2MU05_Bitmap.pdf **cAMÜ_EA**

INDEXSTRUKTÚRÁK III.

Molina-Ullman-Widom:

Adatbázisrendszerek megvalósítása

Panem, 2001könyv „5.4. Bittérkép indexek” fejezete alapján

Oracle: Indexek a gyakorlatban

Oracle® Database Concepts Part II. 5. fej. Séma objektumok dokumentáció alapján

5.4. Bitmap indexek

Személy

név	nem	kor	kereset
Péter	férfi	57	350000
Dóra	nő	25	30000
Salamon	férfi	36	350000
Konrád	férfi	21	30000
Erzsébet	nő	20	30000
Zsófia	nő	35	160000
Zsuzsanna	nő	35	160000

érték	vektor
férfi	1011000
nő	0100111

érték	vektor
30000	0101100
160000	0000011
350000	1010000

Bitmap indexek haszna

SELECT COUNT(*)

FROM személy

WHERE nem='nő' and kereset = 160000;

- 0100111 AND 0000011 = 0000011, az eredmény: 2.

SELECT név

FROM személy

WHERE kereset > 100000;

- 0000011 (160000) OR 1010000 (350000) = 1010011, azaz az 1., 3., 6. és 7. rekordokat tartalmazó blokko(ka)t kell beolvasni.

Tömörítés (bitmap)

- Ha a táblában n rekord van, a vizsgált attribútum pedig m különböző értéket vehet fel, ekkor, ha m nagy, a bitmap index túl naggyá is nőhet (n*m méret). Ebben az esetben viszont a bitmap indexben az egyes értékekhez tartozó rekordokban kevés az 1-es. A tömörítési technikák általában csak ezeknek az 1-eseknek a helyét határozzák meg.
- Tegyük fel, hogy i db 0-t követ egy 1-es. Legegyszerűbb megoldásnak tűnik, ha i-t binárisan kódoljuk. Ám ez a megoldás még nem jó: (a 000101 és 010001 vektorok kódolása is 111 lenne).
- Tegyük fel, hogy i binárisan ábrázolva j bitből áll. Ekkor először írjunk le j-1 db 1-est, majd egy 0-t, és csak ez után i bináris kódolását.
- Példa: a 000101 kódolása: 101101,

  a 010001 kódolása: 011011.

Tömörítés (bitmap)

- Állítás: a kód így egyértelművé válik.
- Tegyük fel, hogy m=n, azaz minden rekordérték különböző a vizsgált attribútumban.
- Ekkor, mivel a bitmap indexekben n hosszú rekordokról van szó, egy rekord kódolása legfeljebb $2\log_2 n$. Az indexet alkotó teljes bitek száma pedig $2n\log_2 n$, n^2 helyett.

