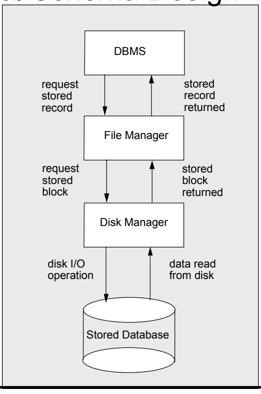
Internal Schema Design, Performance and Indexing

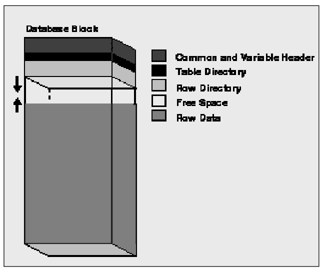
CS2312

Internal Schema Design



Data Blocks

- Oracle manages data in datafiles as data blocks
  - the smallest unit of I/O used by a database.
  - the smallest units of storage that Oracle can use or allocate.



In contrast, at the physical, operating system level, all data is stored in bytes. Each operating system has what is called a block size. Oracle requests data in multiples of Oracle data blocks, not operating system blocks. Set the data block size for each Oracle database when you create the database as a multiple of the operating system's block size within the maximum (port-specific) limit to avoid unnecessary I/O.

Performance Profiling

- Query Profile
  - frequency of certain queries
  - hit rate on relations
  - certain relations used together
  - selection attributes
- Update Profile
  - dynamic or static
  - hit rate of certain updates
  - predictable—pre-fetch strategies

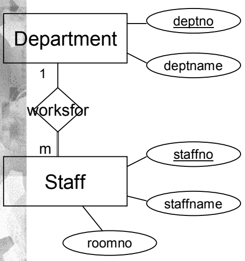
- Analysis and Monitoring using tuning tools

Performance: Joins with Composite FK.

Flight(<u>flightcode</u>, ukairport,holairport, depday, deptime…….)
Hotel(<u>hotelid</u>, hotelname,...)
GenericPackage(<u>flightcode,</u> <u>hotelid</u>, reservedrooms, reservedseats)
SpecificPackage(<u>flightcode,</u> <u>hotelid</u>, depdate,availseats,availrooms, …)
Booking(bookingid, contact, <u>flightcode,</u> <u>hotelid</u>, depdate, noofpeople,noofrooms,…)

GenericPackage(<u>gpackid</u>,flightcode, hotelid, reservedrooms, reservedseats)
SpecificPackage(<u>spackid</u>,gpackid, depdate, availseats, availrooms, …)
Booking(bookingid, contact, spackid, noofpeople,noofrooms,…)

Performance: Frequent Joins



Dept(<u>deptno</u>, deptname)

Staff(<u>staffno</u>, staffname, roomno, deptno)

1. Denormalise to 2NF

- Replicate attribute values

Staff(<u>staffno</u>, staffname, roomno, deptno, deptname)

staffno → staffname, roomno

deptno → deptname

<u>staffno</u>	staffname	roomno	deptno	deptname
10	Goble	2.82	1	Computer Sci
22	Paton	2.83	1	Computer Sci
31	Smith	1.100	2	Maths
49	Leuder	2.23	2	Maths

2. Physically storing a file resulting from a join

- Materialised View
- Update integrity management

<u>staffno</u>	staffname	roomno	deptno	deptname
10	Goble	2.82	1	Computer Sci
22	Paton	2.83	1	Computer Sci
31	Smith	1.100	2	Maths
49	Leuder	1.23	2	Maths

<u>staffno</u>	staffname	roomno	deptno
10	Goble	2.82	1
22	Paton	2.83	1
31	Smith	1.100	2
49	Leuder	1.23	2

deptno	deptname
1	Computer Sci
2	Maths

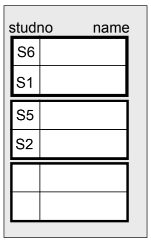
File Organisation

- Organisation of the data of a file into records, blocks and access structures
- Organisation
  - Unordered records
  - Ordered records
  - Hashing

File Organisation: Unordered Records

- Place records in the order they are inserted. New records inserted at the end of the file HEAP / PILE

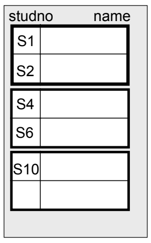
Insertion	efficient	
Deleting	expensive	reorganisation fragmentation
Searching	expensive	linear n/2
Retrieval in order	expensive	sort



