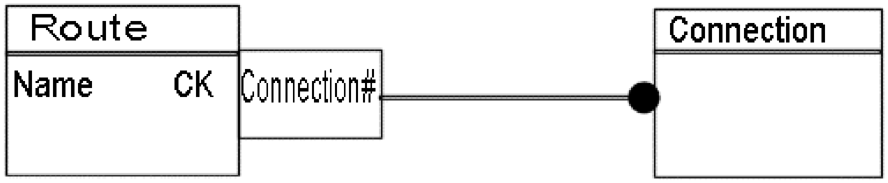
Databases Study Material

Qualified Association: when translated into relational schema's you must first migrate attributes across association.



Translating the above schema into relational schema, we migrate Connection# and Route.Name to Connection to create primary key for Connection class. The resulting schema is as follows:

Route:
------
Name
Constraint pk primary key(Name)

Connection
----------
Connection#
Route_name
constraint fk foreign key(Route_name) references Route(Name)
constraint pk2 primary key(Connection#, Route_name)

Indexing
--------

Situation		Index
attribute=value	-	hash based index on selected attributes
attribute>value	-	B-Tree based index on selected attributes
attribute1=value1 AND attribute2=value2	-	multiple-attribute index key
index enables index only query computation	-	multiple-attribute index key

An index is implemented as a B-Tree – from 22.pdf page 3

Case: Rows are stored in random order in disk memory, and rows are not clustered. The cost to bring 't' rows from disk memory into main memory is:

(1) best case: proportional to t/bfr.
(2) Worst case: proportional to t.
(3) Average case: proportional to (t/bfr + t)/2

Time which is needed to perform an operation on an index (select, update, insert, delete) is proportional to $F(a_i)$, where:
(1) a_I is an attribute over which index is constructed
(2) if B-Tree technique is used for implementation then **$F(a_I) = ceil(log_n N_i)$** where N_i is a number of distinct values (keys) of attribute a_I and n is B-Tree's fanout (number of pointers per node).

In a selection with a multiple condition where clause, ie

Select * from table_name where C1 AND C2 AND C3;

For example, select * from mytable where name='tom' and age=15 and sex='m';

Index processing stage

For each attribute a_k used in a query and such that respective index I_k exists access I_k and create a list L_k of identifiers of all rows such that attribute a_k has a value specified in the query.

List processing stage

Compute intersection of all L_1 , ..., L_m lists and store the result in L_{result}.

The following formula can be used to work out the **index processing costs**:

Index Processing Costs

$$C_i = \sum_{k=1}^{m} F(a_k)$$

Where **$F(a_k) = ceil(\log_n N_k)$**

List Processing Costs

For condition true AND condition true

$| L_{result} | = N * ((((1/N_1 * 1/N_2 * … * 1/N_m)$

For condition true OR condition true (MORE COMPLICATED)

- **Use Ven-Diagram**

Indexing Total costs = Index processing costs + table processing costs

CASE STUDY:

Consider a relational table; EMPLOYEE(name, salary, age, address)

$N = 1000$
$N_{name} = 1000$
$N_{salary} = 10$
$N_{age} = 40$
$N_{address} = 1000$
bfr = 3 (3 rows per block)
n =10 (B-tree fanout)

Consider a query with conditions (salary = 40K and age = 25)
Savings from using an index
The costs of sequential read = 1000/3 ˜ 333 blocks (not indexed)

If age attribute indexed (and salary is NOT indexed) then that means that we need to transverse all relevant age value rows only.

Index processing costs = $(\log_{10}40 + 1)$ = 2 blocks
Table processing costs = ((1000/(40))/3 = 9 blocks

Total costs = 11;

update of index costs is same as transversal, except that it must do it twice as much, because update is delete and add.

Insert and delete is same as transversal.

Benefits from index existence are equal to
r qry * benefits from query processing -
(r upd * update processing costs +
r ins * insert processing costs +
r del * delete processing costs)
where r qry , r upd , r ins , r de l are the probabilities of
query, update, insert and delete operations

Question 2 (2 marks)

Assume that relation table Product (P#, PNAME, PRICE, MANUFACTURER) contains on 10^4 rows, attribute P# is a primary key, products are manufactured by 50 manufacturers, average number of rows per disk block is equal to 20, block size is 2K.

Assume that database administrator created a clustered B-tree index on attribute MANUFACTURER and that height of B-Tree is equal to 2.

(1) how many read block operations are needed to compute the following query

```
SELECT PNAME, PRICE
FROM PRODUCT;
```

Answer:

Total rows /number of rows per blocks = number of block reads.
$10^4/20=500$ block reads

(2) How many read block operations are needed to compute the following query

```
SELECT PNAME, PRICE
FROM PRODUCT
WHERE PRICE > 100 AND MANUFACTURER = ‘GOLDENBOLTS’;
```

Answer:

```
//////////////////////////////////////////////////////////
///////////    USE IF NEEDED TO BE VERY ACCURATE    //////
///////////        OTHERWISE JUST USE HEIGHT        //////
///////////         AS NUMBER OF INDEX BLOCKS READ  //////
```

B-Tree Theory of height

$h <= \log_t ((n+1)/2)$

h= height
t = fanout (“branchiness”)
n = number of keys

SO:

$2 <= \log_t ((50+1)/2)$
t2 = (50+1)/2
t = 5.049

Number of distinct keys of manufacturer = $10^4/50 = 200$

$F(a_i) = \text{ceil}(\log_n N_i)$
$= \text{ceil}(\log_{5.049} 200)$
$= 4$

index processing costs using B-Tree theory of height method = 4

```
////////////////////////////////////////////////////////////
```

In most cases we just use height of the tree, as given, directly as the processing costs (A accurately real-life problem would yield much more similar results for both!). From this we can say:

Because we have around 200 keys, and 20 rows per block.

```
List processing costs   = number of values(keys)/rows per block
                        = 200/20
                        = 10

Total processing costs  = List processing costs + index processing costs
                        = 10 + 2
                        = 12
```

(3) How many read block operations are needed to compute the following query:

```
SELECT MANUFACTURER
FROM PRODUCT
WHERE P# = 123456 AND MANUFACTURER = ‘GoldenBolts’;
```

Answer:

Because P# is a key, and is automatically indexed, and because it is unique, we don’t use MANUFACTURER index (we don’t need to). Because We must transverse the index:

Index processing costs = height of tree = 2 block reads;
List processing costs = cost of reading one row = 1 block reads;

Total Block reads = 3;

(4) How many read block operations are needed to compute the following query:

```
SELECT count(*)
FROM PRODUCT
WHERE PNAME = ‘bolt’ and MANUFACTURER = ‘GoldenBolts’;
```

Answer:

Here we have count on an index (MANUFACTURER).

So what we must do is access index, and transverse only those rows that are for manufacturer = ‘GoldenBolts’;

```
Total read  = 2 + (104/50)
            = 2 + 200
            = 202;
```

Question 3

Consider the following CREATE TABLE statement

```
CREATE TABLE PRODUCT(
P#    NUMBER(5)   NOT NULL PRIMARY KEY,
PNAME VARCHAR(20) NOT NULL,
PRICE NUMBER(9,2) NOT NULL,
MANUFACTURER      VARCHAR(50) NOT NULL)
TABLESPACE HARDWARE
STORAGE (INITIAL 20K NEXT 40K MINEXTENTS 2 MAXEXTENTS 3
PCTINCREASE 100)
PCTFREE 10
PCTUSED 80;
```

Assume that size of database disk block is equal to 2K.

(1) Find the maximum size of table PRODUCT.

Answer:

20 + 40 + 80 = 140K

(2) Assume that tablespace HARDWARE contains only the following free extents:

3 extents 20K each
1 extent 100K

Is it possible to allocate all extents for table PRODUCT ? justify.

Answer:

First, explain how allocation of space works.

An extent is a contiguous collection of blocks.

A new extent can only be allocated, therefore, over contiguous blocks.

This means that the first 20K extent required can take the first 20K available.

The next extent required is 40K. It cannot use the next two available 20K extents because the new extent must be made of contiguous blocks. Therefore it must use the available 100K extent and break it up, leaving a 60K extent free.

The remaining required 80K extent needs 80K of contiguous blocks. We have available 2 X 20K + 1 X 60K free blocks remaining.
This is a total of 100K. That is more than what is required BUT because they are not contiguous blocks, and because there are no one single extent large enough to hold the 80k extent, **You cannot allocate all the extents for this table PRODUCT.**

Question 4 Clustering

Assume that relational table R occupies 300 data blocks, relational table S occupies 400 data blocks, relational table T occupies 200 blocks.

Join of R and S requires 1000 read block operations and it is computed 8 times per day,

Join or R and T requires 600 read block operations and it is computed 10 times per day.

Find the best clustering schema for the relational tables R, S, T.

Answer:

Clustering can only be done on two of the 3 involved tables.

Therefore, we must find the best two providing the most benefits.

Benefits (R,S) = (1000 – (300 + 400)) * 8 = 2400;
Benefits (R,T) = (600 – (300 + 200)) * 10 = 1000;

Therefore clustering using table R and S is more beneficial.

Relational Algebra and Query Processing

Selection

σ = Selection

ϕ = Condition

$\sigma_{\phi}(r)$

$\sigma_{salary>80000}(Employee)$

```
select *
from employee
where salary > 80000;
```

Projection

π = Projection

$\pi_x(r)$

πname,salary(employee)

```
Select name, salary
From Employee;
```

Join

$R \bowtie S$

Find full information about departments and employees from each department

Employee dname $\bowtie$ deptname Department

```
Select *
From Employee, Department
where Dname = Department;
```

Cartesian Product

X = Cartesian Product

Employee X Department

```
Select *
from Employee, Department;
```

Set Operations

$R \cup S$, $R \cap S$, $R - S$

Union, intersection, minus

Find all employees who are not managers

```
select *
from EMPLOYEE
MINUS
select *
from MANAGER;
```

Optimisation

Optimisation of queries may be done through **syntax based** optimisation, or **Cost based** Optimisation.

Syntax Based Optimisation

First you must know how to translate select statements into relational algebra expressions.

```
SELECT P#, PNAME, S#, SNAME, QTY
FROM PART, SP, SUPPLIER
WHERE PART.P# = SP.P# AND
SP.S# = SUPPLIER.S# AND
PRICE < 100 AND
(COLOUR = ‘RED’ OR COLOUR = ‘BLUE’) AND
CNAME = ‘GOLDENBOLTS’;
```

Becomes

$$\pi_{\text{p\#, pname, s\#,sname, qty}}(\sigma_{\text{price<100 and (colour = 'red' or colour='blue') and cname = 'goldenbolts'}}(\text{Part} \rhd\lhd \text{SP} \rhd\lhd \text{Supplier}))$$

To implement Syntax Based implementation means to re-arrange the order of the syntax of the expression so to make the query more efficient in the way that it is executed.

For instance, it is more efficient to join only the parts of a table that you need, rather than the whole table.

$$\pi_{\text{p\#,pname, s\#, sname, qty}}((\sigma_{\text{price<100 and (colour='red' or colour='blue)}}(\text{Part})) \rhd\lhd (\sigma_{\text{cname='goldenbolts'}}(\text{Supplier})) \rhd\lhd \text{SP})$$