II.Rész
Oracle: Indexek a gyakorlatban

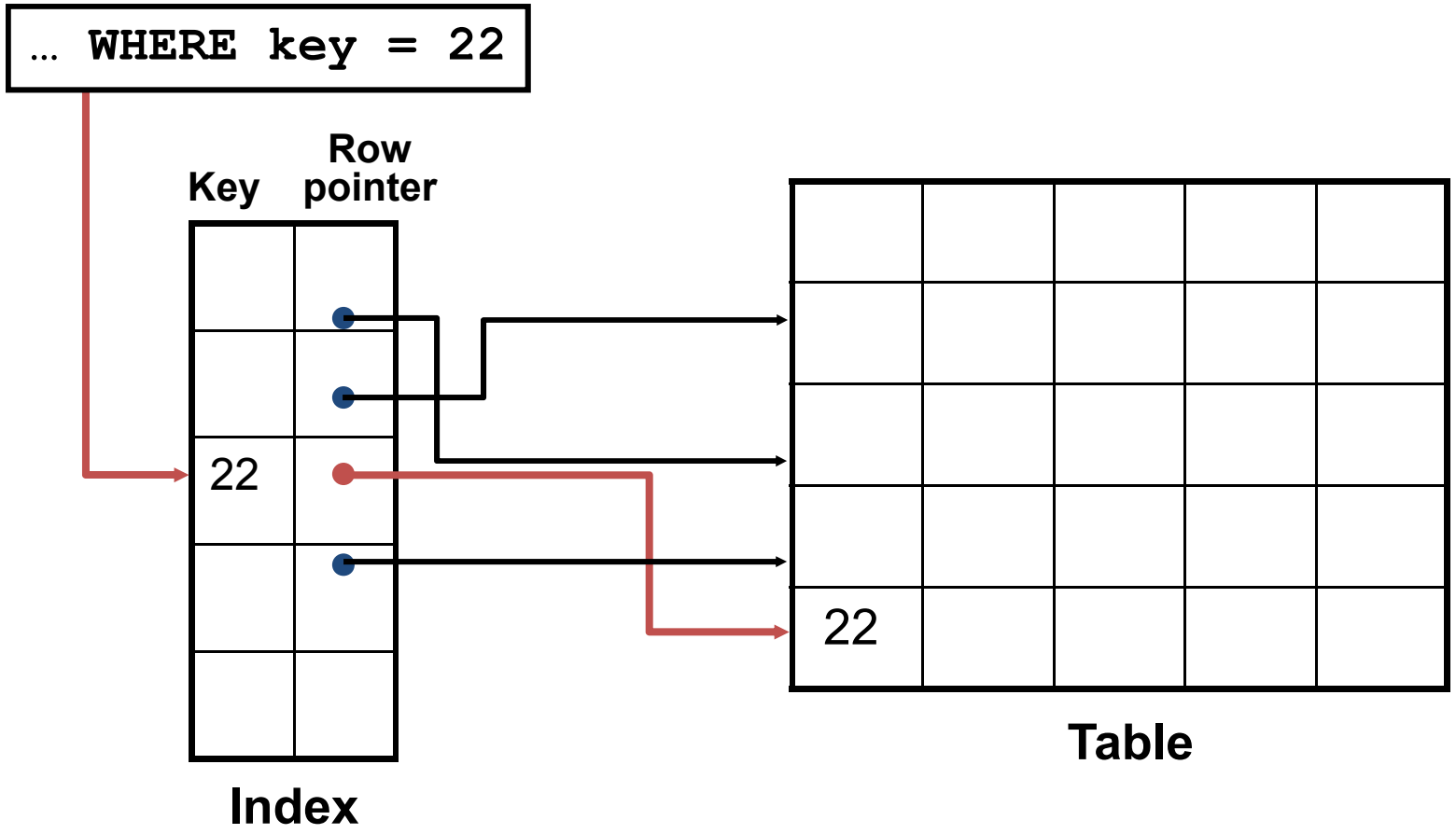
Oracle® Database 10.2 Concepts

Part II. Oracle Database Architecture

5. fejezet Séma objektumok:

- Indexek
  - -- B-fa (B+ fa)
  - -- bittérkép index
  - -- klaszterindex
- Indexszervezett táblák
- Partícionált táblák

