Onderzoek naar de mogelijkheden en de performance van Oracle

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computing Science

Memorandum 1984-08

Augustus 1984

ONDERZOEK NAAR DE MOGELIJKHEDEN EN DE PERFORMANCE VAN ORACLE

door

E.H. Reitsma

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513, 5600 MB Eindhoven
The Netherlands

Onderzoek naar de mogelijkheden en de performance van Oracle

Door E.H. Reitsma

augustus 1984

Inleiding

In deze notitie worden ervaringen met het gebruik van het rationele database management system Oracle beschreven en gedeeltelijk geëvalueerd (versie Oracle (3.0.7. (rev 03)).
Dit systeem was geïmplementeerd op de VAX 11/780 van het Instituut voor Perceptie Onderzoek, Eindhoven. Deze VAX beschikt ten tijde van het gebruik over 3½ Mb werkgeheugen, 2 RM05 schijven (256 Mb) en een RM03 schijf (67 Mb).
Als test-databases werden gebruikt "BIGHOS" en "SMALLHOS", overgenomen van de DEC-20 van de afdeling Bedrijfskunde van de TH-Eindhoven. De definitie ervan staat in Remunen [4]. De definitie in Oracle termen vindt men in bijlage 1. Deze databases worden in deze notitie steeds als voorbeeld gebruikt, met name in query voorbeelden. Om storende herhalingen te voorkomen, worden verwijzigingen naar [4] of naar bijlage 1 daarbij weggelaten.
Meetgegevens hebben betrekking op SMALLHOS, tenzij anders wordt vermeld.
Er is uitsluitend gewerkt met de bovengenoemde versie van Oracle. Als hieronder Oracle genoemd wordt, dient dus gelezen te worden Oracle versie 3.0.7. (rev. 03). Het is waarschijnlijk dat in de volgende versie verbeteringen aangebracht zullen worden.

2. Logische structuur en indices

De gebruiker definieert zijn database door tabellen te specificeren met attributen. Bij elk attribuut moet een basistype (NUMBER voor getallen, CHAR voor karakter, LONG voor dubbele precisie, DATE voor data) en het maximaal aantal benodigde posities gespecificeerd worden (zie bijlage voor een voorbeeld). Tevens kan per attribuut geëist worden dat deze in elk tupel een waarde moet hebben (NOT NULL is).
Het concept "constraint" zoals bijvoorbeeld gedefinieerd in [4], komt in Oracle niet voor. Alleen op attribuutniveau wordt bij wijziging van de database getest of de nieuwe waarde is toegelaten.
De specificatie van tabellen en attributen kan men te allen tijde wijzigen. Attributen kunnen worden weggelaten of toegevoegd. (Dit laatste kan alleen aan de "rechterkant" van een tabel.) Ook gehele tabellen kunnen worden verwijderd of toegevoegd.

Ten behoeve van de query-verwerking kunnen indices worden gecreëerd op een of meer attributen van een tabel (inverted files). Ook deze kunnen altijd verwijderd worden of nieuwe bijgemaakt.
Als men een UNIQUE INDEX creëert, dan wordt bij wijzigingen van de database getest of de waarde van het betreffende attribuut uniek is (desgewenst meer één attribuut). Dit attribuut of deze attributen vormen dus een sleutel van de betreffende tabel. Het definiëren van een sleutel kan alleen op deze manier. Het is onmogelijk een sleutel te definiëren zonder een index te creëren.

3. Opslag

Naast de gewone manier van opslag kunnen tabellen ook "geclusterd" worden opgeslagen. Daarbij wordt een aantal (meer dan één) tabellen, die minstens één attribuut gemeenschappelijk hebben als 1 tabel opgeslagen, waarbij het (de) gemeenschappelijke attribuut (attributen) maar één maal wordt (worden) opgeslagen. Er ontstaat dan een tabel met "repeating groups". (Omdat deze faciliteit tijdens het experiment met Oracle nog niet probleemloos werkte, is hiervan geen gebruik gemaakt in het voorbeeld van bijlage 1.)

Tabellen worden met variabele recond- en attribuutlengte opgeslagen. "Trailing spaces" van attribuutwaarden worden niet opgeslagen. De extra ruimte die voor het opslaan van de lengtes van attributen en records benodigd is, bedraagt per record: (aantal attributen + 1) * 2 bytes. Bij de opslag wordt tussen groepen van records door een percentage van de ruimte leeggelaten ten behoeve van eventuele latere toevoegingen. Dit percentage kan door de gebruiker worden opgegeven.

Het ruimtebeslag van onze test-databases SMALLHOSS (S) en de vier grootste tabellen van BIGHOS (B) bedroeg:

	S	B (alleen de tabellen MV, OPN, P, PB)
1. opgegeven percentage vrije ruimte	20	5
2. kale data	321 kb	2955 kb
3. door Oracle gebruikte ruimte voor tabellen + overhead + vrije ruimte	700 kb	5600 kb
4. gecorrigeerd voor vrije ruimte	560 kb	5320 kb
5. verhouding tussen 4 en 2	1.74	1.80

Voor BIGHOS zijn alleen de tabellen MV, OPN, P, PB beschouwd. De overige zes tabellen zijn identiek aan de overeenkomstige tabellen van SMALLHOS.

