ECE 356 - Assignment 3

Zahin Mohammad - 20669584 - z5mohamm@uwaterloo.ca

List Of Tables

Part 1

For this question, we refer to instance 1 as the mysql instance that performs the update transaction, and instance 2 as the mysql instance that performs the database read transaction. From the 4x4 permutation of the different isolation levels presented in table 1, it seems that the isolation level of instance 1 does not affect the output of instance 2's database read. Instance 2's results remain the same regardless of instance 1's isolation level. From this we can say that an instance's output is dependent on that instance's isolation level, not the isolation level of other instances. The results

Instance 1 (Update)	Instance 2 (Select)	Instance 2 Result
Read-Uncommitted	Read-Uncommitted	+------------+ | Enrollment | +------------+ | 64 | | 103 | +------------+
Read-Uncommitted	Read-Committed	+------------+ | Enrollment | +------------+ | 64 | | 123 | +------------+
Read-Uncommitted	Repeatable-Read	+------------+ | Enrollment | +------------+ | 64 | | 123 | +------------+
Read-Uncommitted	Serializeable	Timeout
Read-Committed	Read-Uncommitted	+------------+ | Enrollment | +------------+ | 64 | | 103 | +------------+
Read-Committed	Read-Committed	+------------+ | Enrollment | +------------+

		| 64 | | 123 | +------------+
Read-Committed	Repeatable-Read	+------------+ | Enrollment | +------------+ | 64 | | 123 | +------------+
Read-Committed	Serializeable	Timeout
Repeatable-Read	Read-Uncommitted	+------------+ | Enrollment | +------------+ | 64 | | 103 | +------------+
Repeatable-Read	Read-Committed	+------------+ | Enrollment | +------------+ | 64 | | 123 | +------------+
Repeatable-Read	Repeatable-Read	+------------+ | Enrollment | +------------+ | 64 | | 123 | +------------+
Repeatable-Read	Serializeable	Timeout
Serializeable	Read-Uncommitted	+------------+ | Enrollment | +------------+ | 64 | | 103 | +------------+
Serializeable	Read-Committed	+------------+ | Enrollment | +------------+ | 64 | | 123 |

		+------------+
Serializeable	Repeatable-Read	+------------+ | Enrollment | +------------+ | 64 | | 123 | +------------+
Serializeable	Serializeable	Timeout

Table 1. Summary of outputs observed with changing isolation levels.

```
SET autocommit = 0;

SET SESSION TRANSACTION ISOLATION LEVEL [Level];

BEGIN;
update Offering set Enrollment = Enrollment - 20 where courseID="ECE356" and
section=2 and termCode=1191;
update Offering set Enrollment = Enrollment + 20 where courseID="ECE356" and
section=2 and termCode=1191;
COMMIT;
```

Table 2. Source code for instance 1.

```
SET autocommit = 0;

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN;select Enrollment from Offering where courseID='ECE356';COMMIT;
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN;select Enrollment from Offering where courseID='ECE356';COMMIT;
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN;select Enrollment from Offering where courseID='ECE356';COMMIT;
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN;select Enrollment from Offering where courseID='ECE356';COMMIT;
```

Table 3. Source code for instance 2.

Part 2

The sql for the stored procedure is provided in table 4. The test suite for this stored procedure is provided in table 5.
It is not possible to get equivalent functionality as the stored procedure from checks and triggers. This is because checks and triggers are independent of each other.

