Оптимизация запросов в реляционных базах данных

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;
```

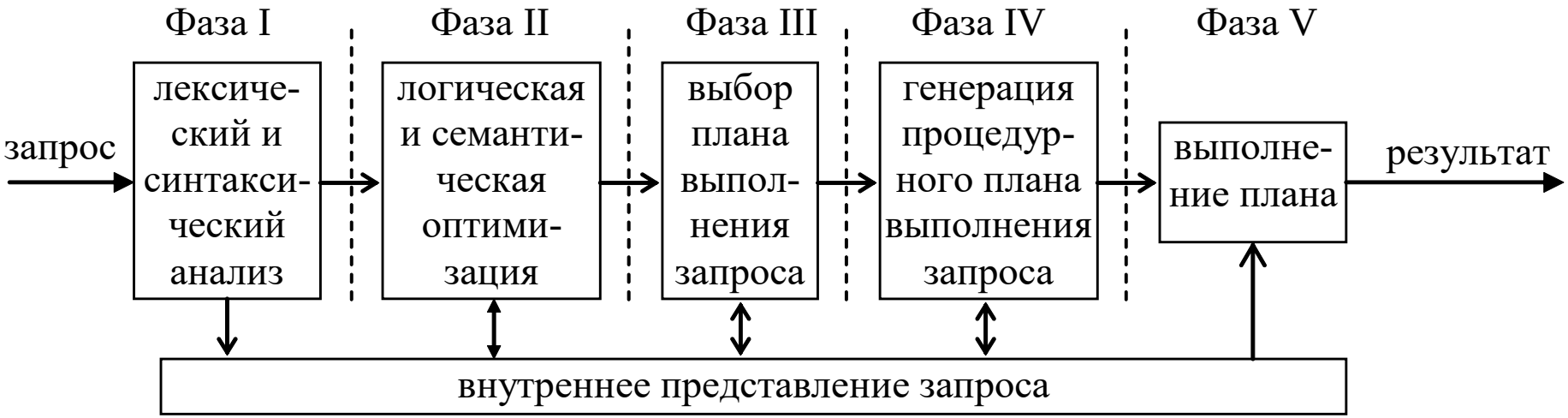
Если по полям depNo (Номер отдела), post (Должность) и salary (Зарплата) есть индексы, то способы выполнения этого запроса могут быть такими:

- 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 выполнения запроса

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.



Направление исследований

- Проблемы преобразований запроса к более эффективному непроцедурному представлению (логическая оптимизация); проблемы выбора набора альтернативных процедурных планов выполнения запроса; проблемы оценок стоимости выполнения запроса по выбранному плану и т.д.
- Поиск новых, более эффективных стратегий выполнения элементарных составляющих запроса и способов композиции более сложных стратегий на основе элементарных.
- Оптимизация распределенных запросов (запросов в распределенных базах данных).
- Глобальная оптимизация запросов в системах баз данных.

Обзоры:

- Дейт К. Введение в системы баз данных.- М.: Наука.1980.- 463 с.
- Ульман Д. Основы систем баз данных.- М.: Финансы и статистика. - 1983.- 335 с.
- Мейер Д. Теория реляционных баз данных.- М.: Мир.1987.- 608 с.
- Date C.J. An Introduction to Database Systems. V.1. 4th ed.- Reading, Mass.: Addison-Wesley.- 1984.- 639 с.

Логическая оптимизация запросов

В ходе логической оптимизации система приводит запрос к каноническому виду. При этом обычно используются законы преобразования операций РА.

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

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

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

…

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:
    $\sigma_F (R1 \cap R2) = \sigma_F (R1) \cap \sigma_F (R2)$.

Логические преобразования

1. Преобразования, связанные с приведением предикатов, задающих условие выборки в данном запросе, к каноническому представлению.
В общем случае такой предикат имеет вид

<арифметическое выражение> op <арифметическое выражение>,

где арифметические выражения левой и правой частей в общем случае содержат имена полей отношений и константы (среди констант могут быть и литеральные константы, и имена переменных объемлющей программы, значения которых становятся известными только при реальном выполнении запроса). Канонические виды предикатов:

1) <имя поля> op <константное арифметическое выражение>
2) <имя поля> op <арифметическое выражение>
3) <арифм. выражение> op <константное арифметическое выражение>

В каноническом виде в выражениях полностью раскрыты скобки и произведено некоторое лексикографическое упорядочение. Цель: в дальнейшем это позволит произвести поиск общих арифметических выражений в разных предикатах запроса. Такая работа может быть оправдана, поскольку при реальном выполнении запроса вычисление арифметических выражений будет производиться при выборке каждого очередного кортежа, т.е. потенциально очень большое число раз.

Логические преобразования

2. Приведение к каноническому виду логического выражения, задающего условие выборки.

Как правило, используются либо дизъюнктивная, либо конъюнктивная нормальные формы.

При приведении логического условия к каноническому представлению можно производить поиск общих предикатов (они могут существовать изначально, могут появиться после приведения предикатов к каноническому виду или в процессе нормализации логического условия), и, кроме того, может быть произведено упрощение логического выражения за счет, например, выявления конъюнкции взаимно противоречащих предикатов.

Например:

... **(A>5) AND (A<5)** ... можно заменить на ...FALSE...

... **(A>B) AND (B=5)** . .. можно дополнить условием **AND (A>5)**

Как видно из последнего примера, такие упрощения могут оказаться очень существенными для дальнейшей обработки запроса: в запросе с логическим условием первого вида предполагалось выполнение соединения двух отношений; после преобразования первое условие можно проверить без выполнения соединения.

Логические преобразования

4. Приведение запросов к каноническому виду. Типы предикатов:

1) Простые предикаты. Это предикаты вида **Ri.Ck op X,**
где X константа или список констант, и op – оператор скалярного сравнения (=, !=, >, >=, <, <=) или оператор проверки вхождения во множество (IS IN, IS NOT IN).

2) Предикаты со вложенными подзапросами: **Ri.Ck op Q,**
где Q - блок запроса, а **op** может быть таким же, как для простых предикатов. Предикат может также иметь вид **Q op Ri.Ck**. В этом случае оператор принадлежности ко множеству заменяется на CONTAINS или DOES NOT CONTAIN. Блоком запроса называется допустимая конструкция языка, начинающаяся с ключевого слова SELECT, т.е. в блоке запроса не допускаются конструкции UNION, INTERSECT и MINUS.

3) Предикаты соединения. Это предикаты вида **Ri.Ck op Rj.Cn,**
где Ri != Rj и **op** - оператор скалярного сравнения.

4) Предикаты деления. Это предикаты вида **Qi op Qj,**
где Qi и Qj - блоки запросов, а **op** может быть оператором скалярного сравнения или оператором проверки вхождения в множество.

Каноническим представлением запроса на ***n*** отношениях называется запрос, содержащий n-1 предикат соединения и не содержащий предикатов со вложенными подзапросами (классов 2 и 4).

Логические преобразования

Предикаты с вложенными подзапросами типа A. Пример:

```
select Ri.Ck from Ri where Ri.Cn =
        (select max (Rj.Cm) from Rj where Rj.Cl > a);
```

Единственно разумными способом выполнения подобного запроса является изолированное вычисление подзапроса и сведение тем самым предиката со вложенным подзапросом к простому.

Предикаты с вложенными подзапросами типа N. Пример:

```
select Ri.Ck from Ri where Ri.Cn in
        (select Rj.Cm from Rj where Rj.Cl > a);
```

Аналогично предыдущему случаю.

Предикаты с вложенными подзапросами типа J содержат предикаты соединения с отношением внешнего блока и не содержат агрегатных функций**. Пример:**

```
select Ri.Ck from Ri where Ri.Cn in
        (select Rj.Cm from Rj where Rj.Cr = Ri.Ch and Rj.Cl > a);
```

Предикаты с вложенными подзапросами типа JA содержат предикаты соединения с отношением внешнего блока и агрегатные функции**. Пример:**

```
select Ri.Ck from Ri where Ri.Cn =
        (select avg (Rj.Cm) from Rj
                where Rj.Cr = Ri.Ch and Rj.Cl > a);
```

Логические преобразования

Среди предикатов класса 4 выделяется подкласс таких предикатов, вложенные блоки которых Qi или Qj (или и тот, и другой) содержат предикаты соединения с отношением внешнего запроса. Обозначим этот подкласс – D.
Примером запроса с предикатом типа D может быть:

```
select Ri.Ck from Ri where
       (select Rj.Cm from Rj where Rj.Cn = Ri.Cl)
              contains (select Rp.Cm from Rp where Rp.Cr > a);
```

Логические преобразования

Приведение к канонической форме запросов с предикатами типов N и J, содержащих оператор проверки вхождения в множество IN основано на следующей лемме:

Пусть запрос Q1 - это

```
select   Ri.Ck  from   Ri, Rj  where  Ri.Cn = Rj.Cm
```

и запрос Q2 - это

```
select  Ri.Ck  from  Ri  where   Ri.Cn  in
        (select   Rj.Cm  from  Rj )
```

Тогда запросы Q1 и Q2 эквивалентны, т.е. дают одинаковый результат.

При этом предполагается, что если каноническое представление запроса на n отношениях получено из запроса с N или J предикатами с операциями сравнения IN, то до выполнения соединений производится необходимая фильтрация и проецирование отношений, соответствующих внутреннему подзапросу с удалением возможных дубликатов. Только при таком выполнении преобразованного запроса его семантика будет совпадать с семантикой запроса в начальном виде.

Логические преобразования

Примером преобразования запроса с предикатом класса J к канонической форме может быть следующий:

```
select   Ri.Ck  from  Ri   where   Ri.Ch   IS IN
        select   Rj.Cm   from  Rj   where   Ri.Cn = Rj.Cp
```

эквивалентно

```
select   Ri.Ck   from   Ri, Rj
        where  Ri.Ch = Rj.Cm and Ri.Cn = Rj.Cp
```

Сложнее с предикатами типов N и J, в которых используется оператор проверки невхождения во множество NOT IN. Например, запрос

```
select   Ri.Ck   from   Ri   where   Ri.Ch  NOT IN
                (select   Rj.Ch   from   Rj   where   Rj.Cn = a )
```

не эквивалентен запросу

```
select   Ri.Ck   from  Ri   where   Ri.Ch  IN
                (select   Rj.Ch   from   Rj   where   Rj.Cn != Rj.Cp.).
```

Потенциально лучшим способом выполнения такого запроса является выработка временного отношения X, соответствующего внутреннему подзапросу, сортировка обоих отношений по Ch и затем одновременное сканирование отсортированных файлов с тем, чтобы найти кортеж отношения Ri, не входящий в X.

Логические преобразования

Рассмотренные возможные преобразования обобщаются в следующем алгоритме NEST-N-J, преобразующем запросы с предикатами типов N и J к "канонической" форме (понятно, что ее можно считать только псевдоканонической, поскольку отсутствует единая семантика предикатов соединения):

1. Объединить все списки отношений, встречающихся во всех разделах FROM, в один список FROM.
2. Объединить через AND логические условия всех разделов WHERE в одно логическое условие.
3. Заменить предикаты вида Ri.Cn op (select Rj.Cm) на предикаты Ri.Cn op Rj.Cm, если op отлично от IS IN и IS NOT IN; на предикаты Ri.Cn = Rj.Cm, если op - это IS IN; на предикаты NOT(Ri.Cn = Rj.Cm), если op - это IS NOT IN.
4. Оставить список выборки внешнего запроса.

Логические преобразования

Преобразования запросов с предикатами типа JA основывается на наблюдении, что запрос Q3

select Ri.Ck from Ri where Ri.Ch =

(select AGG (Rj.Cm) from Rj where Rj.Cn = Ri.Cp)

эквивалентен запросу Q4

select Ri.Ck from Ri where Ri.Ch =

(select Rt.C2 from Rt where Rt.C1 = Ri.Cp,

где Rt (C1, C2) =(SELECT Rj.Cn, AGG (Rj.Cm) FROM Rj GROUP BY Rj.Cn).

Поскольку запрос Q4 содержит предикат типа J, он может быть преобразован к канонической форме с помощью алгоритма NEST-N-J.

В более общем случае для запроса вида

select R1.Cn+2 from R1 where R1.Cn+1 op

(select AGG (R2.Cn+1) from R2 where

R2.C1 = R1.C1 AND ... AND R2.Cn=R1.Cn

можно применить алгоритм преобразования NEST-JA.

Логические преобразования

Алгоритм преобразования NEST-JA.

- Генерируется временное отношение Rt(C1,..., Cn, Cn+1), соответствующее запросу

  select C1,..., Cn, AGG (Cn+1) from R2 group by C1,..., Cn.

- Внутренний блок запроса в начальной форме преобразуется путем замены всех вхождений имен полей отношения R2 на соответствующие имена полей отношения Rt; идентификатор агрегатной функции удаляется.
- Алгоритм естественным образом обобщается на случай произвольной глубины вложенности внутренних подзапросов.

Описанный алгоритм не является вполне корректным, поскольку искажает семантику запроса с предикатом, включающим агрегатную функцию COUNT.

Например, запрос:

select Ri.Ck from Ri where Ri.Ch =

(select count (Rj.Cm) from Rj where Rj.Cn = Ri.Cp)

на самом деле не эквивалентен запросу

select Ri.Ck from Ri, Rt where Ri.Ch = Rt.Cm AND Ri.Cp = Rt.Cn,

где Rt (Cp, Cn) = (select Rj.Cp, count (Rj.Cn) from Rj group by Rj.Cp).

Логические преобразования

- Преобразования запросов, содержащих предикаты типа D, основаны на следующей лемме:

Пусть Q5 есть запрос