Bij SMALLHOS werden 18 indices gecreëerd, ieder op één attribuut. Deze werden zo gekozen dat een vergelijking met de Netwerk implementatie van dezelfde databases zo eerlijk mogelijk zou zijn (zie Reitsma [2]). Elf van deze indices

betroffen de relatief grote tabellen MV, OPN, P, PB. Dit kostte nog eens 475 kb extra, zodat de totaal benodigde ruimte voor SMALLHOS 1175 kb bedroeg, 3.66 maal zoveel als de kale data. (Na correctie voor de vrije ruimte bedraagt deze factor: 2.83.)

Voor een index bleek tussen 10 en 20 bytes per tupel nodig te zijn bij voldoende grote tabellen. Het creëren van een index kostte nooit meer dan 0.007 CPUs/tupel.

Het laden van tabellen ging relatief snel. Voor BIGHOS bijvoorbeeld:

P tabel	0.045 CPUs/tupel
OPN	0.054
PB/MV	0.057

4. SQL

Queries worden geformuleerd in SQL. Er is naar gestreefd deze taal zoveel mogelijk te laten lijken op de IBM-versie van SQL.

4.1. Beschrijving van de syntax

De formele beschrijving van de syntax van SQL in de documentatie [1] en [7] is zeer gebrekkig. Nemen we het SELECT statement als voorbeeld (dit is immers "the primary clause of an SQL-query command" aldus [7], pag. 33).
De definitie luidt:
"SELECT [DISTINCT] {* column-name 1, column-name 2, column-name 3, ...}
FROM [creator 1.] table-name 1, [creator 2.], table-name 2, ...
[WHERE predicate]
[GROUP BY column-name [HAVING predicate]]"
of
"SELECT [DISTINCT] {expression 1, column-names, expression 2, ...}
expressions include arithmetic statements or functions."

Wat tussen vierkante haken staat, is optioneel. Tussen accolades dient één van de door | gescheiden alternatieven te worden gekozen, als die vertikale streep voorkomt.

In of bij bovenstaande definities ontbreekt in ieder geval:
- het "re-namen" (ten behoeve van uitvoer een andere naam geven) van column-names (bijv. SELECT PNR NUMMER FROM P),
- het opgeven van "tabel-variabelen" (zoals bij P1 in SELECT PNR FROM P P1),
- de mogelijkheid om sortering te vragen met ORDER BY
- het feit dat het gebruik van DISTINCT aan beperkingen onderhevig is.

Verder is nergens een duidelijke definitie van "expression" te vinden.
Ook is niet vermeld dat een tabel-name ook een view-name mag zijn. In de tweede definitie is onduidelijk wat op de plaats van de drie punten mag staan en deze definitie is klaarblijkelijk onvolledig (FROM en WHERE ontbreken).

4.2. Onvolkomenheid

NOT IN werkt niet op een lege verzameling. Om dit probleem te omzeilen, dient men de query anders te formuleren.
Zo zou men de vraag: "Geef namen van de specialisten die nooit de behandeling met code 1020 hebben verricht", met NOT IN aldus formuleren:

```
SELECT SNM
FROM   SP
WHERE  SNR NOT IN
          (SECLECT SNR
           FROM    PB
           WHERE   PCD = 1020)
```

Hierin wordt eerst de verzameling van de nummers van specialisten gevormd die behandeling met code 1020 hebben verricht. Daarna wordt vastgesteld welke specialisten daar niet in voorkomen.

Zonder NOT IN wordt dat iets ingewikkelder:

```
SELECT SNM
FROM   SP
WHERE  0 = (SELECT COUNT(*)
            FROM PB
            WHERE BCD = 1020
            AND SP.SNR = PB.SNR)
```

Nu wordt van elke specialist geteld hoevaak hij de behandeling 1020 heeft uitgevoerd.

4.3. GROUP BY

Door GROUP BY aan een query toe te voegen, kan men tabellen opsplitsen in deelverzamelingen (groups in SQL-terminologie), die de waarden van één of meer attributen gemeenschappelijk hebben. Elke groep kan één regel (Oracle betekenis) uitkomst opleveren die kan bevatten:
- de gemeenschappelijke attribuutwaarde(n)
- de uitkomsten van set-functies (Oracle terminologie) min, max, avg, sum en count.

Desgewenst kan men groepen nog selecteren met voorwaarden na HAVING. Deze voorwaarden kunnen ook weer set-functies bevatten die per groep werken.

Een query om het aantal opnames per dag op te vragen:

```
SELECT INDAT, COUNT(*)
FROM  OPN
GROUP BY INDAT
```

Wil men hetzelfde per jaar weten, dan is dat op een soortgelijke manier echter niet mogelijk en dan moet dat voor ieder jaar apart gevraagd worden, of er moet een extra attribuut jaar toegevoegd worden.

