Database Systems - Project 2

825008857 Chiawei Chang

1. Overview

In this TinySQL DBMS program, all required features are fully supported including but not limit to insertion, deletion, selection, projection, sorting, duplication removal and join, with unlimited size of tuples in any relations. Not only aim to pass the testing script, I also implemented some user friendly method such as the draw the table while showing information on console and user can simply type top arrow to see previous query. All of the operations, especially selection is put down in logical query plan and join is applied by efficient algorithm, are optimized to reduce the overall disk I/Os.

The system interprets the input SQL query and executes the query using provided StorageManager library as an abstraction for Disk and Main memory. The StorageManager library helps simulating the Disk I/Os and execution times. Moreover, my program also provides decent error detections: every invalid query will be stopped immediately and report the error log after that, with the location of potential problems being pointed out clearly.

In addition, all features are designed to be as similar to MySQL as possible, such as table display, commands, and interactive user interface with autocomplete and history support. This program is also flexible enough to further support other queries which is now not yet defined in current TinySQL grammar. This report is organized as follows the section 2 will provide the information of setup and usage which will also record in README file; section 3 will explicate the structure of this program; the details of functionality and performance of every operation will be discussed in section 4; the optimization details will be addressed in section 5, and finally some conclusion and personal experience will be provided in section 6.

2. README

2.1 Build & execute TinySQL

This section provides instructions to build and execute TinySQL, as well as folder structure and development environment. First, since I used the library of GUN Readline to implement the very first step of reading script in the file in, you will have to install this library if you don't have it by the command following below.

```
sudo apt-get install libreadline-dev
```

Type "make" under the project folder "tinysql-master_2/" to build the project. The executable binary file "tinysql" will be in "/bin" folder after compilation.

```
Shell> cd tinysql-master_2
Shell> make
```

To execute TinySQL in you can either choose interactive mode by typing "./bin/tinysql " in folder or load script that you want to test by "./bin/tinysql $FILE". The prompt "tinysql>" indicates that you can start using TinySQL now for interative mode. If you want to stop TinySQL, use command "quit" to exit.

All the results will be display on screen by standard output. If you want to dump the results to a file, please redirect the output.

```
Shell> ./bin/tinysql $FILE > $OUT_FILE
```

If you want the results to be both on screen and in a file, please use linux pipe and tee

```
Shell> ./bin/tinysql $FILE | tee $OUT_FILE
```

To run multiple commands without leaving, use the “source” command that I design to load file in interactive mode in interactive mode.

```
Shell> ./bin/tinysql
Tinysql> source $FILE
```

```
[mfchang➜~/Downloads» cd tinysql-master_2                                [20:01:12]
[mfchang➜~/Downloads/tinysql-master_2» cd bin                            [20:01:16]
[mfchang➜Downloads/tinysql-master_2/bin» ls                              [20:01:29]
tinysql
[mfchang➜Downloads/tinysql-master_2/bin» rm tinysql                      [20:01:30]
[mfchang➜Downloads/tinysql-master_2/bin» ls                              [20:01:33]
[mfchang➜Downloads/tinysql-master_2/bin» cd ..                           [20:01:33]
[mfchang➜~/Downloads/tinysql-master_2» make                              [20:01:35]
g++ -o bin/tinysql obj/main.o obj/cmd.o obj/dbMgr.o obj/query.o obj/test.o obj/o
bj_util.o obj/wrapper.o obj/tiny_util.o obj/y.tab.o obj/lex.yy.o obj/parser.o db
_sim_api/StorageManager.o -gdwarf-2 -g3 -Wall -Iinclude -Idb_sim_api -ly -ll -lr
eadline
[mfchang➜~/Downloads/tinysql-master_2» ./bin/tinysql                     [20:01:37]
[tinysql> quit
Bye

mfchang➜~/Downloads/tinysql-master_2»                                    [20:01:49]
```

```
[tinysql> quit
Bye

[mfchang➜~/Downloads/tinysql-master_2» ./bin/tinysql TinySQL_linux_updated.txt
TinySQL_linux_updated.txt:1> CREATE TABLE course (sid INT, homework INT, project
 INT, exam INT, grade STR20)
Query OK (0 disk I/O, 0 ms)

TinySQL_linux_updated.txt:2> INSERT INTO course (sid, homework, project, exam, g
rade) VALUES (1, 99, 100, 100, "A")
Query OK (1 disk I/O, 74.63 ms)

TinySQL_linux_updated.txt:3> SELECT * FROM course
+-----+----------+---------+------+-------+
| sid | homework | project | exam | grade |
+-----+----------+---------+------+-------+
|   1 |       99 |     100 |  100 | A     |
+-----+----------+---------+------+-------+
1 row in set (1 disk I/O, 75.26 ms)

TinySQL_linux_updated.txt:4> INSERT INTO course (sid, homework, project, exam, g
rade) VALUES (2, NULL, 100, 100, "E")
Warning: does not support 'NULL' now, set to default value !!
Query OK (1 disk I/O, 74.89 ms)
```

2.2 Project Structure

The whole file that I turned in is including the following file and directory

- makefile.
- bin/: executable.
- db_sim_api/: StorageManager library.
- dep/: dependency files generated by gcc.
- include/: some additional header files.
- obj/: object files generated by gcc.
- results/: results of automatic test by “make test”.
- src/: source codes.
- testcases/: some testcases I made to test my code. Note that those with suffix ∗.in will be executed by typing “make test”, and the results will be compared with ∗.out.

2.3 Development Environment

This program was developed under MacOS X Sierra 10.12.1 with c++11 and c99 that means also support UNIX-like environment such as Linux.

- compiler: gcc 4.8.1, g++ version 4.2.1
- lex: 2.5.35
- yacc parser: GNU Bison 2.5
- make: GNU Make 3.81
- libraries: The GNU Readline Library

3. Program Architecture

The followings are basic architecture of the TinySQL program:

3.1 Command Line Interface

TinySQL supports two kinds of user interface mode: interactive mode and batch mode. I use GNU readline library to support history and autocomplete in the interactive mode. In batch mode, either by specifying the file or by “source” command, queries will be executed consecutively, and the prompt will become the file name and line number of the current command. The commands other than TinySQL syntax, such as “source” and “quit”, are implemented here.