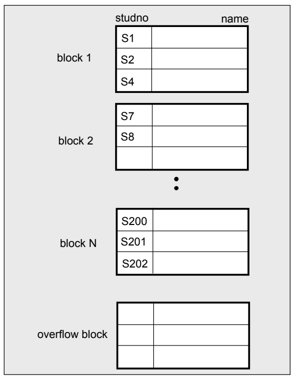
File Organisation: Ordered Records

- Physically order the records of a file on disk based on values of one of the fields — *ordering field / ordering key*

Insertion	expensive	reposition
Deleting	expensive	reorganisation & fragmentation
Searching on ordering key	more efficient	binary $\log_2(n)$
Searching on non-ordering key	expensive	linear n/2
Retrieval in order of ordering key	efficient	no sort

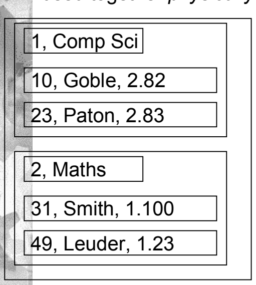


Overflow Blocks



3. Inter-file clustering

- Store records that are logically related and frequently used together *physically* close together on disk



cluster applied across multiple files
e.g. frequently access depts with their staff

Therefore *interleave* Dept and Staff

Oracle Inter file Clustering

Create a cluster named PERSONNEL with the cluster key column DEPTNO

Cluster key

CREATE CLUSTER personnel (deptno NUMBER(2)) ;

- Add the STAFF and DEPT tables to the cluster:

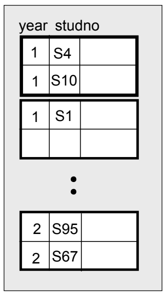
```
CREATE TABLE staff
(staffno  NUMBER  PRIMARY
KEY,
staffname  VARCHAR2(10)
roomno NUMBER(2,3),
deptno  NUMBER(2)  NOT
NULL )
CLUSTER personnel (deptno);
```

```
CREATE TABLE dept
(deptno  NUMBER(2),
deptname  VARCHAR2(9))
CLUSTER personnel
(deptno);
```

Intra file clustering

- cluster around a clustering field in a single stored file
- e.g. frequently access STUDENT by year

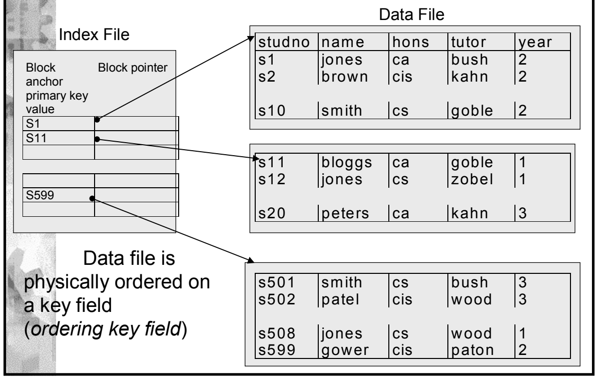
```
create table student
(studentno number(8) primary key,
givenname char(20),
surname char(20),
hons char(3),
tutorid number(4),
yearno number(1) not null,
cluster year(yearno),
...);
```



4. Construct Access Structures for the join attributes

- Access Structures / Access Paths
  - Indexes
  - Multi-level indexes
  - B trees, B$^+$ trees
  - Hashing
  - Bitmap
- Access Methods
  - routines that allow operations to be applied to a file

Primary Index



studno	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s10	smith	cs	goble	2

s11	bloggs	ca	goble	1
s12	jones	cs	zobel	1
s20	peters	ca	kahn	3

s501	smith	cs	bush	3
s502	patel	cis	wood	3
s508	jones	cs	wood	1
s599	gower	cis	paton	2

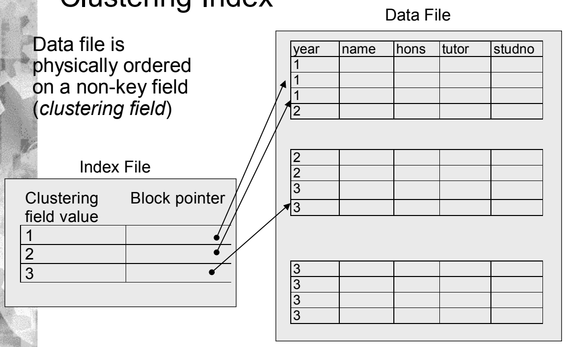
Data file is physically ordered on a key field (*ordering key field*)