Evenmin kan men meer rijen uitvoer vragen per groep, bijvoorbeeld alle voorkomende waarden van een bepaald attribuut. De vraag "Geef per opnamedatum één maal de datum en vervolgens alle opname-redenen van die dag" kan in SQL niet gesteld worden, want in de uitkomst van

```
SELECT INDAT, OPNR
FROM   OPN
```

wordt bij iedere opname-reden de datum herhaald. Merk op dat in een relationeel complete vraagtaal het niet vereist is dat bovenstaande vraag gesteld mag worden, want met de operatoren van de relational algebra is hij niet te formuleren (zie bijvoorbeeld Ullman [5]).

4.4. Subqueries

In een logische expressie mag aan de linkerzijde van een logische operator geen subquery staan. Dit betekent dat men niet in één query twee subqueries kan evalueren en de uitkomsten vergelijken. Het is noodzakelijk eerst een view te creëren met het resultaat van de eerste subquery en in een volgende query de VIEW te vergelijken met het resultaat van de tweede subquery.
Voorbeeld: Geef de namen van de specialisten die de behandeling met code 1020 vaker hebben verricht dan specialist 13.
Niet is toegestaan:

```
SELECT SNM
FROM   SP
WHERE (SELECT COUNT(*)
       FROM   PB
       WHERE  SP.SNR = PB.SNR AND BCD = 1020)
       >
      (SELECT COUNT(*)
       FROM   PB
       WHERE  BCD = 1020 AND PB.SNR = 13)
```

Wel goed is:

```
CREATE VIEW TOT13 (AANTAL) AS
SELECT COUNT(*)
FROM   PB
WHERE  BCD = 1020 AND SNR = 13
```

en vervolgens:

```
SELECT SNM
FROM   SP, TOT13
WHERE  TOT13.AANTAL <
      (SELECT COUNT(*)
       FROM   PB
       WHERE  BCD = 1020 AND PB.SNR = SP.SNR)
```

Als in een subquery gerefereerd wordt naar een tabel van een omvattende query, dan wordt die subquery "gecorreleerd" genoemd. Een dergelijke subquery moet per waarde van de omvattende query precies één regel opleveren, d.w.z. één tupel of de uitkomst(en) van set-functions.

Bijvoorbeeld: Geef de gegevens van de specialisten die minstens eenmaal de behandeling met code 1020 hebben verricht.

Niet toegestaan is:

```
SELECT *
FROM   SP
WHERE  1020 IN
      (SELECT DISTINCT BCD
       FROM   PB
       WHERE  PB.SNR = SP.SNR)
```

Wel goed is:

```
SELECT *
FROM   SP
WHERE  SNR IN
      (SELECT DISTINCT SNR
       FROM   PB
       WHERE  BCD = 1020)
```

Er zijn omstandigheden denkbaar waaronder de eerste formulering een efficiënter toegangspad suggereert en daarom meer voor de hand ligt. Dit is bijvoorbeeld het geval als de meeste specialisten erg vaak de behandeling 1020 hebben uitgevoerd.
Overigens kan dezelfde vraag ook als join geformuleerd worden, dan heeft men geen last van beperkingen:

```
SELECT DISTINCT SP.SNR, SNM, SADR, WSPL, AB, IND
FROM   SP, PB
WHERE  PB.SNR = SP.SNR
  AND  BCD = 1020
```

Subqueries die niet worden voorafgegaan door IN of ANY worden verondersteld een regel (Oracle betekenis) op te leveren. Als het er meer zijn, volgt pas tijdens de uitvoering van de query een foutmelding.

4.5. Tabel-type, naam en variabele

In CREATE TABLE specificeert men het type van een tabel. Dat betekent in ORACLE termen dat men van een aantal attributen de naam en het type vermeldt en of de attribuut waarden uniek moeten zijn of NULL mogen zijn. Bovendien wordt een lege tabel van dat type gecreëerd.
De in CREATE TABLE opgegeven tabel naam wordt vervolgens gebruikt bij wijzigingen en queries betreffende die tabel. Alleen als er geen eenduidigheid meer bestaat, moet men achter de tabelnaam een tabelvariabele vermelden.
Bijvoorbeeld: Geef de nummers van de patiënten die zijn opgenomen op de dag dat hun vorige opname beëindigd werd:

```
SELECT OP1.PNR
FROM   OPN OP1, OPN OP2
WHERE  OP1.INDAT = OP2.UITDAT
  AND  OP1.PNR   = OP2.PNR
```

In andere gevallen fungeert de tabelnaam ook als tabelvariabele. Het geringe onderscheid dat in Oracle gemaakt wordt tussen tabeltype, naam en variabele kan verwarrend werken.

4.6. Constanten en variabelen van basistypes

- Constanten, laat staan variabelen, van het type BOOL bestaan niet in Oracle.
- Het is niet mogelijk lokale variabelen van de basistypes (NUMERIC, CHAR) te declareren. Daarvoor moet een VIEW gecreëerd worden met één attribuut en één regel. Zie bijvoorbeeld 4.4, waarin een view gecreëerd is met het attribuut aantal dat aangeeft hoevaak specialist 13 behandeling 1020 heeft uitgevoerd. De inhoud van deze view (één geheel getal) kan meer malen gebruikt worden. Dit gebeurt echter op een relatief omslachtige wijze omdat steeds de view-naam na FROM en de attribuut-naam elders vermeld moet worden (zie ook 4.4.).

