Experiment One

Hypothesis

This experiment will aim to alter the query plan of a query by introducing an index on the winmember attribute of the matches table. This factor should effect queries which join or restrict using the winmember attribute. The winmember attribute is a foreign key.

Method

A plausible query that would be run on the database is one to determine a leader board of the members, ranking them in order based on how many matches they have won and also how many matches they have lost, these will be my queries; they are shown below to the right. They both require a join; the first will use winmember and the second losemember as the foreign key. The output will be different for each but as each query performs the same operations the query plan will be the same.

Query 1

```
SELECT name, count(winmember) wins FROM members, matches WHERE members.memberno = matches.winmember GROUP BY members.name, matches.winmember ORDER BY wins DESC;
```

Query 2

```
SELECT name, count(losemember) wins FROM members, matches WHERE members.memberno = matches. losemember GROUP BY members.name, matches. losemember ORDER BY wins DESC;
```

By introducing an index on the winmember attribute, the plan table for query 1 will change as it now includes an indexed attribute, the second query plan table will not change as it does not use the winmember attribute. This makes the second query suitable for an unchanging control query.

I had a few issues running the queries using sqlplus as the plan tables where being truncated despite my linesize. I instead used iSQLplus which is web interface giving me usable plan table outputs.

I ran the two queries, then created the index on the winmember attribute and then re-ran the two queries.

The generated query plans are shown on the next page.

Query Plan for Query 1, Before Introduction of Factor

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			4,999	1	14	289,942		
SORT	ORDER BY		4,999	1	14	289,942		
HASH	GROUP BY		4,999	1	14	289,942		
HASH JOIN			4,999	1	12	289,942		"MEMBERS"."MEMBERNO" = "MATCHES"."WINMEMBER"
TABLE ACCESS	FULL	MEMBERS	2,148	1	6	96,660		
TABLE ACCESS	FULL	MATCHES	4,999	1	5	64,987		

Query Plan for Query 2, Before Introduction of Factor

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			4,999	1	14	289,942		
SORT	ORDER BY		4,999	1	14	289,942		
HASH	GROUP BY		4,999	1	14	289,942		
HASH JOIN			4,999	1	12	289,942		"MEMBERS"."MEMBERNO" = "MATCHES"."LOSEMEMBER"
TABLE ACCESS	FULL	MEMBERS	2,148	1	6	96,660		
TABLE ACCESS	FULL	MATCHES	4,999	1	5	64,987		

Query Plan for Query 1, After Introduction of Factor (Affected)

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			4,999	1	14	289,942		
SORT	ORDER BY		4,999	1	14	289,942		
HASH	GROUP BY		4,999	1	14	289,942		
HASH JOIN			4,999	1	12	289,942		"MEMBERS"."MEMBERNO" = "MATCHES"."WINMEMBER"
TABLE ACCESS	FULL	MEMBERS	2,148	1	6	96,660		
INDEX	FAST FULL SCAN	EXP1	4,999	1	5	64,987		

Query Plan for Query 2, After Introduction of Factor (Not affected)

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			4,999	1	14	289,942		
SORT	ORDER BY		4,999	1	14	289,942		
HASH	GROUP BY		4,999	1	14	289,942		
HASH JOIN			4,999	1	12	289,942		"MEMBERS"."MEMBERNO" = "MATCHES"."LOSEMEMBER"
TABLE ACCESS	FULL	MEMBERS	2,148	1	6	96,660		
TABLE ACCESS	FULL	MATCHES	4,999	1	5	64,987		

Results

Adding the index has affected the first query and not the second as expected. The first query, after the factor was introduced now performs a fast index scan instead of a full table access on the matches table. As there is no restriction being applied both queries are still dealing with the entire table which means the number of rows being dealt with has not changed.

The query plan for query has changed because the index file is now being used instead of the matches table to retrieve the information.

Hypothesis

In this experiment I will look at the effect of an index on a query which performs a restriction, my control query will be one which does not restrict. The introduction of the index on the clubno attribute should allow.

Method

Two plausible queries that could be run on this database could be the listing of all members for all clubs or for a specific club. The two queries I will use in this experiment output that data. My primary goal is to view the effect of an index on the restriction (selecting a specific club), both of the queries I have chosen will be affected by the creation of the index, but the second query (which restricts) should be affected in a more significant way. The first query doesn't actually require the join as it doesn't do anything with the clubs table, however to keep the similar enough to be useful I need to join on the clubs table.

The second query (which restricts) should be much more efficient, it will no longer be necessary to read the entire table or (looking back on the first experiment) to look at the same number of rows but instead looking in the index file.

I ran the two queries, then created the index on the winmember attribute and then re-ran the two queries.

The generated query plans are shown on the next page.

Query 1

```
SELECT name FROM members,
clubs WHERE clubs.clubno =
members.clubno;
```

Query 2

```
SELECT name FROM members,
clubs WHERE clubs.clubno =
members.clubno AND
clubs.clubno = 6;
```

Query Plan for Query 1, Before Introduction of Factor

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			2,148	1	9	75,180		
HASH JOIN			2,148	1	9	75,180		"CLUBS"."CLUBNO" = "MEMBERS"."CLUBNO"
TABLE ACCESS	FULL	CLUBS	8	1	2	24		
TABLE ACCESS	FULL	MEMBERS	2,148	1	6	68,736		

Query Plan for Query 2, Before Introduction of Factor

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			269	1	9	9,953		
HASH JOIN			269	1	9	9,953		"CLUBS"."CLUBNO" = "MEMBERS"."CLUBNO"
TABLE ACCESS	FULL	CLUBS	1	1	2	3	""""CLUBNO" = 6	
TABLE ACCESS	FULL	MEMBERS	269	1	6	9,146	""""CLUBNO" = 6	

Query Plan for Query 1, After Introduction of Factor

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			2,148	1	7	75,180		
NESTED LOOPS			2,148	1	7	75,180		
TABLE ACCESS	FULL	MEMBERS	2,148	1	6	68,736		
INDEX	RANGE SCAN	EXP2	1	1	0	3		"MEMBERS"."CLUBNO" = "CLUBS"."CLUBNO"

Query Plan for Query 2, After Introduction of Factor

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			269	1	6	9,415		
NESTED LOOPS			269	1	6	9,415		
TABLE ACCESS	FULL	MEMBERS	269	1	6	8,608	""""CLUBNO" = 6	
INDEX	RANGE SCAN	EXP2	1	1	0	3		"CLUBS"."CLUBNO" = 6

Results

As I expected, both plan tables where affected, however they have both been affected in different ways. The first query (which does not restrict) was accessing the information in the two base tables, joining them together (using a hash join) and then projecting. After the index is introduced the join is performed by a nested loop, so the rows of the outer table are searched for matches in the where clause and only if there is a match is the inner table searched. Although the same amount of rows are being compared in this way as with the hash algorithm used before the index introduction – the nested loop row comparison is more efficient and has slightly reduced the cost.

As I hoped, the second query plan has been significantly altered. The nested loop is now performing the join instead of the has join which means that only the rows from the outer table (in this case clubs) to search for matches in the members table. The introduction of the index has removed the need for the filter predicate to be applied on both tables which significantly reduces processing as seen by the reduced cost on all operations on the plan table than the plan table before the introduction of the index.