Indexes

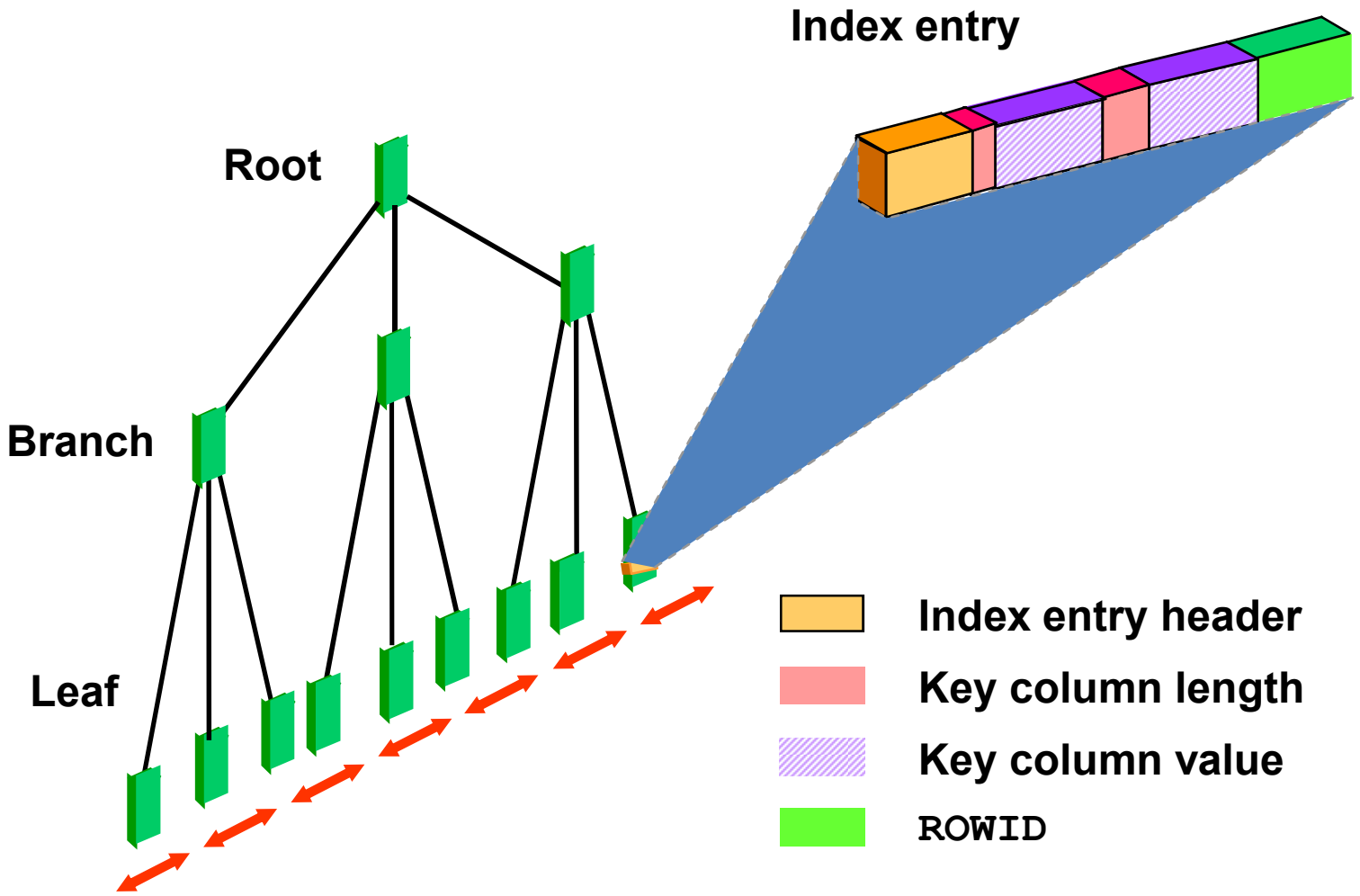


Types of Indexes

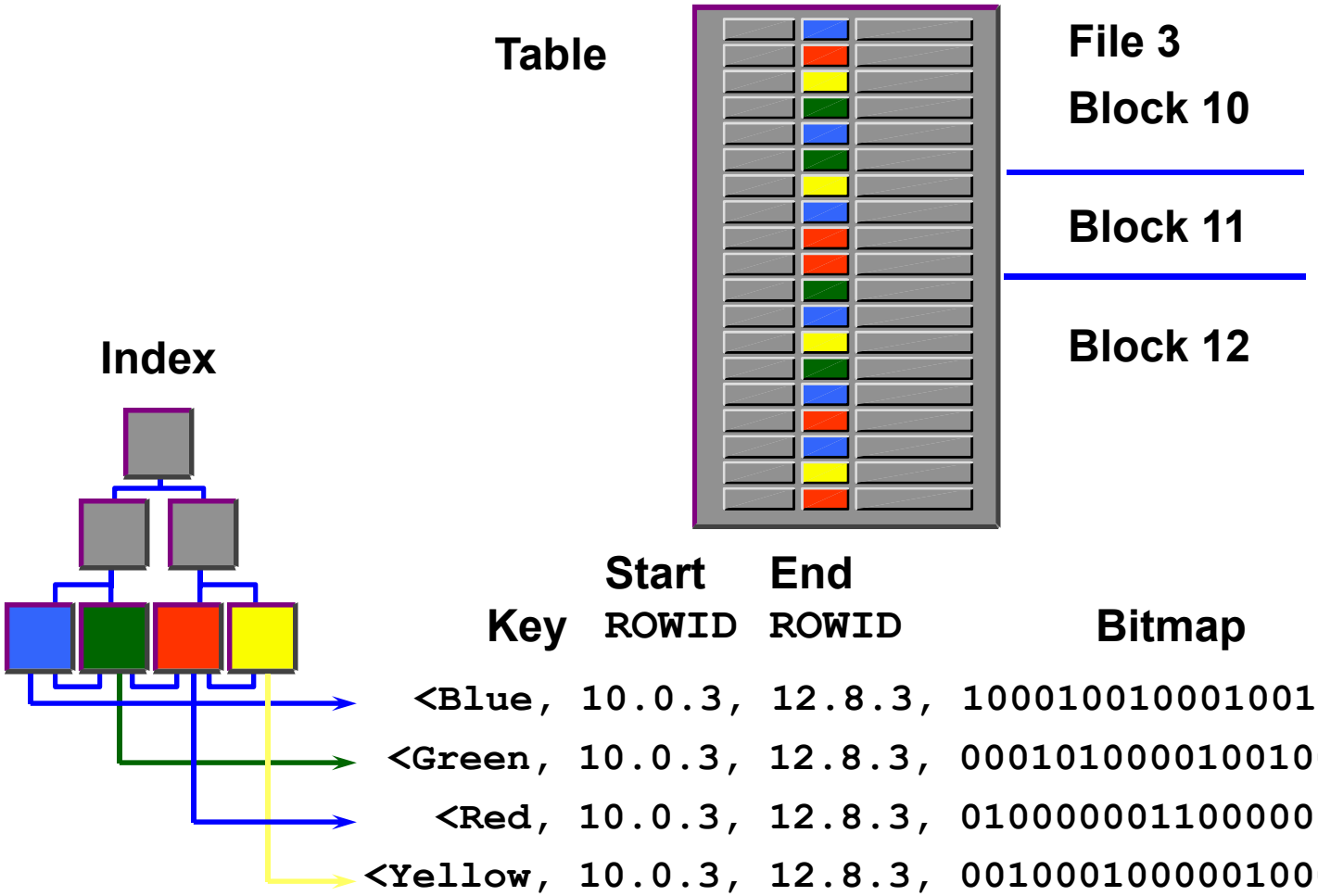
These are several types of index structures available to you, depending on the need:

- A B-tree index is in the form of a binary tree and is the default index type.
- A bitmap index has a bitmap for each distinct value indexed, and each bit position represents a row that may or may not contain the indexed value. This is best for low-cardinality columns.

B-Tree Index


Index entry
Root
Branch
Leaf
Index entry header
Key column length
Key column value
ROWID