4.7. Overige beperkingen

- EXISTS (∃) bestaat niet expliciet. Dit moet worden geformuleerd als
  0 < (SELECT COUNT(*) ...)

- CONTAINS (⊃) bestaat niet expliciet. Als men na de WHERE wil controleren of de specialist met nummer 13 de behandelingen 1020, 1021 en 1022 alle drie minstens één maal heeft uitgevoerd, dan kan dat alleen op omslachtige manieren:

```
WHERE 1020 IN (SELECT BCD FROM PB WHERE SNR = 13)
  AND 1021 IN (SELECT BCD FROM PB WHERE SNR = 13)
  AND 1022 IN (SELECT BCD FROM PB WHERE SNR = 13)
```

  danwel

```
WHERE 13 IN (SELECT SNR FROM PB WHERE BCD = 1020)
  AND 13 IN (SELECT SNR FROM PB WHERE BCD = 1021)
  AND 13 IN (SELECT SNR FROM PB WHERE BCD = 1022)
```

- DISTINCT mag maar éénmaal per query-block worden vermeld. Daarom moet een query als SELECT COUNT (DISTINCT PNR), COUNT (DISTINCT OPNR) FROM OPN
  (Geef het aantal verschillende patiënten dat is opgenomen en het aantal verschillende opnameredenen)
  geformuleerd worden als twee afzonderlijke queries (één met COUNT (DISTINCT

PNR) en één met COUNT (DISTINCT OPNR).

- UNION mag worden gebruikt om de resultaten van twee queries te verenigen, maar niet op andere plaatsen bijvoorbeeld na WHERE.

- Expressies na SELECT mogen functies bevatten, expressies na WHERE niet.
  Bijvoorbeeld:
  SELECT 3 * AVG (UITDAT-INDAT) is toegestaan,
  maar
  WHERE 3 * AVG (UITDAT-INDAT) > 0 niet.
  Het laatste moet met een subquery geformuleerd worden:

```
WHERE 0 < (SELECT 3 * AVG (UITDAT-INDAT)
           FROM OPN)
```

5. Query analyse en executie

5.1. Toegangspaden

Bij het uitvoeren van een query worden tupels van één of meer tabellen ter inspectie van schijf gehaald. Per tabel bestaan hiervoor in Oracle twee methoden, ook wel toegangspaden genoemd. De eerste genaamd "sequential scan" haalt de tupels van schijf in de volgorde waarin zij zijn opgeslagen. De tweede is via een (door de gebruiker gedefinieerde) index. Dit is voordelig als er een selectie wordt uitgevoerd met het attribuut waarvoor de index bestaat, bijv.

```
SELECT *
FROM   P
WHERE  PNR < 100
```

Met behulp van de index op patiëntnummer kan men gericht de tupels van schijf halen die aan de voorwaarden PNR < 100 voldoen. De kosten die verbonden zijn met het creëren van een index worden snel terug verdiend: bij de index op patiëntnummer op de patiënttabel van SMALLHOS (1000 tupels) is dat al het geval na het opvragen van tien patiënten op nummer.

Een zogenaamde "optimizer" beslist bij de query-analyse welk toegangspad bij executie gebruikt zal worden. Deze optimizer kiest voor de index waar mogelijk, dus ook als een sequential scan efficënter zou zijn. Dit laatste gebeurt bijvoorbeeld als een gehele tabel wordt opgevraagd als in

```
SELECT *
FROM   P
WHERE  PNR > 0
```

en de patiënttabel niet gesorteerd is op nummer.
Bij gebruik van de index worden de patiënt-tupels op volgorde van PNR van schijf gehaald. Omdat transport van en naar schijf met pagina's tegelijk gaat, komen met het gevraagde tupel ook andere tupels mee. Later wordt diezelfde pagina weer van schijf gehaald ten behoeve van zo'n ander tupel. Bij een sequential scan worden alle pagina's slechts éénmaal van schijf gehaald en dat is in dit voorbeeld voordeliger voor wat betreft schijftransport.

Merkwaardig genoeg worden indices niet gebruikt als een OR voorkomt in de voorwaarde na WHERE, zoals in:

```
SELECT *
from   P
WHERE  PNR = 1 OR PNR = 2
```

Nu worden alle patiënt-tupels van schijf gehaald om te bekijken of PNR de waarde 1 of 2 heeft. Kennelijk is de optimizer niet in staat in dit toch vrij simpele geval de meest efficiënte methode te kiezen.

In Oracle bestaan niet de zogenaamde links of DBTG-sets, die in andere systemen voorkomen. In een DBTG-netwerk database zou men een set-type kunnen declareren tussen de P en de OPN tabel. Bij elk patiënt-tupel wordt dan een pointer-claim of een pointer-array gemaakt om snel de opname-tupels van de betreffende patiënt te vinden. In Oracle zou men in zo'n geval een index op het patiëntnummer van de opname-tabel maken. De vraag naar opnames van een patiënt wordt dan beantwoord door in die index het betreffende patiëntnummer op te zoeken, en daar vindt men dan de verwijzingen naar de gezochte opname-tupels.

