


# CS50's Introduction to Databases with SQL

OpenCourseWare

Donate  (<https://cs50.harvard.edu/donate>)



Carter Zenke (<https://carterzenke.me>)

[carter@cs50.harvard.edu](mailto:carter@cs50.harvard.edu)

 (<https://github.com/carterzenke>)  (<https://www.linkedin.com/in/carterzenke/>)

David J. Malan (<https://cs.harvard.edu/malan/>)

[malan@harvard.edu](mailto:malan@harvard.edu)

 (<https://www.facebook.com/dmalan>)  (<https://github.com/dmalan>)  (<https://www.instagram.com/davidjmalan/>)  (<https://www.linkedin.com/in/malan/>)  (<https://www.reddit.com/user/davidjmalan>)  (<https://www.threads.net/@davidjmalan>)  (<https://twitter.com/davidjmalan>)

## your.harvard



*The my.harvard logo*

## Problem to Solve

If you're not already familiar, Harvard has a course shopping tool called [my.harvard \(https://my.harvard.edu/\)](https://my.harvard.edu/), with which students explore (and ultimately register for!) classes. To keep track of courses, students, and their registrations, my.harvard presumably uses some kind of underlying database. And yet, if you've ever used it, you'll know that my.harvard isn't especially... quick.

Here's your chance to make my.harvard just a little bit faster! In this problem, take some Harvard course data and create indexes to speed up typical queries on the database. Keep in mind that indexing every column isn't always the best solution: you'll need to consider trade-offs in terms of space and time, ultimately representing Harvard's courses and students in the most efficient way possible.

## Demo

```
$ sqlite3 harvard.db
sqlite> SELECT "title", "
```

Recorded with [asciinema](#)

## Distribution Code

For this problem, you'll need to download `harvard.db` and an `indexes.sql` file in which you'll write your SQL statements to create your indexes.

### ▼ Download the distribution code

Log into [cs50.dev](https://cs50.dev) (<https://cs50.dev/>), click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
wget https://cdn.cs50.net/sql/2023/x/psets/5/harvard.zip
```

in order to download a ZIP called `harvard.zip` into your codespace.

Then execute

```
unzip harvard.zip
```

to create a folder called `harvard`. You no longer need the ZIP file, so you can execute

```
rm harvard.zip
```

and respond with “y” followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd harvard
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

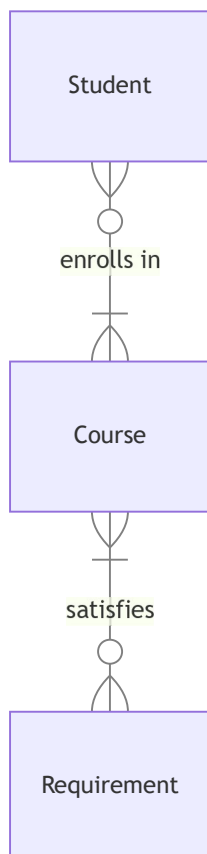
```
harvard/ $
```

If all was successful, you should execute

```
ls
```

and see a database named `harvard.db` alongside a SQL file named `indexes.sql`. If not, retrace your steps and see if you can determine where you went wrong!

## Schema



Within `harvard.db`, you'll find five tables that implement the relationships described in the ER diagram above. Click the drop-downs below to learn more about the schema of each individual table.

#### ▼ **students** table

The `students` table contains the following columns:

- `id`, which is the student's ID.
- `name`, which is the student's name.

#### ▼ **courses** table

The `courses` table contains the following columns:

- `id`, which is the courses's ID.
- `department`, which is the department in which the course is taught (e.g., "Computer Science", "Economics", "Philosophy").
- `number`, which is the course number (e.g., 50, 12, 330).
- `semester`, which is the semester in which the class was taught (e.g., "Spring 2024", "Fall 2023").
- `title`, which is the title of the course (e.g., "Introduction to Computer Science").

#### ▼ **enrollments** table

The `enrollments` table contains the following columns:

- `id`, which is the ID to identify the enrollment.
- `student_id`, which is the ID of the student enrolled.
- `course_id`, which is the ID of the course in which the student is enrolled.

#### ▼ **requirements** table

The `requirements` table contains the following columns:

- `id`, which is the ID of the requirement.
- `name`, which is the name of the requirement.

#### ▼ **satisfies** table

The `satisfies` table contains the following columns:

- `id`, which is the ID of the course-requirement pair.
- `course_id`, which is the ID of a given course.
- `requirement_id`, which is the ID of the requirement which the given course satisfies.

# Specification

In `indexes.sql`, write a set of SQL statements that create indexes which will speed up typical queries on the `harvard.db` database. The number of indexes you create, as well as the columns they include, is entirely up to you. Be sure to balance speed with disk space, only creating indexes you need.

When engineers optimize a database, they often care about the typical queries run on the database. Such queries highlight patterns with which a database is accessed, thus revealing the best columns and tables on which to create indexes. Click the spoiler tags below to see the set of typical queries run on `harvard.db`.

## ▼ Typical **SELECT** queries on **harvard.db**

- Find a student's historical course enrollments, based on their ID:

```
SELECT "courses"."title", "courses"."semester"
FROM "enrollments"
JOIN "courses" ON "enrollments"."course_id" = "courses"."id"
JOIN "students" ON "enrollments"."student_id" = "students"."id"
WHERE "students"."id" = 3;
```

- Find all students who enrolled in Computer Science 50 in Fall 2023:

```
SELECT "id", "name"
FROM "students"
WHERE "id" IN (
    SELECT "student_id"
    FROM "enrollments"
    WHERE "course_id" = (
        SELECT "id"
        FROM "courses"
        WHERE "courses"."department" = 'Computer Science'
        AND "courses"."number" = 50
        AND "courses"."semester" = 'Fall 2023'
    )
);
```

- Sort courses by most- to least-enrolled in Fall 2023:

```
SELECT "courses"."id", "courses"."department", "courses"."number", "courses"
FROM "courses"
JOIN "enrollments" ON "enrollments"."course_id" = "courses"."id"
WHERE "courses"."semester" = 'Fall 2023'
GROUP BY "courses"."id"
ORDER BY "enrollment" DESC;
```

- Find all computer science courses taught in Spring 2024:

```
SELECT "courses"."id", "courses"."department", "courses"."number", "courses"
FROM "courses"
WHERE "courses"."department" = 'Computer Science'
```

```
AND "courses"."semester" = 'Spring 2024';
```

- Find the requirement satisfied by “Advanced Databases” in Fall 2023:

```
SELECT "requirements"."name"
FROM "requirements"
WHERE "requirements"."id" = (
    SELECT "requirement_id"
    FROM "satisfies"
    WHERE "course_id" = (
        SELECT "id"
        FROM "courses"
        WHERE "title" = 'Advanced Databases'
        AND "semester" = 'Fall 2023'
    )
);
```

- Find how many courses in each requirement a student has satisfied:

```
SELECT "requirements"."name", COUNT(*) AS "courses"
FROM "requirements"
JOIN "satisfies" ON "requirements"."id" = "satisfies"."requirement_id"
WHERE "satisfies"."course_id" IN (
    SELECT "course_id"
    FROM "enrollments"
    WHERE "enrollments"."student_id" = 8
)
GROUP BY "requirements"."name";
```

- Search for a course by title and semester:

```
SELECT "department", "number", "title"
FROM "courses"
WHERE "title" LIKE "History%"
AND "semester" = 'Fall 2023';
```

### ▼ Typical INSERT, UPDATE, and DELETE queries on harvard.db

- Add a course:

```
INSERT INTO "courses" ("department", "number", "semester", "title")
VALUES ('Computer Science', '151', 'Fall 2023', 'Introduction to Databases v
```

- Register a student for a course:

```
INSERT INTO "enrollments" ("student_id", "course_id")
VALUES ((
    SELECT "id"
    FROM "students"
    WHERE "name" = 'Carter'
), (
    SELECT "id"
    FROM "courses"
    WHERE "title" = 'Introduction to Databases with SQL'
));
```

- Update the title of a course:

```
UPDATE "courses"  
SET "title" = 'Introduction to Shakespeare'  
WHERE "title" = 'Shakespeare'  
AND "number" = 20  
AND "department" = 'English';
```

- Remove a course from the catalog:

```
DELETE FROM "courses"  
WHERE "department" = 'Philosophy'  
AND "number" = 178  
AND "semester" = 'Spring 2024'  
AND "title" = 'Philosophy of Mind'
```

## Advice

- Type `.timer on` or `.timer off` to turn SQLite's built-in query timer on or off, respectively.
- Recall that creating an index on a column can ensure that `SELECT` queries which search that column run faster. At the same time, an index requires additional space.
- Recall that an index organizes data in such a way as to facilitate the search process. Yet, when new rows are added, updated, or deleted, the index needs to be re-arranged –resulting in some `INSERT`, `UPDATE`, and `DELETE` queries actually taking longer when an index exists!
- Recall that in SQLite you can execute

```
EXPLAIN QUERY PLAN  
...
```

where `...` is a full SQL query. SQLite will reveal the process it intends to use to execute the given query. You'll know that your query is using an index when `EXPLAIN QUERY PLAN` lists steps that includes "USING INDEX" or "USING COVERING INDEX." Keep in mind that a query can involve multiple steps and, depending on which indexes you've created, some of those steps may use an index while others may not!

## Usage

To test your indexes as you write them in `indexes.sql`, you can use

```
.read indexes.sql
```

Keep in mind you can also use

```
DROP INDEX name;
```

where `name` is the name of your index, to remove an index before creating it anew.

You may want to use `VACUUM` to free up disk space after you delete an index!

## How to Test

---

While `check50` is available for this problem, you're encouraged to also test your code on your own.

## Correctness

```
check50 cs50/problems/2023/sql/harvard
```

## How to Submit

---

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2023/sql/harvard
```