Базы данных

Оптимизация реляционных запросов

Оптимизация запросов

- Оптимизация как написание "оптимальных" запросов. Это задача программиста (или квалифицированного пользователя): она заключается в написании таких реляционных запросов, для которых СУБД могла бы использовать более эффективные способы нахождения данных.
- Оптимизация как внутренняя задача СУБД, которая заключается в определении наиболее оптимального (эффективного) способа выполнения реляционных запросов.
- Под **оптимизацией** понимается построение квазиоптимального процедурного плана выполнения декларативного запроса.

Пример выполнения запросов

Список программистов 4-го отдела с «чистой» зарплатой не менее 40000 рублей:

```
SELECT * FROM Emp
        WHERE depNo=4 AND post LIKE 'программист%'
        AND salary*0.87>=40000;
```

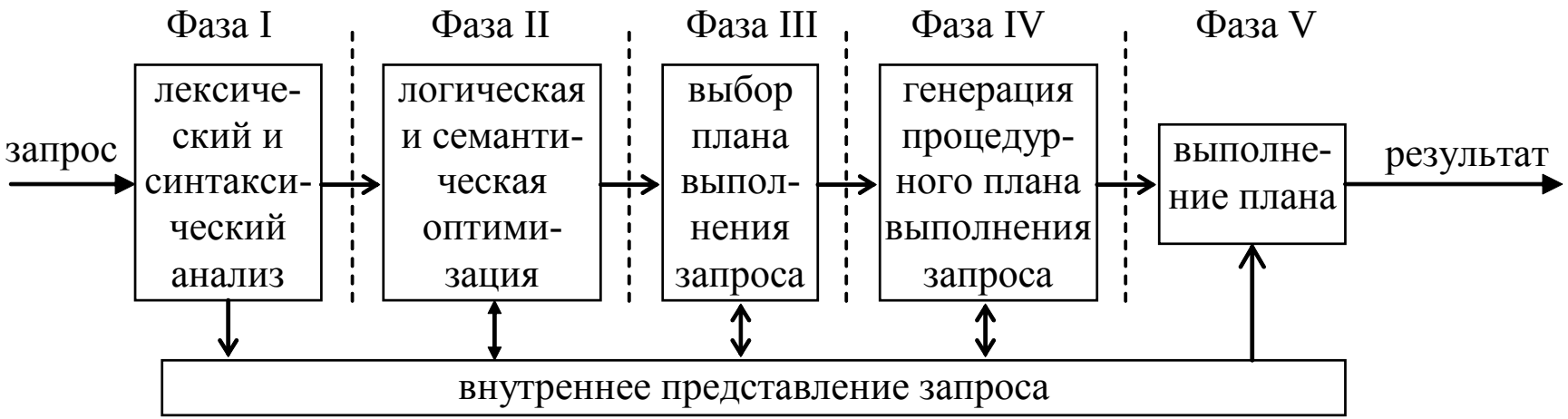
Если по полям depNo (Номер отдела), post (Должность) и salary (Зарплата) есть индексы, то способы выполнения этого запроса могут быть такими:

- Найти по индексу INDEX(depNo) записи, удовлетворяющие первому условию, и проверить для найденных записей второе условие.
- Найти по индексу INDEX(post) записи, удовлетворяющие второму условию, и проверить для найденных записей первое условие.
- Последовательно считать все записи таблицы Emp и проверить для каждой записи оба условия.

Индексом по полю salary система воспользоваться не может, т.к. это поле находится внутри выражения.

Этапы оптимизации запросов в РСУБД

План выполнения запроса состоит из последовательности шагов, каждый из которых либо физически извлекает данные из памяти, либо делает подготовительную работу. Построением этого плана занимается оптимизатор – специальная компонента СУБД.



Преобразования операций реляционной алгебры

Два выражения реляционной алгебры считаются **эквивалентными**, если они описывают одно и то же отображение.