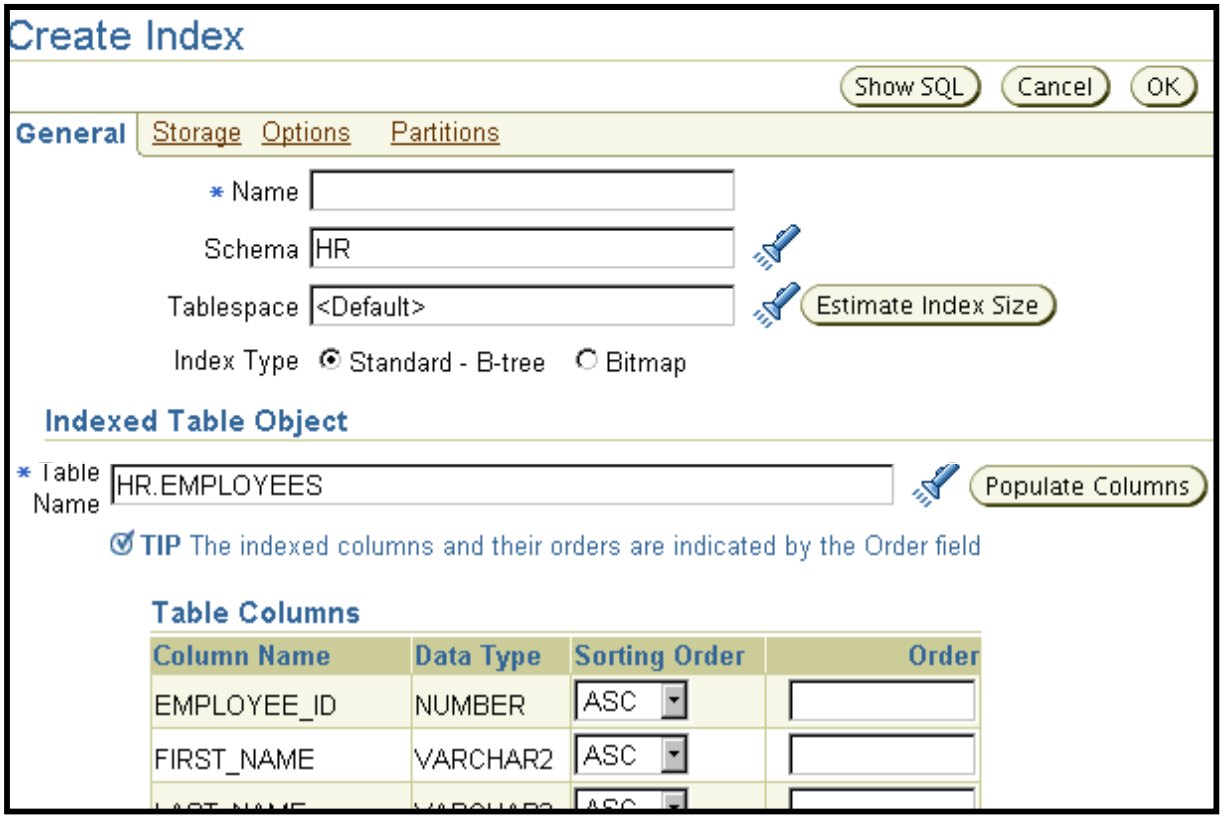
Bitmap Indexes



Index Options

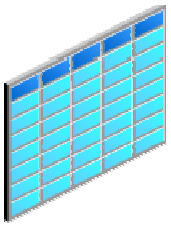
- A unique index ensures that every indexed value is unique.
- An index can have its key values stored in ascending or descending order.
- A reverse key index has its key value bytes stored in reverse order.
- A composite index is one that is based on more than one column.
- A function-based index is an index based on a function's return value.
- A compressed index has repeated key values removed.

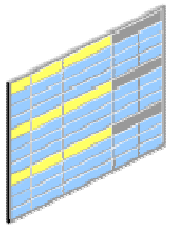
Creating Indexes



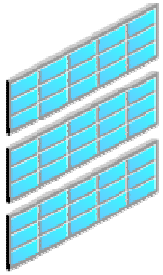

```
CREATE INDEX my_index ON
employees(last_name, first_name);
```

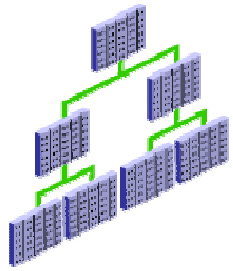
Table Types





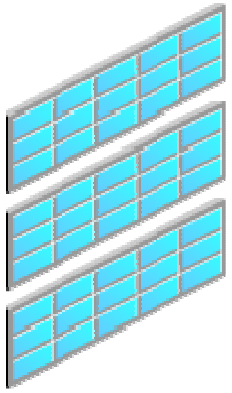
•Type	•Description
•Ordinary (heap-organized) table	•Data is stored as an unordered collection (heap).
•Partitioned table	•Data is divided into smaller, more manageable pieces.
•Index-organized table (IOT)	•Data (including non-key values) is sorted and stored in a B-tree index structure.
•Clustered table	•Related data from more than one table are stored together.





