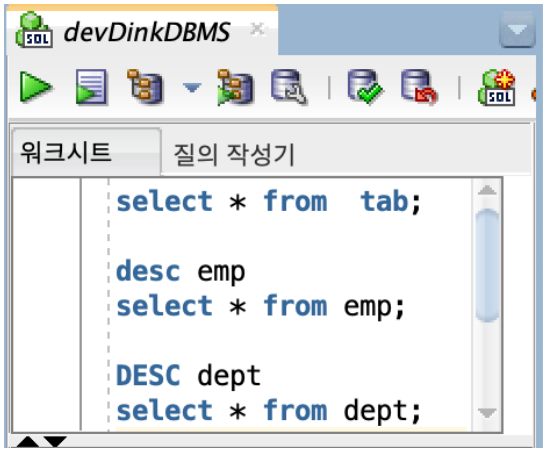
04. Execution Plan(실행계획) 실습

0. SQL Developer

❑ Oracle 社에서 무료로 제공하는 Java 기반 GUI IDE(Integrated Development Environment: 통합개발환경)를 갖춘 SQL 개발(개발자,분석가) 및 관리툴(DBA : Database Administrator,Tuner)

- 유료제품: Toad, Orange

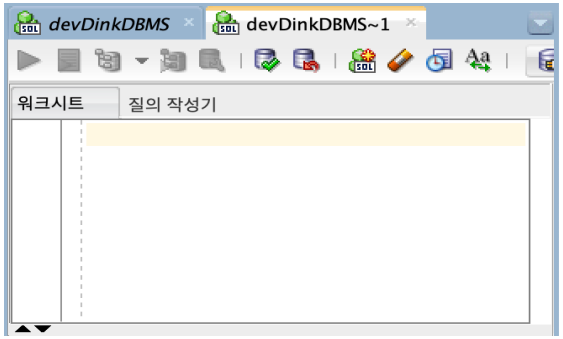
❑ SQL 워크시트에서 N개 SQL 작성



* 해당 SQL 차례로 선택(클릭)후 실행(ctrl + enter)

❑ N개 SQL 워크시트 생성

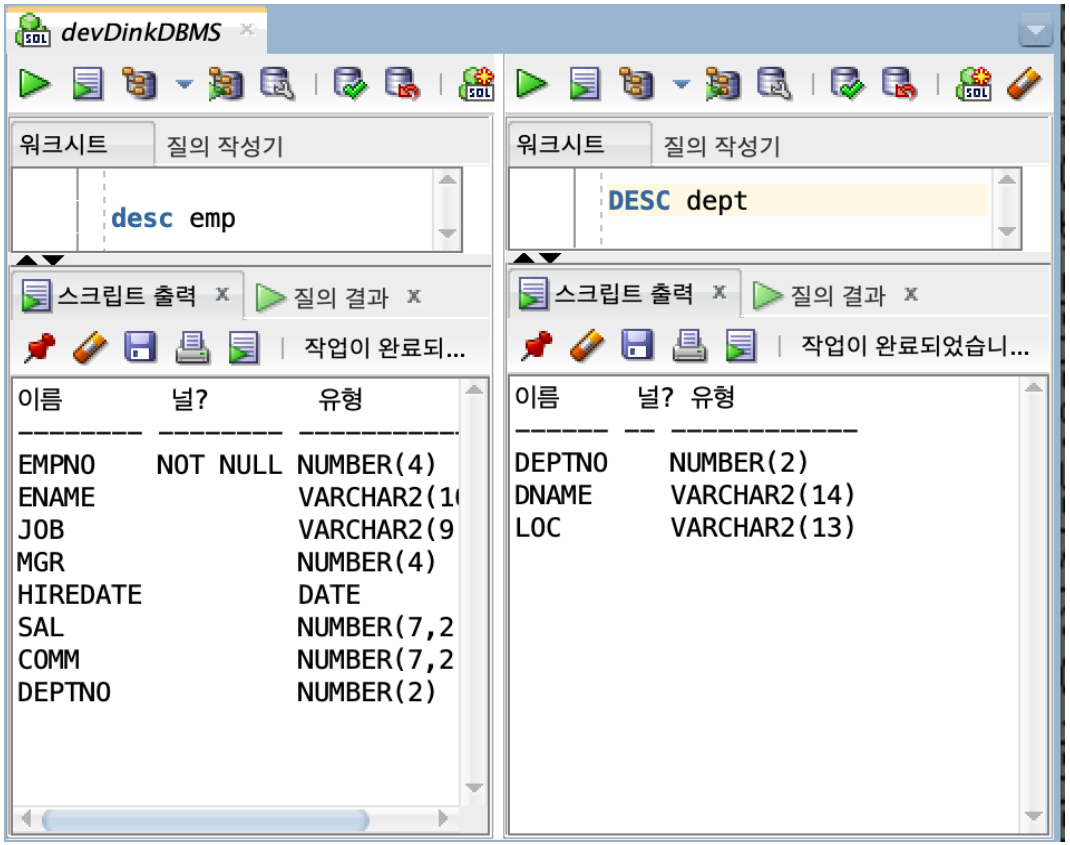
– 도구(T) > SQL 워크시트 > devDinkDBMS 선택



❑ SQL 워크시트 화면 N개 분할 (SQL공유, 실행결과 분할)