An example of bad state that can be caused by using checks and triggers opposed to a stored procedure:

- Current state
    - Offering ('ECE356' , 1 , 1191 , 'E74417' , 1 , 64)
    - Offering ('ECE356' , 2 , 1191 , 'E74417' , 3 , 123)
    - Classroom ('E74417', 'E7', 4417, 1000)
- Try and remove 70 people from section 1, this should fail the check/trigger as 64-70 < 0
    - Offering ('ECE356' , 1 , 1191 , 'E74417' , 1 , 64)
    - Offering ('ECE356' , 2 , 1191 , 'E74417' , 3 , 123)
    - Classroom ('E74417', 'E7', 4417, 1000)
- Try and add 70 people to section 2, this passes the check/trigger as 64+70 < 1000
    - Offering ('ECE356' , 1 , 1191 , 'E74417' , 1 , 64)
    - Offering ('ECE356' , 2 , 1191 , 'E74417' , 3 , 193)
    - Classroom ('E74417', 'E7', 4417, 1000)

Since the checks/triggers are independant, we cannot stop an update after a previous check/trigger has failed. This is why we need a stored procedure.

```
DELIMITER //

DROP PROCEDURE IF EXISTS switchSection;
CREATE PROCEDURE switchSection(
    IN courseID CHAR(8),
    IN section1 int,
    IN section2 int,
    IN termCode decimal(4),
    IN quantity int,
    OUT errorCode int
)
proc_label:BEGIN

    set errorCode = 0;

    START TRANSACTION;
        -- if (courseID,section1,termCode) or (courseID,section2,termCode)
do not exist in Offering,
        -- or quantity is 0 or less or section1 = section2, set the error
code to -1.
        IF (
            SELECT count(*)
            FROM Offering
            WHERE courseID = Offering.courseID
```

```
        AND section1 = Offering.section
        AND termCode = Offering.termCode) = 0
        THEN
                set errorCode = -1;
                ROLLBACK;
                LEAVE proc_label;
    END IF;

    IF (
        SELECT count(*)
        FROM Offering
        WHERE courseID = Offering.courseID
        AND section2 = Offering.section
        AND termCode = Offering.termCode) = 0
        THEN
                set errorCode = -1;
                ROLLBACK;
                LEAVE proc_label;
    END IF;

    -- attempt to reduce the enrollment in section1 by "quantity";
    UPDATE Offering
    SET enrollment =  enrollment - quantity
    WHERE courseID = Offering.courseID
    AND termCode = Offering.termCode
    AND section1 = Offering.section;

    -- if the result is a negative enrollment, set the error code to -2
    IF (
        SELECT enrollment
        FROM Offering
        WHERE courseID = Offering.courseID
        AND termCode = Offering.termCode
        AND section1 = Offering.section) < 0
        THEN
                set errorCode = -2;
                ROLLBACK;
                LEAVE proc_label;
    END IF;

    -- attempt to increase the enrollment in section2 by "quantity";
    UPDATE Offering
```

```
        SET enrollment =  enrollment + quantity
        WHERE courseID = Offering.courseID
        AND termCode = Offering.termCode
        AND section2 = Offering.section;

        -- if the result is that enrollment in section2 exceeds room
capacity, then set the error code to -3.
        IF (
            SELECT capacity
            FROM Offering JOIN Classroom
            ON Offering.roomID = Classroom.roomID
            WHERE courseID = Offering.courseID
            AND termCode = Offering.termCode
            AND section2 = Offering.section
            ) <
            (
            SELECT enrollment
            FROM Offering JOIN Classroom
            ON Offering.roomID = Classroom.roomID
            WHERE courseID = Offering.courseID
            AND termCode = Offering.termCode
            AND section2 = Offering.section
            )
            THEN
                    set errorCode = -3;
                    ROLLBACK;
                    LEAVE proc_label;
        END IF;
    COMMIT;

END//

DELIMITER ;
```

