

CSE 135

when to use indices

Sample relational database

Classes					
id	name	number	date_code	start_time	end_time
1	Web stuff	CSE135	TuTh	2:00	3:20
2	Databases	CSE132A	TuTh	3:30	4:50
4	VLSI	CSE121	F	null	null

Enrollment			
id	class	student	credits
1	1	1	4
2	1	2	3
3	4	3	4
4	1	3	3

Students			
id	pid	first_name	last_name
1	8888888	John	Smith
2	1111111	Mary	Doe
3	2222222	null	Chen

Declaration of schemas in SQL's Data Definition Language – without indices

```

CREATE TABLE classes (
  ID          SERIAL PRIMARY KEY,
  name        TEXT,
  number       TEXT,
  date_code   TEXT,
  start_time  TIME,
  end_time    TIME
)
CREATE TABLE students (
  ID          SERIAL PRIMARY KEY,
  pid         TEXT,
  first_name  TEXT,
  last_name   TEXT
)
CREATE TABLE enrollment (
  ID          SERIAL,
  class       INTEGER REFERENCES classes (ID) NOT NULL,
  student     INTEGER REFERENCES students (ID) NOT NULL,
  credits     INTEGER
)

```

Indices for queries with selections

- Index when a query involves a condition
<attribute> = <value>
- also when
<attribute> = ?

No index
 Find first names and last names of all students
SELECT first_name, last_name FROM students;

Find all students whose first name is John; project all attributes
SELECT * FROM students WHERE first_name = 'John';
Index on first_name

After you have created table **students**, issue command
CREATE INDEX students_first_name ON students(first_name)

You can remove an index with "DROP INDEX" command, e.g.,
DROP INDEX students_first_name

When is an index beneficial to query performance?

- Imagine that you were not using an SQL database
- Instead, you had stored the tuples in file-based data structures
 - File-based because they are too big for main memory and because they must stay around even if server is restarted
 - Assume the students table has hundreds of thousands tuples
- How would you find quickly the tuples of table **t** where an attribute **a** has a given value **v**? Eg, **students.first_name = v**
 - Scanning all student tuples to find the ones where **first_name = v** is unacceptably slow.
- You would build an index data structure and an algorithm such that your algorithm would quickly return the tuples **t** where **t.a = v** (in example, **students.first_name = v**)
 - E.g., a hash table or a binary tree data structure such that given the student first name **v**, your algorithm quickly returns the students with first name **v**
- This data structure would be an "index on **a**" (index on first name)

When is an index beneficial to query performance?

- Now, stop imagining that you do not use a database. When you use a database you need not build and maintain an index manually.
 - A database builds it and maintains it automatically, after you issue the command
- CREATE INDEX <index name> ON <table>(<attribute>)**
- You should issue the CREATE INDEX after the CREATE TABLE.
 - You may DROP later if you find it useless
 - You do not need to change your SQL queries. The database will automatically find whether your index is useful for answering a query quickly. If it is useful, the database will use it.

Do I have anything to lose by creating indices on all attributes? Yes, you do

- If you have built an index on attribute **a** of table **t** the database will have to update the index every time an insert, delete or update happens on **t**. This slows down inserts, deletes, updates

7

Primary keys and indices

- You do not need to create an index on a primary key. The database has automatically created it already.

8

Which indices are useful on queries that join many tables (example)

```
SELECT students.pid, students.first_name,
       students.last_name, enrollment.credits
FROM   students, enrollment
WHERE  students.id = enrollment.student
       AND enrollment.class = 1 ;
```

Enrollment.class,
Students.id (default)

9

The logic behind choosing indices to speed up selection-join queries

- Again, imagine that you did not have a database. You just have the tables stored in files.
- Then, what index data structures and algorithms would you manually make to run these queries fast?
- For the query of the previous example you would need indices on enrollment.class and on students.id
 - the indices point in positions in the files
- Given these indices, you can write an algorithm that answers the query very fast in the following steps:
 - Uses the enrollment.class index to quickly retrieved the enrollment tuples for the given class
 - For each student id in the retrieved enrollment tuples it finds the corresponding student record using the students id index

10

(example)

Produce a table that shows the pid, first name and last name of every student enrolled in the CSE135 class along with the number of credit units in his/her 135 enrollment

Create indices on
Classes.number,
Enrollment.class
Students.id (default)

```
SELECT students.pid, students.first_name,
       students.last_name, enrollment.credits
FROM   students, enrollment, classes
WHERE  classes.number = 'CSE135'
       AND students.id = enrollment.student
       AND enrollment.class = classes.id ;
```

11

(example)

produce a table where each row has the name of a 135 student and the name of another class he/she takes

```
SELECT c_others.name, first_name, last_name
FROM   classes AS c_135, enrollment AS e_135,
       students,
       enrollment AS e_others, classes AS c_others
WHERE  c_135.number = 'CSE135'
       AND c_135.id = e_135.class
       AND e_135.student = students.id
       AND students.id = e_others.student
       AND e_others.class = c_others.id
       AND NOT (c_others.number = 'CSE135')
```

Classes.number,
Enrollment.class,
Students.id,
Enrollment.student,
Classes.id

12

Should you use an index here? No

- Find the average salary in each department that has more than 1 employee:

```
SELECT Dept,AVG(Salary) AS AvgSal
FROM Employee
GROUP BY Dept
HAVING COUNT (Name) >1
```

13

Should you use an index here? No

- Problem:** List all enrolled students and the number of total credits for which they have registered

```
SELECT students.id, first_name, last_name, SUM(credits)
FROM students, enrollment
WHERE students.id = enrollment.student
GROUP BY students.id, first_name, last_name
```

14

Should you use an index here? Yes

- Problem:** List all the classes (id's only) in which students of the class "?" are enrolled and also show the number of students (of the class "?") in each one of them. (The "?" is a parameter that will be changed into a class id when a query is executed.)

```
SELECT e_others.class, COUNT(e.student)
FROM enrollment e, enrollment e_others
WHERE e.class = ?
AND e.student = e_others.student
GROUP BY e_others.class
```

```
enrollment.class,
enrollment.student
```

15