

DDL Commands

Reservations:

```
CREATE TABLE Reservations (  
    ReservationID INT AUTO_INCREMENT NOT NULL,  
    StartTime DATETIME NOT NULL,  
    ReturnTime DATETIME,  
    Deadline DATETIME NOT NULL,  
    NetID VARCHAR(100),  
    ItemID INT,  
    PRIMARY KEY (ReservationID, NetID, ItemID),  
    FOREIGN KEY (ItemID) REFERENCES Inventory(ItemID) ON UPDATE CASCADE,  
    FOREIGN KEY (NetID) REFERENCES Credentials(netID) ON DELETE CASCADE ON UPDATE  
    CASCADE  
);
```

Credentials:

```
CREATE TABLE Credentials (  
    netID VARCHAR(100) NOT NULL,  
    password VARCHAR(50),  
    permission Bool Default False,  
    PRIMARY KEY (netID)  
);
```

Facilities:

```
CREATE TABLE Facilities (  
    LocationID INT PRIMARY KEY NOT NULL AUTO_INCREMENT,  
    BldgName VARCHAR(20),  
    FloorSection VARCHAR(20), -- Renamed to avoid using a slash  
    Longitude FLOAT,  
    Latitude FLOAT,  
    MapURL VARCHAR(200)  
);
```

Has:

```
CREATE TABLE Has(  
    MajorID INT,  
    NetID VARCHAR(100),  
    PRIMARY KEY (NetID, MajorID),  
    FOREIGN KEY(MajorID) REFERENCES Majors(MajorID) ON UPDATE CASCADE,  
    FOREIGN KEY(NetID) REFERENCES Credentials(netID) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Inventory:

```
CREATE TABLE Inventory (  
    ItemID INT AUTO_INCREMENT PRIMARY KEY,  
    ItemName VARCHAR(100),  
    Availability BOOL,  
    `Condition` INT,  
    LocationID INT,  
    Duration INT,  
    FOREIGN KEY (LocationID) REFERENCES Facilities(LocationID) ON DELETE SET NULL ON UPDATE  
    CASCADE  
);
```

MajorRestrictions:

```
CREATE TABLE MajorRestriction(  
    MajorID INT,  
    ItemID INT,  
    PRIMARY KEY (MajorID, ItemID),  
    FOREIGN KEY (MajorID) REFERENCES Majors(MajorID) ON DELETE CASCADE ON UPDATE CASCADE,  
    FOREIGN KEY (ItemID) REFERENCES Inventory(ItemID) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

Majors:



```
CREATE TABLE Majors (  
    MajorID INT NOT NULL AUTO_INCREMENT,  
    MajorName VARCHAR(100),  
    CollegeName VARCHAR(100),  
    PRIMARY KEY (MajorID)  
);
```

1000+ rows verification:

4 • **SELECT** COUNT(*) **FROM** Inventory;

5

100% 1:10



Result Grid   Filter Rows:

COUNT(*)	
1371	

4 • **SELECT** COUNT(*) **FROM** Reservations;

5

100% 34:4



Result Grid   Filter Rows: Ex

COUNT(*)	
20565	

4 • **SELECT** COUNT(*) **FROM** Credentials;

5

00% 33:4

Result Grid   Filter Rows:

COUNT(*)	
2004	

Connection INFO:

```
db = mysql.connector.connect(  
    host="35.239.240.215",  
    user="albacore",  
    password="{REDACTED}"  
    database="SampleDB"  
)
```

