Data Definition Language (DDL) commands

```
CREATE TABLE Patient(
        patientId INT Primary Key,
        firstName VARCHAR(32),
        lastName VARCHAR(32),
        age INT,
        sex VARCHAR(16),
        birthDate DATE,
        address VARCHAR(64),
        phone VARCHAR(16),
        notes VARCHAR(128),
        chargesDue REAL,
        insProvider VARCHAR(32),
        insHolder VARCHAR(32),
        insNumber INT,
        email VARCHAR(32),
        password VARCHAR(32));

CREATE TABLE Procedures(
        procedureId INT Primary Key,
        name VARCHAR(64),
        description VARCHAR(128),
        cost REAL);

CREATE TABLE NeedProcedure(
        patientId INT,
        procedureId INT,
        FOREIGN KEY(patientId) REFERENCES Patient(patientId),
        FOREIGN KEY(procedureId) REFERENCES Procedures(procedureId));

CREATE TABLE Practitioner(
        practitionerId INT Primary key,
        name VARCHAR(32),
        title VARCHAR(32));

CREATE TABLE Appointment(
        appointmentId INT Primary key,
        date DATE,
        time TIME,
        reason VARCHAR(128),
        practitionerId INT,
        procedureId INT,
```

```
        patientId INT,
        notes VARCHAR(128),
        status VARCHAR(16),
        FOREIGN KEY(practitionerId) REFERENCES Practitioner(practitionerId),
        FOREIGN KEY(procedureId) REFERENCES Procedures(procedureId),
        FOREIGN KEY(patientId) REFERENCES Patient(patientId));

CREATE TABLE Conditions(
        conditionId INT Primary Key,
        name VARCHAR(32),
        description VARCHAR(128),
        infoLink VARCHAR(128));

CREATE TABLE HasCondition(
        conditionId INT,
        patientId INT,
        FOREIGN KEY(conditionId) REFERENCES Conditions(conditionId),
        FOREIGN KEY(patientId) REFERENCES Patient(patientId));

CREATE TABLE Vitals(
        patientId INT,
        date DATE,
        height INT,
        weight INT,
        systolicBP INT,
        diastolicBP INT,
        pulseRate INT,
        respRate INT,
        FOREIGN KEY(patientId) REFERENCES Patient(patientId));

CREATE TABLE Medications(
        medicationId INT Primary Key,
        name VARCHAR(32),
        description VARCHAR(75));

CREATE TABLE Prescriptions(
        medicationId INT,
        patientId INT,
        Price REAL,
        doseSize VARCHAR(16),
        dosePeriod VARCHAR (16),
        doseCount INT,
        ExpirationDate DATE,
        FOREIGN KEY(medicationId) REFERENCES Medications(medicationId),
```

```
    FOREIGN KEY(patientId) REFERENCES Patient(patientId));

CREATE TABLE Cares(
    practitionerId INT,
    patientId INT,
    FOREIGN KEY(practitionerId) REFERENCES Practitioner(practitionerId),
    FOREIGN KEY(patientId) REFERENCES Patient(patientId));

CREATE TABLE Insurance(
    insProvider VARCHAR(32) Primary Key,
    practitionerId INT,
    FOREIGN KEY(practitionerId) REFERENCES Practitioner(practitionerId));
```

Advanced Queries

First Advanced Query

For a procedure with procedureId X that needs to be done for a patient with patientId Y find doctors who can perform the procedure for the patient that the procedure is assigned to. The doctor must accept the patient's insurance and must have the right title to perform the procedure. We will put a keyword for the type of doctor involved in the procedure's description.

This query is useful for when patients need to find a doctor to do a specific procedure, especially if the procedure involves doctors in a field unrelated to the reasons of a patient's usual visits.