Oracle: Index-organized Tables

```
create table student
(studentno number(8) primary key,
givenname char(20),
surname char(20),
hons char(3),
tutorid number(4),
yearno number(1) not null,
ORGANIZATION INDEX TABLESPACE students
OVERFLOW TABLESPACE students_overflow;
```

Clustering Index

Data file is physically ordered on a non-key field (*clustering field*)



Clustering Index in Oracle

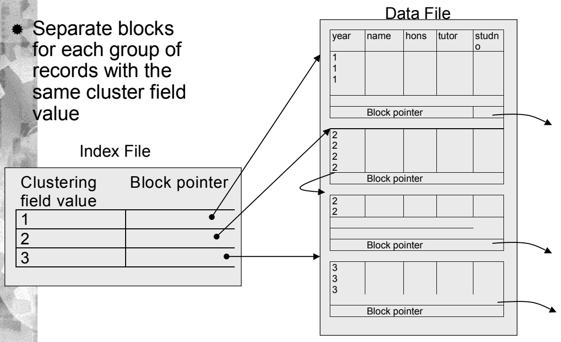
- The following statement creates a cluster index on the cluster key of PERSONNEL:

CREATE INDEX idx_personnel ON CLUSTER personnel;

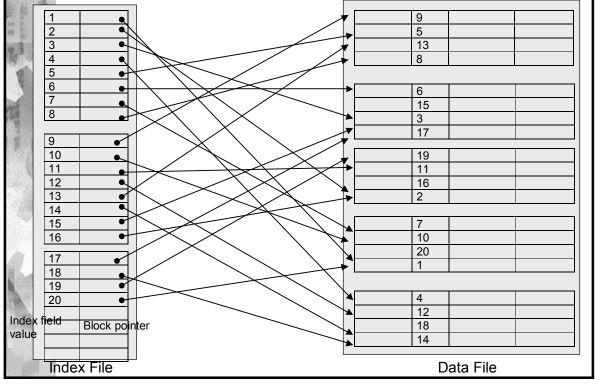
- After creating the cluster index, you can insert rows into either the STAFF or DEPT tables.

Clustering Index with Separate Blocks

- Separate blocks for each group of records with the same cluster field value



Dense Secondary Index on a non-ordering key field

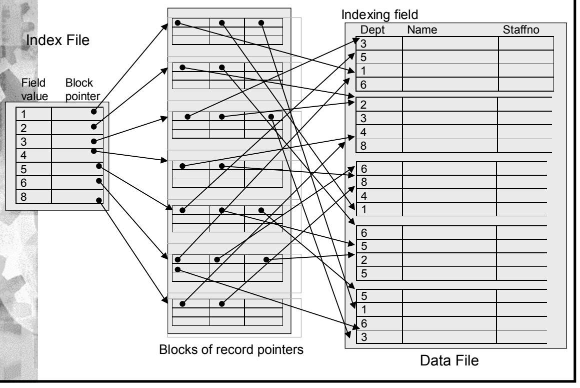


Oracle: Create Index

```
create table enrol
  (studno number(8),
  courseno char(5),
  primary key (studno, courseno),
...);
```

- CREATE INDEX enrol-idx1 ON enrol (studno, courseno);
- CREATE INDEX enrol-idx2 ON enrol (courseno, studno);

Secondary Index on a non-key field



Dense & Sparse Indexes

- Dense Index
  - Every record in the data file is in the index
- Sparse Index
  - Not every record in the data file is in the index.
  - The index indicates the *block* of records
    - takes less space
    - quicker to scan the index
    - efficient

  but...no existence test based on the index
- A file can have one sparse index and many dense indexes, because a sparse index relies on a unique physical ordering of the data file on disk

Types of Keys

- Unordered data files $\Rightarrow$ lots of secondary indexes
- Specify ordering attribute for file
  - primary / clustering index
  - attribute used most often for joins

	Ordering Field	Non-ordering Field
Key field	Primary Index	Secondary Index (key)
Non-key Field	Clustering Index	Secondary Index (non-key)

Analysing database queries and transactions

- Each query
  - files that will be accessed
  - fields whose value is retrieved *access paths*
  - fields specified in selection conditions *access paths*
  - fields specified in joins *access paths*
- Each update transaction
  - files that will be updated
  - type of update operation on each file
  - fields used in selection conditions
  - fields whose value is modified *avoid access structure*