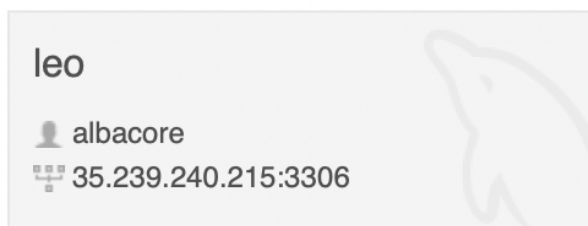
Filter Enter property name or value										?	☰
<input type="checkbox"/>	Instance ID ⓘ ⬆	Issues	Cloud SQL edition	Type	Public IP address	Private IP address	Instance connection name	High availability	Location	⌵	Actions
<input type="checkbox"/>	✔ campuscache	Underprovisioned resource	Enterprise	MySQL 8.0	35.239.240.215		centered-sight-4154... ⌵	ENABLE	us-central1-a		⋮

```
CLOUD SHELL
Terminal centered-sight-415421 x + -
Open Editor

Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_SampleDB |
+-----+
| Computers           |
| Credentials         |
| Facilities          |
| Has                 |
| Inventory           |
| MajorRestriction    |
| Majors              |
| Reservations        |
+-----+
```

MySQL Connections + ↻



(on MySQL Workbench)

Four Advanced Queries

Query #1:

This query will calculate the timely return ratio by major. This will be useful for the admin side of our application.

```
SELECT MajorID, MajorName, Count(*) AS LateCount
FROM Reservations NATURAL JOIN Has NATURAL JOIN Majors
Where Deadline < ReturnTime
GROUP BY MajorID
ORDER BY LateCount DESC;
```

Query Output:

MajorID	MajorName	LateCount
49	Chemistry	61
32	Biochemistry	59
63	Computer Sci & Chemistry	49
184	Musicology	13
26	Asian American Studies	13
94	Engineering	12
128	Horticulture	11
199	Physics	11
238	Teaching of German	11
121	Global Studies	11
213	Secondary Education	11
101	Environ Engr in Civil Engr	10
39	Bioprocessing and Bioen...	10
88	Educ Policy, Orgzn & Le...	10
91	Electrical & Computer Engr	9

Combination of Indexes Tried:

1. Reservations(Deadline)
2. Reservations(ReturnTime)
3. Reservations(Deadline, ReturnTime)

```
1 • USE SampleDB;
2
3 -- CREATE INDEX deadline_idx ON Reservations(Deadline);           -- Uncomment for combination 1
4 -- CREATE INDEX return_time_idx ON Reservations(ReturnTime);       -- Uncomment for combination 2
5 -- CREATE INDEX deadline_return_time_idx ON Reservations(Deadline, ReturnTime); -- Uncomment for combination 3
6
7 • EXPLAIN ANALYZE (
8     SELECT MajorID, MajorName, Count(*) AS LateCount
9     FROM Reservations NATURAL JOIN Has NATURAL JOIN Majors
10    Where Deadline < ReturnTime
11    GROUP BY MajorID
12    ORDER BY LateCount DESC
13 );
14
15 -- DROP INDEX deadline_idx ON Reservations;           -- Uncomment for combination 1
16 -- DROP INDEX return_time_idx ON Reservations;       -- Uncomment for combination 2
17 -- DROP INDEX deadline_return_time_idx ON Reservations; -- Uncomment for combination 3
```

Baseline:

-> Sort: LateCount DESC (actual time=79.635..79.652 rows=251 loops=1)
-> Table scan on <temporary> (actual time=79.368..79.443 rows=251 loops=1)
-> Aggregate using temporary table (actual time=79.363..79.363 rows=251 loops=1)
-> Nested loop inner join (cost=9835.73 rows=9000) (actual time=2.363..78.375 rows=1238 loops=1)
-> Nested loop inner join (cost=385.05 rows=2648) (actual time=0.258..2.500 rows=2648 loops=1)
-> Table scan on Majors (cost=26.35 rows=256) (actual time=0.169..0.486 rows=256 loops=1)
-> Covering index lookup on Has using MajorID (MajorID=Majors.MajorID) (cost=0.37 rows=10) (actual time=0.005..0.007 rows=10 loops=256)
-> Filter: (Reservations.Deadline < Reservations.ReturnTime) (cost=2.55 rows=3) (actual time=0.025..0.028 rows=0 loops=2648)
-> Index lookup on Reservations using NetID (NetID=Has.NetID) (cost=2.55 rows=10) (actual time=0.022..0.027 rows=11 loops=2648)