В качестве примера приведём отношения R1 и R2, содержащие по 1000 кортежей, причём только 10 кортежей в каждом отношении удовлетворяют условию F. Если выполнять следующую последовательность операций:

$$\sigma_F (R1 \cup R2),$$

то после выполнения объединения получится 2000 кортежей (если отношения не содержат одинаковых кортежей), а после селекции останется 20 записей. Если изменить последовательность выполнения операций:

$$\sigma_F (R1) \cup \sigma_F (R2),$$

то после селекции останется по 10 записей из каждого отношения, объединение которых даст 20 требуемых кортежей. Если учитывать, что объединение обычно выполняется путем сортировки данных (для удаления одинаковых кортежей) и промежуточный результат надо хранить, то выигрыш и по объёму памяти и по времени очевиден: гораздо быстрее отсортировать 20 кортежей, а не 2000.

Преобразования операций реляционной алгебры

Закон коммутативности для декартовых произведений: $R \times S = S \times R$

r1
r2

Объем ОП – 4 блока.

s1
s2
s3
s4
…
s200

Содержимое ОП по тактам:

1-й, 2-й:	3-й:	4-й:	5-й:	6-й:
r1	r1	r1	r1	r1
s1	s1	s1	s4	s4
	s2	s2	s2	s5
		s3	s3	s3

…

201-й:	202-й:	203-й:	402:
r1	r1	r1	r1
s198	r2	r2	r2
s199	s199	s1	s1
s200	s200	s200	s2

…

$2 + 200 \times 2 = 402$ физических чтений

Преобразования операций реляционной алгебры

Закон коммутативности для декартовых произведений: $R \times S = S \times R$

s1
s2
s3
s4
…
s200

r1
r2

Объем ОП – 4 блока.

Содержимое ОП по тактам:

1-й, 2-й:	3-й:	4-й:	5-й:	6-й:		202:
s1	s1	s1	s3	s3		s199
r1	r1	r1	r1	r1	…	r1
	r2	r2	r2	r2		r2
		s2	s2	s4		s200

2 + 200 = 202
физических чтения

Преобразования операций реляционной алгебры

1. Закон коммутативности для декартовых произведений:
   $R1 \times R2 = R2 \times R1$.
2. Закон коммутативности для соединений (F – условие соединения):
   $R1 \rhd\lhd_F R2 = R2 \rhd\lhd_F R1$.
3. Закон ассоциативности для декартовых произведений:
   $(R1 \times R2) \times R3 = R1 \times (R2 \times R3)$.
4. Закон ассоциативности для соединений (F1, F2 – условия соединения):
   $(R1 \rhd\lhd_{F1} R2) \rhd\lhd_{F2} R3 = R1 \rhd\lhd_{F1} (R2 \rhd\lhd_{F2} R3)$.
5. Комбинация селекций (каскад селекций):
   $\sigma_{F1} (\sigma_{F2} (R)) = \sigma_{F1 \wedge F2} (R)$.
6. Комбинация проекций (каскад проекций):
   $\pi_{A1,A2,\ldots,Am} (\pi_{B1,B2,\ldots,Bn}(R)) = \pi_{A1,A2,\ldots,Am}(R)$,
   где $\{A_m\} \subset \{B_n\}$.
7. Перестановка селекции и проекции:
   $\sigma_F (\pi_{A1,A2,\ldots,Am}(R)) = \pi_{A1,A2,\ldots,Am}(\sigma_F (R))$.