File: cmd.cpp, cmd.h 3.2 TinySQL Parser

3.2 Query Processer

I used lex/yacc to parse the TinySQL commands and build the parsing tree. The structure of the parsing tree “**struct** tree_node” is as follows:

```
struct tree_node
{
    int type;
    const char* value;
    struct tree_node* next;
    struct tree_node* child;
};
```

File: parser.c, parser.h, sql.l, sql.y

3.3 Query Processer

The main functionalities of TinySQL are implemented in here. All the queries which has been transformed to parsing tree will be processed by routines in “**class** QueryMgr”. Logic query plans will be built for the

"SELECT" queries: it's also a tree-like structure but with additional information. This process was implanted in "**class** QueryNode". After being constructed, the logic query plan will then be optimized and transformed to another logic query plan with estimated fewer overall disk I/Os. The physical query plan will reuse the same tree structure, carrying out the query by choosing desirable functions.

Another tree structure "**class** ConditionNode" will be built under "WHERE" conditions and used for filtering out tuples. After construction, the "**class** ConditionMgr" can take a tuple as the parameter, extract corresponding data from it, and then execute the "**class** ConditionNode". The result "**true**" mean this tuple fit the "WHERE" condition; "**false**", otherwise.

File: query.cpp, query.h

3.4 Database Interface

Most of the utilities for accessing StorageManager library are in "**class** hwMgr". This class is in charge of initiating MainMemory, Disk and SchemaManager, as well as managing and allocating their resources.

Other classes such as "**class** TinyRelation" and "**class** TinyTuple" are wrappers that combine low level operations into high level operations. They work with"**class** RelScanner ", "**class** RelWriter", and **class** RelSorter to read/write or sort relations.

class RelScanner: This class is the main routine to scan tables. It will take a relation and the currently available memory spaces for initialization; the range to scan can be further specified. After initialization, users can use RelScanner::get_next() function to iterate through the table. The RelScanner will keep iterators to the current positions of both disk and memory, and automatically load blocks from disk to memory while it running out of tuples in memory.

class RelSorter: This class is the main routine for sorting relations. It also takes a relation and the currently available memory spaces for initialization, and the order by which tuples to be sorted. and then it will be sorted by the relation automatically by applying n-pass multiway merge, while the pass number depends on the size of the relation. Note that RelSorter will create temporary relations to store the partial sorted tuples.

class RelWriter: This class will load the last block in a relation to the memory, append the new tuples to it, and then write it back to the disk.

File: dbMgr.cpp, dbMgr.h, wrapper.cpp, wrapper.h

3.5 Utilities

These files contain some general utilities, such as those for error message reporting, table drawing, string tokenizing, and debugging.

File: obj_util.cpp, obj_util.h, tiny_util.cpp, tiny_util.h, debug.h, util.h

4. Command and Query

This section will discuss the implementation detail of queries, along with their performance evaluations and examples.

4.1 Create Table

This command defines the schema of a relation. It will report error if a relation with the same name already exists in the database. There is no disk I/O required.

Example: Create a table "example" with 2 attributes, "attribute0" and "attribue1".

```
[mfchang➜~/Downloads/tinysql-master_2» ./bin/tinysql                    [20:04:36]
[tinysql> CREATE TABLE example (attribute0 INT, attribute1 STR20)
Query OK (0 disk I/O, 0 ms)

[tinysql> show tables
+-------------------+
| Tables_in_tinysql |
+-------------------+
| example           |
+-------------------+
1 row in set (0 disk I/O), 0 ms)

tinysql>
```

4.1 Drop Table

This command will delete an existing table. It will report errors if the specified table does not exist in the database. There is no disk I/O required.

Example: Delete the table "example".