Deadline idx - first query

-> Sort: LateCount DESC (actual time=118.511..118.537 rows=251 loops=1)
-> Table scan on <temporary> (actual time=118.301..118.373 rows=251 loops=1)
-> Aggregate using temporary table (actual time=118.298..118.298 rows=251 loops=1)
-> Nested loop inner join (cost=9835.73 rows=9000) (actual time=0.442..116.537 rows=1238 loops=1)
-> Nested loop inner join (cost=385.05 rows=2648) (actual time=0.129..2.985 rows=2648 loops=1)
-> Table scan on Majors (cost=26.35 rows=256) (actual time=0.109..0.351 rows=256 loops=1)
-> Covering index lookup on Has using MajorID (MajorID=Majors.MajorID) (cost=0.37 rows=10) (actual time=0.006..0.009 rows=10 loops=256)
-> Filter: (Reservations.Deadline < Reservations.ReturnTime) (cost=2.55 rows=3) (actual time=0.037..0.043 rows=0 loops=2648)
-> Index lookup on Reservations using NetID (NetID=Has.NetID) (cost=2.55 rows=10) (actual time=0.032..0.041 rows=11 loops=2648)

Return_idx

-> Sort: LateCount DESC (actual time=63.363..63.379 rows=251 loops=1)
-> Table scan on <temporary> (actual time=63.230..63.284 rows=251 loops=1)
-> Aggregate using temporary table (actual time=63.226..63.226 rows=251 loops=1)
-> Nested loop inner join (cost=9835.73 rows=9000) (actual time=0.258..62.328 rows=1238 loops=1)

-> Nested loop inner join (cost=385.05 rows=2648) (actual time=0.065..1.629 rows=2648 loops=1)
-> Table scan on Majors (cost=26.35 rows=256) (actual time=0.050..0.190 rows=256 loops=1)
-> Covering index lookup on Has using MajorID (MajorID=Majors.MajorID) (cost=0.37 rows=10) (actual time=0.003..0.005 rows=10 loops=256)
-> Filter: (Reservations.Deadline < Reservations.ReturnTime) (cost=2.55 rows=3) (actual time=0.019..0.023 rows=0 loops=2648)
-> Index lookup on Reservations using NetID (NetID=Has.NetID) (cost=2.55 rows=10) (actual time=0.016..0.022 rows=11 loops=2648)

Return & Deadline Joint Index (

-> Sort: LateCount DESC (actual time=93.710..93.724 rows=251 loops=1)
-> Table scan on <temporary> (actual time=93.544..93.612 rows=251 loops=1)
-> Aggregate using temporary table (actual time=93.540..93.540 rows=251 loops=1)
-> Nested loop inner join (cost=9835.73 rows=9000) (actual time=0.610..91.956 rows=1238 loops=1)
-> Nested loop inner join (cost=385.05 rows=2648) (actual time=0.172..2.376 rows=2648 loops=1)
-> Table scan on Majors (cost=26.35 rows=256) (actual time=0.127..0.348 rows=256 loops=1)
-> Covering index lookup on Has using MajorID (MajorID=Majors.MajorID) (cost=0.37 rows=10) (actual time=0.004..0.007 rows=10 loops=256)
-> Filter: (Reservations.Deadline < Reservations.ReturnTime) (cost=2.55 rows=3) (actual time=0.028..0.034 rows=0 loops=2648)
-> Index lookup on Reservations using NetID (NetID=Has.NetID) (cost=2.55 rows=10) (actual time=0.023..0.032 rows=11 loops=2648)

Justification:

As the indexes for Primary and Foreign Keys to each tables are created upon table creation, we tried making indexes for attributes that were not primary or foreign keys. Although we created indexes for Reservations(ReturnTime) and Reservations(Deadline), both indexes did not change the cost of the query at all.