```
select   Ri.Ck  from  Ri
          where    (select  Rj.Ch  from  Rj
                    where   Rj.Cn = Ri.Cp)  op (select  Rk.Cm  from  Rk),
```

а Q6 - запрос

```
select Ri.Ck from Ri where Ri.Cp = (select C1 from Rt),
```

где Rt (C1) определяется запросом

```
select   Rj.Cn   from   Rj Rx
          where    (select   Rj.Ch   from   Rj RY
                    where RY.Cn = RX.Cn) op  (select   Rk.Cm   from   Rk).
```

Запросы Q5 и Q6 эквивалентны.

Запрос Q6 содержит предикат типа N, и, следовательно, может быть преобразован к канонической форме. Запрос, определяющий отношение Rt, является формулировкой на SQL реляционной операции деления.

Семантическая оптимизация: работа с представлениями

Примеры подходов к преобразованию запросов на основе семантической информации в известных СУБД System R и INGRES:

Материализация представлений: предварительная и непосредственная. Слияние запроса к представлению с определяющим запросом. Хранение представлений в каталогах базы данных во внутренней форме, получаемом после выполнения грамматического разбора запроса, определяющего данное представление.

Пример: если представление определено с использованием SQL как

```
DEFINE VIEW V (C1, C2) AS
        SELECT C3, AVG (C4) FROM R GROUP BY C3;
```

а запрос имеет вид:

```
SELECT C2, AVG (C1) FROM V GROUP BY C2;
```

то результат слияния внутренней формы запроса с внутренней формой представления, не соответствует никакому запросу на SQL над хранимым отношением.

Влияние подходов к обработке представлений на оптимизацию.

Семантическая оптимизация: работа с представлениями

1) Пример представления:

define view V (C2) AS select C1 from R where C1 > 6;

Запрос формулируется так: select C2 from V where C2 < 6;

Предварительная материализация и слияние запроса с представлением.

2) Немного более сложный пример:

define view V (C3) AS select C2 from R where C1 = 6;

Задается запрос: select C3 from V where C3 < 16;

Если отношение R кластеризовано по значениям поля C1, то слияние запроса с представлением дает более эффективный план выполнения запроса.

Семантическая оптимизация имеет дело со знаниями, представленными в виде ограничений целостности БД. Тогда при семантической оптимизации производится множество преобразованных внутренних представлений запроса, и каждое преобразование использует некоторый поднабор ОЦ. Если, например, в БД определены два ограничения целостности A и B с логическими условиями F1 и F2, и обрабатывается запрос с логическим условием выборки F, то в ходе семантической оптимизации будут получены внутренние представления, эквивалентные запросам с условиями выборки **F, F AND F1, F AND F2 и F AND F1 AND F2**.

Некоторые проблемы синтаксических и семантических преобразований

- Цепочки преобразований.
- Циклы преобразований.
- Запросы, встроенные в программу на языке программирования, которые потенциально в будущем будут выполняться много раз.
- Запросы, обрабатываемые в интерактивном режиме.
- Эвристики. Пример:

Оптимизация производится до тех пор, пока затраты на нее не превзойдут оценочную стоимость наиболее эффективного из всех найденных планов выполнения запроса.

Выбор и оценка альтернативных планов выполнения запросов

Будем называть процедурным представлением или планом выполнения запроса такое его представление, в котором в детализированной форме отображен порядок выполнения операций доступа к базе данных физического уровня.

Как правило, в реляционных СУБД, ориентированных на использования традиционной аппаратуры без использования специализированных процессоров или устройств внешней памяти, выделяется подсистема управления данными на физическом уровне. Например, в System R, такая подсистема называется RSS (Relational Storage System) и обеспечивает простой интерфейс, позволяющий последовательно просматривать кортежи отношений, удовлетворяющие заданным условиям на значения полей, с использованием индексов или простым сканированием страниц базы данных. Кроме того, RSS позволяет производить отсортированные временные файлы и заносить, удалять и модифицировать индивидуальные кортежи. Аналогичные подсистемы явно или неявно выделяются во всех подобных СУБД.

Выбор и оценка альтернативных планов выполнения запросов

Две задачи:

1) Исходя из получаемого после проведения логической оптимизации внутреннего представления запроса и информации, характеризующей управляющие структуры базы данных (например, индексы), выбрать набор потенциально возможных планов выполнения данного запроса.
2) Оценить стоимость выполнения запроса в соответствии с каждым альтернативным планом и выбрать план с наименьшей стоимостью.

При традиционном подходе к организации оптимизаторов обе задачи решаются на основе фиксированных встроенных в оптимизатор алгоритмов.

Пример:

```
select   NAME   from   EMP
            where   DEPTNO = 6 and SALARY > 45.000 ;
```

Алгоритмы: последовательное чтение данных таблицы EMP, сканирование таблицы через индекс по полю DEPTNO (условие равенства константе), сканирование таблицы через индекс по полю SALARY (открытый интервал).

Наиболее эффективно выполняются унарные операции – селекция, проекция.

Наиболее проблемными с точки зрения эффективности являются бинарные операции: ДП, соединение, разность, объединение и пересечение.

Алгоритмы выполнения запросов соединения

Основные алгоритмы выполнения операций эквисоединения в System R:

1. Алгоритм последовательного сканирования соединяемых отношений (метод вложенных циклов, NESTED LOOPS).
2. Алгоритм сортировка-слияние (SORT-MERGE): предварительная сортировка обоих отношений в соответствии со значениями полей соединения.
3. Алгоритм, основанный на использовании некластеризованных индексов обоих отношений на полях соединения. Путем сканирования только индексов производится список пар (TID1, TID2) таких, что TID1 и TID2 определяют соединяемые кортежи первого и второго отношений. Далее этот список сортируется и используется для реального выполнения соединения.
4. Иногда для ускорения выполнения бинарных операций предлагаются новые типы управляющих структур, например, **индекс соединения**. В общем виде, если f (R.C1, S.C2) - предикат соединения отношений R и S по полям C1 и C2, то индекс соединения R и S по f определяется как бинарное отношение JI = { (TIDi, TIDj) | f (кортеж (TIDi).C1, кортеж(TIDj).C2) = true }. (Поля C1 и C2 могут быть составными, а f - не обязательно предикат эквисоединения.)

Алгоритм соединения хешированием

Меньшая из двух входных таблиц помещается в специальную структуру данных в памяти: хеш-таблицу, которая обеспечивает очень высокую скорость поиска. Затем для каждой строки из большей таблицы выполняется поиск значений, соответствующих условию соединения. Результаты помещаются в выходную таблицу.

Выбор плана выполнения запроса

В традиционных оптимизаторах запросов есть фиксированные стратегии, на основе которых вырабатываются планы выполнения запросов. Соответствующие компоненты оптимизаторов имеют достаточно сложную организацию; генерация плана выполнения сложного запроса - это многоэтапный процесс, в ходе которого учитываются свойства создаваемых при выполнении запроса по данному плану временных объектов базы данных. Например, пусть запрос задан над тремя отношениями и в нем имеются два предиката соединения:

```
select   R1.C1, R2.C2, R3.C3   from   R1, R2, R3
            where     R1.C4 = R2.C5   AND   R2.C5 = R3.C6;
```

Тогда, если в плане запроса выбирается порядок выполнения соединений сначала R1 с R2, а затем полученного временного отношения - с R3, и при этом для выполнения первого соединения выбирается метод сортировок со слиянием, то временное отношение будет заведомо отсортировано по C5, и одна сортировка не потребуется, если и второе соединение будет выполняться тем же методом.

Выбор плана выполнения запроса

Компонент оптимизатора, ведающий порождением множества альтернативных планов выполнения запроса, базируется на стратегиях **декомпозиции запроса** на элементарные составляющие и **стратегиях выполнения элементарных составляющих**.

Первая группа стратегий: обеспечивает пространство поиска оптимального плана выполнения запроса.

Вторая: направлена на то, чтобы в этом пространстве действительно находились эффективные планы выполнения запроса.

Наличие эффективных стратегий выполнения элементарных составляющих.

Проблема обоснованного выбора плана выполнения запроса из множества альтернативных планов.

Критерии оптимизации:

- время выполнения запроса (реактивность системы);
- общая пропускная способность системы по отношению к смеси параллельно выполняемых запросов;
- бюджетная стоимость выполнения запроса и т.д.

Выбор плана выполнения запроса

Стоимость плана выполнения запроса.

Основные ресурсы, которые расходуются при выполнении запроса:

- ресурсы процессора;
- ресурсы устройств внешней памяти;
- сетевые ресурсы (для распределенных БД).

Буферизация блоков базы данных в оперативной памяти.

Компонент стоимости выполнения запроса, который связан с ресурсами устройств внешней памяти, монотонно зависит от числа блоков внешней памяти, доступ к которым потребуется при выполнении запроса.

Число блоков внешней памяти, доступ к которым требуется при выполнении запроса, монотонно зависит от числа кортежей, затрагиваемых запросом.

Степень селективности предиката **R.C op const** зависит от: вида операции сравнения, значения константы и от распределения значений поля C отношения R.

Соотношение мощности результирующего отношения и числа блоков. Сложности оценки мощности соединения отношений, объединения, разности и т.п.

Оценка стоимости плана выполнения

Подход System R базируется на двух основных предположениях о распределениях значений атрибутов отношений:

1) Значения полей всех отношений базы данных распределены равномерно.
2) Значения любых двух полей распределены независимо.

В ССД для каждого отношения R поддерживается следующая информация:

T – число кортежей в данном отношении;

N – число блоков внешней памяти, в которых располагаются кортежи.

Для каждого поля отношения R:

CD – число различных значений этого поля;

CMin, CMax – минимальное и максимальное хранимое значение этого поля.

Для более точной оценки доступа к отношению через индексы для каждого индекса хранится число уровней этого индекса и число листовых страниц.

Вычисление степени селективности простых предикатов:

Обозначим степень селективности предиката P как SEL (P). Тогда

SEL (R.C = const) = CD / (CMax - CMin)

SEL (R.C > const) = (CMax - const) / (CMax - CMin)

SEL (R.C < const) = (const - CMin) / (CMax - CMin)

При этом SEL (R.C <= const) полагается равной SEL (R.C < const),

а SEL (R.C >= const) равной SEL (R.C > const).

Оценка стоимости плана выполнения

Оценка числа блоков, в которых могут располагаться кортежи, удовлетворяющие предикату R.C op const:

Различаются два случая: отношение кластеризовано по полю C или не кластерозовано по этому полю. Если отношение кластеризовано, то селективность предиката по отношению к кортежам распространяется и на блоки, занимаемые этими кортежами. Например, если в блоке внешней памяти базы данных размещается K кортежей отношения R (параметр K в System R определяется при создании отношения и не изменяется во все время его существования), то число блоков кластеризованного отношения, содержащих кортежи, удовлетворяющие предикату R.C op const, определяется по формуле

$$T * SEL (R.C \text{ op const}) / K$$

При оценках числа блоков, в которых могут располагаться кортежи отношения, удовлетворяющие предикату R.C op const, в случае, когда отношение не кластеризовано по полю C, при разработке начального варианта оптимизатора исходили из того, что в этом случае кортежи могут быть произвольным образом разбросаны по блокам базы данных, содержащим кортежи данного отношения. Поэтому оценка производилась по формуле

$$T * SEL (R.C \text{ op const})$$

Однако впоследствии было замечено, что эта формула дает завышенный результат, если число кортежей отношения, удовлетворяющих предикату, достаточно велико (т.е. селективность предиката низкая).

Оценка стоимости плана выполнения

Записи листовых блоков индекса отношения R на поле C в System R имеют следующую структуру: значение ключа, список уникальных идентификаторов кортежей отношения R, у которых поле C имеет тоже значение, что и ключ. Уникальный идентификатор кортежа (tid) обеспечивает прямой доступ к кортежу и содержит, в частности, номер блока, в котором располагается кортеж.

Если при организации списка tid'ов в записи индекса поддерживать в каждом списке упорядоченность tid'ов в соответствии с номерами блоков, содержащих соответствующие кортежи, то при последовательном просмотре кортежей с фиксированным значением ключа появляется некоторый элемент порядка. В уточненном варианте оптимизатора System R для оценки числа блоков, обращения к которым должны произойти при таком просмотре, используется формула

$$N * (1 - (1 - 1/N) ** CD)$$

Эта формула была впервые выведена и обоснована Яо [Yao S.B. Approximating Block Acess in Database Organizations // Commun. ACM.- 1977.- 20, N 4.- C. 260-261]. Понятно, что на основе этой формулы можно оценить и число блоков, обращения к которым потребуются при сканировании отношения через индекс для ограничения отношения в соответствии с предикатом с известной (оцененной ранее) селективностью.

Заметим, что и в этом подходе к оценке числа блоков с жестким разделением случаев кластеризованности отношения R по полю C и отсутствия такой кластеризованности проявляется некоторая ограниченность System R (предположение System R о независимости распределений значений разных полей отношения). Существуют подходы к более адекватному учету реально существующих корреляций атрибутов отношений.

Оценка стоимости плана выполнения

Оценка мощности отношений, являющихся результатами двухместных реляционных и теоретико-множественных операций над отношениями.

Предикат эквисоединения R1.C1 = R2.C2. Пусть Ti и CDi - число кортежей в отношении Ri и число различных значений поля Ci, соответственно. Тогда в отношении Ri существует Ti/CDi групп кортежей с одинаковыми значениями поля Ci. Очевидно, что при выполнении операции соединения кортежи каждой группы кортежей отношения R1 могут быть соединены с кортежами не более, чем одной группы отношения R2 (и наоборот). Если кортежи двух групп соединяются, то в результирующем отношении образуется (T1/CD1) * (T2/CD2) кортежей. (Оценка числа кортежей в каждой группе в виде Ti/CDi правомерна в соответствии с предположением о равномерности распределения). Очевидно, что соединиться могут не более, чем Min (CD1, CD2) групп. Поэтому верхней оценкой мощности результирующего отношения может быть

Min (CD1, CD2) * (T1/CD1) * (T2/CD2)

Соответственно, степень селективности предиката эквисоединения можно оценить, как

Min (CD1, CD2) / (CD1 * CD2)

Оценка стоимости плана выполнения

Оценка мощности отношений, являющихся результатами двухместных реляционных и теоретико-множественных операций над отношениями (продолжение).

Получим оценку степени селективности предиката соединения R1.C1 > R2.C2. Обозначим через CiMax и CiMin, соответственно, максимальное и минимальное значения поля Ci. Пусть кортежи из j-ой группы кортежей отношения R1 имеют значение поля C1, равное cj. Тогда число кортежей отношения R2, удовлетворяющих предикату соединения относительно j-ой группы отношения R1 можно оценить как

$$((C2Max - cj) / (C2Max - C2Min)) * T2$$

Число кортежей в результирующем отношении, возникающих при соединении j-ой группы кортежей отношения R1 с кортежами отношения R2 оценивается как

$$(T1/CD1) * ((C2Max - cj) / (C2Max - C2Min)) * T2$$

Для получения оценки общего числа кортежей в результирующем отношении необходимо просуммировать полученную формулу по j для всех групп отношения R1. При этом можно воспользоваться тем, что в силу предположения о равномерности распределения значений поля C1 сумму cj можно оценить как

$$CD1 * AVG (cj) = CD1 * (C1Max - C1Min) / CD1 = C1Max - C1Min$$

Результирующая формула для оценки мощности результирующего отношения:

$$T1 * T2 * (C2Max - C1Max + C2Min) / (C2Max - C2Min)$$

Соответственно, селективность предиката оценивается по формуле

$$(C2Max - C1Max + C2Min) / (C2Max - C2Min)$$

Оценка стоимости плана выполнения

Пример:

```
select  emp.Empname,  dept.Deptname   from   emp,  dept
        where  emp.Salary = 20.000  and  dept.Dept# > 4  and  emp.Dept# = dept.Dept#
```

План выполнения:

1) выполнить ограничение отношения EMP с использованием некластеризованного индекса на поле EMP.SALARY и порождением временного отношения EMP1 (ОП1);
2) выполнить ограничение отношения DEPT с использованием кластеризованного индекса на поле DEPT.DEPT# и порождением временного отношения DEPT1 (ОП2);
3) отсортировать отношение EMP1 в соответствии со значениями поля EMP.DEPT# с порождением отсортированного файла EMP2 (ОП3);
4) отсортировать отношение DEPT1 в соответствии со значениями поля DEPT.DEPT# с порождением отсортированного файла DEPT2 (ОП4);
5) соединить файлы EMP2 и DEPT2 по полю DEPT# (ОП5).

Стоимость элементарных операций ОП1 и ОП2 оценивается на основе формул и статистической информации об отношениях EMP и DEPT. Стоимость элементарных операций ОП3, ОП4 и ОП5 оценивается на основе формул и оценок операций ОП1 и ОП2 (мощности отношений EMP1 и DEPT1). Оценка общей стоимости плана выполнения запроса складывается из оценок для ОП1, ОП2, ОП3, ОП4 и ОП5.

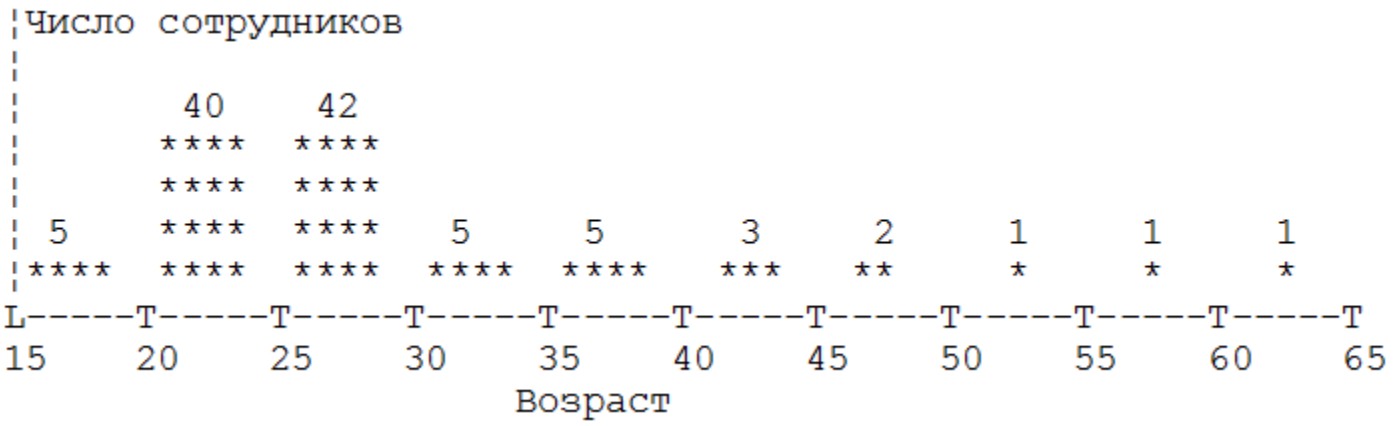
Поддержание статистической информации об отношениях БД: эта информация не корректируется при каждом изменении базы данных. В System R существует специальная утилита сбора статистики, которая и производит соответствующие коррекции либо периодически, либо по явному указанию оператора.

Оценка стоимости плана выполнения

Отказ от предположения о равномерности распределения значений поля отношения.

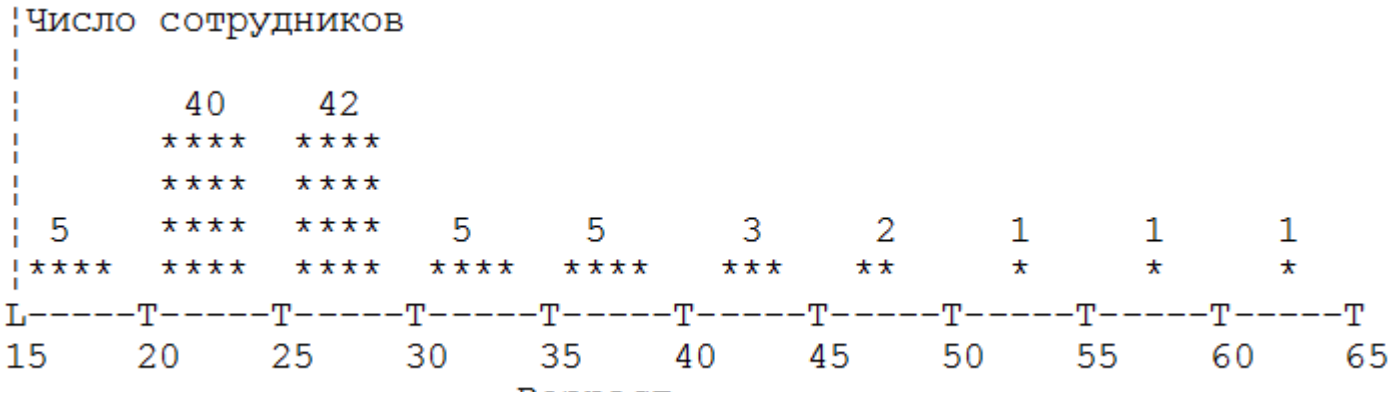
Два базовых подхода к оценкам распределения значений поля отношения: параметрический и основанный на методе сигнатур. С.Христодолакис [Christodoulakis S. Estimating Block Selectivities // Inf. Syst.- 1984.- 9, N 1.- С. 69-79] предложил использовать для оценки реального распределения значений поля отношения серию распределений Пирсона, в которую входят распределения от равномерного до нормального. Выбор распределения из серии производится путем вычисления нескольких параметров на основе выборок реально встречающихся значений. Метод оценки распределения на основе сигнатур в общих словах можно описать следующим образом. Область значений поля разбивается на несколько интервалов. Для каждого интервала некоторым образом устанавливается число значений поля, попадающих в этот интервал. Внутри интервала значения считаются распределенными по некоторому фиксированному закону.

Построение гистограмм:



Оценка стоимости плана выполнения

Построение гистограмм и псевдогистограмм:



Оценка стоимости плана выполнения

Вычислительная сложность построения гистограмм и псевдогистограмм.

Получение достоверной псевдогистограммы без необходимости сортировки всего отношения:

Подход основывается на **статистике Колмогорова**: если из отношения выбирается образец из 1064 кортежей, и b - доля кортежей в образце со значениями поля C < V, то с достоверностью 99% доля кортежей во всем отношении со значениями поля

C < V находится в интервале [b-0.05, b+0.05].

При уменьшении мощности образца достоверность, естественно, уменьшается.

Оценка числа блоков внешней памяти, обращения к которым потребуются при выполнении запроса.

Основным предположением, на котором основываются оценки System R, является предположение о независимости распределений значений различных полей отношения. При этом по-разному оценивается число блоков, обращения к которым потребуются при сканировании отношения без использования индекса, при сканировании через некластеризованный и кластеризованный индексы.

Отказ от предположения о независимости распределений значений разных полей отношения. Особенности учета кластеризованных данных.

Оценка стоимости плана выполнения

Работа оптимизатора в системе Oracle.

Для таблицы статистика может включать в себя:

- общее количество блоков данных, выделенных таблице;
- количество пустых блоков данных;
- количество записей в таблице;
- среднюю длину записи в таблице;
- среднее количество записей на блок памяти;
- количество мигрировавших строк.

Для каждого индекса статистика может содержать такие данные, как:

- глубина индекса;
- количество листовых блоков;
- количество различных ключей;
- среднее количество листовых блоков на ключ;
- среднее количество блоков данных на ключ;
- количество узлов индекса;
- фактор кластеризации (количество блоков, которое надо выбрать для выборки всех строк из таблицы по индексу).

Статистика по столбцу:

- количество различных значений;
- минимальное значение;
- максимальное значение;
- количество *null*-значений.

Оценка стоимости плана выполнения

Оптимизация запросов в системе Oracle.

Для хранения распределения значений по *диапазонам*. используются *гистограммы*. Существует два типа гистограмм - частотные гистограммы и сбалансированные по высоте. Максимальное количество брикетов в гистограмме - 254. Если количество различных значений в столбце меньше указанного размера гистограммы, то строится *частотная* гистограмма. В противном случае - *сбалансированная по высоте*.

Сбалансированные по высоте гистограммы содержат одинаковое количество строк в каждом брикете гистограммы. Значение *ячейки* соответствует максимальному значению столбца в группе. Частотные гистограммы содержат столько брикетов, сколько есть различных значений в столбце. Значение ячейки содержит количество строк со значениями столбца, меньше либо равными данному.

Примеры:

analyze table T compute statistics for columns S size 254;