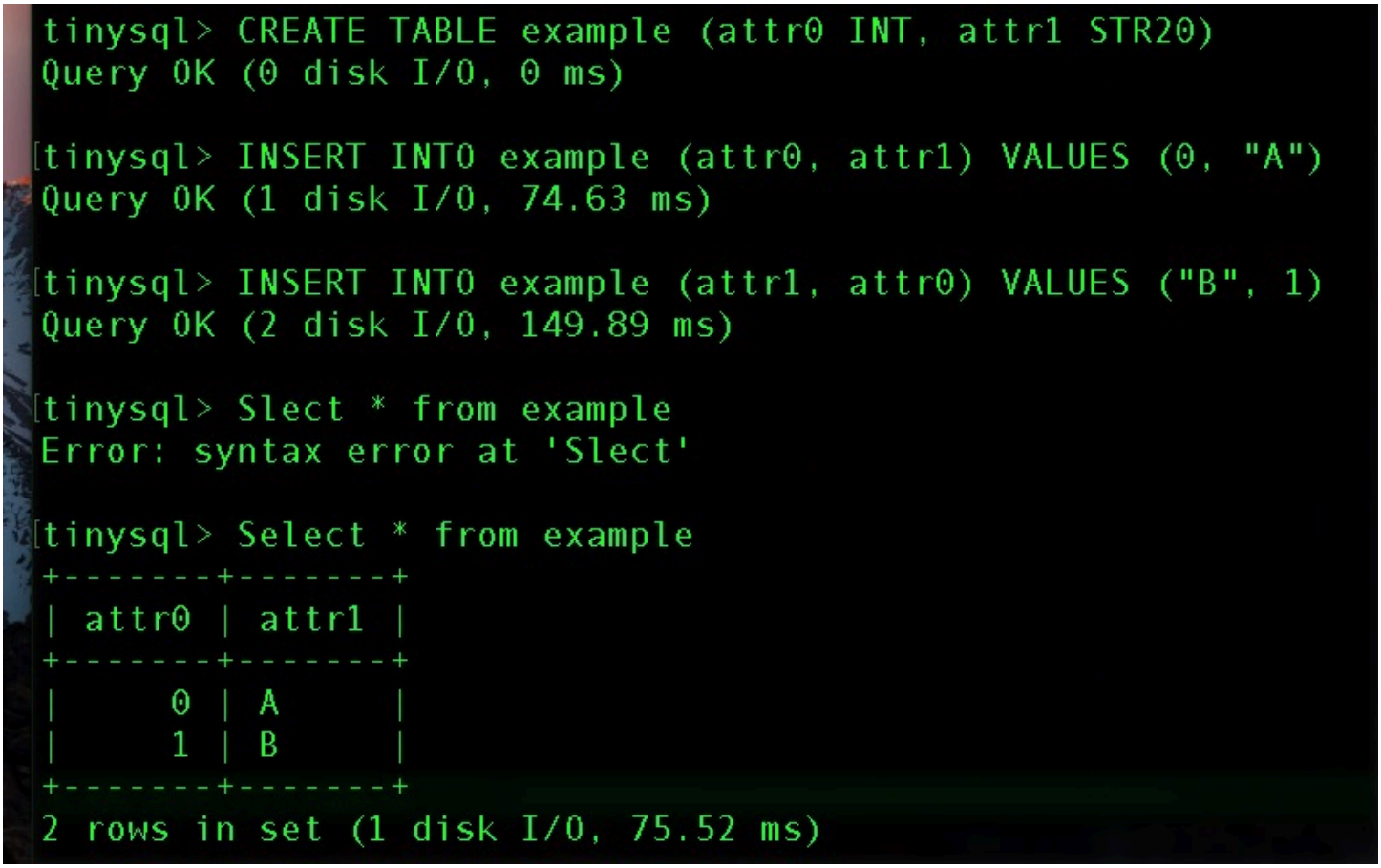
```
tinysql> CREATE TABLE example2( att INT)
Query OK (0 disk I/O, 0 ms)

tinysql> show tables
+--------------------+
| Tables_in_tinysql  |
+--------------------+
| example            |
| example2           |
+--------------------+
2 rows in set (0 disk I/O), 0 ms)

tinysql> DROP TABLE example
Query OK (0 disk I/O, 0 ms)

tinysql> show tables
+--------------------+
| Tables_in_tinysql  |
+--------------------+
| example2           |
+--------------------+
1 row in set (0 disk I/O), 0 ms)

tinysql>
```

4.3 INSERT INTO

The INSERT INTO command adds a new tuple into an existing relation. The number of disk I/O varies based on the current size of the relation: if the last block of the relation is not full, then TinySQL will load it to the memory first, and then write it back after appending the new tuple. It will cost total 2 disk I/Os. On the other hand, if the last block is empty, then TinySQL will overwrite it directly, with only 1 disk I/O.

In current implementation, TinySQL always appends the new tuple to the back of the relation, even if there might be some "hole" where the original tuple was deleted. Note that TinySQL doesn't support *NULL*, due to the limitation of StoreManager library (It uses *int* instead of *int** to storage integer value). Every *NULL* value, including those caused by attributes not being specified in the query, will be set to default value (0 for *INT* and empty string for *STR20*).

Example: Insert tuples into table "example".



4.4 DELETE FROM

This command deletes tuples from relation. It will load, from disk to memory, all non- empty blocks in the relation, and then delete every tuple which matches the WHERE condition. If any tuple is deleted in a block, then the block will be written back to disk, otherwise the block in memory will be simply discarded. The total number of disk I/Os for a relation R would be B(R) + number of blocks modified.

Example: Delete tuples from table "example".

```
tinysql> DELETE FROM example WHERE attr1="A"
Query OK (2 disk I/O, 149.78 ms)

tinysql> Select * from example
+-------+-------+
| attr0 | attr1 |
+-------+-------+
|     1 | B     |
+-------+-------+
1 row in set (1 disk I/O, 75.41 ms)
```

4.5 SELECT FROM

The SELECT command prints out tuples which comply with specified requirements. Before generating any outputs, TinySQL will first construct a logic query plan and then optimize it. A physical query plan will reuse the same tree structure, and calculate desired results with different algorithms. The functionalities and performances of different operations would be discussed in the following sections.

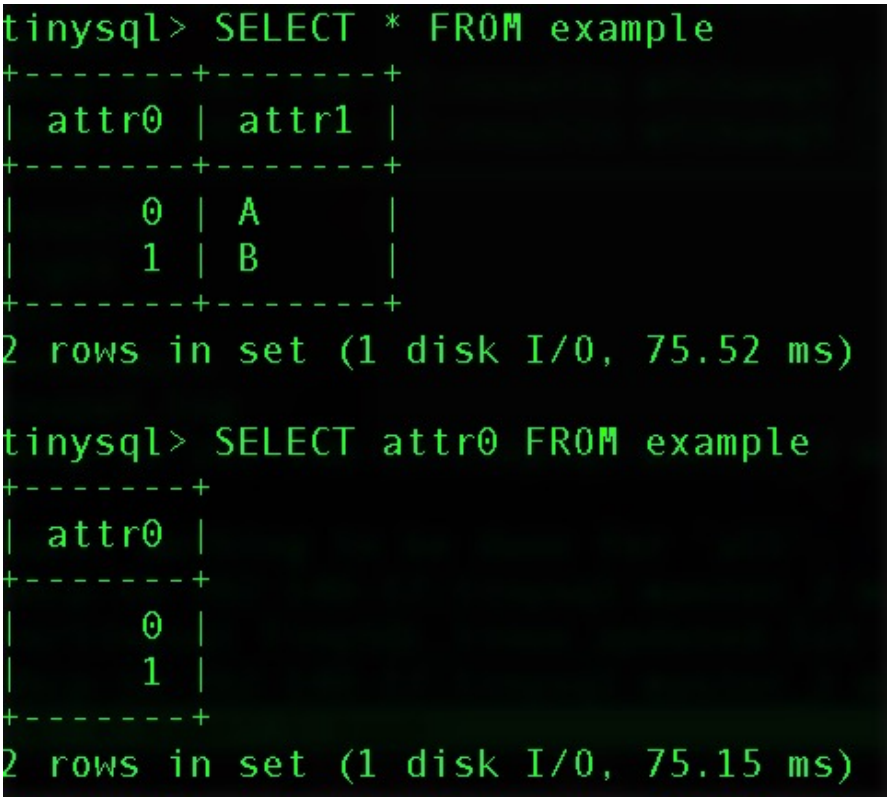
Example: A query with its corresponding logic query plan

```
SELECT example.attr0, example.attr1, example2.attr0
FROM example, example2
WHERE example.attr0 < example2.attr0
ORDER BY example.att

root:
 `- PROJECTION: {example.attr0, example.attr1, example2.attr0}
     `- ORDER_BY: example.attr0
         `- WHERE
             |- WHERE_OPTION
             |    `- COMP_OP: <
             |         |- COLUMN_NAME: example.attr0
             |         `- COLUMN_NAME: example2.attr0
             `- CROSS_PRODUCT
                 |- BASE_NODE: example
                 `- BASE_NODE: example2
```