Query #2:

This query will calculate the average return time for an item, ignoring ongoing reservations.

```
SELECT itemID, itemName, AVG(TIMESTAMPDIFF(MINUTE, StartTime, ReturnTime))
FROM Reservations NATURAL JOIN Inventory
WHERE ReturnTime IS NOT NULL -- to ignore ongoing reservations
GROUP BY itemID;
```

Query output:

itemID	itemName	AVG(TIMESTAMPDIFF(MINUTE, StartTime, R...
1	3D Printer	151.2667
2	3D Printer	52.7333
3	3D Printer	1110.2667
4	3D Printer	44.6000
5	3D Printer	798.2667
6	3D Printer	658.2667
7	3D Printer	76.2000
8	3D Printer	60.8000
9	3D Printer	54.6667
10	3D Printer	690.3333
11	3D Printer	69.1333
12	3D Printer	215.7333
13	3D Printer	69.7333
14	3D Printer	82.4667
15	3D Printer	153.2000

Combination of Indexes Tried:

1. Reservations(StartTime)
2. Reservations(ReturnTime)
3. Reservations(StartTime, ReturnTime)

```
1 • USE SampleDB;
2
3 -- CREATE INDEX start_time_idx ON Reservations(StartTime);           -- Uncomment for combination 1
4 -- CREATE INDEX return_time_idx ON Reservations(ReturnTime);         -- Uncomment for combination 2
5 -- CREATE INDEX start_and_return_time_idx ON Reservations(StartTime, ReturnTime); -- Uncomment for combination 3
6
7 • EXPLAIN ANALYZE (
8     SELECT itemID, itemName, AVG(TIMESTAMPDIFF(MINUTE, StartTime, ReturnTime))
9     FROM Reservations NATURAL JOIN Inventory
10    WHERE ReturnTime IS NOT NULL -- to ignore ongoing reservations
11    GROUP BY itemID
12 );
13
14 -- DROP INDEX start_time_idx ON Reservations;           -- Uncomment for combination 1
15 -- DROP INDEX return_time_idx ON Reservations;         -- Uncomment for combination 2
16 -- DROP INDEX start_and_return_time_idx ON Reservations; -- Uncomment for combination 3
```


Baseline, Combination 1, Combination 3:

-> Table scan on <temporary> (actual time=60.618..61.117 rows=1371 loops=1)
-> Aggregate using temporary table (actual time=60.614..60.614 rows=1371 loops=1)
-> Nested loop inner join (cost=7089.06 rows=17882) (actual time=0.352..41.807 rows=20565 loops=1)
-> Table scan on Inventory (cost=135.05 rows=1333) (actual time=0.193..0.744 rows=1371 loops=1)
-> Filter: (Reservations.ReturnTime is not null) (cost=3.73 rows=13) (actual time=0.025..0.029 rows=15 loops=1371)
-> Index lookup on Reservations using ItemID (ItemID=Inventory.ItemID) (cost=3.73 rows=15) (actual time=0.025..0.027 rows=15 loops=1371)

Combination 2:

-> Table scan on <temporary> (actual time=33.973..34.432 rows=1371 loops=1)
-> Aggregate using temporary table (actual time=33.969..33.969 rows=1371 loops=1)
-> Nested loop inner join (cost=5643.70 rows=10217) (actual time=0.088..19.127 rows=20565 loops=1)
-> Filter: (Reservations.ReturnTime is not null) (cost=2067.75 rows=10217) (actual time=0.070..9.612 rows=20565 loops=1)
-> Table scan on Reservations (cost=2067.75 rows=20435) (actual time=0.069..7.934 rows=20565 loops=1)
-> Single-row index lookup on Inventory using PRIMARY (ItemID=Reservations.ItemID) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=20565)