– 창(W) > Configure Window > 세로로 분할

– 기존창 : desc emp 실행 , 새로운창: DESC dept 실행

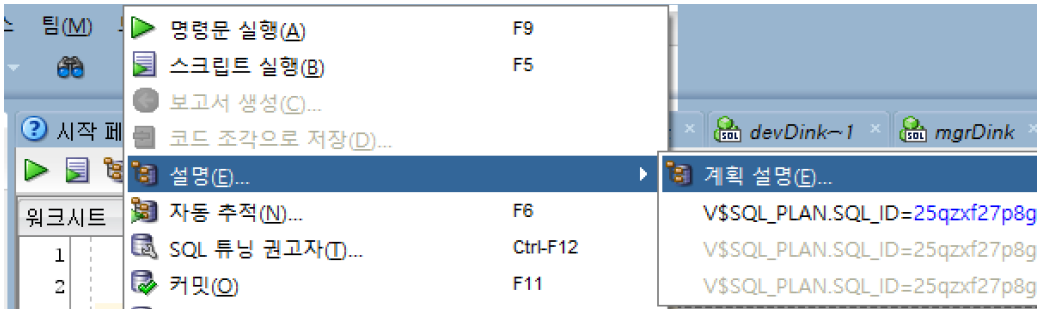


0. SQL Developer에서 실행계획 조회

❑ 실행계획 조회(Execution Plan)

① SELECT * FROM EMP WHERE SAL = '3000';

② SQL 선택 > 마우스 오른쪽 버튼 > 설명 > 계획설명 (Explain execution plan)



③[에러 발생시] 권한이 없는 경우 DBA 계정으로 SQL 워크시트 생성

- 도구(T) > SQL 워크시트 > mgrDinkDBMS 클릭
- grant select_any_catalog to scott; ## 권한부여
- 도구(T) > SQL 워크시트 > devDinkDBMS 클릭 ## 신규 session 생성부터 적용

④

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	EMP	FULL
Filter Predicates		
SAL=3000		

* 문자 < 숫자 : '3000' ➔ 3000 * 문자 < 숫자 < 날짜

0. SQL Developer에서 실행계획 조회

❑ 실행계획 조회(Execution Plan)

⑤ 명시적 형변환
숫자➔문자

SELECT * FROM EMP WHERE TO_CHAR(SAL) = '3000'; ## SQL선택 ➔ F10

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	EMP	FULL
Filter Predicates		
TO_CHAR(SAL)='3000'		

⑥ 암시적 형변환
날짜 > 문자

SELECT * FROM EMP WHERE HIREDATE = '80/12/17';

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	EMP	FULL
Filter Predicates		
HIREDATE='80/12/17'		

⑦ 명시적 형변환
날짜 ➔ 문자

SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'YY/MM/DD') = '80/12/17'

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	EMP	FULL
Filter Predicates		
TO_CHAR(INTERNAL_FUNCTION(HIREDATE),'YY/MM/DD')='80/12/17'		

⑧ 암시적 형변환
날짜 ➔ 문자

SELECT * FROM EMP WHERE HIREDATE LIKE '80/12/17%';

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
TABLE ACCESS	EMP	FULL
Filter Predicates		
INTERNAL_FUNCTION(HIREDATE) LIKE '80/12/17%'		

0. SQL Developer에서 autotrace 사용

▣ SQLDEV 에서 autotrace (explain execution plan + Statistics)

SQL Developer는 설명(Explain Execution Plan) 과 자동추적(Autotrace) 을 사용하여 실행 계획을 조회할수 있다. 설명 보다는 자동추적이 더 많은 분석 정보를 제공 한다.

자동추적 기능을 사용하여 실습해보자, 아래의 SQL을 2번 실행한후 각각 응답시간을 비교한후 확인한후 자동 추적 실습(응답시간이 각각 다른 이유는?)