4.5.1 Projection (SELECT)

This operation simply scans the table and selects specified columns. It takes B(R) disk I/Os to scan the table; However, if the results need to be materialized, it will take additional B(R) disk I/Os to write the results into the temporary table. Thus, a simple projection query will take as many I/Os as the amount of tuple in that table, such like 200 tuples relation will need 200 I/Os if you apply select * from that relation.

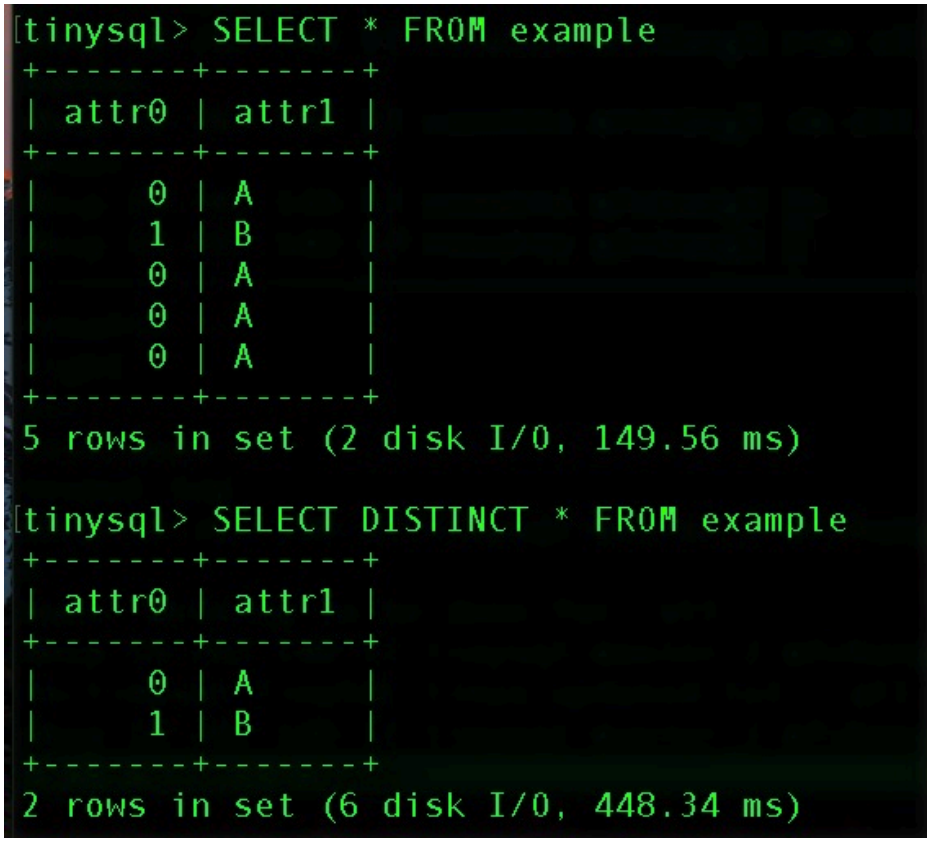
Example: Project column "attr0".



4.5.2 Duplication Removal (DISTINCT)

This operation removes duplicated tuples. TinySQL uses sort-based algorithm: first sort the relation if it has not yet been sorted, and then scan the relation and output only the tuple which is not the same as the next tuple. Therefore, it will take B(R) disk I/Os with a potential sorting to complete its job.