Justification:

For this query, we did not observe any changes when adding the indices for combination 1 and combination 3. However when we did the index for combination 2, we noticed some changes. We saw that the cost for the Nested loop inner join went down from 7089.06 to 5643.70. However, we noticed that it used a different order and performed the Filter before the Table scan and went through more rows for both and had a higher cost. It seems that combination 2 over all had higher costs. We are uncertain as to why this happened but suspect it may have been trying to optimize and use a different algorithm since it had the index. There probably were not enough entries with null return times such that the number of entries would be significantly reduced in order to justify the different algorithm. So we are sticking with the baseline indexing on the primary keys to not have unnecessary indices and not get worse performance.

Query #3:

This query will find the locations with the most amount of broken items.

```
SELECT BldgName, Count(*) as Num_Broken
FROM Inventory NATURAL JOIN Facilities
WHERE `Condition` = 0
GROUP BY BldgName
ORDER BY Num_Broken DESC;
```

Our database has less than 15 buildings, so this output is less than 15 rows.

Query output:

BldgName	Num_Broken
Main Library	80
BIF	70
M&P Arts Library	30
Grainger Library	29
Noyes Lab	27
English Building	21

Combination of Indexes Tried:

1. Inventory(`Condition`)
2. Facilities(BldgName)
3. Inventory(`Condition`) and Facilities(BldgName)

Baseline cost without indexes:

```
3 • EXPLAIN ANALYZE
4 SELECT BldgName, Count(*) as Num_Broken
5 FROM Inventory NATURAL JOIN Facilities
6 WHERE `Condition` = 0
7 GROUP BY BldgName
8 ORDER BY Num_Broken DESC;
9
```

-> Sort: Num_Broken DESC (actual time=0.966..0.967 rows=6 loops=1)
-> Table scan on <temporary> (actual time=0.951..0.952 rows=6 loops=1)
-> Aggregate using temporary table (actual time=0.949..0.949 rows=6 loops=1)
-> Nested loop inner join (cost=181.71 rows=133) (actual time=0.158..0.779 rows=257 loops=1)
-> Filter: ((Inventory.`Condition` = 0) and (Inventory.LocationID is not null)) (cost=135.05 rows=133) (actual time=0.145..0.653 rows=257 loops=1)
-> Table scan on Inventory (cost=135.05 rows=1333) (actual time=0.061..0.530 rows=1371 loops=1)
-> Single-row index lookup on Facilities using PRIMARY (LocationID=Inventory.LocationID) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=257)

Cost with condition index:

```
4 • CREATE INDEX condition_idx ON Inventory(`Condition`);
5 • EXPLAIN ANALYZE
6 SELECT BldgName, Count(*) as Num_Broken
7 FROM Inventory NATURAL JOIN Facilities
8 WHERE `Condition` = 0
9 GROUP BY BldgName
10 ORDER BY Num_Broken DESC;
11 • DROP INDEX condition_idx ON Inventory;
12
```

-> Sort: Num_Broken DESC (actual time=0.786..0.786 rows=6 loops=1)
-> Table scan on <temporary> (actual time=0.768..0.769 rows=6 loops=1)
-> Aggregate using temporary table (actual time=0.767..0.767 rows=6 loops=1)
-> Nested loop inner join (cost=120.90 rows=257) (actual time=0.231..0.606 rows=257 loops=1)
-> Filter: (Inventory.LocationID is not null) (cost=30.95 rows=257) (actual time=0.220..0.468 rows=257 loops=1)
-> Index lookup on Inventory using condition_idx (Condition=0) (cost=30.95 rows=257) (actual time=0.219..0.448 rows=257 loops=1)
-> Single-row index lookup on Facilities using PRIMARY (LocationID=Inventory.LocationID) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=257)