Table 4. Stored procedure switchSection.

```
DROP DATABASE IF EXISTS unitest;
CREATE DATABASE unitest;
use unitest;
source createUni.sql;
set autocommit=0;
source q2.sql;
-- Should pass
call switchSection('ECE356', 1, 2, 1191, 0, @errorCode);
```

```
SELECT CONCAT('EXPECTED 0 GOT ', @errorCode) AS 'result';

DROP DATABASE IF EXISTS unitest;
CREATE DATABASE unitest;
use unitest;
source createUni.sql;
set autocommit=0;
source q2.sql;
-- should pass
call switchSection('ECE356', 1, 2, 1191, 14, @errorCode);
SELECT CONCAT('EXPECTED 0 GOT ', @errorCode) AS 'result';

DROP DATABASE IF EXISTS unitest;
CREATE DATABASE unitest;
use unitest;
source createUni.sql;
set autocommit=0;
source q2.sql;
-- should fail
call switchSection('ECE356', 1, 2, 1191, 20, @errorCode);
SELECT CONCAT('EXPECTED -3 GOT ', @errorCode) AS 'result';

DROP DATABASE IF EXISTS unitest;
CREATE DATABASE unitest;
use unitest;
source createUni.sql;
set autocommit=0;
source q2.sql;
-- should fail
call switchSection('ECE356', 1, 2, 1191, 100, @errorCode);
SELECT CONCAT('EXPECTED -2 GOT ', @errorCode) AS 'result';

set autocommit=1;
```

Table 5. Test for stored procedure switchSection.

Part 3

Using the example queries shown in tables 13-17, average timings were recorded after changing the keys on tables.
It can be seen that the greatest performance improvement can be seen with Primary Key, Foreign Keys and Indexes all enabled, and the worst performance was observed when all these keys were turned off or removed.

Interestingly it can be seen that Foreign keys provide marginal performance improvement when we look at the case of Primary Key on, Foreign Key on, Index on vs Primary key on, Foreign key off, and Index on. All these results are summarized in tables 6-8.

Average Tlme in mS	PK: ON FK: ON IND: ON	PK: ON FK: ON IND: OFF	PK: ON FK: OFF IND: ON	PK: ON FK: OFF IND: OFF	PK:OFF FK: OFF IND: OFF
Query 1	5837.7248	9634.21020000	5864.4370	7802.2104	17251.97480
Query 2	132840.5008	157434.7098	135214.6006	135979.1682	76949746.87100
Query 3	4357.3764	7996.3006	5123.2024	6987.5210	9742.52280
Query 4	132869.5422	180253.7926	129635.9020	127548.8456	72416305.83740000
Query 5	165408.6374	178687.5120	170509.2602	163197.7116	137500.66940000

Table 6. Average time per query.

Minimum Tlme in mS	PK: ON FK: ON IND: ON	PK: ON FK: ON IND: OFF	PK: ON FK: OFF IND: ON	PK: ON FK: OFF IND: OFF	PK:OFF FK: OFF IND: OFF
Query 1	3888.9400	7350.6160	4041.3180	6418.2580	12801.0580
Query 2	130588.3830	142179.4510	131798.0490	129679.0120	72252366.2330
Query 3	3865.5630	6750.6140	3634.6020	6440.6150	8967.3420
Query 4	127363.5520	127472.2020	125534.5060	125903.9000	71536399.8190
Query 5	164000.1590	167207.6690	165014.5090	147579.5810	134340.1550

Table 7. Minimum time per query

Maximum Tlme in mS	PK: ON FK: ON IND: ON	PK: ON FK: ON IND: OFF	PK: ON FK: OFF IND: ON	PK: ON FK: OFF IND: OFF	PK:OFF FK: OFF IND: OFF
Query 1	8459.5880	12200.1410	7569.2080	9368.9760	22298.1850
Query 2	138985.8450	202323.9340	138894.3070	139568.0170	88240290.8620
Query 3	5211.4210	8786.9650	7360.8280	7264.0210	10724.1090
Query 4	143089.1090	280593.2990	134201.0880	129632.7520	73066133.2820
Query 5	168839.3960	192524.9700	176335.5830	170099.7250	145392.8270