In this case, X = 'Physical Examination' and Y = 1

```
SELECT name as doctorName, title
FROM Practitioner
NATURAL JOIN Insurance
WHERE insProvider = (SELECT insProvider FROM Procedures NATURAL JOIN
NeedProcedure NATURAL JOIN Patient
WHERE name = 'Physical Examination' AND patientId = 1) AND LOCATE(title,(SELECT
description FROM Procedures WHERE name = 'Physical Examination')) > 0
ORDER BY name
LIMIT 15;
```

```
mysql> SELECT name as doctorName, title
    -> FROM Practitioner
    -> NATURAL JOIN Insurance
    -> WHERE insProvider = (SELECT insProvider FROM Procedures NATURAL JOIN NeedProcedure NATURAL JOIN Patient WHERE name = 'Physical Examination' AND patientId = 1) AND LOCATE(title,(SELECT description FROM Proce
dures WHERE name = 'Physical Examination')) > 0 ORDER BY name LIMIT 15;
+-------------------------+-----------+
| doctorName              | title     |
+-------------------------+-----------+
| Alexis Golden           | LPN       |
| Brett Carr              | LPN       |
| Cristina Richardson     | CNA       |
| Danielle Roberts        | RN        |
| Dr. Mary Baker MD       | physician |
| Dr. Zachary Browning MD | physician |
| Erin Costa              | LPN       |
| Jared Morrison          | RN        |
| Jeffery Coleman         | LPN       |
| Joan Lindsey            | CNA       |
| Joshua Harris           | CNA       |
| Mr. David Lawson        | CNA       |
| Mrs. Erica Gardner      | CNA       |
| Nicole Perez            | RN        |
| Richard Sanchez         | LPN       |
+-------------------------+-----------+
15 rows in set (0.04 sec)
```

Second Advanced Query

Get the number of females over the age of 30 affected by all conditions. Sort by descending order by the number of females affected and the condition name in ascending order

This query will be useful in conjunction with similar queries to compare the condition rates between various groups that can allow us to communicate high condition risk warnings to the patients who belong to a certain group.

SELECT name as conditionName, count(patientId) as FemAffected
FROM Patient NATURAL JOIN HasCondition NATURAL JOIN Conditions
WHERE sex = ‘F’ AND age > 30
GROUP BY conditionName
ORDER BY FemAffected DESC, avgAge DESC, conditionName
LIMIT 15;

*NOTE: if you try to copy and paste the command from here, the single quotation marks didn’t translate to single quotes when pasted into the terminal, so you may have to manually input single quotes when you paste the command.

```
mysql> SELECT name as conditionName, count(patientId) as FemAffected FROM Patient NATURAL JOIN HasCondition NATURAL JOIN Conditions WHERE sex = 'F' AND age > 30 GROUP BY conditionName ORDER BY
 FemAffected DESC, conditionName  LIMIT 15;
+---------------------+-------------+
| conditionName       | FemAffected |
+---------------------+-------------+
| Heart attack        |           2 |
| Varicose veins      |           2 |
| Abdominal hernia    |           1 |
| Abscess of nose     |           1 |
| Acne                |           1 |
| Actinic keratosis   |           1 |
| Acute bronchospasm  |           1 |
| Acute glaucoma      |           1 |
| Acute pancreatitis  |           1 |
| Adrenal cancer      |           1 |
| Alcoholic hepatitis |           1 |
| Alzheimer disease   |           1 |
| Anal fissure        |           1 |
| Aplastic anemia     |           1 |
| Arthritis           |           1 |
+---------------------+-------------+
15 rows in set (0.01 sec)
```

Indexing Analysis

- +3% on trying at least three different indexing designs (excluding the default index) for each advanced query.
- 4% on the indexing analysis reports.

- 1% on the accuracy and thoroughness of the analyses.

First Advanced Query

Indexes:

1. CREATE INDEX procedures_name_idx ON Procedures(name)
2. CREATE INDEX procedures_description10_idx ON Procedures(description(10))
3. CREATE INDEX procedures_description30_idx ON Procedures(description(30))

- Index 2 and 3 may not bring a better effect, because the number of characters before ‘title’ is unknown. This can be covered in the analyses part.