Cost with building name index:

```

4 • CREATE INDEX bldgname_idx ON Facilities(BldgName);
5 • EXPLAIN ANALYZE
6 SELECT BldgName, Count(*) as Num_Broken
7 FROM Inventory NATURAL JOIN Facilities
8 WHERE `Condition` = 0
9 GROUP BY BldgName
10 ORDER BY Num_Broken DESC;
11 • DROP INDEX bldgname_idx ON Facilities;
--

```

Building Name idx

```

-> Sort: Num_Broken DESC (actual time=0.841..0.842 rows=6 loops=1)
-> Table scan on <temporary> (actual time=0.824..0.825 rows=6 loops=1)
-> Aggregate using temporary table (actual time=0.823..0.823 rows=6 loops=1)
-> Nested loop inner join (cost=181.71 rows=133) (actual time=0.141..0.667 rows=257 loops=1)
-> Filter: ((Inventory.`Condition` = 0) and (Inventory.LocationID is not null)) (cost=135.05 rows=133) (actual time=0.129..0.559 rows=257 loops=1)
-> Table scan on Inventory (cost=135.05 rows=1333) (actual time=0.050..0.436 rows=1371 loops=1)
-> Single-row index lookup on Facilities using PRIMARY (LocationID=Inventory.LocationID) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=257)

```

Cost with building name and condition index:

```

4 • CREATE INDEX bldgname_idx ON Facilities(BldgName);
5 • CREATE INDEX condition_idx ON Inventory(`condition`);
6 • EXPLAIN ANALYZE
7 SELECT BldgName, Count(*) as Num_Broken
8 FROM Inventory NATURAL JOIN Facilities
9 WHERE `Condition` = 0
10 GROUP BY BldgName
11 ORDER BY Num_Broken DESC;
12 • DROP INDEX bldgname_idx ON Facilities;
13 • DROP INDEX condition_idx ON Inventory;
14

```

Building Name + Condition

```

-> Sort: Num_Broken DESC (actual time=0.744..0.745 rows=6 loops=1)
-> Table scan on <temporary> (actual time=0.720..0.722 rows=6 loops=1)
-> Aggregate using temporary table (actual time=0.719..0.719 rows=6 loops=1)
-> Nested loop inner join (cost=120.90 rows=257) (actual time=0.192..0.528 rows=257 loops=1)
-> Filter: (Inventory.LocationID is not null) (cost=30.95 rows=257) (actual time=0.181..0.414 rows=257 loops=1)
-> Index lookup on Inventory using condition_idx (Condition=0) (cost=30.95 rows=257) (actual

```

time=0.179..0.394 rows=257 loops=1)
-> Single-row index lookup on Facilities using PRIMARY (LocationID=Inventory.LocationID) (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=257)

Justification:

Combination #1:

The cost of scanning through the Inventory table and filtering out records that were not null and had 'Condition = 0' decreased from 135.05 to 30.95. Before performing the join between Inventory and Facilities tables, the query first scans through the entire Inventory table and filters out records which have non-null LocationIDs and condition = 0. The index of the Condition attribute in the Inventory table sped up this process, which lowered the overall cost of the query.

Combination #2:

Although BldgName was an attribute used in the Group By clause, creating an index of the BldgName attribute did not alter the cost of the query. The index created indexes the BldgName for all records in the Facility table, and this seemed to have no effect on the Group By aggregation performed on the temporary table (natural join between Inventory and Facilities). Therefore, the cost of this index combination resembled that of the baseline.

Combination #3:

As the BldgName index did not affect the cost of the query, we expected the combination of both the Inventory(Condition) index and Facility(BldgName) index to cost the same as the first combination. As expected the cost of this combination was the same as that of combination #1.

Because the Condition index led to a noticeable decrease in costs, while the BldgName index did not lead to any change in costs, we have decided to add the Condition index to improve performance for this query, and to not include the BldgName index.