Преобразования операций реляционной алгебры

8. Перестановка селекции с объединением:
   $\sigma_F (R1 \cup R2) = \sigma_F (R1) \cup \sigma_F (R2)$.
9. Перестановка селекции с декартовым произведением:
   - $\sigma_F (R1 \times R2) = (\sigma_{F1} (R1)) \times (\sigma_{F2} (R2))$,
     (если $F = F1 \wedge F2$, где F1 содержит атрибуты, присутствующие только в R1, а F2 содержит атрибуты, присутствующие только в R2);
   - $\sigma_F (R1 \times R2) = (\sigma_F (R1)) \times R2$,
     (если F содержит атрибуты, присутствующие только в R1);
   - $\sigma_F (R1 \times R2) = R1 \times (\sigma_F (R2))$,
     (если F содержит атрибуты, присутствующие только в R2);
   - $\sigma_F (R1 \times R2) = \sigma_{F2} (\sigma_{F1} (R1) \times R2)$,
     (если $F = F1 \wedge F2$, где F1 содержит атрибуты, присутствующие только в R1, а F2 содержит атрибуты, присутствующие и в R1, и в R2).
10. Перестановка селекции с разностью:
    $\sigma_F (R1 - R2) = \sigma_F (R1) - \sigma_F (R2)$.
11. Перестановка селекции с пересечением:
    $\sigma_F (R1 \cap R2) = \sigma_F (R1) \cap \sigma_F (R2)$.

Метод оптимизации, основанный на синтаксисе

План составляется на основании существующих путей доступа и их рангов. Все пути доступа ранжируются на основании знаний о правилах и последовательности осуществления этих путей.

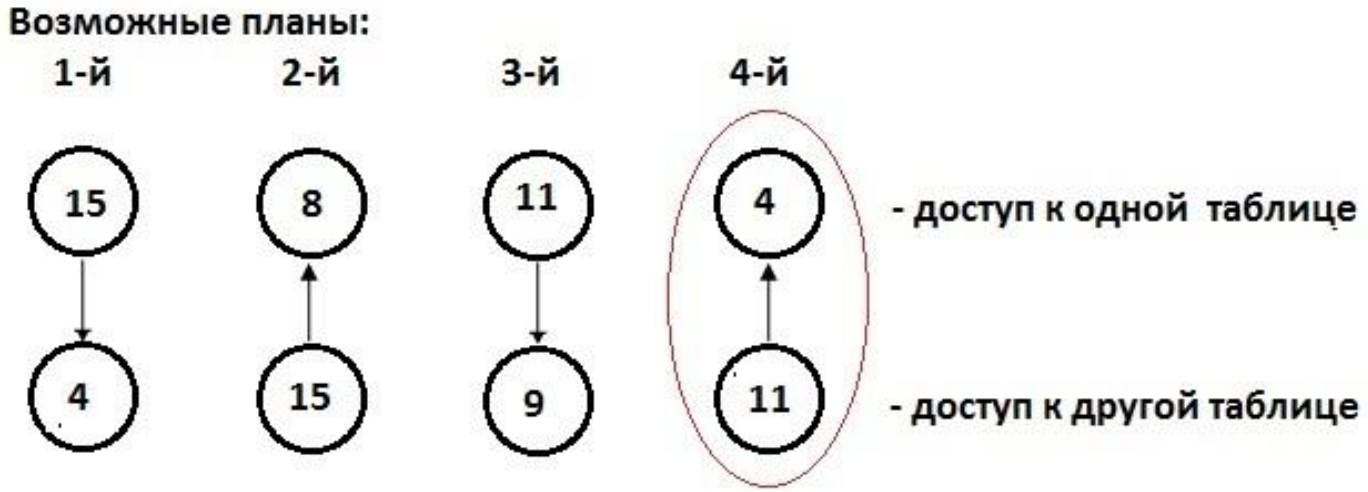
Для Oracle ранги такие:

Ранг	Пути доступа
1	Одна строка по ROWID*
2	Одна строка по кластерному соединению
3	Одна строка по хеш-кластеру с уникальным или первичным ключом
4	Одна строка по уникальному или первичному ключу
5	Кластерное соединение
6	Ключ хеш-кластера
7	Ключ индексного кластера
8	Составной индекс
9	Индекс по одиночному столбцу (по условию равенства)
10	Индексный поиск по закрытому интервалу
11	Индексный поиск по открытому интервалу
12	Сортировка-объединение
13	MAX и MIN по индексированному столбцу
14	ORDER BY по индексированному столбцу
15	Полный просмотр таблицы

* RowID – ключ базы данных в Oracle

Метод оптимизации, основанный на синтаксисе

Ранг пути доступа определяется *на основании знаний о последовательности реализации этого пути.* Например, самый быстрый способ доступа – это чтение по КБД: если он известен, то это одно физическое чтение. А поиск конкретного значения через индекс (ранг 9) обычно занимает меньше времени, чем поиск в закрытом интервале значений (ранг 10).
Метод оптимизации по синтаксису учитывает ранги путей доступа. Вырабатывается несколько возможных планов, и выбирается тот план, чей ранг выше (т.е. сумма рангов путей доступа меньше), т.к. в общем случае он выполняется быстрее, чем план с более низким рангом.