Analysing database queries and transactions

- Expected frequency of invocation of queries and transactions
  - expected frequency of each field as a selection field or join field over all transactions
  - expected frequency of retrieving and /or updating each record
- Analysing time constraints of queries and transactions
  - stringent performance constraints
  - influence access paths on selection fields
- Analysing expected frequency of update operations
  - volatile files
  - reduce number of access paths

Types and Properties of Indexes

Type of Index	Properties of Index Type		
	Number of (first level) Index Entries	Dense or Non-dense	Block Anchoring on the Data File
Primary	Number of blocks in data file	Non-dense	Yes
Clustering	Number of distinct index field values	Non-dense	Yes/No
Secondary (key)	Number of records in data file	Dense	No
Secondary (non-key)	Number of records of number of distinct index field values	Dense or Non-dense	No

Index Summary

- Speeds up retrieval but slows down inserts and updates
- Improve performance when
  - relations are large
  - queries extract < 25% of all tuple in a relation
  - a where clause is properly constructed
- Two main considerations:

1. Organisation
2. Access
   - sequential *range queries*
   - direct *criteria queries*
   - existence tests

Data Definition: Create Table

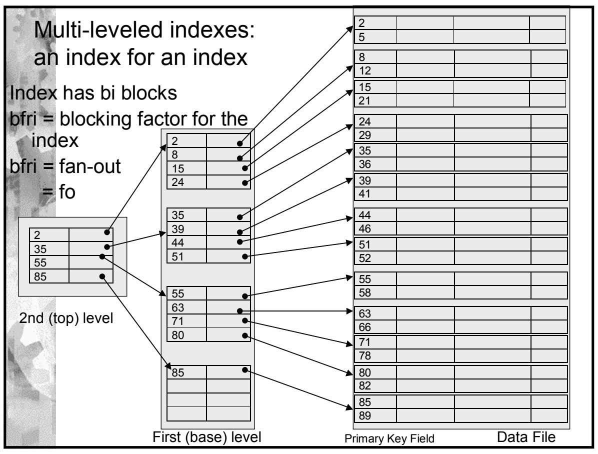
```
create table year
  (yearno number(1) primary key,
  yeartutorid number(4),
     yeartut_uk unique
     exceptions into bad_tutors
     using index
     not null
     constraint tut_fk
     foreign key (yeartutorid) references
  staff(staffid))
     tablespace cags_course
     storage (initial 6144
                    next 6144
                    minextents 1
                    maxextents 5
                    pctincrease 5
                    pctfree 20);
```

Multi-leveled indexes: an index for an index

Index has bi blocks

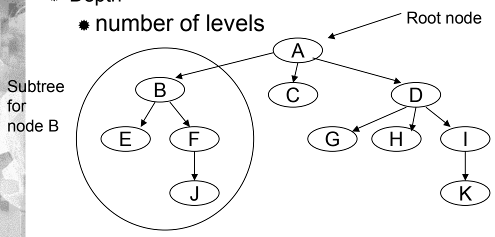
bfri = blocking factor for the index

bfri = fan-out = fo



Tree Indexes

- Order
  - a measure of the number of indexing field values at each node
- Depth
  - number of levels



B-trees Balanced Trees

- Every leaf is at the same level
- Ordered - Search time O(log(n))
- Predictable search time
- Efficiency - each node = block

- A key value is stored once, with the address of the associated data record

B trees Order p

1. at least (p-1)/2 and at most p-1 key values at each internal node and leaf

∴ internal nodes and leaves must always be at least half full (or half empty)

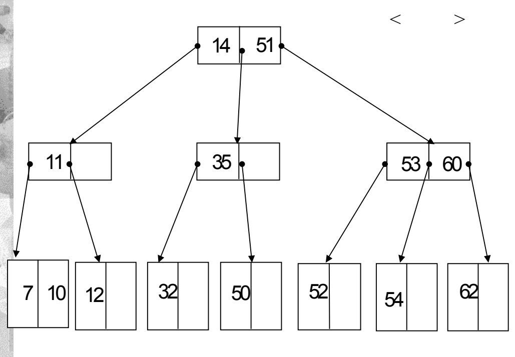
2. the root must contain at least one key and 2 pointers (thus 2 child nodes) unless its a leaf

∴ can't have an empty start point for a non-null tree

3. for k key values at a node, there will be k+1 pointers to child nodes

∴ a node must have between p/2 and p tree pointers (child nodes)

B tree



B trees

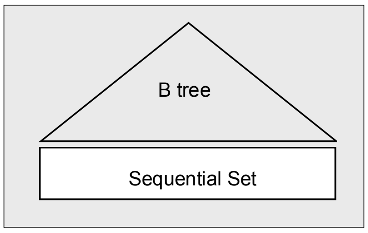
- Predictable search pattern
- at most X node comparisons for a tree of X levels
- Dense index addresses record location index value can lie anywhere in the tree

Cost maintenance
but....