Table 8. Maximum time per query.

```
alter table baseball2016.Teams DROP FOREIGN KEY Teams_ibfk_1;
alter table baseball2016.AllstarFull DROP FOREIGN KEY AllstarFull_ibfk_1;
alter table baseball2016.AllstarFull DROP FOREIGN KEY AllstarFull_ibfk_2;
alter table baseball2016.Appearances DROP FOREIGN KEY Appearances_ibfk_1;
alter table baseball2016.Appearances DROP FOREIGN KEY Appearances_ibfk_2;
alter table baseball2016.AwardsManagers DROP FOREIGN KEY
AwardsManagers_ibfk_1;
alter table baseball2016.AwardsPlayers DROP FOREIGN KEY
AwardsPlayers_ibfk_1;
alter table baseball2016.AwardsShareManagers DROP FOREIGN KEY
AwardsShareManagers_ibfk_1;
alter table baseball2016.AwardsSharePlayers DROP FOREIGN KEY
AwardsSharePlayers_ibfk_1;
alter table baseball2016.Batting DROP FOREIGN KEY Batting_ibfk_1;
alter table baseball2016.Batting DROP FOREIGN KEY Batting_ibfk_2;
alter table baseball2016.BattingPost DROP FOREIGN KEY BattingPost_ibfk_1;
alter table baseball2016.BattingPost DROP FOREIGN KEY BattingPost_ibfk_2;
alter table baseball2016.CollegePlaying DROP FOREIGN KEY
CollegePlaying_ibfk_1;
alter table baseball2016.CollegePlaying DROP FOREIGN KEY
CollegePlaying_ibfk_2;
alter table baseball2016.Fielding DROP FOREIGN KEY Fielding_ibfk_1;
alter table baseball2016.Fielding DROP FOREIGN KEY Fielding_ibfk_2;
alter table baseball2016.FieldingOF DROP FOREIGN KEY FieldingOF_ibfk_1;
alter table baseball2016.FieldingOFsplit DROP FOREIGN KEY
FieldingOFsplit_ibfk_1;
alter table baseball2016.FieldingOFsplit DROP FOREIGN KEY
```

```
FieldingOFsplit_ibfk_2;
alter table baseball2016.FieldingPost DROP FOREIGN KEY FieldingPost_ibfk_1;
alter table baseball2016.FieldingPost DROP FOREIGN KEY FieldingPost_ibfk_2;
alter table baseball2016.HallOfFame DROP FOREIGN KEY HallOfFame_ibfk_1;
alter table baseball2016.HomeGames DROP FOREIGN KEY HomeGames_ibfk_1;
alter table baseball2016.Managers DROP FOREIGN KEY Managers_ibfk_1;
alter table baseball2016.Managers DROP FOREIGN KEY Managers_ibfk_2;
alter table baseball2016.ManagersHalf DROP FOREIGN KEY ManagersHalf_ibfk_1;
alter table baseball2016.ManagersHalf DROP FOREIGN KEY ManagersHalf_ibfk_2;
alter table baseball2016.Pitching DROP FOREIGN KEY Pitching_ibfk_1;
alter table baseball2016.Pitching DROP FOREIGN KEY Pitching_ibfk_2;
alter table baseball2016.PitchingPost DROP FOREIGN KEY PitchingPost_ibfk_1;
alter table baseball2016.PitchingPost DROP FOREIGN KEY PitchingPost_ibfk_2;
alter table baseball2016.Salaries DROP FOREIGN KEY Salaries_ibfk_1;
alter table baseball2016.Salaries DROP FOREIGN KEY Salaries_ibfk_2;
alter table baseball2016.SeriesPost DROP FOREIGN KEY SeriesPost_ibfk_1;
alter table baseball2016.SeriesPost DROP FOREIGN KEY SeriesPost_ibfk_2;
alter table baseball2016.TeamsHalf DROP FOREIGN KEY TeamsHalf_ibfk_1;
```