Метод оптимизации, основанный на стоимости

Оптимизатор сначала строит несколько возможных планов выполнения запроса (обычно, 2-3 плана). Для выбора наиболее перспективных планов он применяет некоторые **эвристики**, т.е. правила, полученные опытным путем.

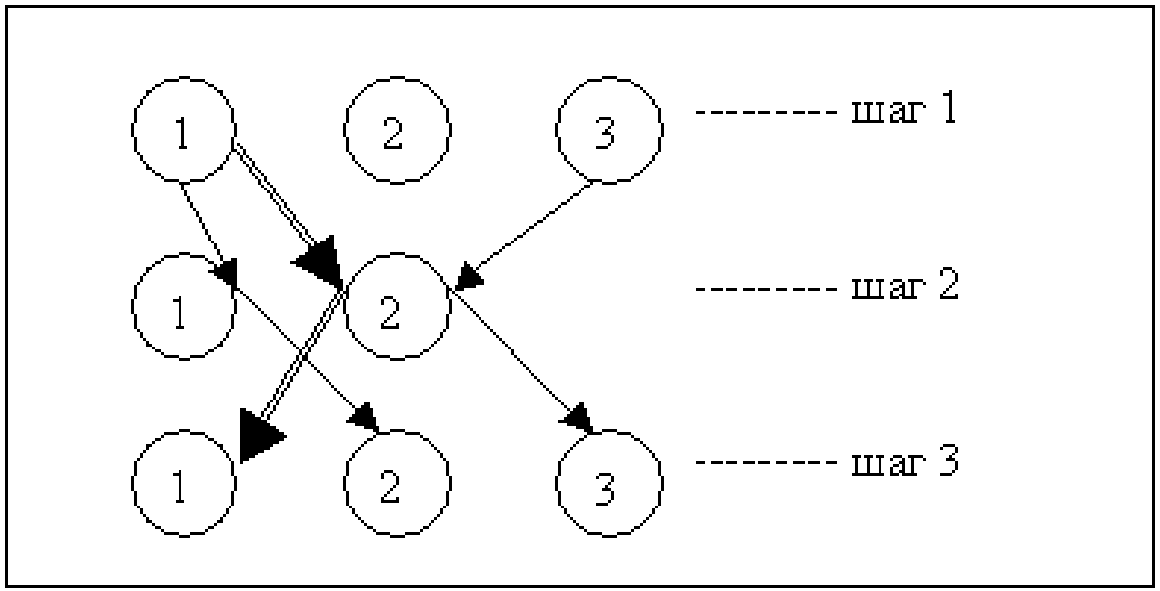
Для каждого из построенных планов рассчитывается его стоимость. **Стоимость** – это оценка ожидаемого времени выполнения запроса с использованием конкретного плана выполнения. При расчёте стоимости оптимизатор может учитывать такие параметры, как количество необходимых ресурсов памяти, время операций дискового ввода-вывода, время процессора.

Из множества возможных планов выполнения запроса оптимизатор в соответствии с критерием выбирает лучший план.

В качестве **критерия оптимизации** может выступать:

- Наилучшая общая производительность системы.
- Минимальное время реакции – время, необходимое для обработки и выдачи первой строки.
- Минимальные затраты времени на обработку всех строк, к которым обращается данная команда.

Метод оптимизации, основанный на стоимости



Стоимость плана выполнения запроса определяется на основании сведений о распределении данных в таблицах, к которым обращается команда, и связанных с ними кластеров и индексов.