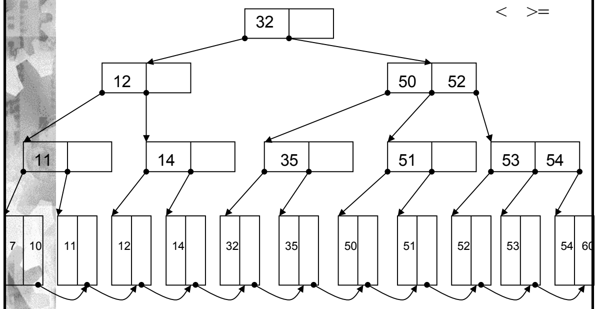
? Sequential access
? Range queries
? Sparse index

B$^+$ trees

- Amendment to B tree: addresses for data records are in the leaves and no where else



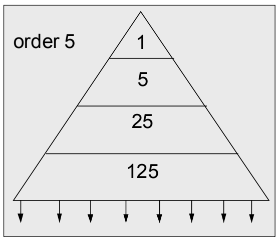
B$^+$ trees



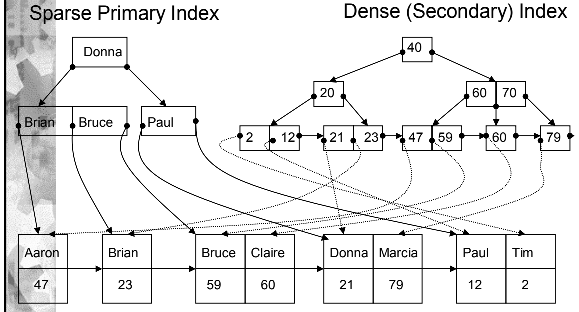
B$^+$ trees

1. Each node has at most p-1 comparisons
2. Number of nodes visited is limited by the depth of the tree

- A tree with k levels has
  - at most $(p)^{(k-1)}$ leaves
  - at least $(p/2)^{(k-1)}$ leaves
- Each leaf has
  - p/2 addresses if dense or
  - 1 block of n if sparse



Sparse / Dense B+ Trees



Data blocks and indexes

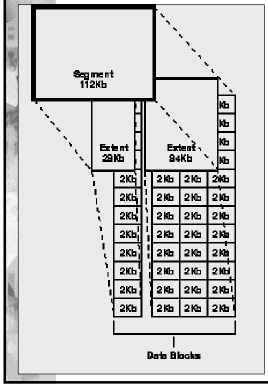
B$^+$ trees

- Sequential and direct access
- Straightforward insertion and deletion maintaining ordering
- Grows as required—only as big as needs to be
- Predictable scanning pattern
- Predictable and constant search time

but .….

- maintenance overhead
- overkill for small static files
- duplicate keys?
- relies on random distribution of key values for efficient insertion

Data Blocks, Extents, and Segments

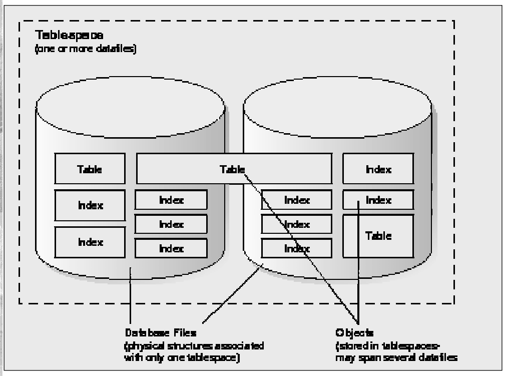


- Data stored in *data blocks* (also called logical blocks, Oracle blocks, or pages).
- One data block corresponds to a specific number of bytes of physical database space on disk
- An *extent* is a specific number of contiguous data blocks allocated for storing a specific type of information.
- A *segment* is a set of extents that have been allocated for a specific type of data structure.
- Each table's data is stored in its own data segment, while each index's data is stored in its own index segment.