Query #4:

Show only items that the user has permission to use (unrestricted / in major) for a certain user.

```
SELECT DISTINCT ItemID, ItemName, BldgName, FloorSection
FROM Inventory NATURAL JOIN Facilities
WHERE ItemID NOT IN
  (SELECT DISTINCT ItemID
   FROM MajorRestriction
   WHERE ItemID NOT IN
     (SELECT DISTINCT ItemID
      FROM MajorRestriction NATURAL JOIN
        (SELECT MajorID
         FROM Has
         WHERE Has.NetID = "acotelardm1") AS UserMajors)); -- replace "acotelardm1" with the
user's netid
```

Query Output:

	ItemID	ItemName	BldgName	FloorSection	
	81	Stanley Linesman	SCD	Lower Level	
	82	Irwin Vice Grip - Curved Jaw Locking Pliers with...	SCD	Lower Level	
	83	Irwin Vice Grip - Curved Jaw Locking Pliers with...	SCD	Lower Level	
	84	Irwin Vice Grip - Curved Jaw Locking Pliers with...	SCD	Lower Level	
	85	Stanley Groove Joint Pliers	SCD	Lower Level	
	86	Stanley Long Nose Pliers	SCD	Lower Level	
	87	Stanley Diagonal Pliers	SCD	Lower Level	
	88	Wire Stripper	SCD	Lower Level	
	89	Stanley Bent Nose Pliers	SCD	Lower Level	
	90	Stanley Linesman	SCD	Lower Level	
	91	Irwin Vice Grip - Curved Jaw Locking Pliers with...	SCD	Lower Level	
	92	Irwin Vice Grip - Curved Jaw Locking Pliers with...	SCD	Lower Level	
	93	Irwin Vice Grip - Curved Jaw Locking Pliers with...	SCD	Lower Level	
	94	Stanley Groove Joint Pliers	SCD	Lower Level	
	95	Stanley Long Nose Pliers	SCD	Lower Level	

Combination of Indexes Tried:

1. Facilities(BuildingName)
2. Inventory(ItemName)
3. Inventory(ItemName), Facilities(BuildingName)

Baseline Cost:

-> Table scan on <temporary> (cost=320.36..339.51 rows=1333) (actual time=6.434..6.607 rows=1273 loops=1)

-> Temporary table with deduplication (cost=320.35..320.35 rows=1333) (actual time=6.430..6.430

rows=1273 loops=1)
-> Nested loop inner join (cost=187.05 rows=1333) (actual time=2.745..5.478 rows=1273 loops=1)
-> Table scan on Facilities (cost=1.25 rows=10) (actual time=0.023..0.031 rows=10 loops=1)
-> Filter: <in_optimizer>(Inventory.ItemID,Inventory.ItemID in (select #2) is false) (cost=6.58 rows=133) (actual time=0.345..0.532 rows=127 loops=10)
-> Index lookup on Inventory using LocationID (LocationID=Facilities.LocationID) (cost=6.58 rows=133) (actual time=0.096..0.168 rows=137 loops=10)
-> Select #2 (subquery in condition; run only once)
-> Filter: ((Inventory.ItemID = `<materialized_subquery>`.ItemID)) (cost=78873.36..78873.36 rows=1) (actual time=0.002..0.002 rows=0 loops=1372)
-> Limit: 1 row(s) (cost=78873.26..78873.26 rows=1) (actual time=0.002..0.002 rows=0 loops=1372)
-> Index lookup on <materialized_subquery> using <auto_distinct_key> (ItemID=Inventory.ItemID) (actual time=0.002..0.002 rows=0 loops=1372)
-> Materialize with deduplication (cost=78873.26..78873.26 rows=391410) (actual time=2.333..2.333 rows=98 loops=1)
-> Nested loop antijoin (cost=39732.24 rows=391410) (actual time=0.185..1.946 rows=2548 loops=1)
-> Covering index scan on MajorRestriction using ItemID (cost=298.43 rows=2928) (actual time=0.022..0.628 rows=3148 loops=1)
-> Single-row index lookup on <subquery3> using <auto_distinct_key> (ItemID=MajorRestriction.ItemID) (actual time=0.000..0.000 rows=0 loops=3148)
-> Materialize with deduplication (cost=28.67..28.67 rows=134) (actual time=0.161..0.161 rows=200 loops=1)
-> Filter: (MajorRestriction.ItemID is not null) (cost=15.30 rows=134) (actual time=0.053..0.123 rows=200 loops=1)
-> Nested loop inner join (cost=15.30 rows=134) (actual time=0.052..0.109 rows=200 loops=1)
-> Covering index lookup on Has using PRIMARY (NetID="acotelardm1") (cost=0.94 rows=1) (actual time=0.020..0.021 rows=2 loops=1)
-> Covering index lookup on MajorRestriction using PRIMARY (MajorID=Has.MajorID) (cost=8.37 rows=101) (actual time=0.019..0.037 rows=100 loops=2)