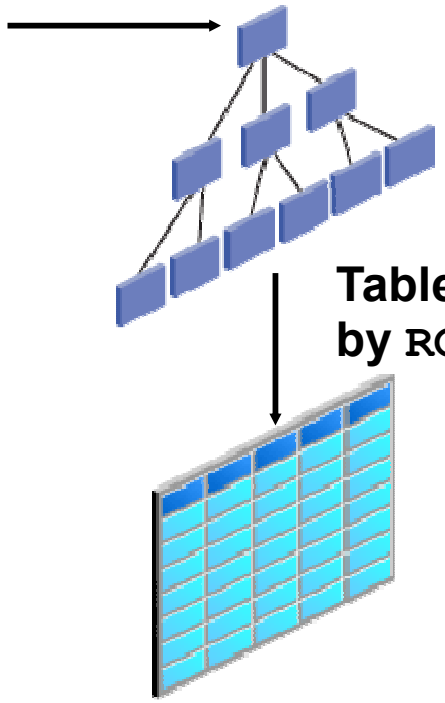
What Is a Partition and Why Use It?

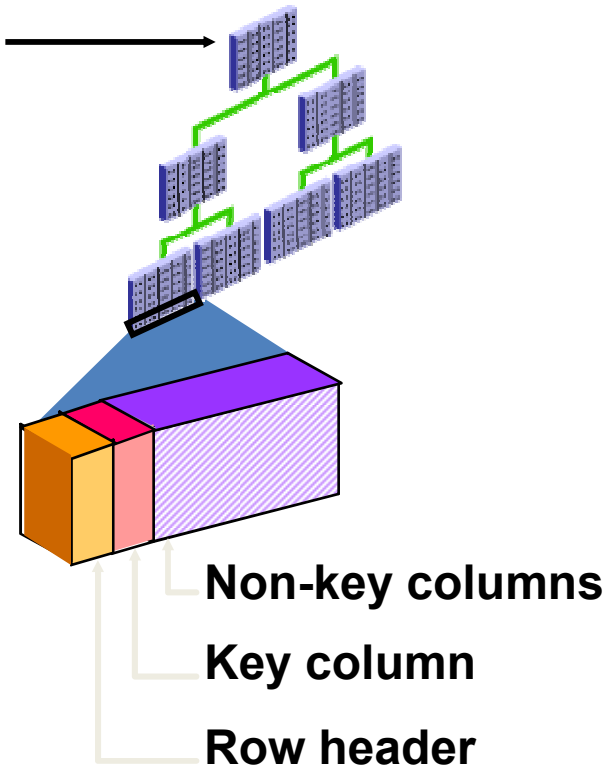
- A partition is:
  - A piece of a “very large” table or index
  - Stored in its own segment
  - Used for improved performance and manageability