We first create an index procedures_name_idx (Index 1) on Procedures:name to possibly speed up the filter process in ‘WHERE name = 'Physical Examination''. We then use the EXPLAIN ANALYZE command to measure the query performance. As we filter the ‘name’ attribute twice, we compare the actual time in each filter process in Table 1. The complete comparison of EXPLAIN ANALYZE result is in appendix [1].

	Original	After Index 1
1st filter	0.016…0.020	0.005…0.006
2nd filter	0.012…0.014	0.005…0.006

Table 1: Comparison of Actual Time

After adding Index 1, the actual time of running ‘WHERE name = 'Physical Examination' was reduced by more than half. Therefore, we keep the Index 1 procedures_name_idx.

We then consider indexing on Procedures:description to speed up the ‘LOCATE(title,(SELECT description FROM Procedures WHERE name = 'Physical Examination')) > 0’. As description is of type VARCHAR(1024), we choose to use the first 10 characters for procedures_description10_idx (Index 2) and the first 30 characters for procedures_description30_idx (Index 3).

	After Index 1	After Index 1&2	After Index 1&3
locate	0.005…0.005	0.004…0.004	0.004…0.005

Table 2: Comparison of Actual Time

After adding Index 2 the time of locating is reduced by 20%, so we choose to use Index 1 and 2. Index 3 takes more time than Index 2 possibly because Indexing on 30 characters takes more time than on 10 characters. The complete comparison of EXPLAIN ANALYZE result is in appendix [2].

Second Advanced Query

```
SELECT name as conditionName, count(patientId) as FemAffected
FROM Patient NATURAL JOIN HasCondition NATURAL JOIN Conditions
WHERE sex = ‘F AND age > 30
GROUP BY conditionName
ORDER BY FemAffected DESC, avgAge DESC, conditionName
LIMIT 15;
```

Indexes:

1. CREATE INDEX patient_sex_idx ON Patient(sex)
2. CREATE INDEX patient_age_idx ON Patient(age)
3. CREATE INDEX conditions_name_idx ON Conditions(name)

We consider patient_sex_idx (index 1) and patient_age_idx (index 2) to speed up the filter process in ‘WHERE sex = ‘F AND age > 30’ and conditions_name_idx (index 3) to speed up the ‘GROUP BY’ and ‘ORDER BY’ process.

We first compare the actual time of the nested inner join and filter process for original, after index 1, after index 2 and after index 1&2 in Table 3.

	Original	After Index 1	After Index 2	After Index 1&2
Actual Time	0.512…5.77	1.983…8.596	0.235…4.827	0.0075…4.372

Table 3: Comparison of Actual time

From Table 3 we notice that After Index 1&2 the actual time spent in the nested inner join and filter process is minimal, so we use Index 1 and 2. We then compare the actual time spent after adding index 1&2 and adding index 1&2&3 in Table 4. As there are no major differences before and after adding Index 3, we do not use Index 3 to speed up our query. [3]

