CIS 560 – Database System Concepts

Lecture 24

Indexes and B+ Trees

October 30, 2013

Credits for slides: Chang, Ullman, Whitehead.

Outline

Last:

- Indexes and B-trees 14.1-14.2

Today:

- Indexes and B-trees 14.1-14.2

Next:

- Query execution 15.1-15.6
- Query optimization 16

2

Planning

- Assignment 7 (concurrency control) due 11/1
- Assignment 8 (indexes) due 11/8
- Assignment 9 (query optimization) due 11/15
- Exam 2 (assignments 6-9) – 11/20
- Project assignment due 11/22
- Quiz from special topics – 12/06
- Project presentations 12/9, 12/11, 12/13
- Project reports – finals weeks

3

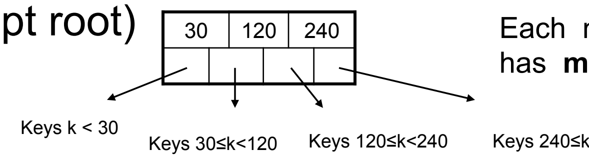
Review

- Clustered vs unclustered indexes
- Dense/sparse
- Primary/secondary

4

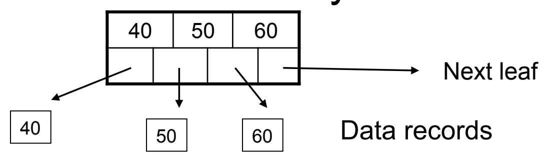
B+ Trees Basics

- Parameter d = the *degree*
- Each internal node has d≤m≤2d keys (except root)



Each node also has **m+1 pointers**

- Each leaf has d≤m≤2d keys:

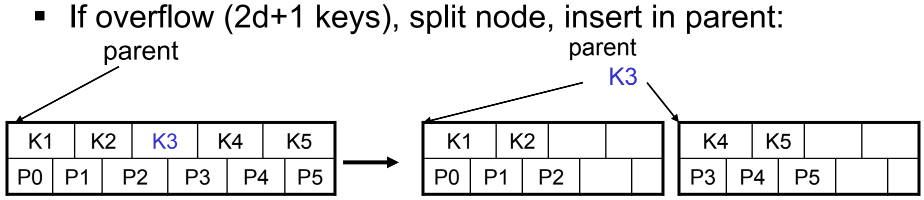


Data records

Insertion in a B+ Tree

Insert (K, P)

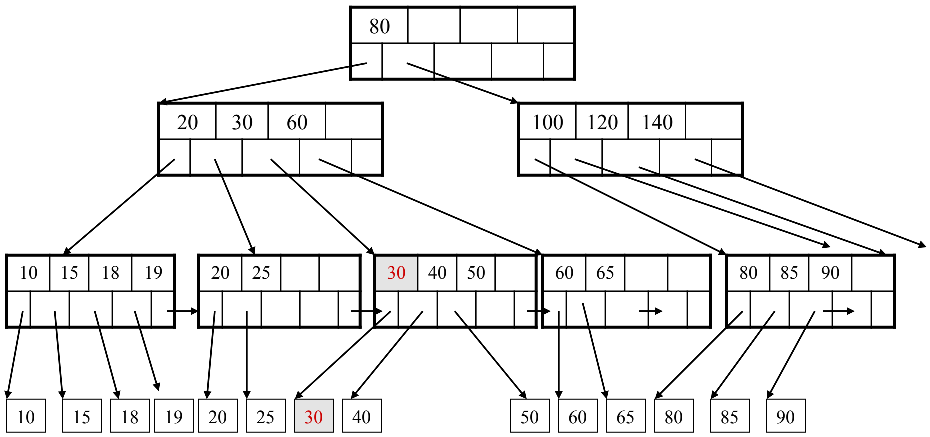
- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:



- If leaf, keep K3 too in right node
- When root splits, new root has 1 key only

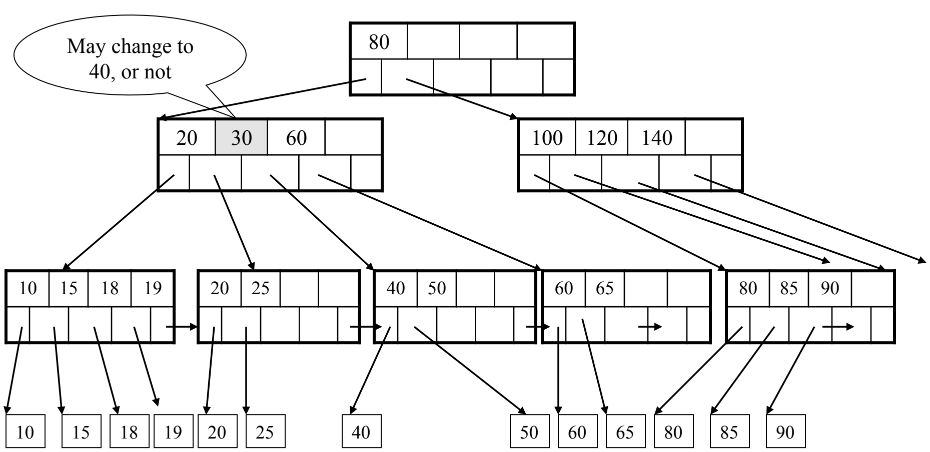
Deletion from a B+ Tree

Delete 30



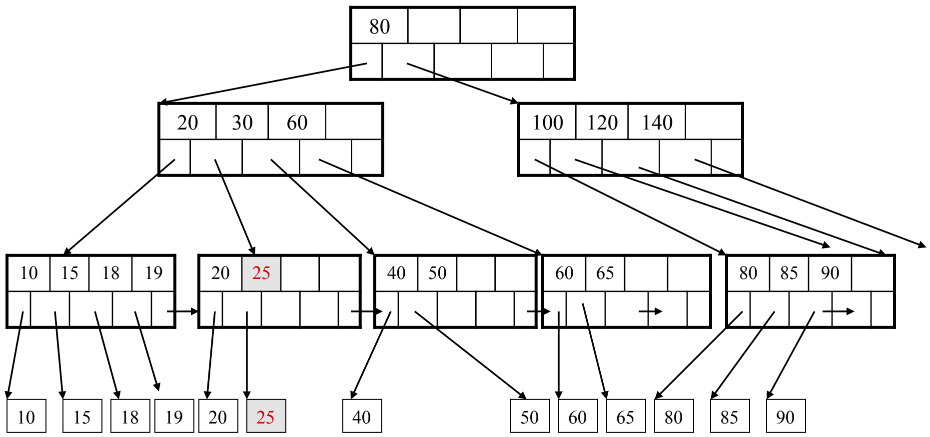
Deletion from a B+ Tree

After deleting 30



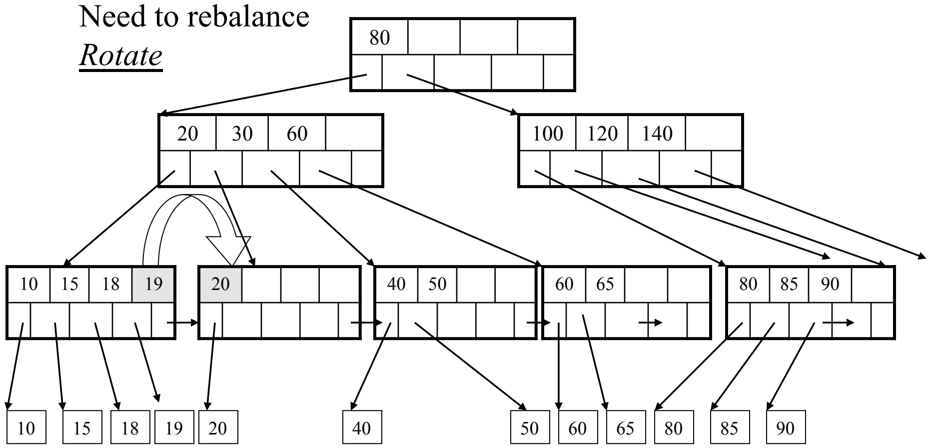
Deletion from a B+ Tree

Now delete 25



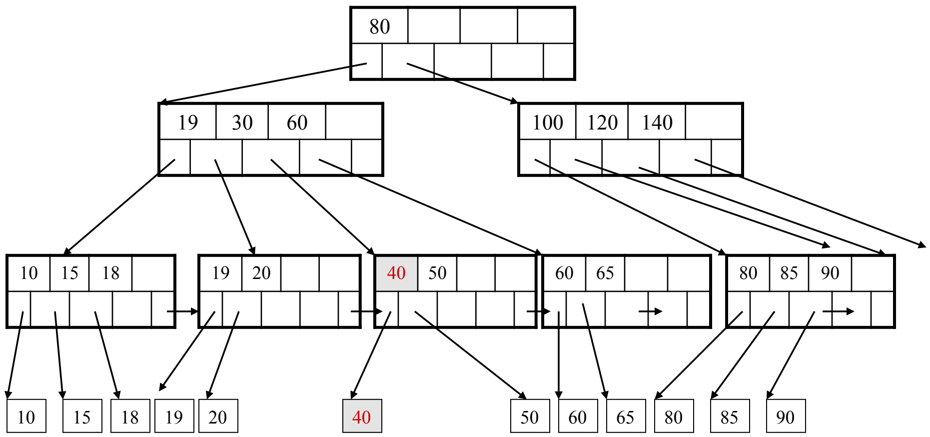
Deletion from a B+ Tree

After deleting 25
Need to rebalance
Rotate



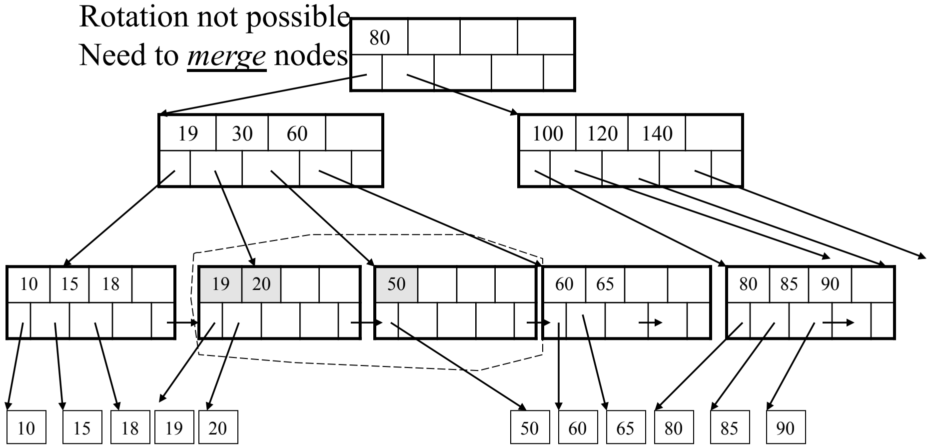
Deletion from a B+ Tree

Now delete 40



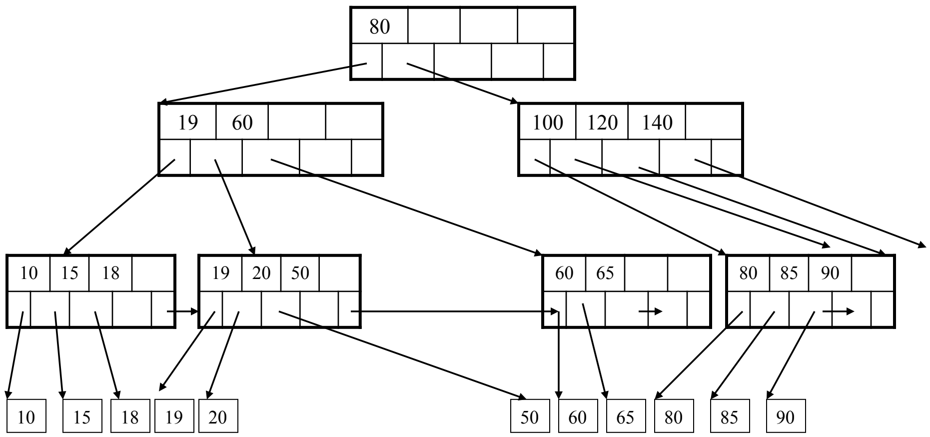
Deletion from a B+ Tree

After deleting 40
Rotation not possible
Need to *merge* nodes

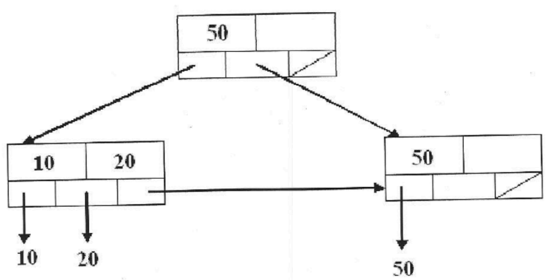


Deletion from a B+ Tree

Final tree



Home Exercise



Insert record 60
Insert record 15
Delete record 50
Delete record 20
Delete record 10

Summary on B+ Trees

- Default index structure on most DBMS
- Very effective at answering ‘point’ queries:
  productName = ‘gizmo’
- Effective for range queries:
  50 < price AND price < 100
- Less effective for multirange:
  50 < price < 100 AND 2 < quant < 20

Indexes in MySQL

CREATE [UNIQUE] INDEX *index_name*
[USING *index_type*]
ON *tbl_name* (*col_name,…*)

index_type BTREE | HASH

Example:

CREATE INDEX *Ind_ItemPrice*
USING BTREE
ON *Items (Price)*

The Index Selection Problem

- Given a database schema (tables, attributes)
- Given a "query workload":
  - Workload = a set of (query, frequency) pairs
  - The queries may be both SELECT and updates
  - Frequency = either a count, or a percentage
- Select a set of indexes that optimizes the workload

In general this is a very hard problem

Index Selection Decisions

- To index or not to index?
- Which key?
- Multiple keys?
- Clustered or unclustered?
- Hash or trees?