SQL	SELECT E.EMPNO,C.GENDER,COUNT(C.ID) AS CNT_CUSTOMER FROM EMP E, CUSTOMER C WHERE E.EMPNO = C.ACCOUNT_MGR AND E.JOB != 'PRESIDENT' GROUP BY E.EMPNO,C.GENDER;

SQL | 인출된 모든 행: 26(3,902초)

	EMPNO	GENDER	CNT_CUSTOMER
1	7499	M	404288
2	7566	F	223022
3	7900	F	222004
4	7499	F	443241
5	7782	M	201719

해당 SQL을 선택한후 F6키를 누르거나 마우스 오른쪽 버턴을 클릭하여 자동 추적을 실행한다

```
SELECT   E.EMPNO,C.GENDER,COUNT(C.ID) AS CNT_CUSTOMER
FROM     EMP E, CUSTOMER C
WHERE    E.EMPNO = C.ACCOUNT_MGR AND
         E.JOB  != 'PRESIDENT'
GROUP BY E.EMPNO,C.GENDER;
```

- 명령문 실행(A) F9
- 스크립트 실행(B) F5
- 보고서 생성(C)...
- 코드 조각으로 저장(D)...
- 설명(E)
- 자동 추적(N)... F6

[에러 발생시] 권한이 없는 경우 DBA 계정으로 SQL 워크시트 생성

① grant select_any_catalog to scott;

② grant select on sys.v_$session to demo;
grant select on sys.v_$mystat to demo;
grant select on sys.v_$statname to demo;
* system 계정에 권한이 없는경우 sys 계정에서 각각의 v_$에 대한 권한을 system계정에게 부여한후 재실행
grant select on v_$session to system with grant option;

0. SQL Developer에서 autotrace 사용

▣ SQLDEV 에서 autotrace 사용

자동추적 결과의 각 항목

OPERATION	Row Source Operation
OBJECT_NAME	Operation 수행 대상이 되는 테이블 ,인덱스의 이름
CARDINALITY	Operation 실행시 생성될거라 예측(예상)되는 Row 개수
COST	Operation의 수행 비용
LAST_CR_BUFFER_GETS	현재 실행된 Operation이 Data Buffer Cache내에서 CR(Consistent Read) 모드로 읽은 Block의 개수 (SELECT시 Block를 CR모드로 읽는다)
LAST_ELAPSED_TIME	현재 실행된 Operation의 경과 시간 (microsecond , 백만분의 1초단위)
LAST_OUTPUT_ROWS	현재 실행된 Operation에 의해 생성된 Row 개수

<참고> 실행계획 분석시 CARDINALITY(예상 되는 Row 개수) 와 LAST_OUTPUT_ROWS(실제 생성된 Row 개수)의 차이가 크게 발생하는 이유는 오브젝트에 대한 통계정보가 정확하지 않기 때문이다.

CARDINALITY는 실행계획 수립시 데이터 딕셔너리에 저장되어 있는 오브젝트 통계정보를 기반으로 예측 하기에 정확하지 않는 통계정보는 비효율적인 실행계획을 생성하는 주요 원인이 된다.

통계정보와 실제 데이터를 비교한후 통계정보가 정확하지 않는 경우 DBA에게 통계정보 재생성 요청

(주! ANALYZE ,DBMS_STATS를 사용하여 통계정보 재생성 가능)

0. SQL Developer에서 autotrace 사용

▣ SQLDEV 에서 autotrace 사용

결과

SQL 핫스팟 | 4.005초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				33654	123074	3807366	26
HASH		GROUP BY	19	33654	123074	3807366	26
MERGE JOIN			19	33653	123074	3807022	26
TABLE ACCESS	EMP	BY INDEX ROWID	13	2	2	66	13
Filter Predicates							
E.JOB<>'PRESIDENT'							
INDEX	EMP_EMPNO_PK	FULL SCAN	14	1	1	25	14
SORT		JOIN	19	33651	123072	3806934	26
Access Predicates							
E.EMPNO=ITEM_1							
Filter Predicates							
E.EMPNO=ITEM_1							
VIEW	VW_GBF_7		19	33650	123072	3806826	26
HASH		GROUP BY	19	33650	123072	3806814	26
TABLE ACCESS	CUSTOMER	FULL	5895966	33488	123072	3057384	5895966

① 4.579초는 SQL을 실행한후 결과를 보여주는데 소요된 응답시간

② 성능 문제가 발생하는 부분을 찾는 쉬운 방법은 **Cost가 많이 사용된 Operation을 우선 분석하는 것.**

핫스팟은[N번 클릭하면서 관찰] 실행계획 분석시 요긴하게 사용하는 유틸리티로 핫스팟을 통해 이를 쉽게 찾을 수 있다.