	After Index 1&2	After Index 1&2&3
Actual Time	0.010..0.380	0.012..0.303

Table 4: Comparison of Actual time

Appendix

[1] Comparison of Query 1 Original and After Index 1

Original	After Index 1
'-> Limit: 15 row(s) (actual time=0.152..0.155 rows=15 loops=1)\n -> Sort: Practitioner.`name`, limit input to 15 row(s) per chunk (actual time=0.152..0.153 rows=15 loops=1)\n -> Stream results (cost=2.55 rows=2) (actual time=0.064..0.134 rows=15 loops=1)\n -> Nested loop inner join (cost=2.55 rows=2) (actual time=0.060..0.124 rows=15 loops=1)\n -> Filter: ((Insurance.insProvider = (select #2)) and (Insurance.practitionerId is not null)) (cost=1.95 rows=2) (actual time=0.037..0.055 rows=15 loops=1)\n -> Table scan on Insurance (cost=1.95 rows=17) (actual time=0.007..0.018 rows=17 loops=1)\n -> Select #2 (subquery in condition; run only once)\n -> Nested loop inner join (cost=0.70 rows=1) (actual time=0.017..0.023 rows=1 loops=1)\n -> Filter: (Procedures.`name` = \'Physical Examination\') (cost=0.35 rows=1) (actual time=0.010..0.012 rows=1 loops=1)\n -> Table scan on Procedures (cost=0.35 rows=1) (actual time=0.006..0.008 rows=1 loops=1)\n -> Filter: (NeedProcedure.procedureId = Procedures.procedureId) (cost=0.35 rows=1) (actual time=0.006..0.009 rows=1 loops=1)\n -> Index lookup on NeedProcedure using patientId (patientId=1) (cost=0.35 rows=1) (actual time=0.006..0.009 rows=1 loops=1)\n -> Filter: (locate(Practitioner.title,(select #3)) > 0) (cost=0.31 rows=1) (actual time=0.004..0.004 rows=1 loops=15)\n -> Single-row index lookup on Practitioner using PRIMARY (practitionerId=Insurance.practitionerId) (cost=0.31 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n -> Select #3 (subquery in condition; run only once)\n -> Filter: (Procedures.`name` = \'Physical Examination\') (cost=0.35 rows=1) (actual time=0.007..0.008 rows=1 loops=1)\n -> Table scan on Procedures (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n'	'-> Limit: 15 row(s) (actual time=0.171..0.173 rows=15 loops=1)\n -> Sort: Practitioner.`name`, limit input to 15 row(s) per chunk (actual time=0.170..0.171 rows=15 loops=1)\n -> Stream results (cost=2.55 rows=2) (actual time=0.056..0.150 rows=15 loops=1)\n -> Nested loop inner join (cost=2.55 rows=2) (actual time=0.054..0.142 rows=15 loops=1)\n -> Filter: ((Insurance.insProvider = (select #2)) and (Insurance.practitionerId is not null)) (cost=1.95 rows=2) (actual time=0.037..0.055 rows=15 loops=1)\n -> Table scan on Insurance (cost=1.95 rows=17) (actual time=0.017..0.028 rows=17 loops=1)\n -> Select #2 (subquery in condition; run only once)\n -> Nested loop inner join (cost=0.70 rows=1) (actual time=0.012..0.014 rows=1 loops=1)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Filter: (NeedProcedure.procedureId = Procedures.procedureId) (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Index lookup on NeedProcedure using patientId (patientId=1) (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Filter: (locate(Practitioner.title,(select #3)) > 0) (cost=0.31 rows=1) (actual time=0.005..0.005 rows=1 loops=15)\n -> Single-row index lookup on Practitioner using PRIMARY (practitionerId=Insurance.practitionerId) (cost=0.31 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n -> Select #3 (subquery in condition; run only once)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n'

