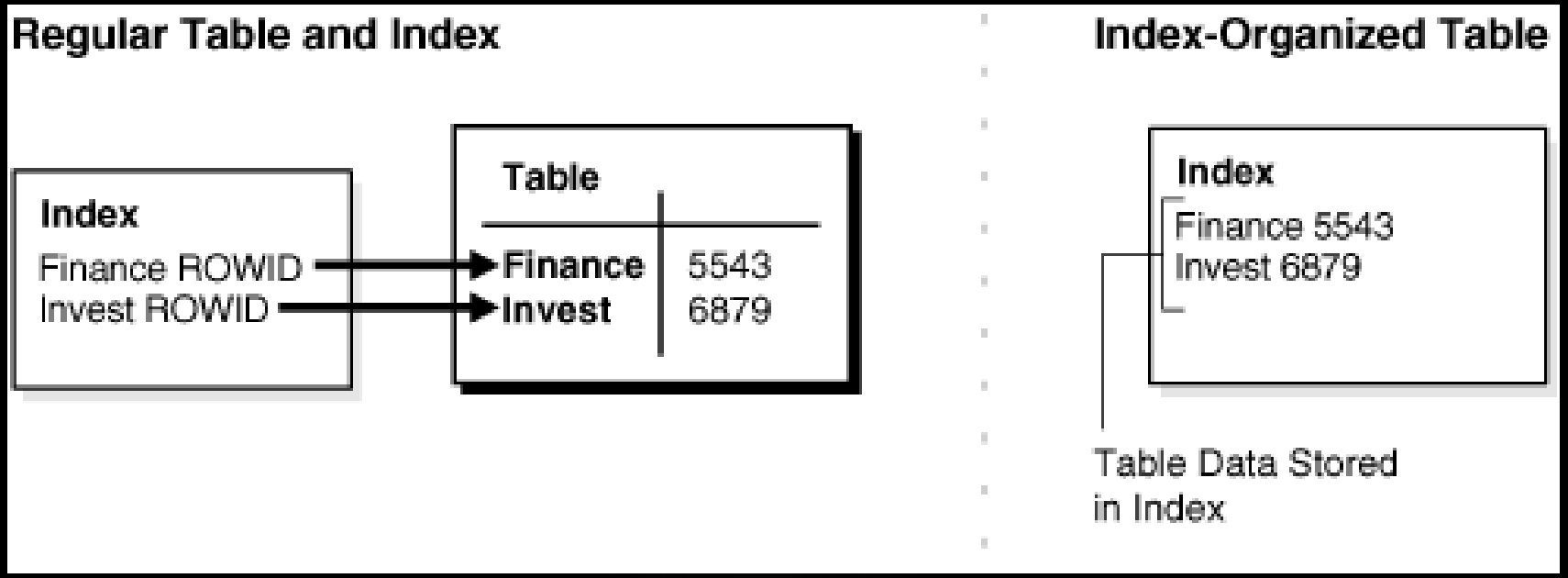
Indexes on partitioned table can be global or local partitioned.

- Local
  - Maps on table partitions, so easier to maintain
  - Unique index can be local, but partitioning key must be part of index
- Global
  - Has own partitions definitions
  - Can be partitioned by range or hash

INDEX ORGANISED - *KEY FEATURES*

- Stored in a variation of a B-tree index structure
- Primary key stores all rows
- Fast access by primary key (specially range scan)
- DML performance might be worse (in compare to HEAP)
- Take less room on disk (no separate primary key storage required)
- Cannot be stored in table cluster

INDEX ORGANISED - *KEY FEATURES*



INDEX ORGANISED - *KEY FEATURES*

PROS

- Fast access by PK
- Less storage for the same amount of data
- Data is ordered by PK

CONS

- Slower DML
- Secondary index can be slower and take more storage in compare to index in HEAP table

TABLE CLUSTER

Group of tables that share common columns and store related data in the same blocks.