Example: Remove duplication tuples



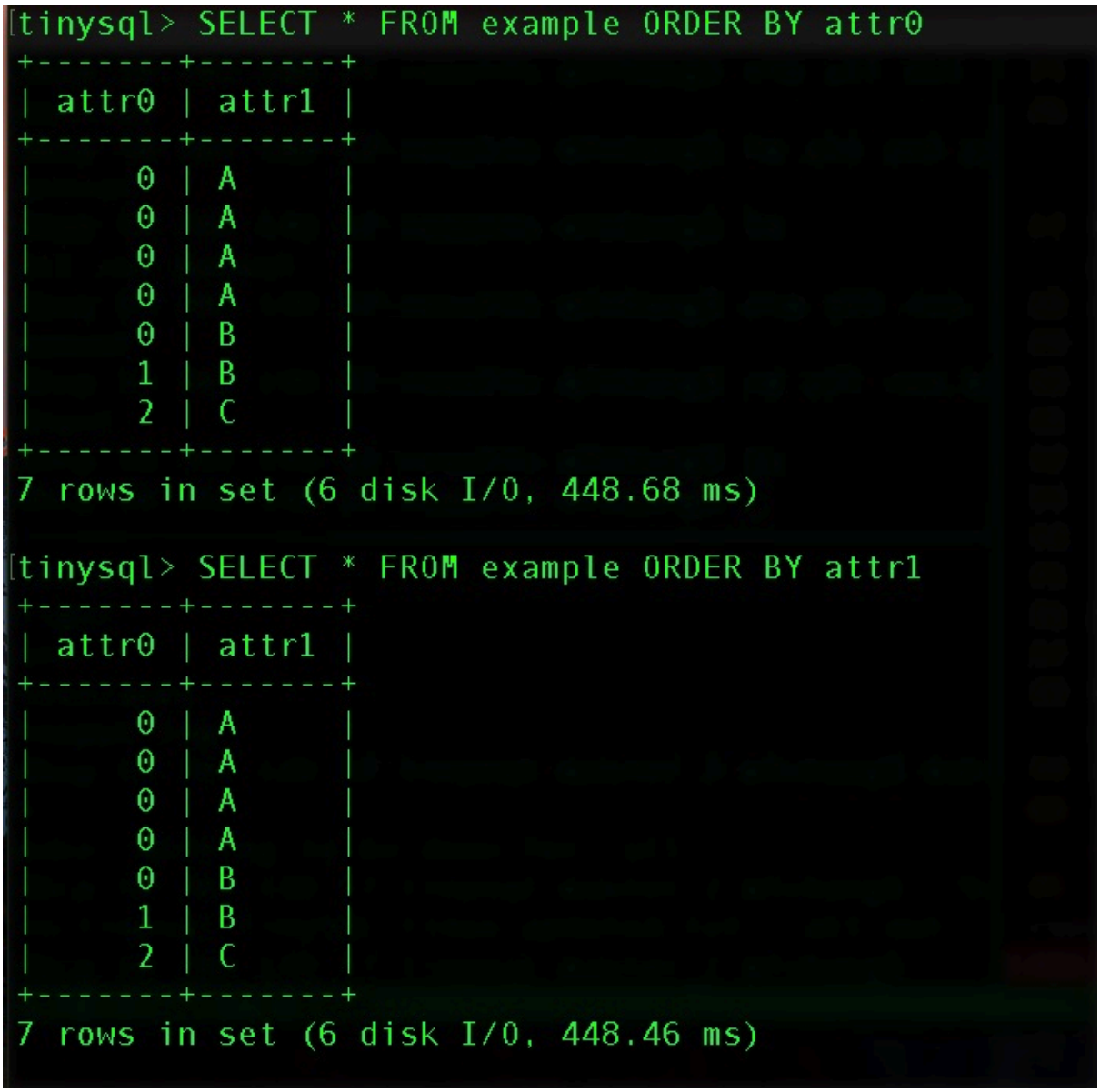
4.5.3 Sorting (ORDER BY)

This operation uses multiway merge sort. It could be 1-pass, 2-pass, or n-pass, which is based on the size of the relation. If it takes more than 1 pass to complete the sort,
TinySQL will create temporary relations to store the partially sorted tuples. It requires 2 × B(R) × n (pass) disk I/Os to sort a relation.

table size	1	2	3	4	5	6	7	8	9	10	20	30
disk I/O	3	6	9	12	15	18	21	24	27	50	100	150
table size	40	50	60	70	80	90	100	110	120			
disk I/O	200	250	300	350	400	630	700	770	840			

Table 1: disk I/Os with table size for 1 sorting + 1 scan

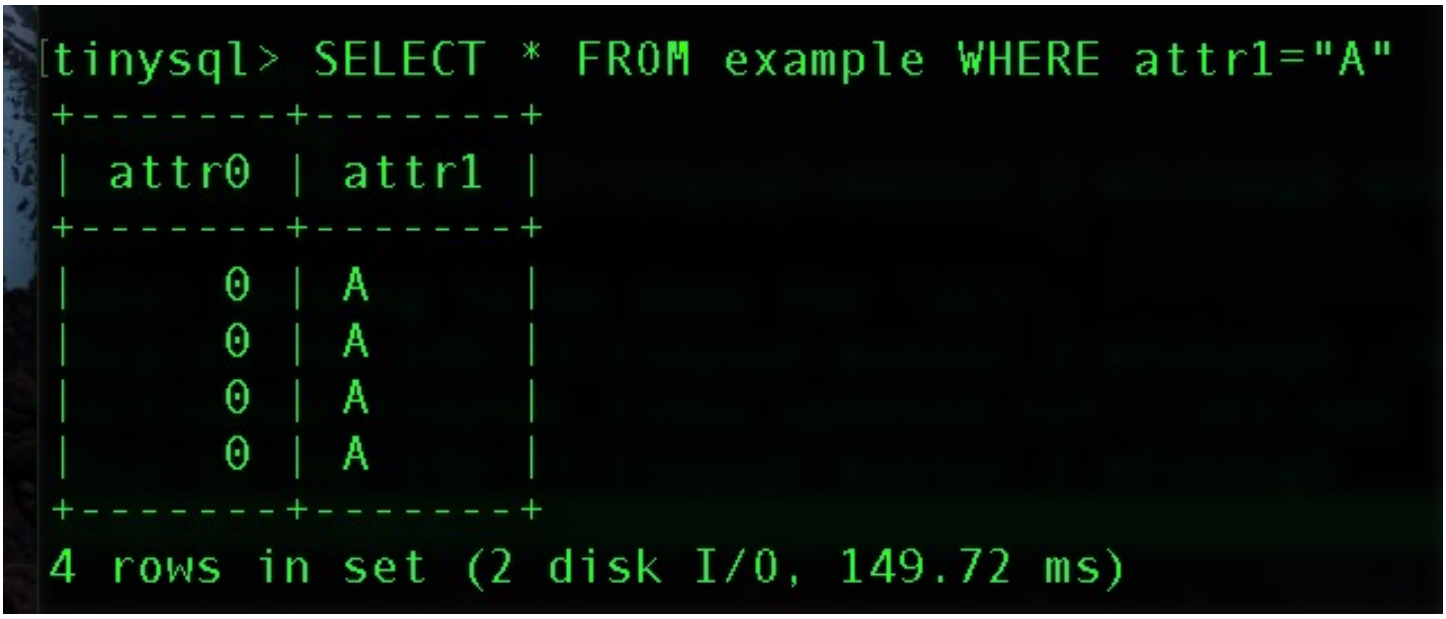
Example: Sorting



4.5.4 Selection (WHERE)

The selection operation only filters out the output of other operations, it doesn't involve disk I/O.