[2] Comparison of Query 1 After Index 1, After Index 1&2, After Index 1&3

After Index 1	After Index 1&2	After Index 1&3
'-> Limit: 15 row(s) (actual time=0.171..0.173 rows=15 loops=1)\n -> Sort: Practitioner.`name`, limit input to 15 row(s) per chunk (actual time=0.170..0.171 rows=15 loops=1)\n -> Stream results (cost=2.55 rows=2) (actual time=0.056..0.150 rows=15 loops=1)\n -> Nested loop inner join (cost=2.55 rows=2) (actual time=0.054..0.142 rows=15 loops=1)\n -> Filter: ((Insurance.insProvider = (select #2)) and (Insurance.practitionerId is not null)) (cost=1.95 rows=2) (actual time=0.037..0.055 rows=15 loops=1)\n -> Table scan on Insurance (cost=1.95 rows=17) (actual time=0.017..0.028 rows=17 loops=1)\n -> Select #2 (subquery in condition; run only once)\n -> Nested loop inner join (cost=0.70 rows=1) (actual time=0.012..0.014 rows=1 loops=1)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Filter: (NeedProcedure.procedureId = Procedures.procedureId) (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Index lookup on NeedProcedure using patientId (patientId=1) (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Filter: (locate(Practitioner.title,(select #3)) > 0) (cost=0.31 rows=1) (actual time=0.005..0.005 rows=1 loops=15)\n -> Single-row index lookup on Practitioner using PRIMARY (practitionerId=Insurance.practitionerId) (cost=0.31 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n -> Select #3 (subquery in condition; run only once)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n' (actual time=0.005..0.006 rows=1 loops=1)\n'	'-> Limit: 15 row(s) (actual time=0.141..0.143 rows=15 loops=1)\n -> Sort: Practitioner.`name`, limit input to 15 row(s) per chunk (actual time=0.141..0.142 rows=15 loops=1)\n -> Stream results (cost=2.55 rows=2) (actual time=0.050..0.123 rows=15 loops=1)\n -> Nested loop inner join (cost=2.55 rows=2) (actual time=0.048..0.115 rows=15 loops=1)\n -> Filter: ((Insurance.insProvider = (select #2)) and (Insurance.practitionerId is not null)) (cost=1.95 rows=2) (actual time=0.029..0.048 rows=15 loops=1)\n -> Table scan on Insurance (cost=1.95 rows=17) (actual time=0.008..0.019 rows=17 loops=1)\n -> Select #2 (subquery in condition; run only once)\n -> Nested loop inner join (cost=0.70 rows=1) (actual time=0.013..0.015 rows=1 loops=1)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.006..0.007 rows=1 loops=1)\n -> Filter: (NeedProcedure.procedureId = Procedures.procedureId) (cost=0.35 rows=1) (actual time=0.006..0.007 rows=1 loops=1)\n -> Index lookup on NeedProcedure using patientId (patientId=1) (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=1)\n -> Filter: (locate(Practitioner.title,(select #3)) > 0) (cost=0.31 rows=1) (actual time=0.004..0.004 rows=1 loops=15)\n -> Single-row index lookup on Practitioner using PRIMARY (practitionerId=Insurance.practitionerId) (cost=0.31 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n -> Select #3 (subquery in condition; run only once)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.006..0.007 rows=1 loops=1)\n' (actual time=0.005..0.006 rows=1 loops=1)\n'	'-> Limit: 15 row(s) (actual time=0.150..0.152 rows=15 loops=1)\n -> Sort: Practitioner.`name`, limit input to 15 row(s) per chunk (actual time=0.149..0.150 rows=15 loops=1)\n -> Stream results (cost=2.55 rows=2) (actual time=0.052..0.132 rows=15 loops=1)\n -> Nested loop inner join (cost=2.55 rows=2) (actual time=0.049..0.123 rows=15 loops=1)\n -> Filter: ((Insurance.insProvider = (select #2)) and (Insurance.practitionerId is not null)) (cost=1.95 rows=2) (actual time=0.031..0.049 rows=15 loops=1)\n -> Table scan on Insurance (cost=1.95 rows=17) (actual time=0.007..0.018 rows=17 loops=1)\n -> Select #2 (subquery in condition; run only once)\n -> Nested loop inner join (cost=0.70 rows=1) (actual time=0.014..0.016 rows=1 loops=1)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.006..0.006 rows=1 loops=1)\n -> Filter: (NeedProcedure.procedureId = Procedures.procedureId) (cost=0.35 rows=1) (actual time=0.007..0.008 rows=1 loops=1)\n -> Index lookup on NeedProcedure using patientId (patientId=1) (cost=0.35 rows=1) (actual time=0.007..0.008 rows=1 loops=1)\n -> Filter: (locate(Practitioner.title,(select #3)) > 0) (cost=0.31 rows=1) (actual time=0.004..0.005 rows=1 loops=15)\n -> Single-row index lookup on Practitioner using PRIMARY (practitionerId=Insurance.practitionerId) (cost=0.31 rows=1) (actual time=0.002..0.002 rows=1 loops=15)\n -> Select #3 (subquery in condition; run only once)\n -> Index lookup on Procedures using procedures_name_idx (name=\'Physical Examination\') (cost=0.35 rows=1) (actual time=0.007..0.007 rows=1 loops=1)\n'

