Physical Design and Tuning Example

1

The Database

- A *hockey league* with rinks, teams, and players:
  - Rink (name, phone, capacity)
  - Team (tname, city, color, wins, losses, tie, rname FK references Rink(name))
  - Player (id, name, num, pos, tname, tcity, FK (tname, tcity) references Team (tname, city))
- All relations are in BCNF
- The only FDs are PK→all other fields
- Constraint: All players with uniform number 9 must be goalies.

2

The Workload

- A *mix* of queries and updates:
  - Q1 (20%): given player id, return num and pos
  - Q2 (40%): given player position, return names, nums
  - Q3 (5%): retrieve player name, num, pos, and rink
  - Q4 (10%): get W-L-T record of a team (given name, city)
  - Q5 (10%): produce an alphabetical team listing
  - Q6 (5%): list all rink names from smallest to largest
  - U1 (10%): update W-L-T information

3

Index Choices

- Q1: non-clustered index on player id
- Q2: *clustered* index on player pos
- Q3: index won't help without harming Q2
- Q4, Q5: *clustered* index on team (tname, city)
- Q6: *clustered* index on rink capacity
- U1: team (tname, city) index will help, as will lack of indices on wins, losses, tie attributes

4

Subsequent Tuning

- The system runs fine for a week, so you take a vacation. When you return….
  - General performance complaints abound
    - you *rebuild indices*
    - you *create and update statistics*
    - you *check optimizer plans*
  - Q3 is still particularly bad, and the league president wants it to be fast.
    - you wisely decide to give him what he wants
    - you *denormalize* to achieve a *precomputed join*

5

Denormalization

- Add rname field to player to avoid join:
  - newplayer (id, name, num, pos, tname, tcity, rname FK references rink(rname), FK (tname, tcity) references team (tname, city))
  - newplayer is 2NF (YIKES!)
  - DB is still *lossless, dependency-preserving*
- **Must manage redundancy!**
  - Updates to newplayer (tname, tcity, rname) must *check* for correct value of rname
  - Updates to team (rname) must *propagate* to newplayer
- Create a view to **preserve external schema**

6

Vertical Decomposition

- Suppose we want to speed up Q6. We can make it read fewer pages by *decomposing*:
  - Rink_phone (name, phone), clust. index on name
  - Rink_cap (name, capacity), clust. index on capacity
- Create a view to preserve external schema
- IC (FD) maintenance choices:
  - leave it to user (scary!)
  - allow inserts to "rink" view, not to base relations
  - application pgm. to force user to enter both (atomic)
  - insert into one base relation triggers insert into other

7

Unpreserved Dependencies

- Suppose the users now decide that all rinks in the same city have the same capacity:
  - city $\rightarrow$ capacity
  - While trying to remain calm, you realize that:
  - This FD doesn't exist in any single relation, so a *join is required to check it* each time we add or change a capacity value.
- The tradeoff:
  - Expensive to check, but
  - may not be checked often enough to justify creating a dependency-preserving decomposition.

8

Other SQL Server Tuning

- Has a general performance *profiling* tool
  - Generates execution traces
- Queries can give optimizer *hints*:
  - Use loop, hash, or merge *join*
  - Use hash or sort to do *grouping*
  - Force use of an *index*
  - Force a *join ordering*
  - Optimize for *time-to-nth-result-tuple*
  - Adjust lock granularities and concurrency (next week…)

9

Summary

- Tunability varies among systems
- B trees nearly universal
- Denormalization, decomposition possible
- Storage structure size and growth tunable
- Optimizer hints common
- "Check" constraints very useful
- Triggers, assertions for IC (FD) enforcement

10