Table 9. Script to remove foreign keys.

```
alter table baseball2016.Master DROP PRIMARY KEY;
alter table baseball2016.TeamsFranchises DROP PRIMARY KEY;
alter table baseball2016.Schools DROP PRIMARY KEY;
alter table baseball2016.Teams DROP PRIMARY KEY;
alter table baseball2016.Parks DROP PRIMARY KEY;
alter table baseball2016.AllstarFull DROP PRIMARY KEY;
alter table baseball2016.Appearances DROP PRIMARY KEY;
alter table baseball2016.AwardsManagers DROP PRIMARY KEY;
alter table baseball2016.AwardsPlayers DROP PRIMARY KEY;
alter table baseball2016.AwardsShareManagers DROP PRIMARY KEY;
alter table baseball2016.AwardsSharePlayers DROP PRIMARY KEY;
alter table baseball2016.Batting DROP PRIMARY KEY;
alter table baseball2016.BattingPost DROP PRIMARY KEY;
alter table baseball2016.CollegePlaying DROP PRIMARY KEY;
alter table baseball2016.Fielding DROP PRIMARY KEY;
alter table baseball2016.FieldingOF DROP PRIMARY KEY;
alter table baseball2016.FieldingOFsplit DROP PRIMARY KEY;
alter table baseball2016.FieldingPost DROP PRIMARY KEY;
alter table baseball2016.HallOfFame DROP PRIMARY KEY;
alter table baseball2016.HomeGames DROP PRIMARY KEY;
alter table baseball2016.Managers DROP PRIMARY KEY;
```

```
alter table baseball2016.ManagersHalf DROP PRIMARY KEY;
alter table baseball2016.Pitching DROP PRIMARY KEY;
alter table baseball2016.PitchingPost DROP PRIMARY KEY;
alter table baseball2016.Salaries DROP PRIMARY KEY;
alter table baseball2016.SeriesPost DROP PRIMARY KEY;
alter table baseball2016.TeamsHalf DROP PRIMARY KEY;
```

Table 10. Script to foreign primary keys.

```
ALTER TABLE Master ADD INDEX (birthMonth);
ALTER TABLE Master ADD INDEX (birthYear);
ALTER TABLE Master ADD INDEX (birthDay);
ALTER TABLE Salaries ADD INDEX (yearID);
ALTER TABLE Appearances ADD INDEX (yearID);
ALTER TABLE Managers ADD INDEX (yearID);
ALTER TABLE Batting ADD INDEX (playerID);
ALTER TABLE Batting ADD INDEX (yearID);
```

Table 11. Script to add indexes.

```
CALL sys.ps_truncate_all_tables(FALSE);
source query1x5.sql;

SELECT
    AVG(TIMER_WAIT)/1000000 as 'AverageRunTime',
    min(TIMER_WAIT)/1000000 as 'SmallestRunTime',
    max(TIMER_WAIT)/1000000 as 'LargestRunTime'
FROM performance_schema.events_transactions_history;

CALL sys.ps_truncate_all_tables(FALSE);
source query2x5.sql;

SELECT
    AVG(TIMER_WAIT)/1000000 as 'AverageRunTime',
    min(TIMER_WAIT)/1000000 as 'SmallestRunTime',
    max(TIMER_WAIT)/1000000 as 'LargestRunTime'
FROM performance_schema.events_transactions_history;

CALL sys.ps_truncate_all_tables(FALSE);
source query3x5.sql;

SELECT
    AVG(TIMER_WAIT)/1000000 as 'AverageRunTime',
```

```
    min(TIMER_WAIT)/1000000 as 'SmallestRunTime',
    max(TIMER_WAIT)/1000000 as 'LargestRunTime'
FROM performance_schema.events_transactions_history;

CALL sys.ps_truncate_all_tables(FALSE);
source query4x5.sql;

SELECT
    AVG(TIMER_WAIT)/1000000 as 'AverageRunTime',
    min(TIMER_WAIT)/1000000 as 'SmallestRunTime',
    max(TIMER_WAIT)/1000000 as 'LargestRunTime'
FROM performance_schema.events_transactions_history;

CALL sys.ps_truncate_all_tables(FALSE);
source query5x5.sql;

SELECT
    AVG(TIMER_WAIT)/1000000 as 'AverageRunTime',
    min(TIMER_WAIT)/1000000 as 'SmallestRunTime',
    max(TIMER_WAIT)/1000000 as 'LargestRunTime'
FROM performance_schema.events_transactions_history;
```