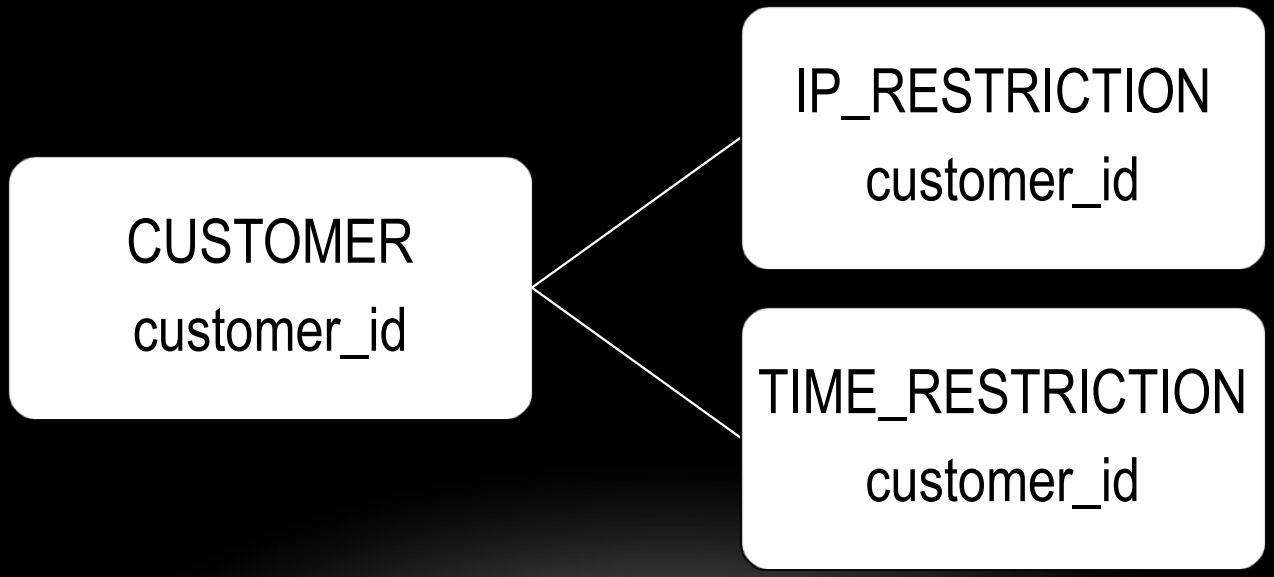


TABLE CLUSTER

```
SELECT* FROM customer WHERE customer_id = :id;
```

```
SELECT * FROM ip_restriction JOIN time_restriction
USING(customer_id) WHERE customer_id = :id;
```

```
SELECT * FROM customer JOIN ip_restriction
USING(customer_id) JOIN time_restriction
USING(customer_id) WHERE customer_id = :id;
```

TABLE CLUSTER

In cluster single data block contains rows from several clustered tables.

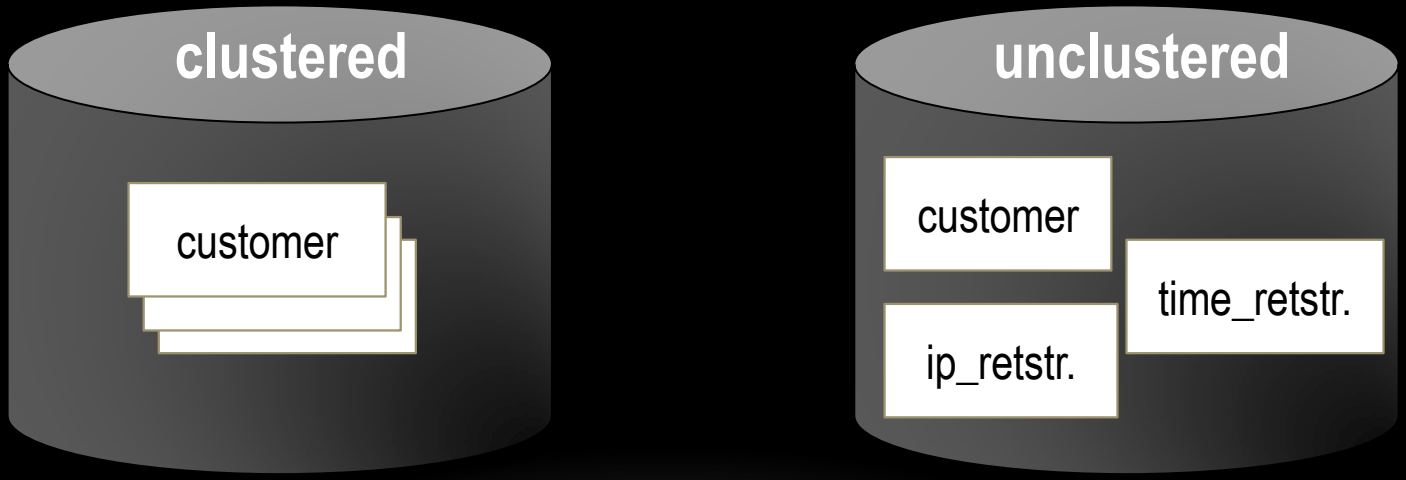


TABLE CLUSTER

And all data, that contains the same clustered key value are stored together.

unclustered

CUSTOMER_ID	FIRST_NAME	LAST_NAME	…
1	AAA	BBB	
2	CCC	DDD	

CUSTOMER_ID	IP
1	192.168.1.1
1	192.168.1.2
2	68.1.1.1

CUSTOMER_ID	TIME
1	MON 8-18
1	WEN 10-15
2	MON 8-18

TABLE CLUSTER

And all data, that contains the same clustered key value are stored together.

clustered

CUSTOMER_ID	FIRST_NAME	LAST_NAME	…
1	AAA	BBB	
	IP		
	192.168.1.1		
	192.168.1.2		
	TIME		
	MON 8-18		
2	CCC	DDD	

TABLE CLUSTER

What you will get

- Disk I/O is reduced for joins of clustered tables.
- Access time improves for joins of clustered tables.
- Less storage is required to store related table and index data because the cluster key value is not stored repeatedly for each row.

But be careful (do not use when)

- The tables are frequently updated.
- The tables frequently require a full table scans.
- The tables require truncating.

TABLE CLUSTER

There are 2 types of clusters

Index cluster

- Data is co-located within a single block based on a common column index, so separate cluster index should be created before adding table into cluster

Hash cluster

- Data is co-located within a single block based on a hash key and a hash function (own hash function can be used)

TABLE CLUSTER

Index VS Hash clusters

- Hash is faster
- Hash take less storage
- For hash you should know number of hash keys
- Hash cluster size and keys number can’t be changed
- Hash cluster loads CPU more, than index cluster

Questions
and
Thank YOU