Justification:

There was no effect in the cost from adding the given indexes that we tried. This is because all the clauses in our query are down of primary and foreign key indexes, creating an index for an attribute other than the primary/foreign keys would not have affected the cost of the query. Thus, for all the index combinations we tried, the cost stayed the same. For choosing our indexes, we chose attributes that were not primary/foreign keys, although we were aware this would have probably have no effect on the cost. Upon confirmation of our results of each of our index combinations and comparing it to the baseline cost, our theory was proven correct. As such, since creating indexes providing no change to the cost, we decided not to add an index.

Explain Analyze Command Screenshots:

```
1
2  USE SampleDB;
3  CREATE INDEX bldg_name_idx ON Facilities(BldgName);
4  EXPLAIN ANALYZE
5  SELECT DISTINCT ItemID, ItemName, BldgName, FloorSection
6  FROM Inventory NATURAL JOIN Facilities
7  WHERE ItemID NOT IN
8     (SELECT DISTINCT ItemID
9      FROM MajorRestriction
10     WHERE ItemID NOT IN
11        (SELECT DISTINCT ItemID
12         FROM MajorRestriction NATURAL JOIN
13          (SELECT MajorID
14           FROM Has
15           WHERE Has.NetID = "acotelardm1") AS UserMajors));
16
17 DROP INDEX bldg_name_idx ON Facilities;
```

```
1
2  USE SampleDB;
3  CREATE INDEX item_name_idx ON Inventory(ItemName);
4  EXPLAIN ANALYZE
5  SELECT DISTINCT ItemID, ItemName, BldgName, FloorSection
6  FROM Inventory NATURAL JOIN Facilities
7  WHERE ItemID NOT IN
8     (SELECT DISTINCT ItemID
9      FROM MajorRestriction
10     WHERE ItemID NOT IN
11        (SELECT DISTINCT ItemID
12         FROM MajorRestriction NATURAL JOIN
13          (SELECT MajorID
14           FROM Has
15           WHERE Has.NetID = "acotelardm1") AS UserMajors));
16
17 DROP INDEX item_name_idx ON Inventory;
```



```
USE SampleDB;
• CREATE INDEX inventory_idx ON Inventory(ItemName);
• CREATE INDEX facilities_idx ON Facilities(BldgName);
• EXPLAIN ANALYZE
SELECT DISTINCT ItemID, ItemName, BldgName, FloorSection
FROM Inventory NATURAL JOIN Facilities
WHERE ItemID NOT IN
  (SELECT DISTINCT ItemID
   FROM MajorRestriction
   WHERE ItemID NOT IN
     (SELECT DISTINCT ItemID
      FROM MajorRestriction NATURAL JOIN
        (SELECT MajorID
         FROM Has
         WHERE Has.NetID = "acotelardm1") AS UserMajors));
• DROP INDEX inventory_idx ON Inventory;
• DROP INDEX facilities_idx ON Facilities;
```