analyze table T compute statistics for table for all indexes for all indexed columns;

Оптимизация распределенных запросов

Особенности распределенных запросов

Будет рассматриваться класс распределенных баз данных, основанных на однородных локальных системах управления базами данных (например, распределенная СУБД System R* или INGRES).

Обязательное требование: локальная автономность узлов сети, на которых выполняются локальные СУБД. Это требование означает:

- отсутствие централизованного администрирования распределенной БД,
- отсутствие централизованного каталога базы данных,
- наличие возможности локального порождения и удаление новых отношений и индексов в локальной базе данных, причем уничтожение индекса не должно приводить к невозможности повторного выполнения ранее откомпилированных глобальных запросов.

Требование локальной автономности существенно влияет на допустимые способы оптимизации глобальных запросов (здесь и далее под глобальным запросом понимается запрос, для выполнения которого требуется сетевой доступ к удаленным локальным базам данных).

Логические уровни оптимизации запросов фактически не связаны с распределенной или локальной природой БД (не считая некоторой специфики обработки запросов через представления). Распределенный характер БД затрагивает главным образом "физические" уровни оптимизации, связанные с выбором и оценкой планов выполнения запроса.

Основные проблемы оптимизации распределенных запросов

Самое большое влияние на эффективность выполнения распределенных запросов оказывает выбор способа реализации (и алгоритма) двуместных реляционных операций над отношениями, хранящимися в разных узлах распределенной системы.

Эта проблема существует и при традиционном хранении каждого отношения целиком в одной локальной базе данных, но становится еще более сложной и допускающей большее число решений в случае так называемых разделенных (partitioned) и раскопированных (replicated) баз данных.

Если данные являются распределенными, стоимость пересылки данных становится переменной, а не константой проблемы оптимизации запросов.

Есть тенденция к преобладанию стоимости коммуникаций над стоимостью локальной обработки для оптимизации запросов требуется другая целевая функция - минимизация коммуникационных задержек, часто представляемая объемом данных, передаваемых из одного узла в другой.

В самой общей постановке задачей компонента распределенной СУБД, оптимизирующего выполнение глобального запроса, является **генерация множества альтернативных планов** выполнения запроса и **выбора** из этого множества на основе некоторых критериев одного плана, на основе которого и производится выполнение запроса.

Подход на основе предварительной компиляции запросов

Этот подход используется, например, в System R* и состоит в том, что фазы порождения выполняемого плана глобального запроса и его реального выполнения разнесены во времени. Это позволяет заранее откомпилировать программу с глобальными запросами на SQL, а затем много раз выполнять ее без необходимости каждый раз вырабатывать план выполнения запросов.

В результате компиляции запроса в System R*, инициированной в некотором узле сети, порождается распределенная программа выполнения этого запроса, которая хранится в распределенной форме. В каждом узле сети, локальная БД которого содержит отношения, затрагиваемые запросом, хранится часть распределенной программы, осуществляющая доступ к локальным данным этого узла и взаимодействующая с другими узлами, содержащими части той же распределенной программы. Выполнение запроса начинается с запуска "главной" части распределенной программы, хранящейся в том узле, в котором инициировалась компиляция запроса ("главном" узле). Эта программа вызывает другие части распределенной программы, хранящиеся в "дополнительных" узлах и т.д. Результат выполнения запроса формируется в главном узле, хотя промежуточные результаты могут быть распределены между другими локальными БД.

Оптимизация распределенных запросов в System R*

Задача оптимизации запросов та же: необходимо построить оптимальный план выполнения запроса в условиях локальной автономности узлов сети.

Основой обработки запроса в System R* является поддержание **распределенного каталога** глобальной базы данных. За счет наличия правил именования объектов глобальной базы данных и специальных протоколов доступа к локальным каталогам баз данных при обработке запроса можно получить достоверную информацию о всех затрагиваемых запросом объектах базы данных. После этого можно было бы генерировать полные распределенные планы выполнения запроса и выбирать из них оптимальный в том узле, в котором начата обработка запроса. Но это противоречит принципу локальной автономности узлов сети.

Например, предположим, что в построенном детальном плане выполнения запроса предполагается сканирование некоторого удаленного отношения R с использованием индекса I. Во время выполнения это сканирование должно производиться в локальной СУБД, база данных которой содержит отношение R. В соответствии с требованием локальной автономности локальный администратор этой базы данных может уничтожить индекс I, и тогда для того, чтобы привести выполняемый план в корректное состояние, потребуется взаимодействие с главным узлом, что противоречит требованиям локальной автономности.

Оптимизация распределенных запросов в System R*

Поэтому при обработке запроса в главном узле генерируются не детальные, а так называемые глобальные планы выполнения запросов.

Каждый **глобальный план** соответствует отдельной схеме межузловых взаимодействий при выполнении запроса. В нем определяются:

- узлы, в которых должны выполняться соединения удаленных отношений,
- методы и порядок передачи кортежей между узлами.

Но глобальный план не предписывает правил выполнения локальных реляционных операций в узлах, включаемых в выполнение запроса.

В любом случае порождается **множество альтернативных планов**, которые необходимо оценивать. Для оценок используется информация распределенного каталога. По существу, необходимо оценить мощности промежуточных отношений, порождаемых в удаленных узлах при выполнении локальных частей запроса.

В отличие от случая локальной СУБД, оценочные формулы глобальных планов в основном базируются на оценках **сетевых накладных расходов** (но могут также учитывать и затраты на чтение данных и выполнение локальных операций).

Оптимизация распределенных запросов в System R*

После порождения множества альтернативных глобальных планов запроса, вычисления оценок их стоимости и выбора наиболее дешевого завершается первая фаза оптимизации глобального запроса и начинается следующая фаза – построение **детального распределенного плана**.

Для этого производится **декомпозиция исходного запроса** в соответствии с выбранным глобальным планом на компоненты. Каждая компонента содержит:

- некоторый локальный подзапрос исходного запроса в непроцедурной форме и
- процедурную часть, предписывающую порядок сетевых взаимодействий.

Полученные компоненты рассылаются по сети в соответствующие локальные СУБД, в каждой из которых осуществляется генерация альтернативных **локальных планов выполнения** подзапроса и выбор наиболее дешевого из них в соответствии с **локальными критериями оценок стоимости**.

Оптимизация распределенных запросов в System R*

На общем уровне процесс компиляции можно разбить на следующие фазы.

1. В главном узле производится грамматический разбор оператора SQL с построением внутреннего представления запроса в виде дерева. На основе информации из локального каталога главного узла и удаленных каталогов дополнительных узлов производится замена имен объектов, фигурирующих в запросе, на их системные идентификаторы.
2. В главном узле генерируется глобальный план выполнения запроса, в котором учитывается лишь порядок взаимодействий узлов при реальном выполнении запроса. Для выработки глобального плана используется расширение техники оптимизации, применяемой в System R. Глобальный план отображается в преобразованном соответствующим образом дереве запроса.
3. Если в глобальном плане выполнения запроса участвуют дополнительные узлы, производится его декомпозиция на части, каждую из которых можно выполнить в одном узле (например локальная фильтрация отношения в соответствии с заданным в условии выборки предикате ограничения). Соответствующие части запроса (во внутреннем представлении) рассылаются в дополнительные узлы.

Оптимизация распределенных запросов в System R*

(продолжение)

4. В каждом узле, участвующем в глобальном плане выполнения запроса (главном и дополнительных), выполняется завершающая стадия выполнения компиляции. Эта стадия включает, по существу, две последние фазы процесса компиляции запроса в System R: оптимизацию и генерацию машинных кодов. Производится проверка прав пользователя, от имени которого производится компиляция, на выполнение соответствующих действий; происходит обработка представлений базы данных; осуществляется локальная оптимизация обрабатываемой части запроса в соответствии с имеющимися индексами; наконец, производится генерация кода.

Оптимизация распределенных запросов в System R*

Порядок сетевых взаимодействий является заранее предписанным, и это является граничным условиям выбора альтернативных локальных планов. Локальной СУБД не требуется учитывать стоимость сетевых накладных расходов - она одна и та же для всех возможных локальных планов.

Если запрос производится над представлениями, то раскрытие представления производится в том узле, в котором оно определялось, и в общем случае, если это представление определено над несколькими удаленными отношениями, дополнительный узел выступает для своего подзапроса как главный, т.е. вырабатывает глобальный план выполнения подзапроса и рассылает его компоненты дополнительным узлам следующего уровня.

Основная проблема остается той же, что и при оптимизации в локальной СУБД, – точность оценок селективности простых предикатов. При использовании для оценок селективности методов, основанных на гистограммах, для выбора глобального плана выполнения запроса могут потребоваться дополнительные сетевые накладные расходы. Поэтому оценки селективность основаны на предположениях о равномерности и независимости распределений значений полей отношений, что резко упрощает получение оценок.

Стратегии выполнения соединений в РБД

Рассматривается соединение двух удаленных отношений S и T, расположенных в узлах 1 и 2, соответственно, под управлением предиката эквисоединения S.C1 = T.C2. Результат соединения должен быть получен в узле 1.

Предполагается также следующее:

- ни одно из отношений не кластеризовано по полям соединения;
- отсутствуют индексы на полях, отличных от полей соединения;
- в запросе не требуется проекция (т.е. кортежи результата включают все поля S и T);
- в запросе отсутствуют предикаты ограничения отношений S и T;
- отношения S и T хранятся в отдельных блоках внешней памяти (любой блок базы данных, содержащий кортежи S или T, не содержит кортежей других отношений).