위의 SQL 실행에 사용되는 **전체 Cost 33,654**(가장 상위 Operation에 표시되는 비용이 **각 단위 Operation이 누적된 최종값**)중 핫스팟으로 표시된 **TABLE ACCESS (FULL) CUSTOMER Operation에서 Cost 33,488 사용. 전체 비용의 97.5%(33,488/34,368) 가 CUSTOMER 테이블 FULL SCAN 하는데 사용 (전체 응답시간중에 차지하는 비중, 전체 Logical read중 비중). 튜닝시 우선 검토 대상**

0. SQL Developer에서 autotrace 사용

▣ SQLDEV 에서 autotrace 사용

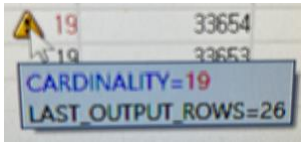
③ 응답시간(Response Time)은 Response time = Elapsed time(Run time +Wait time) + @(ex Transfer time + GUI time)
응답시간은 사용자 관점에서 사용자가 원하는 결과를 얻는데 까지 걸린 시간이고 경과시간(Elapsed time)은 DBMS내부에서 SQL을 처리하는데 사용된 시간이며 대기시간(Wait time)은 DBMS 내부 자원에 대한 대기 시간.
실행계획에 표시되는 시간은 마이크로세컨드(microsecond,㎲,백만분의 일초)

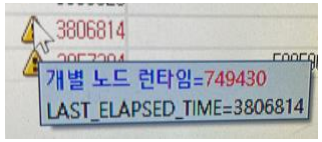
마우스

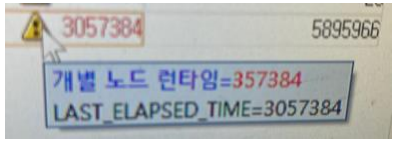
VIEW	VW_GBF_7		19	33650	123072	3806826	26
HASH		GROUP BY	19	33650	123072	3806814	26
TABLE ACCESS	CUSTOMER	FULL	5895966	33488	123072	3057384	5895966

④ CUSTOMER에 대한 FULL SCAN에 3.057384초가 소요 되었고 HASH(GROUP BY) 연산까지 누적된 전체 경과시간은 3.806814초로 HASH(GROUP BY) 연산 수행시간은 3.806814 - 3.057384 = 0.749430 소요

⑤ CARDINALITY=19 , LAST_OUTPUT_ROWS=26 // 예측 vs 실제
LAST_ELAPSED_TIME = 3.806814 , 개별 노드 런타임 = 0.749430
LAST_ELAPSED_TIME = 3.057384 , 개별 노드 런타임 = 0.357384 // 약 2.7초







0. SQL Developer에서 autotrace 사용

▣ SQLDEV 에서 autotrace 사용

⑥ wait **입력** ➔ 2.398905초

wait

V$STATNAME Name	V$MYSTAT Value
file io wait time	2398905
non-idle wait count	15526
non-idle wait time	244
user I/O wait time	244

1. Execution Plan

▣ 실행계획 해석

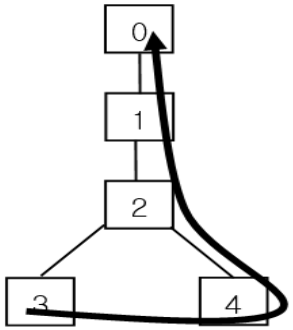
실행계획은 계층적 트리 구조로 구성 되며 실행계획의 순서는 하위 레벨의 Operation이 먼저 실행되고 동일한 레벨이면 위에(위치상으로 윗쪽) 있는 Operation이 먼저 실행 된다.

첫 번째 **[트리]**

튜닝 학습하는 과정에서는 실행계획의 각 Operation을 트리 구조로 그림을 그리면 이해하기 쉽다.

실행계획의 구조가 트리구조로 정식적인 읽는 방법

```
-----------------------------------
| Id  | Operation          | Name |
-----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  HASH GROUP BY     |      |
|*  2 |   HASH JOIN        |      |
|   3 |    TABLE ACCESS FULL| DEPT |
|*  4 |    TABLE ACCESS FULL| EMP  |
-----------------------------------
```



두 번째 **[위안]**

직관적으로 읽는 방법으로 인덴테이션 (Indentation)이 가장 깊은(가장 안쪽에) Operation이 먼저 실행되고

동일한 인덴테이션레벨은 위쪽 Operation이 먼저 실행.

안쪽에서 밖으로 위에서 아래로 읽는다.