Tablespaces and Datafiles

- An Oracle database is divided into one or more logical storage units called tablespaces. The database's data is collectively stored in the database's tablespaces.
- Each tablespace in an Oracle database consists of one or more files called datafiles. These are physical structures that conform with the operating system in which Oracle is running.
- A database's data is collectively stored in the datafiles that constitute each tablespace of the database.
- The simplest Oracle database would have one tablespace and one datafile. A more complicated database might have three tablespaces, each consisting of two datafiles (for a total of six datafiles).

Tablespaces and Datafiles


Tablespace
(one or more datafiles)
Table
Index
Database Files
(physical structures associated with only one tablespace)
Objects
(stored in tablespaces- may span several datafiles)

Why bother with tablespaces?

- Uses tablespaces to:
  - control disk space allocation for database data
  - assign specific space quotas for database users
  - control availability of data by taking individual tablespaces online or offline
  - perform partial database backup or recovery operations
  - allocate data storage across devices to improve performance
- Different functions
  - System tablespace
  - Temporary tablespaces
  - User tablespaces
  - Read-only table spaces

Example

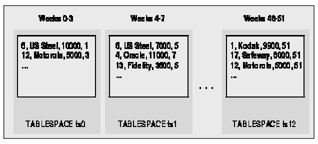
```
create table year
(yearno number(1) primary key,
yeartutorid number(4),
   yeartut_uk unique not null
   constraint tut_fk
   foreign key (yeartutorid) references
staff(staffid))
   tablespace secondyr_course
   storage (initial 6144
                  next 6144
                  minextents 1
                  maxextents 5
                  pctincrease 5
                  pctfree 20);
```

Partitioned Tables in Oracle

- Supports very large tables and indexes by allowing users to decompose them into smaller and more manageable pieces called partitions.
- Each partition is stored in a separate segment and you can store each partition in a separate tablespace, so:
  - contain the impact of damaged data.
  - back up and recover each partition independently.
  - balance I/O load by mapping partitions to disk drives.
- Useful for:
  - Very Large Databases (VLDBs)
  - Reducing Downtime for Scheduled Maintenance
  - Reducing Downtime Due to Data Failures
  - DSS Performance – I/O Performance
  - *Disk Striping*: Performance vs. Availability

Example

- A sales table contains historical data divided by week number into 13 four-week partitions.

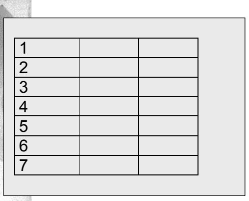


```
CREATE TABLE sales
( acct_no  NUMBER(5),
acct_name  CHAR(30),
amount_of_sale  NUMBER(6),
week_no  INTEGER )
PARTITION BY RANGE ( week_no ) ...
(PARTITION VALUES LESS THAN ( 4) TABLESPACE ts0,
PARTITION VALUES LESS THAN ( 8) TABLESPACE ts1,
  ...
 PARTITION VALUES LESS THAN ( 52 ) TABLESPACE ts12 );
```

Hashing: Hash Clusters

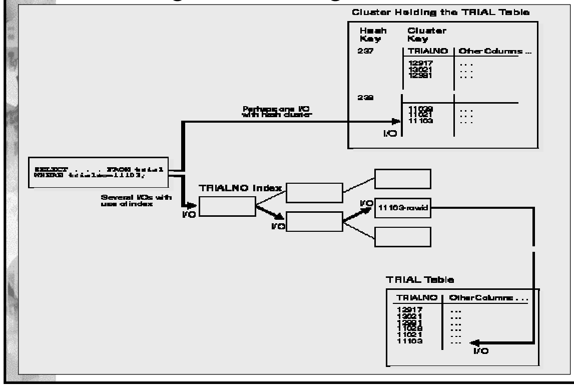
- Physically store the rows of a table in a *hash cluster* and retrieve them according to the results of a hash function.
- A *hash function* generates a distribution of numeric values, called *hash values*, which are based on specific cluster key values. The key of a hash cluster can be a single column or composite key.
- To find or store a row in a hash cluster, apply the hash function to the row's cluster key value; the resulting hash value corresponds to a data block in the cluster, which you then reads or writes on behalf of the issued statement.

Hashing Example



- Hash function
- mod ($\frac{\text{hash key}}{\text{prime number}}$)
- Collisions
- Rehash functions
- Oracle
  - internal hash function
  - user defined hash function

Hashing vs Indexing



Choice of Hashing

- If a key attribute is used mainly for equality selection and join
- Nothing depends on layout order of data file
- Data files are static and of known size