Первый алгоритм заключается в том, что отношение T целиком пересылается в узел 1 и в этом узле для него создается временный индекс на поле C2. После этого в узле 1 выбирается наилучший локальный план выполнения соединения отношения S и временной копии отношения T.

Стратегии выполнения соединений в РБД

Второй алгоритм основывается на использовании полусоединений:

- Оба отношения сортируются в соответствии со значениями полей C1 и C2 с образованием временных отсортированных файлов S' и T'.
- Производится проекция S' на C1 (уничтожаются дубликаты) и результат посылается в узел 2.
- В этом узле выполняется полусоединение T' с полученным унарным отсортированным отношением (т.е. выбираются те кортежи T', значение поля C2 которых совпадает с каким-либо значением C1 полученного списка).
- Полученное промежуточное по-прежнему отсортированное отношение T'' посылается в узел 1, где выполняется соединение S' и T'' методом слияний (оба отношения уже отсортированы) и производится окончательный результат.

При наличии индексов на полях отношений S.C1 и T.C2 может быть использован либо этот же алгоритм, либо его модификация, в которой отсутствует сортировка (используются сканирования отношений через индексы).

Стратегии выполнения соединений в РБД

Третий альтернативный алгоритм соединения двух отношений основан на использовании так называемых фильтров Блюма. В соответствии с этим алгоритмом, если индексы на полях S.C1 и T.C2 не определены, сначала в узле 1 для отношения S по полю C1 строится **фильтр Блюма**. Фильтр представляет собой битовый массив, инициализированный нулями.

- Для формирования фильтра производится сканирование отношения S, и к значению поля C1 каждого кортежа применяется хэш-функция, ставящая в соответствие этому значению позицию бита в массиве. Этот бит устанавливается в единицу.
- Сформированный фильтр посылается в узел 2. В этом узле производится сканирование отношения T и к значениям поля C2 применяется та же хэш-функция, задающая позицию соответствующего бита в фильтре. Если значение этого бита равно 1, то кортеж отношения S посылается в узел 1 в потоке кортежей T'. В этом узле выполняется соединение S и T' и формируется результат.

Наличие индексов на полях S.C1 и T.C2 позволяет модифицировать алгоритм. В частности, фильтр Блюма для отношения S можно построить в этом случае, сканируя только записи индекса без потребности обращаться к кортежам отношения.

Оптимизация наборов запросов

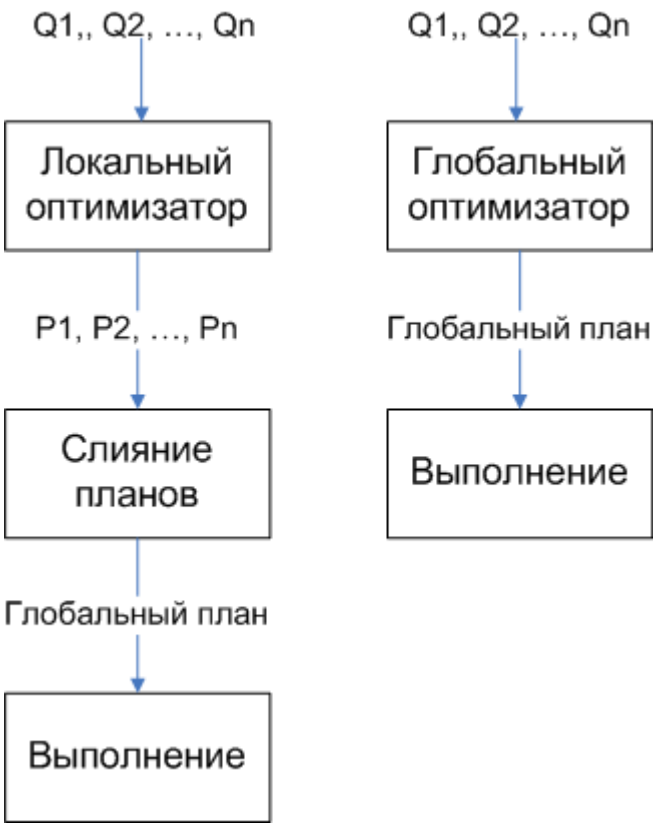
Если целью СУБД является выполнение всех запросов набора, то естественно и возможно производить **совместную оптимизацию запросов**, входящих в набор. Интуитивно ясно, что даже если для каждого запроса набора выработан оптимальный индивидуальный план выполнения, при некоторых условиях может быть сгенерирован глобальный план выполнения набора запросов, обеспечивающий большую суммарную эффективность.

В наиболее общей форме эффект глобальной оптимизации состоит в том, что действия, требуемые для выполнения более, чем одного запроса, выполняются для набора только один раз. Тем самым, основной задачей глобальной оптимизации является выявление таких общих действий и формирование глобального плана выполнения набора запросов, в котором учитывается одноразовое выполнение общих действий.

В результате глобальной оптимизации могут порождаться различные глобальные планы. Поэтому при выборе глобального плана выполнения набора запросов необходимо применять оценки альтернативных планов и выбирать оптимальный план в соответствии с принятыми в системе критериями. Переборный характер проблемы и следующая из этого сложность ее точного решения вынуждает применять уменьшающие сложность эвристики. Достоверность применяемых эвристик определяет качество глобальной оптимизации.

Оптимизация наборов запросов

Первая архитектура предполагает, что каждый запрос, входящий в группу, сначала проходит все стадии локальной оптимизации. После того, как для каждого из запросов из набора Q1, Q2,..., Qn сгенерированы оптимальные в соответствии с критериями локального оптимизатора планы выполнения P1, P2,..., Pn, в действие вступает компонент глобальной оптимизации, осуществляющий слияние локальных планов с образованием глобального плана выполнения набора запросов, в соответствии с которым производится реальное выполнение. Само применение такого подхода является эвристикой при решении проблемы глобальной оптимизации: сокращение пространства поиска при генерации глобального плана происходит за счет того, что рассматриваются фиксированные процедурные представления исходных запросов.



Оптимизация наборов запросов

Первая из приведенных архитектур соответствует базовой организации СУБД, в которой после компиляции индивидуального запроса в той или иной форме сохраняется его выполняемый план. Например, этот подход был бы естественным в System R. Если же в системе предполагается непрерывный цикл выполнения запроса, то можно применять вторую архитектуру обработки набора запросов. Эта архитектура, вообще говоря, обеспечивает большие возможности глобальной оптимизации за счет более широкого пространства поиска возможных вариантов. С другой стороны, реальна опасность существенного увеличения сложности алгоритмов и, как следствие этого, увеличения затрат на глобальную оптимизацию. В этом случае более актуальны эвристические алгоритмы.

Список литературы:

1. Кузнецов С.Д. Методы оптимизации выполнения запросов в реляционных СУБД. - http://citforum.ru/database/articles/art_26.shtml
2. Матиас Ярке, Юрген Кох. Оптимизация запросов в системах баз данных. - http://citforum.ru/database/articles/query_optimization/part2.shtml#6_2