5.2. Join

Een vraag over meer dan één tabel kan men in Oracle als een "join" formuleren door na FROM meer dan één tabelnaam te vermelden. De volgorde waarin die tabellen worden doorlopen bij query-executie kan veel uitmaken voor de performance. Dit wordt veroorzaakt door verschillende hoeveelheden tupels die van schijf gehaald en verwerkt moeten worden bij verschillende tabelvolgorden.
Bijvoorbeeld: Geef de naam van patiënten die na 21 aug. 1900 zijn geboren en die een behandeling hebben gehad, die langer duurde dan 119 minuten alsmede de code van die behandeling.

```
A. SELECT PNM, BCD
   FROM   P, PB
   WHERE  P.PNR = PB.PNR
     AND  P.GBDAT > 00 08 21
     AND  PB.DUUR > 119
```

In dit speciale geval is het efficiënter eerst de PB-tabel te bekijken, omdat weinig tupels aan de voorwaarde PB.DUUR > 119 voldoen en alleen voor die tupels het bijbehorende patiënt-tupel opgezocht hoeft te worden.
Begint men bij de P-tabel, dan voldoen relatief veel tupels aan de voorwaarde P.GBDAT > 00 08 21, zodat ook veel PB-tupels bekeken worden worden.

Omdat Oracle geen gegevens over gegevens bijhoudt, kan het de beste volgorde vooraf niet bepalen. Vergelijken we bovenstaande query (A) met een query die gelijk is aan (A), behalve dat na FROM de tabelnamen zijn verwisseld (B), dan levert een meting de volgende resultaten:

	CPU (S)	Direct I/O
A.	31.61	454
B.	93.63	3474

(Voor een toelichting op CPU-seconden en Direct I/O zie Reitsma [3].)
Kennelijk begint Oracle in geval A. bij de PB-tabel en in geval B bij de P-tabel. De grote verschillen spreken voor zich.

Het algoritme dat Oracle gebruikt voor het vaststellen van de tabelvolgorde wordt door de leverancier geheim gehouden. Uit experimenten is gebleken dat het algoritme in grote lijnen de prioriteiten in de volgende volgorde legt:

1. tabellen die benaderd kunnen worden met een index op een attribuut waarmee geselecteerd wordt met een gelijkheidsvoorwaarde
2. als 1 met een andere voorwaarde
3. overige tabellen in een volgorde die (op een voor mij onbekende manier) afhangt van de door de gebruiker opgegeven volgorde.

(Zie bijlage 2.C.)
De volgorde wordt verder nog beïnvloed door de manier waarop tabellen met join-condities met elkaar verbonden zijn. Zoals boven reeds gedemonstreerd is, levert dit algoritme ook in niet uitzonderlijke gevallen niet optimale oplossingen.
Het is haast onbegrijpelijk dat in bovenstaand algoritme niet is verwerkt dat tabellen met een selectievoorwaarde vóór tabellen komen zonder selectievoorwaarde (zie voor laatste voorbeelden bijlage 2.C).

5.3. Uitvoering van een join

Bij een join van twee tabellen loopt Oracle systematisch één tabel af en zoekt (in de praktijk meestal met een index) de bijbehorende tupels in de andere tabel op. Bij joins van meer dan twee tabellen worden eerst twee tabellen samengenomen en daarna andere tabellen stuk voor stuk toegevoegd. Deze manier van join-en is efficiënt als weinig tupels van de tabellen bekeken moeten worden. Als veel tupels nodig zijn, dan is het efficiënter eerst op het join-attribuut te sorteren. In dat geval is Oracle vooral slecht voor wat betreft

transport van en naar schijfgeheugen. Er worden data-pagina's meer malen van schijf gehaald, telkens vanwege een andere tupel.

5.4. Subqueries

Een query die betrekking heeft op meer dan één tabel kan, behalve als join, ook met behulp van een "subquery" geformuleerd worden. Een subquery is een SELECT opdracht na de WHERE van een andere query. Bijvoorbeeld: Geef de namen van de specialisten die nooit de behandeling met de code 1020 hebben verricht:

```
C. SELECT SNM
   FROM   SP
   WHERE  SNR NOT IN
         (SELECT DISTINCT SNR
          FROM PB
          WHERE BCD = 1020)
```

Bij de uitvoering van subqueries worden enige inefficiënte methodes gebruikt. Zo worden subqueries als
0 < (SELECT COUNT (*) ...)
niet afgebroken zodra de "COUNT" bij 1 is, maar het tellen wordt tot het einde toe uitgevoerd.

Verder worden subqueries in bepaalde gevallen vaker uitgevoerd dan strikt noodzakelijk is. Dit gebeurt bijvoorbeeld in bovenstaande query (C). De subquery daarin wordt voor elke specialist uitgevoerd, uiteraard steeds met hetzelfde resultaat. Merkwaardig genoeg wordt de subquery maar éénmaal uitgevoerd als men NOT weglaat.

