

**Recap: Which indices you should consider for selection queries?**

- Index when a selection query involves a condition
  - index on first name assuming there are plenty of first names, i.e., few tuples per first name
- <attribute> = ?**
  - index on ID, already created

index on student, assuming each student has a relatively small fraction of the total enrollments

**No index is useful**

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';**

Find the student whose ID is ?  
**SELECT \***  
**FROM students**  
**WHERE ID = ?;**

Find the enrollments of the student whose ID is ?  
**SELECT \***  
**FROM enrollment**  
**WHERE student = ?;**

---

---

---

---

---

---

---

---

**Queries with Selections and Joins: Which indices you should try?**

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

Index on enrollment.class assuming most queried classes have relatively few enrollments compared to total enrollments, and index on students.id (default) assuming the students taking the queried class are a small fraction of the total students

---

---

---

---

---

---

---

---

**One selection and two joins: which indices to consider?**

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 (assuming many classes), enrollment.class, students.id (default), with the latter two posing similar considerations to prior example

```
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 ;
```

---

---

---

---

---

---

---

---

**Many selections and many joins: which indices are useful**

Index on classes.number, enrollment.class, students.id (similar considerations to prior example), perhaps enrollment.student (if the enrollments of CSE135 students are not a larger fraction of total enrollments), perhaps classes.id (if the enrollments of CSE135 students are a small fraction of total enrollments), perhaps classes.id (if the classes of CSE135 students are a small fraction of total classes)

```
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')
```

---

---

---

---

---

---

---

---

---

---

**Should you use an index in a plain aggregation query? Likely Not**

- 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
```

The grouping on Dept can use an index on Dept to order the tuples and group them faster. However, this use case is unlikely to produce massive performance difference

---

---

---

---

---

---

---

---

---

---

**Should you use an index here? Most likely 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
```

- **Caveat:** In the unlikely case where the vast majority of this university's students are not enrolled in any class (!) the index on students.id becomes useful

---

---

---

---

---

---

---

---

---

---

**Should you consider 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

---

---

---

---

---

---

---

---

**Exercise:**  
**Which indices are needed here?**

**Sample TPC-H Schema**

```
Nation(NationKey, NName)
Customer(CustKey, CName, NationKey)
Order(OrderKey, CustKey, Status)
Lineitem(OrderKey, PartKey, Quantity)
Product(SuppKey, PartKey, PName)
Supplier(SuppKey, SName)
```

SELECT SName  
FROM Nation, Customer, Order, Lineitem, Product, Supplier  
WHERE Nation.NationKey = Customer.NationKey  
AND Customer.CustKey = Order.CustKey  
AND Order.OrderKey=Lineitem.OrderKey  
AND Lineitem.PartKey= Product.PartKey  
AND Product.SuppKey = Supplier.SuppKey  
AND NName = "Canada"

Find the names of suppliers that sell a product that appears in a line item of an order made by a customer who is in Canada

---

---

---

---

---

---

---

---

**3 ways to avoid the cold Vs warm problem**

- Reset computer after the first run
  - Bulletproof but takes time to reset for each experiment
- Flush the relevant pages out of the cache by running a query on an irrelevant very large table. Example:
  - Create "irrelevant" table H with integer attribute S. The table H must be larger than the RAM of your system. Then run the query QL = SELECT SUM(S) FROM H
  - The query will fetch all pages of H from disk to RAM and will most likely remove from RAM any pages that were there before
  - Not fully bulletproof: Database buffer managers sometimes do not use "Least Recently Used" strategy. Check if you get the same performance for the same experiment before/after QL
- Run experiments in different databases that have identical data
  - Make many copies of the database. Reset.
  - Each experiment should use another database

---

---

---

---

---

---

---

---