Table 12. Performance testing script.

```
select count(playerID)
from Master
where birthYear is null
or birthYear = ""
or birthMonth is null
or birthMonth = ""
or birthDay is null
or birthDay = "";
```

Table 13. Query 1.

```
select
playerID,sum(salary) as totalPay
from Salaries
left outer join Appearances using (playerID,yearID,teamID)
left outer join Managers using (playerID,yearID,teamID)
where G is null
and G_all is null
group by playerID
```

```
order by totalPay desc
limit 3;
```

Table 14. Query 2.

```
select count(playerID)
from Master
where birthYear is null
or birthYear = ""
or birthMonth is null or birthMonth = ""
or birthDay is null or birthDay = "";
```

Table 15. Query 3.

```
select playerID,sum(salary) as totalPay
from Salaries
left outer join Appearances using (playerID,yearID,teamID)
left outer join Managers using (playerID,yearID,teamID)
where G is null and
G_all is null
group by playerID
order by totalPay desc
limit 3;
```

Table 16. Query 4.

```
select nameFirst,nameLast,max(RBI)
from Batting
inner join Master using (playerID)
where HR = 0
group by nameLast, nameFirst
limit 1;
```

Table 17. Query 5.

Part 4

The following questions are answered using the output of the SQL queries.

1. What fraction of games does white win?

```
+-------------------+
| FractionWhiteWins |
+-------------------+
```

```
|            0.4994 |
+-------------------+
```

Table 18. Fraction of games where White wins.

```
SELECT COUNT(*)/(SELECT COUNT(*) from Games) AS FractionWhiteWins
FROM Games WHERE Winner = 'White';
```

Table 19. SQL statement to find fraction of games where White wins.

2.

a. What is the average number of moves per game when white wins?

```
+-----------------+
| AVG(num_moves) |
+-----------------+
|        57.7785 |
+-----------------+
```

Table 20. Average number of moves in games where White wins.

```
SELECT AVG(num_moves) FROM
    (SELECT COUNT(Games.game_id) as num_moves
    FROM Games
    JOIN Moves
    ON Games.game_id = Moves.game_id
    Where Games.Winner = 'White'
    GROUP BY Moves.game_id) AS q1;
```

Table 21. SQL to find the average move in games where White wins.

b. And if black wins?

```
+-----------------+
| AVG(num_moves) |
+-----------------+
|        60.7959 |
+-----------------+
```

Table 22. Average number of moves in games where Black wins.

```
SELECT AVG(num_moves) FROM
    (SELECT COUNT(Games.game_id) as num_moves
    FROM Games
    JOIN Moves
    ON Games.game_id = Moves.game_id
```

```
    Where Games.Winner = 'Black'
    GROUP BY Moves.game_id) AS q1;
```

Table 23. SQL to find the average move in games where Black wins.

3. What fraction of games start with a pawn move?

```
+------------------+
| FractionPawnMove |
+------------------+
|           0.9599 |
+------------------+
```

Table 24. Fraction of games where the opening move was pawn.

```
SELECT
(
    SELECT COUNT(*)
    FROM Games
    INNER JOIN Moves
    ON Games.game_id = Moves.game_id
    WHERE CHAR_LENGTH(move) = 3
    AND move_num = 0
)/
(
    SELECT COUNT(distinct game_id)
    FROM Moves
) As FractionPawnMove;
```

Table 25. SQL to find the fraction of games where the opening move was pawn.