Index Selection Guidelines

- Columns in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
- Try to choose indexes that benefit as many queries as possible.
- At most one clustered index per table!
- Think of trade-offs before creating an index!

Updates

- Indexes speed up queries
  - SELECT FROM WHERE
- But they usually slow down updates:
  - INSERT, DELETE, UPDATE
  - However some updates benefit from indexes

```
UPDATE R
SET A = 7
WHERE K=55
```

Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

What indexes?

Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

A: V(N) and V(P) (hash tables or B-trees)

Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

What indexes?

Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```


A: definitely V(N) (must B-tree); unsure about V(P)

Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

What indexes?

Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```


A: V(N, P)

Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

What indexes?

Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

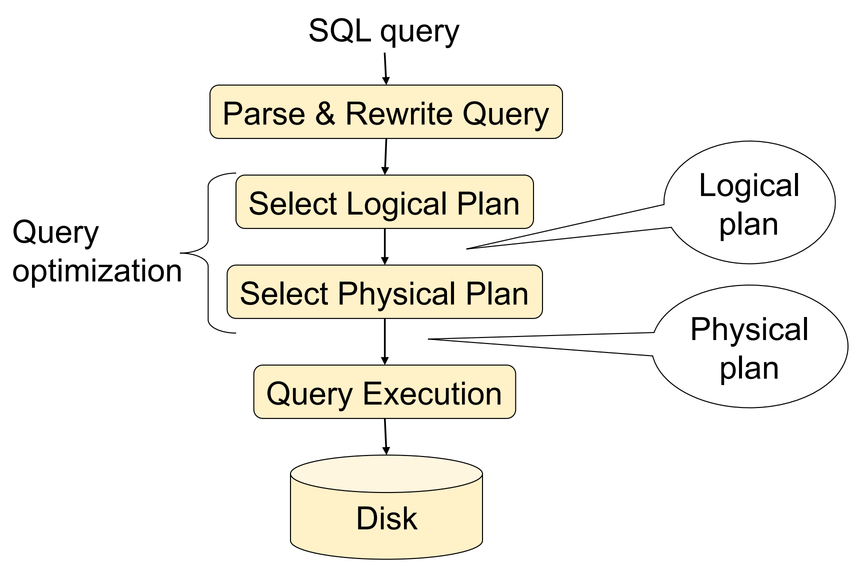
```
SELECT *
FROM V
WHERE P>? and P<?
```


A: V(N) secondary (unclustered), V(P) primary index (clustered)

The Index Selection Problem

- SQL Server:
  - Automatically, through the *AutoAdmin* project
  - Much acclaimed successful research project from mid 90's, similar ideas adopted by the other major vendors

Steps of the Query Processor



Steps in Query Evaluation

- **Step 0: Admission control**
  - User connects to the db with username, password
  - User sends query in text format
- **Step 1: Query parsing**
  - Parses query into an internal format
  - Performs various checks using catalog
    - Correctness, authorization, integrity constraints
- **Step 2: Query rewrite**
  - View rewriting, flattening, etc.

Example Database Schema

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```

View: Suppliers in Manhattan, KS

```
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Manhattan' AND sstate='KS'
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Example Query

Find the names of all suppliers in Manhattan who supply part number 2

```
SELECT sname FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Rewritten Version of Our Query

Original query:

```
SELECT sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

View:

```
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Manhattan' AND
sstate='KS'
```

Rewritten query:

```
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Manhattan' AND S.sstate='KS'
AND S.sno = U.sno
AND U.pno = 2;
```

Continue with Query Evaluation

- **Step 3: Query optimization**
  - Finds an efficient query plan for executing the query
- A **query plan** is
  - **Logical query plan**: an extended relational algebra tree
  - **Physical query plan**: with additional annotations at each node
    - Access method to use for each relation
    - Implementation to use for each relational operator

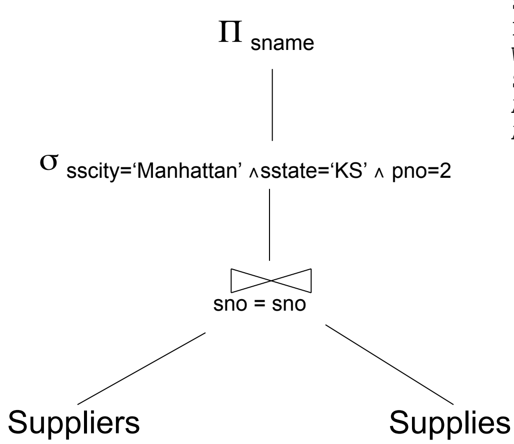
Logical versus Physical Plans

- Logical plan -> logical operators
  - *what* they do
  - e.g., union, selection, project, join, grouping
- Physical plan -> physical operators
  - *how* they do it
  - e.g., nested loop join, sort-merge join, hash join, index join

Extended Algebra Operators

- Union ∪, intersection ∩, difference -
- Selection σ
- Projection π
- Join ⋈
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ
- Rename ρ

Logical Query Plan



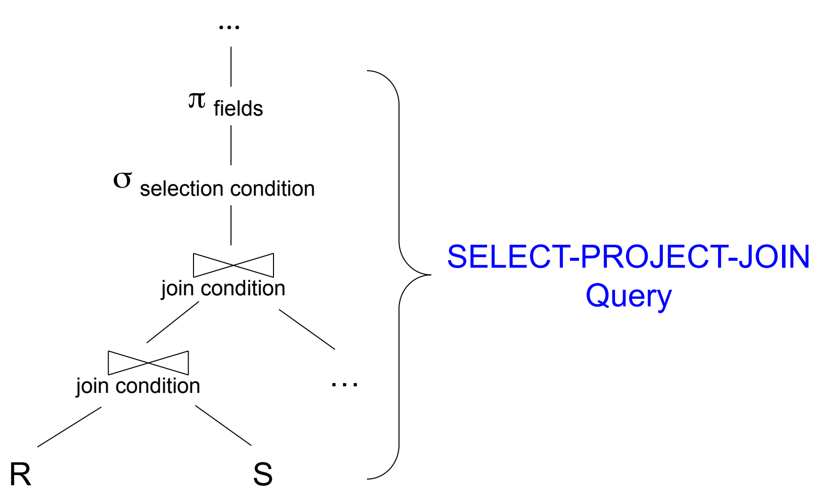
Rewritten query:

```
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Manhattan' AND
S.sstate='KS'
AND S.sno = U.sno
AND U.pno = 2;
```

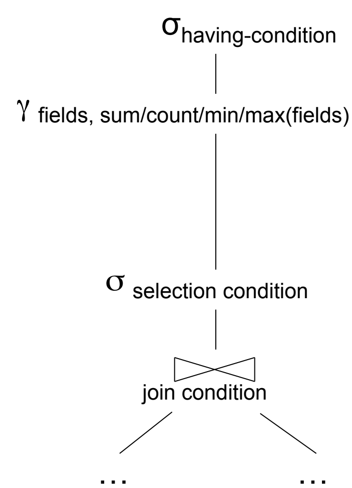
Query Block

- Most optimizers operate on individual query blocks
- A query block is an SQL query with **no nesting**
  - **Exactly one**
    - SELECT clause
    - FROM clause
  - **At most one**
    - WHERE clause
    - GROUP BY clause
    - HAVING clause

Typical Plan for Block (1/2)



Typical Plan For Block (2/2)



Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'KS'
   and not exists
     SELECT *
     FROM Supplies P
     WHERE P.sno = Q.sno
           and P.price > 100
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'KS'
   and not exists
     SELECT *
     FROM Supplies P
     WHERE P.sno = Q.sno
           and P.price > 100
```

Correlation!

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'KS'
   and not exists
     SELECT *
     FROM Supplies P
     WHERE P.sno = Q.sno
           and P.price > 100
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'KS'
   and Q.sno not in
     SELECT P.sno
     FROM Supplies P
     WHERE P.price > 100
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

How about Subqueries?

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'KS')
   EXCEPT
 (SELECT P.sno
   FROM Supplies P
   WHERE P.price > 100)
```

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'KS'
   and Q.sno not in
      SELECT P.sno
      FROM Supplies P
      WHERE P.price > 100
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

How about Subqueries?

Finally…

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'KS')
   EXCEPT
 (SELECT P.sno
   FROM Supplies P
   WHERE P.price > 100)
```