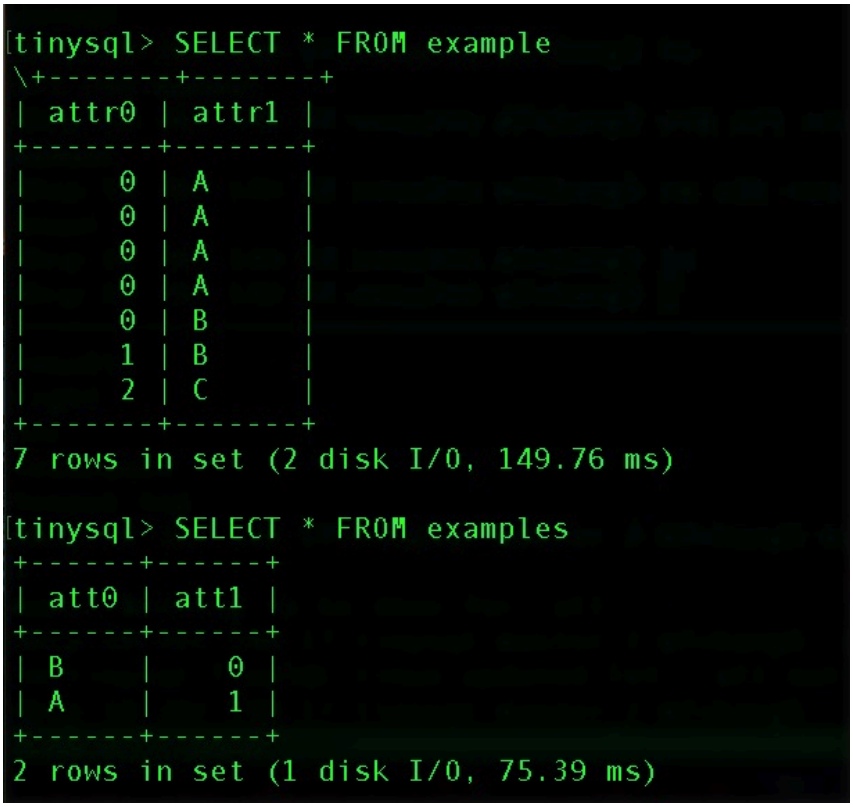
Example: Selection

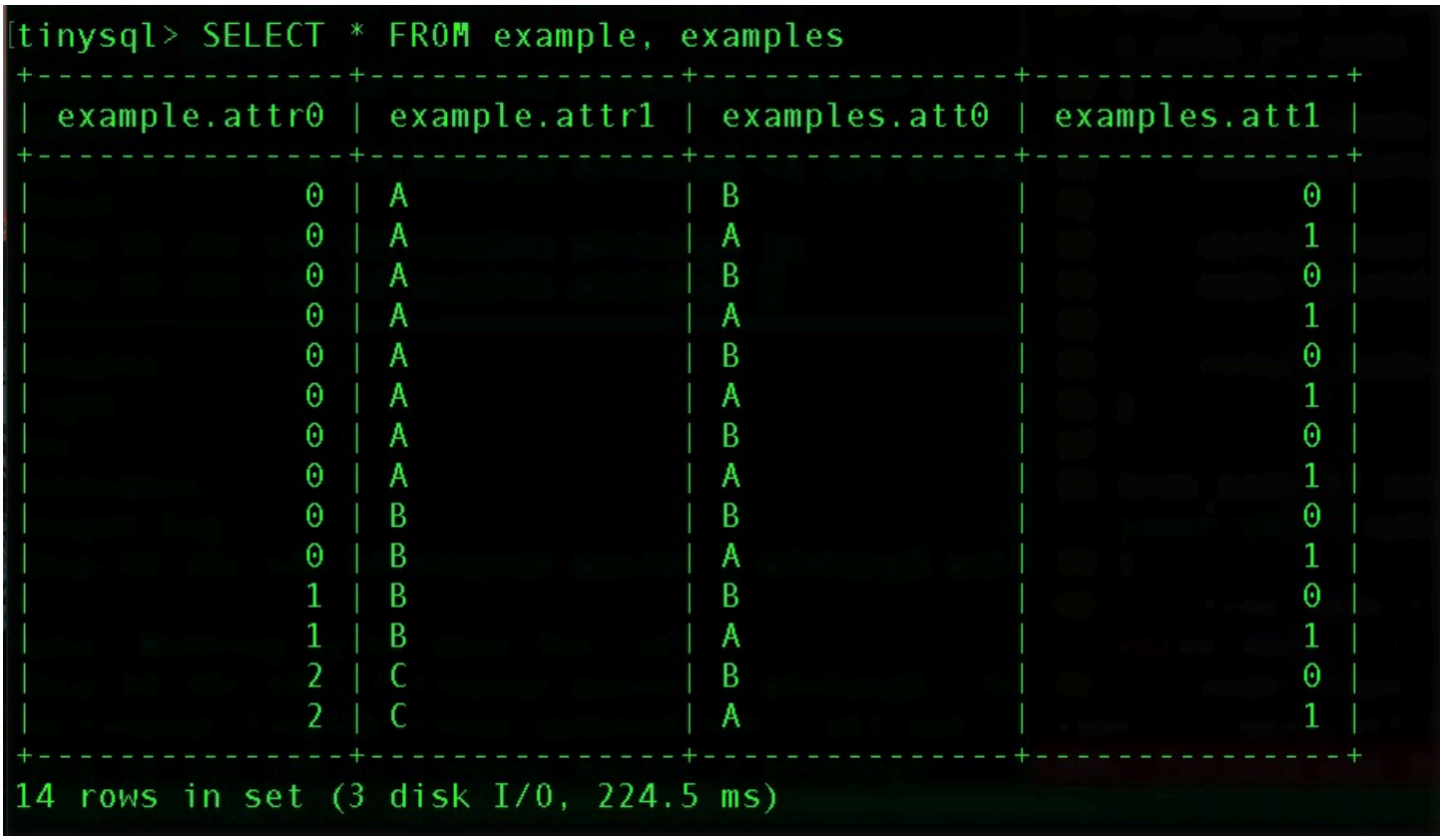


4.5.5 Cross Product (FROM)

The cross product operation combines every tuple in the relations. It takes B(R)×B(S)/M disk I/Os, where M = available memory size. The physical query plan will assign 1 memory block to the larger relation, while the rest of memory blocks to the smaller relation.