1. Execution Plan

▣ 실행계획 해석

```
-----------------------------------------------------
| Id  | Operation                        | Name        |
-----------------------------------------------------
|   0 | SELECT STATEMENT                 |             |
|   1 |  HASH GROUP BY                   |             |
|   2 |   MERGE JOIN                     |             |
|*  3 |    TABLE ACCESS BY INDEX ROWID| EMP            |
|   4 |     INDEX FULL SCAN              | EMP_EMPNO_PK |
|*  5 |    SORT JOIN                     |             |
|   6 |     VIEW                         | VW_GBF_7    |
|   7 |      HASH GROUP BY               |             |
|   8 |       TABLE ACCESS FULL          | CUSTOMER    |
-----------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("E"."JOB"<>'PRESIDENT')
   5 - access("E"."EMPNO"="ITEM_1")
       filter("E"."EMPNO"="ITEM_1")
```

▣ 실행계획 실습 –접근경로(Access path)

- 스캔(SCAN) : 데이터를 읽는 작업
- 실행계획 : Operation의 집합
- Row Source : Operation에 의해 생성된 데이터 집합(a set of rows)
  (ex 테이블,뷰, 조인의 결과, 그룹행 연산의 결과)
- 접근 경로(Access path) : 스캔을 수행하는 방식 , Row Source에서 데이터를 추출하는 방식(경로=path)

- 주요 접근 경로(Access path)

FULL TABLE SCAN	테이블에서 전체 데이터를 읽어 조건에 맞는 데이터 집합을 추출
INDEX SCAN	인덱스를 사용하여 테이블에서 조건에 맞는 데이터 집합을 추출
ROWID SCAN	ROWID를 직접 사용하여 데이터(집합)을 추출

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) - FULL TABLE SCAN (TABLE ACCESS FULL)

SQL	SELECT * FROM CUSTOMER; // 조건절(X)

실행계획

SQL | 0.054초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	33488
TABLE ACCESS	CUSTOMER	FULL	5895966	33488
Other XML				

* 0.054초?

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				33488	4	214	50
TABLE ACCESS	CUSTOMER	FULL	5895966	33488	4	214	50
Other XML							

SQL	SELECT * FROM CUSTOMER WHERE ZIPCODE = '573-738'; // 조건절(O) & Index(X)

실행계획

SQL | 0.034초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			186	33465
TABLE ACCESS	CUSTOMER	FULL	186	33465
Filter Predicates				
ZIPCODE='573-738'				

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) - FULL TABLE SCAN (TABLE ACCESS FULL)

```
create index customer_account_mgr_id_idx on customer(account_mgr,id)
tablespace users;                                  // index record(entry)
```

ACCOUNT_MGR	ID	ROWID
7369	00000021	AAASqQAAFAAAAJLAAU
7369	00000028	AAASqQAAFAAAAJLAAb
7369	00000029	AAASqQAAFAAAAJLAAc
7369	00000055	AAASqQAAFAAAAJOAAH
7369	00000071	AAASqQAAFAAAAJOAAX
7369	00000085	AAASqQAAFAAAAJOAA1

```
select index_name,column_position,column_name from user_ind_columns
where table_name = 'CUSTOMER'
order by 1,2;
```

	INDEX_NAME	COLUMN_POSITION	COLUMN_NAME
1	CUSTOMER_ACCOUNT_MGR_ID_IDX	1	ACCOUNT_MGR
2	CUSTOMER_ACCOUNT_MGR_ID_IDX	2	ID
3	CUSTOMER_ID_PK	1	ID
4	CUSTOMER_MOBILE_NO_UK	1	MOBILE_NO
5	CUSTOMER_NAME_IDX	1	NAME

SQL	SELECT * FROM CUSTOMER WHERE ACCOUNT_MGR= 7499; // 조건절(O) & Index(O)
실행계획	SQL | 0.054초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	33488
TABLE ACCESS	CUSTOMER	FULL	5895966	33488
Other XML				

* Index ??

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) - Index SCAN 1/4

SQL	SELECT * FROM CUSTOMER WHERE ID = '05333333'; // **INDEX UNIQUE SCAN**
실행계획	

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3
TABLE ACCESS	CUSTOMER	BY INDEX ROWID	1	3
INDEX	CUSTOMER_ID_PK	UNIQUE SCAN	1	2
Access Predicates				
ID='05333333'				

```
select constraint_name,constraint_type,search_condition,index_name
from  user_constraints
where table_name = 'CUSTOMER';
```

// type='p' , Primary key ➔ Unique Index

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) - Index SCAN 2/4

SQL	SELECT * FROM CUSTOMER WHERE NAME = '김민준'; // **INDEX RANGE SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			8	11
TABLE ACCESS	CUSTOMER	BY INDEX ROWI...	8	11
INDEX	CUSTOMER_NA...	RANGE SCAN	8	3
Access Predicates				
NAME='김민준'				