In meer complexe gevallen waarbij een logische analyse nodig is om te komen tot een beperking van het aantal malen dat een subquery wordt uitgevoerd, laat Oracle het geheel afweten. Zie ook bijlage 2.E.

5.5. Queries betreffende meer dan twee tabellen

Bij queries die meer dan twee tabellen betreffen, heeft men ook weer te maken met de onder 5.3 en 5.4 beschreven aspecten. Een nog niet genoemd verschil tussen joins en subqueries dat hierbij een rol speelt, is het volgende. Met

subquery-formuleringen is het mogelijk een bepaalde volgorde van tabellen af te dwingen. Als men in query (C) bijvoorbeeld NOT weglaat, kan men er zeker van zijn dat eerst de PB en dan de SP-tabel wordt doorzocht. Bij een join is men overgeleverd aan de door Oracle bepaalde volgorde. Daar staan tegenover de in 5.4 beschreven inefficiënties van subqueries.
In Bijlage 2.A en 2.B worden twee voorbeelden gegeven, waarbij in het ene geval een formulering als join veel beter uitkomt en in het andere geval juist veel slechter. De voorbeelden gaan niet over uitzonderlijke gevallen, maar de verschillen zijn al erg groot. (Ruwweg een factor 4 en een factor groter dan 10.)

In het algemeen kan gezegd worden dat naarmate de query meer tabellen bevat, een formulering met subqueries aantrekkelijker wordt. Dit omdat een verkeerde tabelvolgorde vaak een meer nadelige invloed op de performance heeft dan de subquery-inefficiënties uit 5.4.

5.6. SELECT DISTINCT

Als de gebruiker SELECT DISTINCT opgeeft, worden duplicaten uit het antwoord verwijderd. Hiervoor gebruikt Oracle een efficiënt algoritme dat vermoedelijk gebruik maakt van sortering. Zie bijlage 2.D.

6. (On)betrouwbaarheid

De betrouwbaarheid van de gebruikte Oracle-versie laat te wensen over. Hieronder volgen drie voorbeelden van onbetrouwbaarheid:

- Na het indrukken van control-C tijdens het laden van een tabel, bleek de gehele database onbruikbaar te zijn. Zonder enige foutmelding kwamen er onjuiste resultaten en lange response-tijden. Ook het verwijderen van de betrokken tabel was niet meer mogelijk.
- Een keer bleek een index niet meer te werken, ook zonder enige foutmelding. Dit kon verholpen worden door de index te verwijderen en opnieuw te creëren.
- Oracle biedt de mogelijkheid om vanwege veiligheidsredenen een copie van de database te maken ("exporteren"). Deze copie kan later desgewenst geïmporteerd worden. Na het importeren van een tabel volgde de mededeling "9998 records loaded": Het bleek dat slechts 1700 records aanspreekbaar waren.

7. Tot slot

Tot slot een resumé van de belangrijkste punten:

- Oracle beschouwt een database als een aantal files zonder gebruik te maken van of te controleren op bepaalde verbanden tussen die files.
- Bij de opslag van de database moet men er rekening mee houden dat de benodigde ruimte voor data en indices veel meer is dan de ruimte die de hele data in beslag neemt (zie hoofdstuk 2).
- De query-taal SQL mag zich in een toenemende pupulariteit verheugen (steeds meer leveranciers maken er gebruik van voor hun produkten). Gezien de opmerkingen in hoofdstuk 4 is het de vraag of dat terecht is. Want daaruit blijkt dat deze taak tekort schiet in inzichtelijkheid en systematiek. Een grondige herziening van SQL lijkt beslist geen overbodige luxe.
- De bepaling van de toegangspaden gaat op een grove manier. Het optimale pad wordt in het algemeen niet gevonden.
- Het invoeren van queries gebeurt vaak inefficiënt. Belangrijke voorbeelden daarvan zijn de query-onderdelen join en subquery. Deze conclusie wordt ondersteund door Bitten e.a. [6].
- De betrouwbaarheid van de gebruikte Oracle-versie laat te wensen over (zie hoofdstuk 6).

8. Literatuur

[1] Oracle Terminal Users Guide, RSI, USA? 1983.

[2] Reitsma, E.H., Database Notitie 8308: "Over sets en indexes", 1983.

[3] Reitsma, E.H., Database Notitie 8309: "VAX en statistics", 1983.

[4] Remunen, F., "Databases, grondslagen voor een logische structuur", Academic Service, Den Haag, 1982.

[5] Ullmann, J.D., "Principles of database systems", London, 1983.

[6] Bitten, D., DeWitt, D.J., Turbyfill, C., "Benchemarking Databases Systems, a systematic approach", Computer Sciences Technical Report ≠ 526, University of Wisconsin, 1983.

[7] Oracle Terminal Reference Manual, version 3.1, RSI, USA, 1982.

