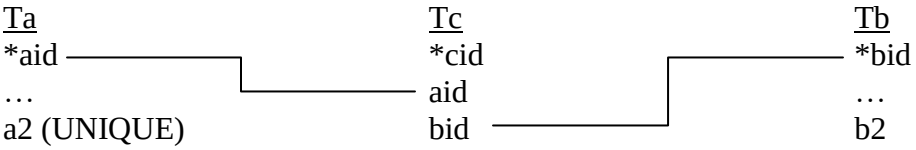
Lab 5. Indexes - Explanation of the requirements

Work on 3 tables of the form Ta(aid, a2, ...), Tb(bid, b2, ...), Tc(cid, aid, bid, ...), where:

- aid, bid, cid, a2, b2 are integers;
- the primary keys are underlined;
- a2 is UNIQUE in Ta;
- aid and bid are foreign keys in Tc, referencing the corresponding primary keys in Ta and Tb, respectively.

You have to create 3 tables that satisfy the conditions bellow (contain the following structure).

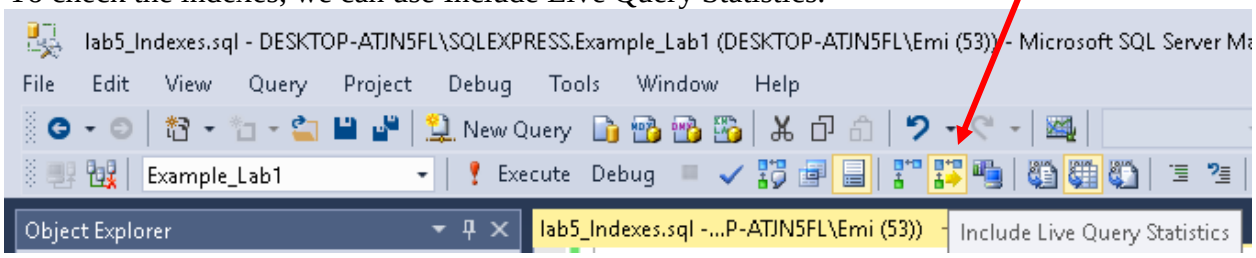


where aid, bid, cid, a2, b2 are INT

a. Write queries on Ta such that their execution plans contain the following operators:

- clustered index scan;
- clustered index seek;
- nonclustered index scan;
- nonclustered index seek;
- key lookup.

To check the indexes, we can use Include Live Query Statistics.



On the table Ta automatically was created a clustered index on the primary key (aid). This index is used in all the queries, if another non-clustered index is created.
The non-clustered indexes can and should be created on the field(s) involved in SELECT, ORDER BY, WHERE and JOIN's clauses.
Index Scan (=Table Scan) retrieves all the rows from the table.
Index Seek retrieves selective rows from the table.
Key Lookup is similar to Index Seek and can specify an additional pre-fetch argument.

- You have to create some queries, so that you will have all the execution plans required. The queries can be done with order by or where clauses (whatever you like).
- For the clustered requirements, you will have to take into account the primary key field
- For the non-clustered requirements, you will have to take into account the field(s) involved in Select, order by or where clauses; check first without a non-clustered index

on those fields; then, create a non-clustered index on the field(s) involved; then, check again and see the differences ☺

For example,

For Primary Key	For other fields (unique)	For other fields
Select * from Ta order by aid	Select * from Ta order by a2	Select * from Ta order by price
Select * From Ta where aid>2	Select * From Ta where a2 between 7 and 10	Select * From Ta where price in (10, 20, 30)
- already a clustered index	- already an unique index	- create non-clustered index on the field(s) involved (here, price)
		- execute again the queries to check the non-clustered requirements
		- test with different fields in SELECT clause, like: 1) Primary key 2) Primary key + the field with non-clustered index 3) Primary key + a field that has no non-clustered index on it 4) The field with non-clustered index 5) The field with non-clustered index + a field with no non-clustered index created on it 6) The field with non-clustered index + the primary key + a field with non-clustered index created on it … Extract some conclusions ☺

b. Write a query on table Tb with a WHERE clause of the form *WHERE b2 = value* and analyze its execution plan. Create a nonclustered index that can speed up the query. Recheck the query's execution plan (operators, SELECT's *estimated subtree cost*).

Here, you have to create a query with a where clause, in which is involved then field b2; create a non-clustered index on the field b2; execute again the query and extract the conclusions. Pay attention to the field(s) involved in the Select clause.

```
Select *
From Tb
Where b2=8
```

```
-- create non-clustered index on the field b2
IF EXISTS (SELECT NAME FROM sys.indexes WHERE name='N_idx_Tb_b2)
DROP INDEX N_idx_Tb_b2 ON Tb
CREATE NONCLUSTERED INDEX N_idx_Tb_b2 ON Tb(b2)
```

```
-- execute the same query and extract conclusions
Select *
From Tb
Where b2=8
** please try to replace * from Select with different fields and understand what happens
```

c. Create a view that joins at least 2 tables. Check whether existing indexes are helpful; if not, reassess existing indexes / examine the cardinality of the tables.

You can create a view on the tables Ta and Tc, Tb and Tc, or Ta, Tc, and Tb. It is up to you.
Execute the view to see the used indexes (`SELECT * FROM view_name`)
Then, create index(es) on the foreign key(s) involved in the Join('s).
Execute, again, the view to see the used indexes (`SELECT * FROM view_name`), and extract conclusions.

```
CREATE VIEW v1
AS
SELECT *
FROM Ta INNER JOIN Tc ON Ta.aid=Tc.aid
-- WHERE …
GO
```

```
CREATE VIEW v2
AS
SELECT *
FROM Tb INNER JOIN Tc
ON Tb.bid=Tc.bid
-- WHERE …
GO
```

```
CREATE VIEW v3
AS
SELECT *
FROM Ta INNER JOIN Tc ON Ta.aid=Tc.aid
INNER JOIN Tb ON Tb.bid=Tc.bid
-- WHERE …
GO
```

```
-- Instead of * in Select clause, you can take only some of the fields (but it is up to you)
```

```
-- execution
SELECT * FROM v1
-- or
SELECT * FROM v1
ORDER BY …
```

```
-- execution
SELECT * FROM v2
-- or
SELECT * FROM v2
ORDER BY …
```

```
-- execution
SELECT * FROM v3
-- or
SELECT * FROM v3
ORDER BY …
```

```
-- create non-clustered indexes on the fields involved in the JOIN's (the foreign keys)
-- Tc.aid, Tc.bid
```

```
-- Re-execute, the Select's from the view and extract conclusions.
-- for example, SELECT * FROM v1
```

* Note:

For checking and see that the speed of the query is increased or that the indexes are helpful, you have to follow the steps:

1. Check - Include Live Query Statistics
2. Execute the query (pay attention to the fields used in Select, Order By, Where, Join's), without having non-clustered indexes on the fields involved in Select, Order By, Where, Join's
3. Create non-clustered indexes on the fields involved in Select, Order By, Where, Join's
4. Re-execute the queries from 2.
5. Extract conclusions related to the performance.

If the performance is ok, then finish ☺
If the performance is not helping you, please try to add records in the tables involved, and re-execute the queries from 2 and extract the conclusions (steps 4. and 5.)