[3] Result of adding different indices to speed up Query 2

ORIGINAL

'-> Limit: 15 row(s) (actual time=3.012..3.015 rows=15 loops=1)\n
-> Sort: FemAffected DESC, avgAge DESC, Conditions.`name`, limit input to 15 row(s) per chunk (actual time=3.011..3.013 rows=15 loops=1)\n
-> Table scan on <temporary> (actual time=0.002..0.022 rows=204 loops=1)\n
-> Aggregate using temporary table (actual time=2.748..2.785 rows=204 loops=1)\n
-> Nested loop inner join (cost=135.05 rows=45) (actual time=0.154..2.434 rows=206 loops=1)\n
-> Nested loop inner join (cost=119.28 rows=45) (actual time=0.140..1.948 rows=206 loops=1)\n
-> Filter: ((Patient.sex = \'F\') and (Patient.age > 30)) (cost=103.50 rows=33) (actual time=0.113..0.837 rows=406 loops=1)\n
-> Table scan on Patient (cost=103.50 rows=1000) (actual time=0.105..0.551 rows=1000 loops=1)\n
-> Filter: (HasCondition.conditionId is not null) (cost=0.34 rows=1) (actual time=0.002..0.003 rows=1 loops=406)\n
-> Index lookup on HasCondition using patientId (patientId=Patient.patientId) (cost=0.34 rows=1) (actual time=0.002..0.002 rows=1 loops=406)\n
-> Single-row index lookup on Conditions using PRIMARY (conditionId=HasCondition.conditionId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=206)\n'

AFTER INDEX 1

'-> Limit: 15 row(s) (actual time=3.684..3.687 rows=15 loops=1)\n
-> Sort: FemAffected DESC, avgAge DESC, Conditions.`name`, limit input to 15 row(s) per chunk (actual time=3.683..3.685 rows=15 loops=1)\n
-> Table scan on <temporary> (actual time=0.002..0.020 rows=204 loops=1)\n
-> Aggregate using temporary table (actual time=3.496..3.530 rows=204 loops=1)\n
-> Nested loop inner join (cost=187.03 rows=228) (actual time=0.509..3.150 rows=206 loops=1)\n
-> Nested loop inner join (cost=107.20 rows=228) (actual time=0.501..2.553 rows=206 loops=1)\n
-> Filter: (Patient.age > 30) (cost=27.36 rows=169) (actual time=0.488..1.482 rows=406 loops=1)\n
-> Index lookup on Patient using patient_sex_idx (sex=\'F\') (cost=27.36 rows=506) (actual time=0.485..1.411 rows=506 loops=1)\n
-> Filter: (HasCondition.conditionId is not null) (cost=0.34 rows=1) (actual time=0.002..0.002 rows=1 loops=406)\n
-> Index lookup on HasCondition using patientId (patientId=Patient.patientId) (cost=0.34 rows=1) (actual time=0.002..0.002 rows=1 loops=406)\n
-> Single-row index lookup on Conditions using PRIMARY (conditionId=HasCondition.conditionId) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=206)\n'