Bijlage 1

```
CREATE TABLE P (
PNR    NUMBER(5) NOT NULL,
PNM    CHAR(20),
PADR   CHAR(20),
PWPL   CHAR(15),
GBDAT  CHAR(6),
BLGR   CHAR(2),
RHF    CHAR(1),
GESL   CHAR(1) );

CREATE TABLE VK (
VKNR   NUMBER(3) NOT NULL,
VKNM   CHAR(19),
VKADR  CHAR(18),
VKWPL  CHAR(9),
VKANR  NUMBER(2) );

CREATE TABLE SP (
SNR    NUMBER(2) NOT NULL,
SNM    CHAR(19),
SADR   CHAR(18),
SWPL   CHAR(9),
SPANR  NUMBER(2),
AB     NUMBER(2),
IND    NUMBER(1) );

CREATE TABLE AFD (
ANR    NUMBER(2) NOT NULL,
ANM    CHAR(20),
VKNR   NUMBER(3),
SNR    NUMBER(2) );

CREATE TABLE VR (
VNR    NUMBER(2) NOT NULL,
VANR   NUMBER(2),
AB     NUMBER(2) );

CREATE TABLE OPN (
PNR    NUMBER(5) NOT NULL,
OPNR   CHAR(40),
VNR    NUMBER(2),
INDAT  NUMBER(6) NOT NULL,
UITDAT NUMBER(6),
SNR    NUMBER(2) );

CREATE TABLE BEH (
BCD    NUMBER(4) NOT NULL,
BNM    CHAR(50),
BSRT   CHAR(3),
BTAR   NUMBER(3) );

CREATE TABLE PB (
PNR    NUMBER(5) NOT NULL,
BCD    NUMBER(4) NOT NULL,
SNR    NUMBER(2),
DAT    NUMBER(6) NOT NULL,
DUUR   NUMBER(3),
BEHR   CHAR(4),
INDAT  NUMBER(6) ) ;
```

```
CREATE TABLE M (
MCD    CHAR(6) NOT NULL,
MNM    CHAR(20),
MSRT   CHAR(4) );

CREATE TABLE MV (
MCD    CHAR(6) NOT NULL,
PNR    NUMBER(5) NOT NULL,
SNR    NUMBER(2),
DAT    NUMBER(6) NOT NULL,
DUUR   NUMBER(2),
FD     NUMBER(1),
AM     NUMBER(1) );



CREATE UNIQUE INDEX PPNR      ON P(PNR);
CREATE INDEX VKVKANR          ON VK(VKANR);
CREATE UNIQUE INDEX SPSNR     ON SP(SNR);
CREATE INDEX SPSPANR          ON SP(SPANR);
CREATE INDEX VRVANR           ON VR(VANR);
CREATE UNIQUE INDEX VRVNR     ON VR(VNR);
CREATE INDEX OPNVNR           ON OPN(VNR);
CREATE INDEX OPNPNR           ON OPN(PNR);
CREATE INDEX OPNSNR           ON OPN(SNR);
CREATE UNIQUE INDEX BEHBCD    ON BEH(BCD);
CREATE INDEX PBPNR            ON PB(PNR);
CREATE INDEX PBINDAT          ON PB(INDAT);
CREATE INDEX PBSNR            ON PB(SNR);
CREATE INDEX PBBCD            ON PB(BCD);
CREATE UNIQUE INDEX MMCD      ON M(MCD);
CREATE INDEX MVPNR            ON MV(PNR);
CREATE INDEX MVSNR            ON MV(SNR);
CREATE INDEX MVMCD            ON MV(MCD);
EXIT
```

Bijlage 2.A

Vergelijking subquery en join (1)

1. Geef nummer en naam van patiënten die zowel medicijn ANDB03 als ANBC08 hebben gebruikt.

A.
```
SELECT DISTINCT PNR, PNM
FROM   P, MV MV1, MV MV2
WHERE  P.PNR = MV1.PNR
  AND  P.PNR = MV2.PNR
  AND  MV1.MCD = 'ANDB03'
  AND  MV2.MCD = 'ANBC08'
```

B.
```
SELECT PNR, PNM
FROM   P
WHERE  PNR IN
       (SELECT DISTINCT PNR
        FROM   MV
        WHERE  MCD = 'ANDB03')
  AND  PNR IN
       (SELECT DISTINCT PRN
        FROM   MV
        WHERE  MCD = 'ANBC08')
```

Meting

	CPU(s)	Direct I/O
A.	13.39	381
B.	62.60	1537

Dus de formulering met join is hier veel sneller.

Bijlage 2.B

Vergelijking subquery en join (2)

2. Geef nummer en naam van elke specialist, die minstens twee verschillende soorten handelingen heeft verricht.

A.
```
SELECT DISTINCT SP.SNR, SNM
FROM   SP, BEH BEH1, BEH BEH2, PB PB1, PB PB2
WHERE  SP.SNR = PB1.SNR
  AND  SP.SNR = PB2.SNR
  AND  PB1.BCD = BEH1.BCD
  AND  PB2.BCD = BEH2.BCD
  AND  BEH1.BSRT = BEH2.BSRT
```

B.
```
SELECT SNR, SNM
FROM   SP.
WHERE
       1 < (SELECT COUNT (DISTINCT BSRT)
            FROM   PB, BEH
            WHERE  PB.SNR = SP.SNR
              AND  PB.BCD = BEH.BCD)
```

Meting

	CPU(s)	Direct I/O
A.	> 3600	> 54000
B	214.66	3760

Dus: formulering met subquery is veel sneller.