Index-Organized Tables

- Regular table access

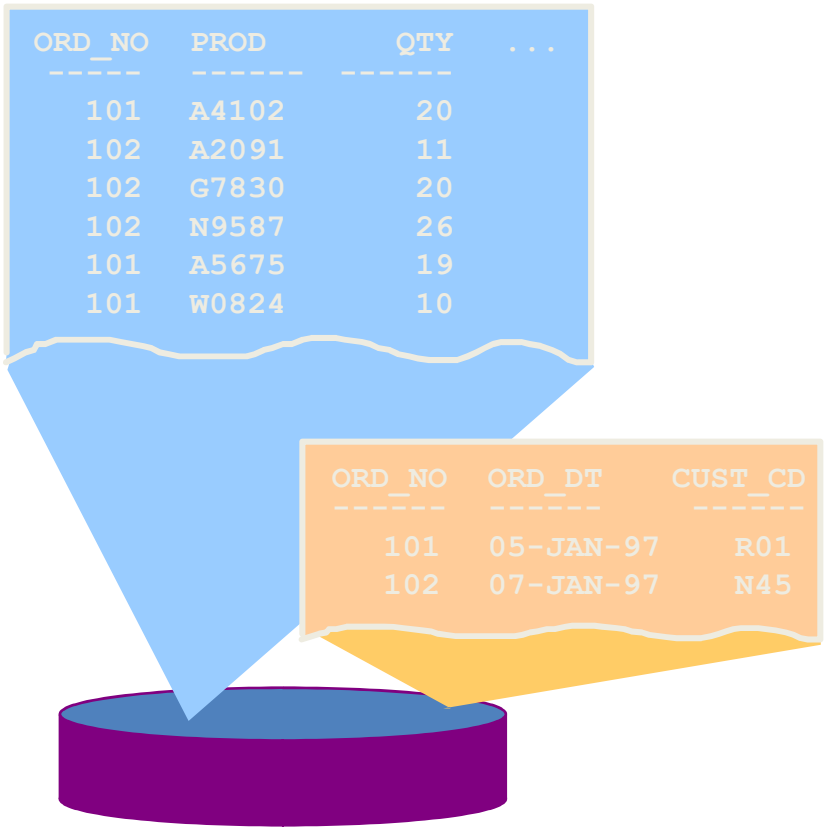




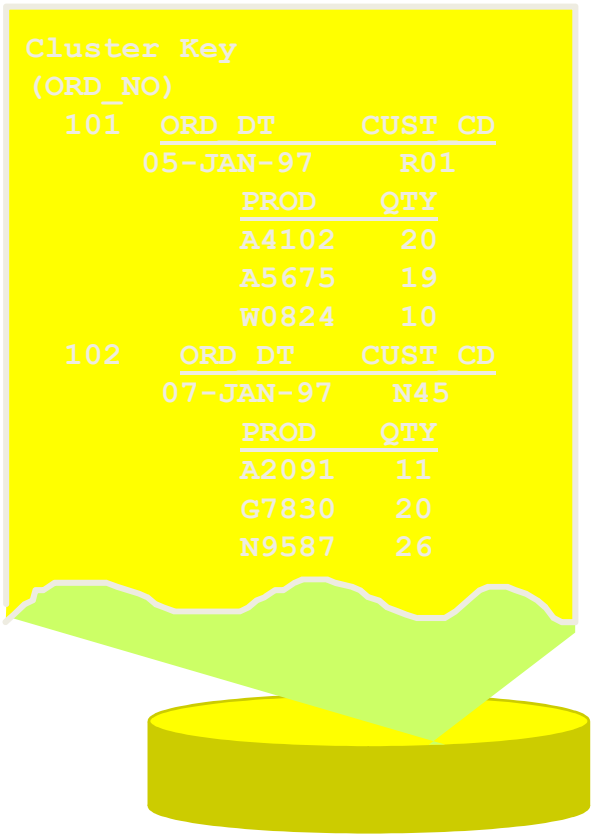
Index-Organized Tables and Heap Tables

- Compared to heap tables, IOTs:
  - Have faster key-based access to table data
  - Do not duplicate the storage of primary key values
  - Require less storage
  - Use secondary indexes and logical row IDs
  - Have higher availability because table reorganization does not invalidate secondary indexes
- IOTs have the following restrictions:
  - Must have a primary key that is not `DEFERRABLE`
  - Cannot be clustered
  - Cannot use composite partitioning
  - Cannot contain a column of type `ROWID` or `LONG`

