when to use indices

Sample relational database

<table>
<thead>
<tr>
<th>id</th>
<th>pid</th>
<th>first_name</th>
<th>last_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8888888</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>2</td>
<td>1111111</td>
<td>Mary</td>
<td>Doe</td>
</tr>
<tr>
<td>3</td>
<td>2222222</td>
<td>Chen</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>class</th>
<th>student</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Indices for queries with selections

Index on first_name

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';

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)
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.

Primary keys and indices

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

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)

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:
  1. Uses the enrollment.class index to quickly retrieved the enrollment tuples for the given class
  2. For each student id in the retrieved enrollment tuples it finds the corresponding student record using the students id index

(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

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

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

(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

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

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

Create indices on Classes.number, Enrollment.class, Students.id (default)
**Should you use an index here? No**

- Find the average salary in each department that has more than 1 employee:
  ```sql
  SELECT Dept, AVG(Salary) AS AvgSal
  FROM Employee
  GROUP BY Dept
  HAVING COUNT(Name) > 1
  ```

**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.)

  ```sql
  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
  ```