Bijlage 2.C

3-tabellen query, verschillende tabel-volgorden

Geef de naam van de patiënt, de opnamereden van zijn opname(s) en de behandelruimte(s) van zijn behandelingen, van die patiënten die geboren zijn vóór 1930, opgenomen door een specialist met een nummer kleiner dan 30 en behandeld door een specialist met een nummer kleiner dan 30.

```
(1) SELECT DISTINCT PNM, OPNR, BEHR
    FROM   P, DPN, PB
    WHERE  P.PNR = OPN.PNR
      AND  OPN.PNR = PB.PNR
      AND  GBDAT < 300000
      AND  OPN.SNR < 30
      AND  PB.SNR < 30
```

Meting

		CPU(s)	Direct I/O	volgorde antwoord
A.	FROM P,OPN,PB	1:10.53	2001	OPN
B.	P,PB,OPN	2:01.28	5196	PB
C.	OPN,PB,P	26.89	900	OPN
D.	OPN,P,PB	1:09.69	2003	OPN
E.	PB,P,OPN	1:57.84	4773	PB
F.	PB,OPN,P	2:02.58	5242	PB

Omdat voor SNR zowel in PB als in OPN een index is, begint Oracle met een van die twee tabellen. Bij B, E en F met PB. Met OPN beginnen, is voordeliger omdat de tabel kleiner is. Als tweede tabel is om dezelfde reden P het voordeligst. Kennelijk is de volgorde in geval C: OPN;P;PB en in gevallen A en D: OPN;PB;P.

Variaties op deze query (1) kunnen licht werpen op de manier waarop Oracle de tabelvolgorde bepaalt.

		Oracle begint bij
Vervang in 1.A	PB.SNR < 30	
door	PB.SNR = 30	PB
Vervang in 1.B	OPN.SNR < 30	
door	OPN.SNR = 30	OPN

Kennelijk gaat een voorwaarde met een =teken vóór een voorwaarde met een <teken.

Vervang in 1.A de laatste drie regels door	
AND PWPL = 'EINDHOVEN'	P
of	
AND GBDAT < 300000	P
Vervang in 1.E de laatste drie regels op dezelfde manier	OPN
Vervang in 1.F de laatste drie regels op dezelfde manier	PB

Kennelijk zijn voorwaarden betreffende attributen waarvoor geen index bestaat niet van invloed op de tabelvolgorde.

Bijlage 2.D

Vergelijking SELECT en SELECT DISTINCT

Geef de namen van de patiënten met een nummer kleiner dan een te specificeren getal.

```
SELECT (DISTINCT) PNM
FROM    P
WHERE   PNR < 100
```

Meting (BIGHOS)

	SELECT PNM			SELECT DISTINCT PNM		
	CPU	Direct I/O	aantal tupel	CPU	Direct I/O	aantal tupels
A. PNR < 100	16.48	211	1	18.36	245	1
B. PNR < 1000	16.57	214	23	18.75	239	23
C. PNR < 10000	18.89	225	231	26.97	250	231
D. PNR < 100000	225.83	10193	100000	701.25	27706	9702

per tupel kost verwijderen duplicaten:

A.	1.88	34
B.	0.095	1.1
C.	0.035	0.11
D.	0.045	0.18

Een slecht algoritme zou van de orde n^2 zijn, een goede (bijvoorbeeld gebaseerd op sortering) van de orde nlogn. De cpu-tijd blijft hier binnen. De direct I/O is in deze niet maatgevend omdat de sortering in de gevallen A t/m C in het werkgeheugen kan plaatsvinden.

Bijlage 2.E

Subquery implementatie

Geef de nummers en de namen van de specialisten die minstens 0 behandelingen hebben verricht die langer duurden dan 1 minuut.
Variant: vervang 0 door 150.

```
SELECT SNR, SNM
FROM   SP
WHERE
       0 ≤ (SELECT COUNT (*)
            FROM   PB
            WHERE  PB.SNR = SP.SNR
              AND  DUUR > 1)
```

Meting

		CPU	Direct I/O	tupels gevonden
A.	0	63.91	2828	30
B.	150	59.33	2775	6

In geval A is het tellen in de subquery overbodig omdat de uitkomst hoe dan ook groter of gelijk aan nul is. Toch wordt kennelijk evenveel geteld als in geval B. Hieruit blijkt dat in de subquery alle tupels geteld worden en dan pas vergelijking plaatsvindt met wat links van het ≤ teken staat. Dit kan grote inefficiënties veroorzaken.
Dat A zelfs meer CPU vergt, kan gedeeltelijk worden verklaard door het feit dat er meer resultaten moeten worden afgedrukt.