Example: Cross product



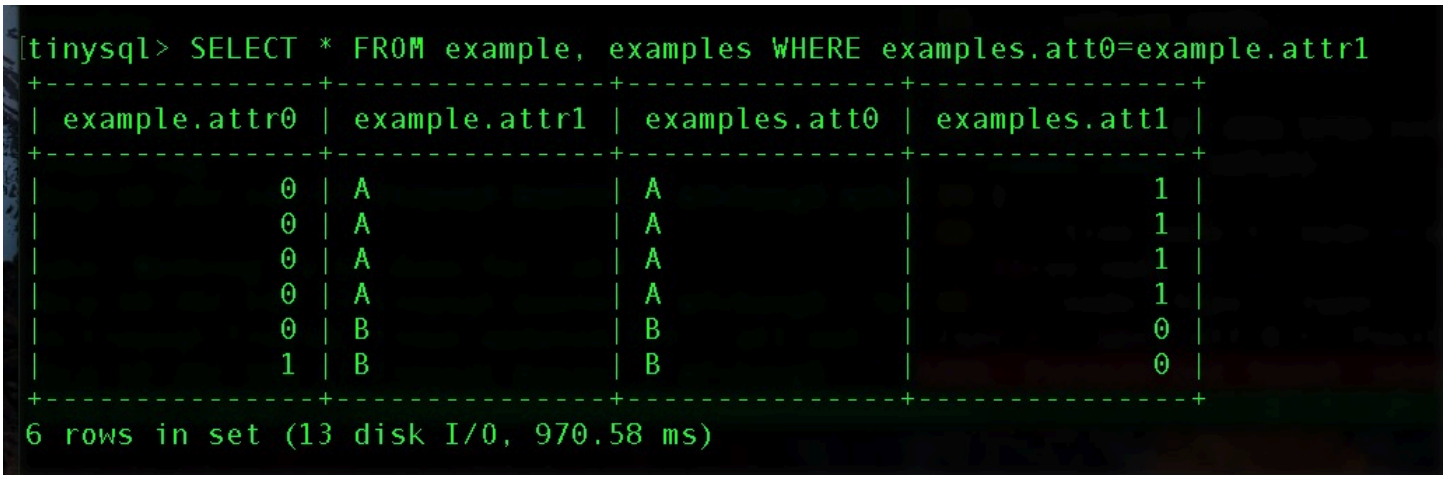


```
tinysql> SELECT * FROM example, examples
+----------------+----------------+----------------+----------------+
| example.attr0  | example.attr1  | examples.att0  | examples.att1  |
+----------------+----------------+----------------+----------------+
|              0 | A              | B              |              0 |
|              0 | A              | A              |              1 |
|              0 | A              | B              |              0 |
|              0 | A              | A              |              1 |
|              0 | A              | B              |              0 |
|              0 | A              | A              |              1 |
|              0 | A              | B              |              0 |
|              0 | A              | A              |              1 |
|              0 | B              | B              |              0 |
|              0 | B              | A              |              1 |
|              1 | B              | B              |              0 |
|              1 | B              | A              |              1 |
|              2 | C              | B              |              0 |
|              2 | C              | A              |              1 |
+----------------+----------------+----------------+----------------+
14 rows in set (3 disk I/O, 224.5 ms)
```

4.5.6 Natural Join (Theta Join)

The natural join operation only exists after optimization. A cross product with equation will be transformed to a natural join. Natural join will, based on the attributes in equation, sort relations first. And then it will scan both table and do cross product for only tuples with matched attribute. The disk I/O depends heavy on the values in the relations. However, most of the time, the time complexity will reduce from O(B(R) × B(S)) of original cross product to O(B(R) + B(S)). In the following example, the relation table of example and examples are remaining the same so I only take the screenshot of natural join outcome.

Example: Natural join (Theta join)



```
tinysql> SELECT * FROM example, examples WHERE examples.att0=example.attr1
+----------------+----------------+----------------+----------------+
| example.attr0  | example.attr1  | examples.att0  | examples.att1  |
+----------------+----------------+----------------+----------------+
|              0 | A              | A              |              1 |
|              0 | A              | A              |              1 |
|              0 | A              | A              |              1 |
|              0 | A              | A              |              1 |
|              0 | B              | B              |              0 |
|              1 | B              | B              |              0 |
+----------------+----------------+----------------+----------------+
6 rows in set (13 disk I/O, 970.58 ms)
```

5. Optimization

To speed up queries, TinySQL implements several optimizations by reducing the main cost time which is disk I/Os.