SQL	SELECT * FROM CUSTOMER WHERE ID >= '05652222'; // **INDEX RANGE SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			341749	8725
TABLE ACCESS	CUSTOMER	BY INDEX ROWI...	341749	8725
INDEX	CUSTOMER_ID_PK	RANGE SCAN	341749	804
Access Predicates				
ID>='05652222'				

SQL	SELECT ID FROM CUSTOMER WHERE ID >= '05652222'; // **INDEX RANGE SCAN or INDEX FULL SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			341749	804
INDEX	CUSTOMER_ID_PK	RANGE SCAN	341749	804
Access Predicates				
ID>='05652222'				

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) - Index SCAN 3/4

SQL	SELECT ID FROM CUSTOMER WHERE ID >= '00002222'; // **INDEX FAST FULL SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5893785	3767
INDEX	CUSTOMER_ID_PK	FAST FULL SCAN	5893785	3767
Filter Predicates				
ID>='00002222'				

SQL	SELECT COUNT(ID) FROM CUSTOMER; // **INDEX FAST FULL SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3759
SORT		AGGREGATE	1	
INDEX	CUSTOMER_ID_PK	FAST FULL SCAN	5895966	3759
Other XML				

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) - Index SCAN 4/4

SQL	SELECT * FROM CUSTOMER ORDER BY name; // Full Table Scan → Sort

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	221801
SORT		ORDER BY	5895966	221801
TABLE ACCESS	CUSTOMER	FULL	5895966	33488

SQL	SELECT * FROM CUSTOMER ORDER BY ID; // **INDEX FULL SCAN ?? FAST FULL SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	150484
TABLE ACCESS	CUSTOMER	BY INDEX ROWID	5895966	150484
INDEX	CUSTOMER_ID_PK	FULL SCAN	5895966	13829

SQL	SELECT ID FROM CUSTOMER ORDER BY ID; // **INDEX FULL SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	13829
INDEX	CUSTOMER_ID_PK	FULL SCAN	5895966	13829

1. Execution Plan

▣ 실행계획 실습 –접근경로(Access path) – Rowid SCAN

SELECT ROWID,ID,NAME,EMAIL FROM CUSTOMER WHERE ROWNUM < 10;

ROWID	ID	NAME	EMAIL
AAASqQAAFAAAAJLAAA	00000001	배재석	howclnc@cau.ac.kr
AAASqQAAFAAAAJLAAB	00000002	천화하	atmysj@lg.com
AAASqQAAFAAAAJLAAC	00000003	진대택	weremega@naver.com

SQL

SELECT * FROM CUSTOMER WHERE ROWID = 'AAASqQAAFAAAAJLAAA'; // **단돈 1원**
// BY USER ROWID vs BY INDEX ROWID

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1
TABLE ACCESS	CUSTOMER	BY USER ROWID	1	1

SQL

SELECT * FROM CUSTOMER WHERE ID = '05333333'; // **INDEX UNIQUE SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3
TABLE ACCESS	CUSTOMER	BY INDEX ROWID	1	3
INDEX	CUSTOMER_ID_PK	UNIQUE SCAN	1	2
Access Predicates				
ID='05333333'				

SQL

SELECT ID FROM CUSTOMER WHERE ID = '05333333'; // **INDEX UNIQUE SCAN**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	2
INDEX	CUSTOMER_ID_PK	UNIQUE SCAN	1	2
Access Predicates				
ID='05333333'				

1. Execution Plan

▣ 실행계획 실습 – Index 사용 1/4

SQL	SELECT MIN(credit_limit) FROM CUSTOMER; // Full Table Scan ➔ ?? Sort , ?? 빠른 이유

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	33473
SORT		AGGREGATE	1	
TABLE ACCESS	CUSTOMER	FULL	5895966	33473

SQL	SELECT MIN(ID) FROM CUSTOMER;

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3
SORT		AGGREGATE	1	
INDEX	CUSTOMER_ID_PK	FULL SCAN (MIN/MAX)	1	3

SQL	SELECT MAX(ID) FROM CUSTOMER;

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3
SORT		AGGREGATE	1	
INDEX	CUSTOMER_ID_PK	FULL SCAN (MIN/MAX)	1	3

SQL	SELECT MAX(ID),MIN(ID) FROM CUSTOMER; **// COST 3759 ?? 6 = 3+3 , 자동추척(autotrace)**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				3759	14198	2292061	1
SORT		AGGREGATE	1		14198	2292061	1
INDEX	CUSTOMER_ID_PK	FAST FULL SCAN	5895966	3759	14198	1410202	5895966