Clusters

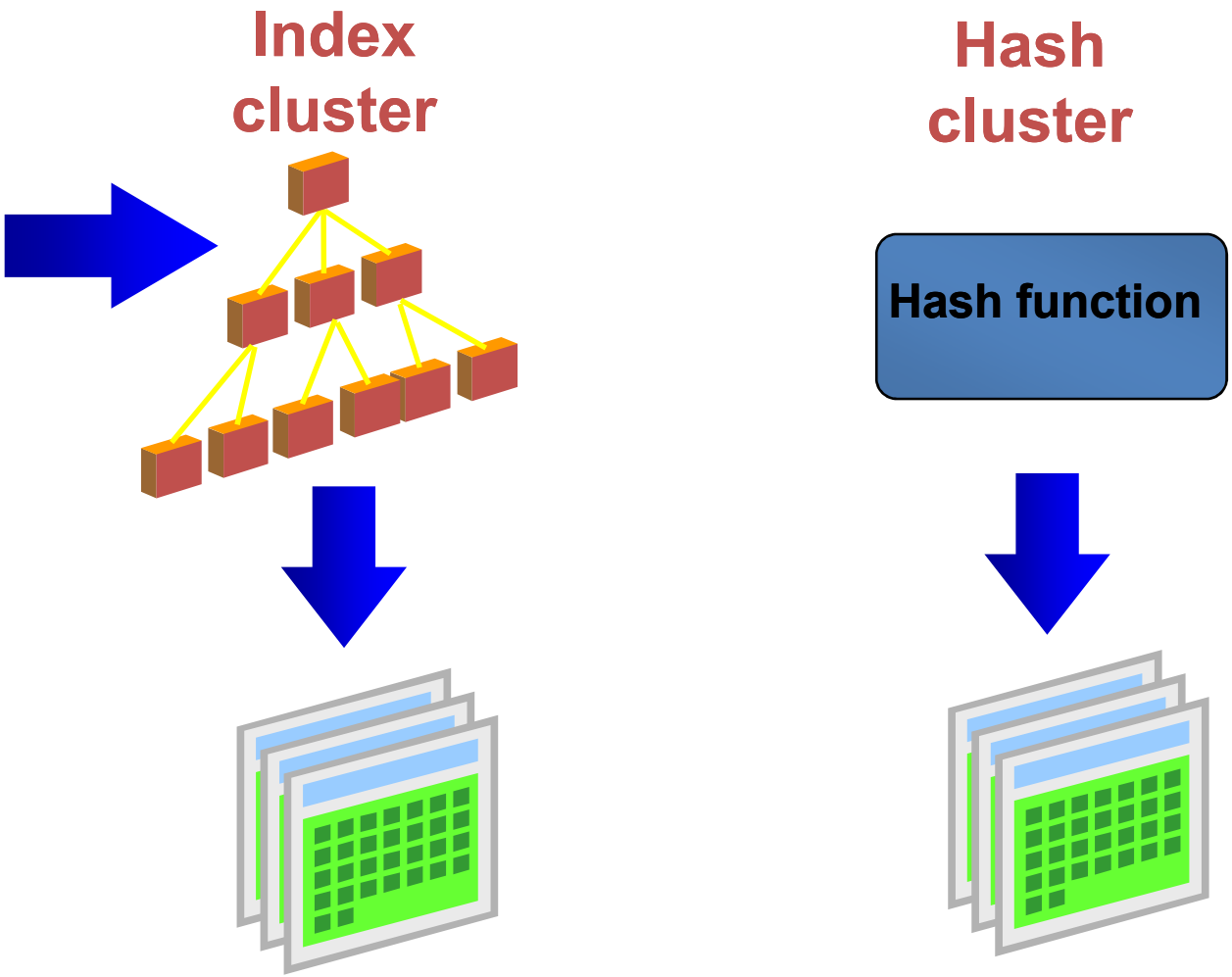


Unclustered `orders` and `order_item` tables



Clustered `orders` and `order_item` tables

Cluster Types



Situations Where Clusters Are Useful

Criterion	Index	Hash
Uniform key distribution	X	X
Evenly spread key values		X
Rarely updated key	X	X
Often joined master-detail tables	X	
Predictable number of key values		X
Queries using equality predicate on key		X

Indexelés

- **Klaszter (nyaláb, fürt)**
- Klaszterszervezés egy tábla esetén egy A oszlopra:
  - az azonos A-értékű sorok fizikailag egymás után blokkokban helyezkednek el.
  - CÉL: az első találat után az összes találatot megkapjuk soros beolvasással.
- **Klaszterindex:**
  - klaszterszervezésű fájl esetén index az A oszlopra
- Klaszterszervezés két tábla esetén az összes közös oszlopra:
  - a közös oszlopokon egyező sorok egy blokkban, vagy fizikailag egymás utáni blokkokban helyezkednek el.
  - CÉL: összekapcsolás esetén az összetartozó sorokat soros beolvasással megkaphatjuk.