Subqueries

CSC 256, SQL Programming

Subqueries

- A subquery is a query nested in another query
- General types of subqueries:
  - Uncorrelated
  - Correlated
- Syntactically, subqueries must be in parentheses
- Note that for many cases we can write the same query using joins or subqueries

Uncorrelated Subqueries

- An uncorrelated subquery has no dependence on the main query
- Example: tracks longer than average length

```
select "Name", "Milliseconds"
from "Track"
where "Milliseconds" >
  (select avg("Milliseconds") from "Track")
order by "Milliseconds";
```

Example

- Payments from each customer with percentage of total

```
select
  "CustomerId",
  sum("Total") as "Total",
  1000 * sum("Total") /
    (select sum("Total") from "Invoice") as "Percent"
from  "Invoice"
group by "CustomerId"
order by "Percent" desc;
```

Example

- Artists that have no ‘Rock’ songs

```
select *
from "Artist"
where "ArtistId" not in
  (select "ArtistId"
   from "Album"
     inner join "Artist" using("ArtistId")
     inner join "Track" using("AlbumId")
     inner join "Genre" using("GenreId")
   where "Genre"."Name" = 'Rock');
```

Correlated subqueries

- A correlated subquery depends on main query
- Example: for each customer, the date of most recent invoice

```
select
  c."CustomerId",
  c."FirstName",
  c."LastName",
  (select max(i."InvoiceDate")
   from "Invoice" as i
   where i."CustomerId" = c."CustomerId") as "Recent In
from "Customer" as c;
```

- In this query, for every row in the outer query, the inner query is run. This is a correlated query because of the reference to `c."CustomerId"` in the inner query.

Example

- Return all customers that spent more than $40

```
select
  c."CustomerId",
  c."FirstName",
  c."LastName"
from "Customer" as c
where
  (select sum("Total")
   from "Invoice" as i
   where i."CustomerId" = c."CustomerId") > 40;
```

Table Subqueries

- A table subquery returns a table
- Table subqueries are useful for:
  - multiple passes over the same data
  - generating a table with values you define (the join for example)

Example

- Example: show the average number of tracks per album

```
select avg(count)
from (select "AlbumId", count(*)
      from "Track"
      group by "AlbumId") as t;
```

- We can redefine the column names in the returned table:

```
select avg(total)
from (select "AlbumId", count(*)
      from "Track"
      group by "AlbumId") as t(album, total);
```

VALUES

- We can create virtual table with VALUES
- Example

```
select *
from
  (values
    ('short', 0, 60000),
    ('medium', 60000, 300000),
    ('long', 300000, 10000000))
  as c("desc", "low", "high");
```

Example

- Return track length descriptions

```
select t."Name", t."Milliseconds", c.desc
from "Track" as t
  inner join
    (values
      ('short', 0, 60000),
      ('medium', 60000, 300000),
      ('long', 300000, 10000000))
  as c("desc", "low", "high")
  on t."Milliseconds" >= c.low and
     t."Milliseconds" < c.high;
```

Lateral Subqueries

- A lateral subquery joins the output of the outer query with the output of the subquery. The output rows returned by the inner subquery are added to the result of the join with the outer query.
- Syntax:

```
SELECT <column names>
FROM <table name>
LATERAL <inner subquery>
```

Example

- Return the three most recent invoices for each customer

```
select c.customer_id, d.rental_id, d.rental_date
from customer as c
  inner join
    (select
     from rental as r
     where r.customer_id = c.customer_id
     order by r.rental_date desc
     limit 3
    ) as d
  on c.customer_id = d.customer_id;
```

Common Table Expressions (CTE)

- A common table expression (CTE) is a temporary named result set.
- Basic syntax:

```
WITH <cte name> AS (
  SELECT <attributes>
  FROM <table>
)
SELECT <attributes>
FROM <cte name>
...
```