1. Execution Plan

▣ 실행계획 실습 – Index 사용 2/4

SQL

SELECT * FROM CUSTOMER WHERE ID <> '05333333'; // **비교연산 , 부정형 비교?**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895965	33496
TABLE ACCESS	CUSTOMER	FULL	5895965	33496
Filter Predicates				
ID<> '05333333'				

SQL

SELECT * FROM CUSTOMER WHERE ID < '05333333'; // **Optimizer의 판단 ? 비효율적**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5240858	33492
TABLE ACCESS	CUSTOMER	FULL	5240858	33492
Filter Predicates				
ID< '05333333'				

SQL

SELECT /*+ rule */ * FROM CUSTOMER WHERE ID < '05333333'; // **Hint→RBO의 결정 ?Cost ?효율성 ?빠른 이유**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS	LAST_CR_BUFFER_GETS
SELECT STATEMENT			106	50	5
TABLE ACCESS	CUSTOMER	BY INDEX ROWID	106	50	5
INDEX	CUSTOMER_ID_PK	RANGE SCAN	76	50	3
Access Predicates					
ID< '05333333'					

SQL

SELECT /*+ index(customer customer_id_pk) */ * FROM CUSTOMER WHERE ID < '05333333'; // **Hint → Cost ??**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				133765	5	61	50
TABLE ACCESS	CUSTOMER	BY INDEX ROWID BATCHED	5240858	133765	5	61	50
INDEX	CUSTOMER_ID_PK	RANGE SCAN	5240858	12293	3	42	50
Access Predicates							
ID< '05333333'							

1. Execution Plan

▣ 실행계획 실습 – Index 사용 3/4

SQL	SELECT * FROM CUSTOMER WHERE ID LIKE '0533333%'; // Index를 사용하는 이유는

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	4
TABLE ACCESS	CUSTOMER	BY INDEX ROWID BATCHED	1	4
INDEX	CUSTOMER_ID_PK	RANGE SCAN	1	3

Access Predicates
ID LIKE '0533333%'
Filter Predicates
ID LIKE '0533333%'

SQL	SELECT * FROM CUSTOMER WHERE ID LIKE '0533%' // Index를 사용하는 이유는 ?

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				4	6	72	50
TABLE ACCESS	CUSTOMER	BY INDEX ROWID BATCHED	16	4	6	72	50
INDEX	CUSTOMER_ID_PK	RANGE SCAN	16	3	4	41	50

Access Predicates
ID LIKE '0533%'
Filter Predicates
ID LIKE '0533%'

SELECT count(*) FROM CUSTOMER WHERE ID LIKE '0533%'; **// 9688(count) vs 16(Cardinality) vs 50(LAST_OUTPUT_ROWS)**

SQL	SELECT * FROM CUSTOMER WHERE ID LIKE '%0533333%'; // Index 비교(?) ➔ Full Scan

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			294798	33465
TABLE ACCESS	CUSTOMER	FULL	294798	33465

Filter Predicates
ID LIKE '%0533333%'

1. Execution Plan

▣ 실행계획 실습 – Index 사용 4/4

SQL	SELECT * FROM CUSTOMER WHERE **ID** = **5333333** ; // **숫자 > 문자 ➔ Index 컬럼 변형(암시적) ➔ 비교(?)**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	33471
TABLE ACCESS	CUSTOMER	FULL	1	33471
Filter Predicates				
TO_NUMBER(ID)=5333333				

SQL	SELECT * FROM CUSTOMER WHERE **SUBSTR(ID,1,6)** ='053333'; // **Index 컬럼 변형(명시적) ➔ 비교(?)**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			58960	33550
TABLE ACCESS	CUSTOMER	FULL	58960	33550
Filter Predicates				
SUBSTR(ID,1,6)='053333'				

SQL	SELECT * FROM CUSTOMER WHERE ID IS NULL; // **NULL(제/비/연) ➔ Index 비교(?)**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	0
FILTER				
Filter Predicates				
NULL IS NOT NULL				
TABLE ACCESS	CUSTOMER	FULL	5895966	33488

1. Execution Plan

▣ 실행계획 실습

SQL

```
SELECT   ENROLL_DT,ACCOUNT_MGR,ID,NAME
FROM    CUSTOMER
ORDER BY  ENROLL_DT desc;                    // order by ➔ sort   ,  ?? 시간차이
```

실행계획

SQL | 0.033초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	83747
SORT		ORDER BY	5895966	83747
TABLE ACCESS	CUSTOMER	FULL	5895966	33485

SQL 핫스팟 | 9.277초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				83747	123072	9160234	50
SORT		ORDER BY	5895966	83747	123072	9160234	50
TABLE ACCESS	CUSTOMER	FULL	5895966	33485	123072	3883341	5895966
Other XML							