Эти сведения о распределении значений данных называются **статистикой** и хранятся в словаре-справочнике данных.

Метод оптимизации, основанный на стоимости

Для таблицы **статистика** может включать в себя:

- общее количество блоков данных (страниц памяти), выделенных таблице;
- количество пустых блоков данных (страниц памяти);
- количество записей в таблице;
- среднюю длину записи в таблице;
- среднее количество записей на блок (страницу) памяти.

Для каждого индекса статистика может содержать такие данные, как:

- общее количество проиндексированных записей (оно может быть меньше, чем количество записей в таблице, т.к. null-значения не индексируются);
- минимальное и максимальное индексированные значения;
- количество различных индексированных значений.

Метод оптимизации, основанный на стоимости

Распределение значений в столбце может быть отражено с помощью гистограммы, которая также входит в статистику. Для этого всё множество значений столбца упорядочивается и разбивается на N интервалов. На гистрограммах ниже приведено разбиение на N =10 интервалов множества значений некоторого столбца F. Для равномерного распределения это означает, что в первых 10% записей это поле имеет значение от 1 до 10, в следующих 10% записей – от 11 до 20 и т.д.

Метод оптимизации, основанный на стоимости

Построение гистограммы бесполезно в следующих случаях:
✓столбец не используется в предикатах запросов;
✓значения столбца уникальны и используются только в предикатах эквивалентности;
✓значения столбца распределены равномерно. При равномерном распределении оценку селективности условия можно провести без построения гистограммы. Например, для условия (*field* > *limit*) по формуле:

(high_value – limit) / (high_value – low_value)

где high_value, low_value – максимальное и минимальное значения поля *field*.

Существуют различные подходы к порядку сбора статистики. Некоторые СУБД постоянно собирают статистическую информацию, но это может уменьшить быстродействие системы. Другие позволяют осуществлять сбор статистики периодически, например, в период минимальной загрузки системы. Третьи предлагают администратору специальные средства для сбора статистики, которые запускаются интерактивно по его команде. В последнем случае в обязанность администратора (администратора данных или приложений) входит выбор таблиц для анализа и периодическое обновление статистики данных.

Подсказки оптимизатору

В некоторых СУБД (например, Oracle, SQL Server и др.) существуют возможности задания подсказок оптимизатору (*hints*). Подсказки являются указаниями оптимизатору, какие пути доступа к данным следует использовать. Подсказки касаются использования индексов (*INDEX*), последовательного чтения (*FULL*), порядка соединения таблиц и проч.
Подсказки в Oracle:

```
select --+ INDEX (emp  ind_salary)
           *  from  emp
           where   depno  IN  (2, 3, 6)  AND  salary > 75000;
select  /*+FULL */  depno, ename, salary
           from  emp
           order by  depno;
```

Для СУБД SQL Server порядок указания подсказок иной:

```
select  *  from  authors
           with  (INDEX(aunmind));
select  *  from  depart  d,  emp  e,  children  c
           where  d.depno = e.depno  AND  e.tabno = c.tabno
           OPTION  (FORCE  ORDER);
```

Примеры использования методов оптимизации запросов

Запрос, выбирающих всех сотрудников информационно-аналитического отдела с зарплатой менее 40000 рублей:

```
SELECT  *
        FROM  Emp  as  e, Depart  as  d
        WHERE  d.did=e.depno  AND  d.name  LIKE  'Информ%аналитич%'
        AND  salary<40000;
```

Пусть таблицы имеют следующие правила целостности и индексы:

- ✓столбец did в Depart - PRIMARY KEY, индекс по первичному ключу;
- ✓существует индекс SALARY_IND для столбца salary в Emp.

Возможны следующие пути доступа:

- ✓полный просмотр таблиц (ранг 15);
- ✓доступ к Emp с помощью открытого интервала salary<40000, используя индекс SALARY_IND (ранг 11);
- ✓доступ к Depart через индекс по первичному ключу, т.к. таблица Depart читается после таблицы Emp, а на этот момент условие на значение поля did известно (оно равно e.depno) (ранг 4).