AFTER INDEX 2

'-> Limit: 15 row(s) (actual time=2.519..2.522 rows=15 loops=1)\n
-> Sort: FemAffected DESC, avgAge DESC, Conditions.`name`, limit input to 15 row(s) per chunk (actual time=2.518..2.520 rows=15 loops=1)\n
-> Table scan on <temporary> (actual time=0.001..0.018 rows=204 loops=1)\n
-> Aggregate using temporary table (actual time=2.324..2.357 rows=204 loops=1)\n
-> Nested loop inner join (cost=179.43 rows=108) (actual time=0.073..2.049 rows=206 loops=1)\n
-> Nested loop inner join (cost=141.46 rows=108) (actual time=0.065..1.651 rows=206 loops=1)\n
-> Filter: ((Patient.sex = \'F\') and (Patient.age > 30)) (cost=103.50 rows=80) (actual time=0.052..0.692 rows=406 loops=1)\n
-> Table scan on Patient (cost=103.50 rows=1000) (actual time=0.045..0.435 rows=1000 loops=1)\n
-> Filter: (HasCondition.conditionId is not null) (cost=0.34 rows=1) (actual time=0.002..0.002 rows=1 loops=406)\n
-> Index lookup on HasCondition using patientId (patientId=Patient.patientId) (cost=0.34 rows=1) (actual time=0.002..0.002 rows=1 loops=406)\n
-> Single-row index lookup on Conditions using PRIMARY (conditionId=HasCondition.conditionId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=206)\n'

AFTER INDEX 1 AND 2

'-> Limit: 15 row(s) (actual time=3.360..3.363 rows=15 loops=1)\n
-> Sort: FemAffected DESC, avgAge DESC, Conditions.`name`, limit input to 15 row(s) per chunk (actual time=3.359..3.361 rows=15 loops=1)\n
-> Table scan on <temporary> (actual time=0.001..0.032 rows=204 loops=1)\n
-> Aggregate using temporary table (actual time=3.143..3.191 rows=204 loops=1)\n
-> Nested loop inner join (cost=415.15 rows=210) (actual time=0.054..2.850 rows=206 loops=1)\n
-> Nested loop inner join (cost=233.85 rows=518) (actual time=0.021..1.522 rows=520 loops=1)\n
-> Filter: ((HasCondition.conditionId is not null) and (HasCondition.patientId is not null)) (cost=52.55 rows=518) (actual time=0.012..0.487 rows=520 loops=1)\n
-> Table scan on HasCondition (cost=52.55 rows=518) (actual time=0.010..0.380 rows=520 loops=1)\n
-> Single-row index lookup on Conditions using PRIMARY (conditionId=HasCondition.conditionId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=520)\n
-> Filter: ((Patient.sex = \'F\') and (Patient.age > 30)) (cost=0.25 rows=0) (actual time=0.002..0.002 rows=0 loops=520)\n
-> Single-row index lookup on Patient using PRIMARY (patientId=HasCondition.patientId) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=520)\n'

After INDEX 1 2 AND 3

'-> Limit: 15 row(s) (actual time=2.979..2.982 rows=15 loops=1)\n
-> Sort: FemAffected DESC, avgAge DESC, Conditions.`name`, limit input to 15 row(s) per chunk (actual time=2.978..2.979 rows=15 loops=1)\n

-> Table scan on <temporary> (actual time=0.001..0.020 rows=204 loops=1)\n
-> Aggregate using temporary table (actual time=2.734..2.770 rows=204 loops=1)\n
-> Nested loop inner join (cost=415.15 rows=210) (actual time=0.061..2.461 rows=206 loops=1)\n
-> Nested loop inner join (cost=233.85 rows=518) (actual time=0.026..1.286 rows=520 loops=1)\n
-> Filter: ((HasCondition.conditionId is not null) and (HasCondition.patientId is not null)) (cost=52.55 rows=518) (actual time=0.014..0.395 rows=520 loops=1)\n
-> Table scan on HasCondition (cost=52.55 rows=518) (actual time=0.012..0.303 rows=520 loops=1)\n
-> Single-row index lookup on Conditions using PRIMARY (conditionId=HasCondition.conditionId) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=520)\n
-> Filter: ((Patient.sex = \'F\') and (Patient.age > 30)) (cost=0.25 rows=0) (actual time=0.002..0.002 rows=0 loops=520)\n
-> Single-row index lookup on Patient using PRIMARY (patientId=HasCondition.patientId) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=520)\n'