4.
    a. How many moves, on average, does white make before moving one of his/her Knights?

```
+-----------------------------+
| AVG(movesBeforeMovingKnight) |
+-----------------------------+
|                  2.42403799 |
+-----------------------------+
```

Table 26. Average number of moves before White moves a knight.

```
SELECT AVG(movesBeforeMovingKnight)
FROM
(
```

```
    SELECT min(move_num)/2 as movesBeforeMovingKnight
    FROM Moves
    WHERE LEFT(move,1) = "n"
    AND move_num%2 = 0
    GROUP BY game_id
) AS q1;
```

Table 27. SQL to find the average number of moves before White moves a knight.

b. How many moves, on average, does white make before moving one of his/her Bishops?

```
+-----------------------------+
| AVG(movesBeforeMovingBishop) |
+-----------------------------+
|                  3.90142447 |
+-----------------------------+
```

Table 28. Average number of moves before White moves a bishop.

```
SELECT AVG(movesBeforeMovingBishop)
FROM
(
    SELECT min(move_num)/2 as movesBeforeMovingBishop
    FROM Moves
    WHERE LEFT(move,1) = "b"
    AND move_num%2 = 0
    GROUP BY game_id
) AS q1;
```

Table 29. SQL to find the average number of moves before White moves a bishop.

c. How many moves, on average, does white make before moving one of his/her Rooks (Castles)?

```
+---------------------------+
| AVG(movesBeforeMovingRook) |
+---------------------------+
|                15.29929480 |
+---------------------------+
```

Table 30. Average number of moves before White moves a rook..

```
SELECT AVG(movesBeforeMovingRook)
```

```
FROM
(
    SELECT min(move_num)/2 as movesBeforeMovingRook
    FROM Moves
    WHERE LEFT(move,1) = "r"
    AND move_num%2 = 0
    GROUP BY game_id
) AS q1;
```

Table 31. SQL to find the average number of moves before White moves a rook.

5.

a. How many moves, on average, does black make before moving one of his/her Knights?

```
+-----------------------------+
| AVG(movesBeforeMovingKnight) |
+-----------------------------+
|                  2.53557708 |
+-----------------------------+
```

Table 32. Average number of moves before Black moves a knight.

```
SELECT AVG(movesBeforeMovingKnight)
FROM
(
    SELECT (min(move_num)-1)/2 as movesBeforeMovingKnight
    FROM Moves
    WHERE LEFT(move,1) = "n"
    AND move_num%2 = 1
    GROUP BY game_id
) AS q1;
```

Table 33. SQL to find the average number of moves before Black moves a knight.

b. How many moves, on average, does black make before moving one of his/her Bishops?

```
+-----------------------------+
| AVG(movesBeforeMovingBishop) |
+-----------------------------+
|                  4.22974308 |
+-----------------------------+
```

Table 34. Average number of moves before Black moves a bishop.

```sql
SELECT AVG(movesBeforeMovingBishop)
FROM
(
    SELECT (min(move_num)-1)/2 as movesBeforeMovingBishop
    FROM Moves
    WHERE LEFT(move,1) = "b"
    AND move_num%2 = 1
    GROUP BY game_id
) AS q1;
```

Table 35. SQL to find the average number of moves before Black moves a bishop.

c. How many moves, on average, does black make before moving one of his/her Rooks (Castles)?

```
+----------------------------+
| AVG(movesBeforeMovingRook) |
+----------------------------+
|                15.53814666 |
+----------------------------+
```

Table 36. Average number of moves before Black moves a rook.

```sql
SELECT AVG(movesBeforeMovingRook)
FROM
(
    SELECT (min(move_num)-1)/2 as movesBeforeMovingRook
    FROM Moves
    WHERE LEFT(move,1) = "r"
    AND move_num%2 = 1
    GROUP BY game_id
) AS q1;
```

Table 37. SQL to find the average number of moves before Black moves a rook.