Оптимизатор выберет для таблицы Depart доступ по индексу с рангом 4 и для таблицы Emp доступ по индексу с рангом 11.

Продолжение примера

```
SELECT  /*+ rule */ *
FROM  Emp e, Depart d
WHERE  d.did=e.depno  AND  d.name  LIKE  'Constr%'
AND  salary<40000;
```

Results | **Explain** | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Filter Predicates *	Access Predicates
SELECT STATEMENT				
NESTED LOOPS				
TABLE ACCESS	BY INDEX ROWID	EMP		
INDEX	RANGE SCAN	SALARY_IND		"SALARY"<40000
TABLE ACCESS	BY INDEX ROWID	DEPART	"D""NAME" LIKE 'Constr%'	
INDEX	UNIQUE SCAN	SYS_C004171		"D"."DID" = "E"."DEPNO"

* Unindexed columns are shown in red

Index Columns

Owner	Table Name	Index Name	Used In Plan	Columns	Uniqueness	Status	Index Type	Join Index
STUD	DEPART	SYS_C004171	✓	DID	UNIQUE	VALID	NORMAL	NO
	EMP	SYS_C004175		TABNO	UNIQUE	VALID	NORMAL	NO
		SALARY_IND	✓	SALARY	NONUNIQUE	VALID	NORMAL	NO

Продолжение примера

Добавление индекса по внешнему ключу (FK_DEPNO) не изменит план, хотя его ранг (9) меньше, чем у используемого индекса SALARY_IND (11).

```
SELECT  /*+ rule */ *
FROM  Emp e, Depart d
WHERE  d.did=e.depno  AND  d.name  LIKE  'Constr%'
AND  salary<40000;
```

Results | Explain | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Filter Predicates *	Access Predicates
SELECT STATEMENT				
NESTED LOOPS				
TABLE ACCESS	BY INDEX ROWID	EMP		
INDEX	RANGE SCAN	SALARY_IND		"SALARY"<40000
TABLE ACCESS	BY INDEX ROWID	DEPART	"D""NAME" LIKE 'Constr%'	
INDEX	UNIQUE SCAN	SYS_C004171		"D"."DID" = "E"."DEPNO"

Index Columns

Owner	Table Name	Index Name	Used In Plan	Columns	Uniqueness	Status	Index Type	Join Index
STUD	DEPART	SYS_C004171	✓	DID	UNIQUE	VALID	NORMAL	NO
	EMP	SYS_C004175		TABNO	UNIQUE	VALID	NORMAL	NO
		SALARY_IND	✓	SALARY	NONUNIQUE	VALID	NORMAL	NO
		FK_DEPNO		DEPNO	NONUNIQUE	VALID	NORMAL	NO

Продолжение примера

Если использовать метод оптимизации по стоимости, то план изменится: к таблице Depart будет полный доступ.

```
SELECT  *
FROM  Emp e, Depart d
WHERE  d.did=e.depno  AND  d.name  LIKE  'Constr%'
AND  salary<40000;
```

Results Explain Describe Saved SQL History

Query Plan

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			2	1	4	940		
TABLE ACCESS	BY INDEX ROWID	EMP	2	1	1	810	"SALARY"<40000	
NESTED LOOPS			2	1	4	940		
TABLE ACCESS	FULL	DEPART	1	1	3	65	"D""NAME" LIKE 'Constr%'	
INDEX	RANGE SCAN	FK_DEPNO	3	1	0			"D"."DID" = "E"."DEPNO"

* Unindexed columns are shown in red

Index Columns

Owner	Table Name	Index Name	Used In Plan	Columns	Uniqueness	Status	Index Type	Join Index
STUD	DEPART	SYS_C004171		DID	UNIQUE	VALID	NORMAL	NO
	EMP	SYS_C004175		TABNO	UNIQUE	VALID	NORMAL	NO
		SALARY_IND		SALARY	NONUNIQUE	VALID	NORMAL	NO
		FK_DEPNO	✓	DEPNO	NONUNIQUE	VALID	NORMAL	NO