5.1 Insert

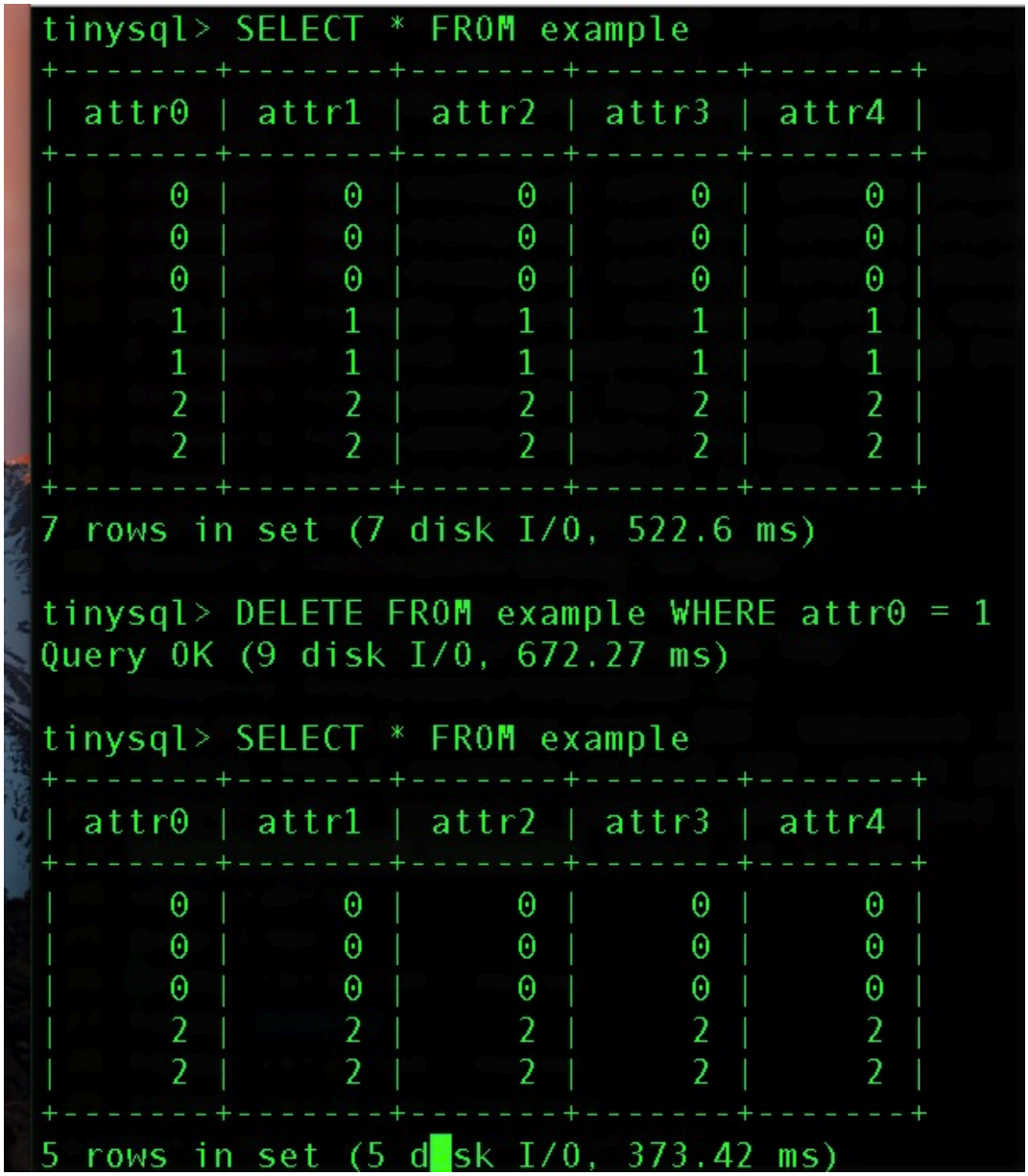
TinySQL will calculate the location for the new tuple before accessing the disk: if the block to load from disk is empty, then TinySQL will overwrite it directly, save 1 disk I/O from loading it to memory.

Example: Overwrite directly

5.2 Scan

TinySQL keep track of the “holes” caused by deletion. If all the tuples in a block are deleted, then scanner will skip it.

Example: Skip empty block



5.3 Selection

TinySQL will decompose the conditions joined by AND operation, and try to push down them to the bottom of the logic query plan. If a selection operation only involves a give table, it will carry out immediately after the scan of the table.

Example: Push down selection

```
Tinysql> SELECT * FROM example, example1 WHERE example.attr0 = 0 AND example1.attr0<1
```

```
optimized logic query plan:
 `- WHERE
     |- WHERE_OPTION
     |   `- AND
     |       |- COMP_OP: =
     |       |   |- COLUMN_NAME: example.attr0
     |       |   `- INTEGER: 0
     |       `- COMP_OP: <
     |           |- COLUMN_NAME: example1.attr0
     |           `- INTEGER: 1
     `- CROSS_PRODUCT
         |- WHERE
         |   |- COMP_OP: =
         |   |   |- COLUMN_NAME: example.attr0
         |   |   `- INTEGER: 0
         |   `- BASE_NODE: example
         `- WHERE
             |- COMP_OP: <
             |   |- COLUMN_NAME: example1.attr0
             |   `- INTEGER: 1
             `- BASE_NODE: example1
```

5.4 Join

Cross product with equation will be transformed to natural join. The disk I/O of natural join varies. In some worst case, it is possible that the overall disk I/Os increases (ex: all the tuple has the same joined attributes). However, most of the time, the time complexity will reduce from O(B(R) × B(S)) of original cross product to O(B(R) + B(S)) of natural join.

For the pure cross product with more than 3 relations, it will do the product on the relations with the smallest number of tuples first. Example: Transform cross product to natural join

Example: Transform cross product to natural join

```
Tinysql> SELECT * FROM example, example1 WHERE example.attr0 = example1.attr0
```

```
optimized logic query plan:
 `- WHERE
     |- WHERE_OPTION
     |    `- COMP_OP: =
     |         |- COLUMN_NAME: example.attr0
     |         `- COLUMN_NAME: example1.attr0
     `- NATURAL_JOIN: {example.attr0, example1.attr0}
         |- BASE_NODE: example
         `- BASE_NODE: example1
```

6. Conclusion and Experience

The I/O and execution time complexity of the system was tested by running the SELECT statement against the size of the table. The increase in the I/O and Runtime is both **LINEAR** with respect to size of the table. To implement this homework, even without the most important log and recovery system, is an extremely hard task. Especially at the beginning when we all clueless of how to start, however the StorageManagement give up a lot of hints. I personally think this project overall is very interesting and a great opportunity to learn, thanks both TA and professor both who help us to learn and been helpful whenever we encounter any problem. Last but not least, I acknowledge that I don't know how to include some of the Latex table into this document, so this report is kind of massy, However I did put all of my effort on it.