SQL

```
SELECT ACCOUNT_MGR FROM CUSTOMER
UNION                                        //  UNION ALL, INTERSECT, MINUS
SELECT EMPNO FROM EMP;                       //  UNION  ➔ SORT (Unique)
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				22213	19732	3390741	14
SORT		UNIQUE	5895980	22213	19732	3390741	14
UNION-ALL					19732	2660837	5895980
INDEX	CUSTOMER_ACCOUNT_MGR_ID_IDX	FAST FULL SCAN	5895966	5349	19731	1154320	5895966
INDEX	EMP_EMPNO_PK	FULL SCAN	14	1	1	602	14

1. Execution Plan

▣ 실행계획 실습

SQL	SELECT COUNT(*),AVG(CREDIT_LIMIT) FROM CUSTOMER; // 그룹행함수(ex AVG,COUNT,SUM)를 수행하기 위한 집계(Aggregate) 연산 수행, 개념적 Sort 연산 ➔ Cost 0

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				33473	123072	3271166	1
SORT		AGGREGATE	1		123072	3271166	1
TABLE ACCESS	CUSTOMER	FULL	5895966	33473	123072	3024169	5895966

SQL 0.033초

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5895966	83747
SORT		ORDER BY	5895966	83747
TABLE ACCESS	CUSTOMER	FULL	5895966	33485

SQL	SELECT * FROM CUSTOMER WHERE **ROWNUM <= 10;** // **COUNT STOPKEY**

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				2	3	82	10
COUNT		STOPKEY			3	82	10
Filter Predicates							
ROWNUM<=10							
TABLE ACCESS	CUSTOMER	FULL	10	2	3	78	10

1. Execution Plan

▣ 실행계획 실습

SQL

```
SELECT DEPTNO,SUM(SAL) AS TOTAL_SAL
FROM   EMP_LARGE
WHERE  JOB <> 'PRESIDENT'                    // where vs having의 공통점? , 차이점?
GROUP  BY DEPTNO
HAVING SUM(SAL) > 6000;
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	1706
HASH		GROUP BY	3	1706
Filter Predicates				
SUM(SAL)>6000				
TABLE ACCESS	EMP_LARGE	FULL	960325	1682
Filter Predicates				
JOB<> 'PRESIDENT'				

SQL

```
SELECT * FROM  CUSTOMER  WHERE  ZIPCODE IN ('369-902','742-140');    //  IN  ➔  OR
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				33473	8126	342447	50
TABLE ACCESS	CUSTOMER	FULL	372	33473	8126	342447	50
Filter Predicates							
OR							
ZIPCODE='369-902'							
ZIPCODE='742-140'							

SQL

```
SELECT * FROM  CUSTOMER
WHERE  ZIPCODE BETWEEN '369-902' AND '742-140';                    // BETWEEN ➔   <=  &   =<
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3900293	33491
TABLE ACCESS	CUSTOMER	FULL	3900293	33491
Filter Predicates				
AND				
ZIPCODE>='369-902'				
ZIPCODE<='742-140'				

1. Execution Plan

▣ 실행계획 실습

SQL

```
SELECT DEPTNO,SUM(SAL) AS TOTAL_SAL
FROM   EMP_LARGE
WHERE  JOB <> 'PRESIDENT'                    // where vs having의 공통점? , 차이점?
GROUP  BY DEPTNO
HAVING SUM(SAL) > 6000;
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3	1706
HASH		GROUP BY	3	1706
Filter Predicates				
SUM(SAL)>6000				
TABLE ACCESS	EMP_LARGE	FULL	960325	1682
Filter Predicates				
JOB<> 'PRESIDENT'				

SQL

```
SELECT * FROM  CUSTOMER  WHERE  ZIPCODE IN ('369-902','742-140');    //  IN  ➔  OR
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST	LAST_CR_BUFFER_GETS	LAST_ELAPSED_TIME	LAST_OUTPUT_ROWS
SELECT STATEMENT				33473	8126	342447	50
TABLE ACCESS	CUSTOMER	FULL	372	33473	8126	342447	50
Filter Predicates							
OR							
ZIPCODE='369-902'							
ZIPCODE='742-140'							

SQL

```
SELECT * FROM  CUSTOMER
WHERE  ZIPCODE BETWEEN '369-902' AND '742-140';                    // BETWEEN ➔   <=  &   =<
```

실행계획

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			3900293	33491
TABLE ACCESS	CUSTOMER	FULL	3900293	33491
Filter Predicates				
AND				
ZIPCODE>='369-902'				
ZIPCODE<='742-140'				