Пример для метода по стоимости

```
SELECT   *
         FROM  Emp e
         WHERE  tabno < 5;
```

Этот запрос выдаст 2 строки (около 2% от объема таблицы).

```
select *
from emp
where tabno<5;
```

Results | Explain | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			1	1	2	187		
TABLE ACCESS	BY INDEX ROWID	EMP	1	1	2	187		
INDEX	RANGE SCAN	SYS_C004175	1	1	1			"TABNO"<5

* Unindexed columns are shown in red

Index Columns

Owner	Table Name	Index Name	Used In Plan	Columns	Uniqueness	Status	Index Type	Join Index
STUD	EMP	SYS_C004175	✓	TABNO	UNIQUE	VALID	NORMAL	NO
		SALARY_IND		SALARY	NONUNIQUE	VALID	NORMAL	NO

Продолжение примера

```
SELECT   *
         FROM  Emp e
         WHERE  tabno > 30000;
```

Этот запрос возвращает 50 строк – больше половины от всех строк таблицы.

```
select *
from emp
where tabno>30000;
```

Results | Explain | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			64	1	3	11 968		
TABLE ACCESS	FULL	EMP	64	1	3	11 968	"TABNO">30000	

* Unindexed columns are shown in red

Index Columns

Owner	Table Name	Index Name	Used In Plan	Columns	Uniqueness	Status	Index Type	Join Index
STUD	EMP	SYS_C004175		TABNO	UNIQUE	VALID	NORMAL	NO
		SALARY_IND		SALARY	NONUNIQUE	VALID	NORMAL	NO

Пример для метода по стоимости

Запрос, выбирающий название отделов и всех сотрудников с максимальной зарплатой в своём отделе (name, salary из таблицы Emp):

```
SELECT  d.name, e.name, salary
    FROM  depart  d, emp  e
    WHERE  d.did=e.depNo AND
    e.salary=(SELECT  max(salary)  FROM  emp  p
                    WHERE  p.depNo=e.depNo);
```

Есть индексы по первичным ключам и по внешнему ключу (Emp.depNo).

Возможные планы:

1. Выбрать все записи из таблиц Emp и Depart, соединить их по условию d.depNo=e.depNo, затем для каждой полученной строки посчитать подзапрос (выбрать максимальную зарплату для данного отдела, обратившись к таблице Emp по индексу) и проверить второе условие.
2. Выбрать все записи из таблицы Emp, для каждой записи найти соответствие по условию d.depNo=e.depNo в таблице Depart через индекс по первичному ключу, затем для каждой полученной строки посчитать подзапрос и проверить второе условие.
3. Выбрать все записи из таблицы Emp, каждую запись соединить по условию d.depNo=e.depNo с таблицей Depart через индекс по первичному ключу. Предварительно посчитать подзапрос, добавив в него условие группировки по номерам отделов GROUP BY depNo. Затем для каждой строки соединения проверить второе условие.

Пример для метода по стоимости

Оптимизатор Oracle выбирает третий план как самый эффективный.

```
SELECT d.name, e.name, salary
FROM depart d, emp e
WHERE d.did=e.depNo AND
e.salary=(SELECT max(salary) FROM emp p
 WHERE p.depNo=e.depNo);
```

Results | Explain | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Rows	Cost	Bytes	Access Predicates
SELECT STATEMENT			9	9	639	
NESTED LOOPS			9	9	639	
HASH JOIN			9	8	414	"E"."SALARY" = "VW_COL_1" AND "DEPNO" = "E"."DEPNO"
VIEW		VW_SQ_1	4	4	104	
HASH	GROUP BY		4	4	24	
TABLE ACCESS	FULL	EMP	96	3	576	
TABLE ACCESS	FULL	EMP	96	3	1 920	
TABLE ACCESS	BY INDEX ROWID	DEPART	1	1	25	
INDEX	UNIQUE SCAN	SYS_C004171	1	0		"D"."DID" = "E"."DEPNO"