Databases and information management

Lecture 2

Piotr Syga

2022-10-13

Base of a query

Schema of a simple query

Select *columns*
From *table*
Where *condition*

Conditions

Logical operators

- Basic comparison: `=`, `<>` (`!=`), `>`, `<`, `<=`, `>=`
- Conjunctions `AND`, `OR`, `NOT`
- `BETWEEN` a `AND` b
- `LIKE` '`%`*pattern*`_`'
- (`NOT`) `IN` *set*
- `IS` (`NOT`) `NULL`
- additional (dialect dependent) features (e.g., REGEX)

Conditions

Logical operators

- Basic comparison: `=`, `<>` (`!=`), `>`, `<`, `<=`, `>=`
- Conjunctions `AND`, `OR`, `NOT`
- `BETWEEN` a `AND` b
- `LIKE` '%*pattern*_'
- (`NOT`) `IN` *set*
- `IS` (`NOT`) `NULL`
- additional (dialect dependent) features (e.g., REGEX)

e.g., `SELECT` speed `FROM` A `WHERE` color `LIKE` '%and%';

Set operations and merging results

Set operations

- `UNION` — $\cup$
- `INTERSECT` — $\cap$
- `EXCEPT` — $\setminus$ (dialect dependent, some require `Where` column `NOT IN`)

Important: the schemas (the number, type and names of the columns) have to be the same

Subqueries

We can merge results with subqueries and derived tables:

```
SELECT color FROM (
SELECT color, speed FROM A WHERE weight <> 400 ) TAB1 WHERE speed
<100;
SELECT color FROM A WHERE weight <> (
SELECT weight FROM A WHERE speed=48);
```

Aliases

- We can alias a selected column to change its name (for presentation reasons or to agree on a common scheme)
- We can alias a table to change its name (e.g., when using two copies of the same table)
- We **have to** alias a derived table (the entity from which we select has to be named)

```
SELECT T1.speed AS s, T2.color
FROM A (AS) T1 JOIN A (AS) T2 ON T1.color=T2.color;
```

Views

The result of `SELECT` queries are not 'saved' by default, neither are the aliases. If we want to reuse the query we have to use some way of preserving the derived tables we may use CTE or a View.

- Views simplify the structure of a query
- We can treat it as an interface – backwards compability of the code even when the structure of the tables changes.
- Views may be easily used to limit the records or the columns, the user has access to.

```
CREATE (algorithm) VIEW name (columns) AS
SELECT ...
```

Important: There are some constraints on the things we may use in a view (subqueries, variables, dynamic queries, triggers, some functions and procedures).

More than a set.. – Ordering and result manipulation

As a general rule we want to interpret a table as a set, yet it may not be true in some specific cases, in particular after a query involving projection. Hence the result of the query is not a set:

sorting — `ORDER BY` *column*(, *column2*, . . .) (`DESC`)

grouping — `GROUP BY` *column*(, *column2*, . . .) – we use aggregation functions so that columns not being grouped may be replaced by a single entry

filtering — `HAVING` *condition* – allows filtering **after** the query, hence filtering by aggregated column is possible

row cap — `LIMIT` (`TOP`, `ROWNUM`< number) – **Important:** the place of a cap in the query differs for each capping keyword

Aggregation

- `MIN(`*column*`)`, `MAX(`*column*`)`, `SUM(`*column*`)`
- `AVG(`*column*`)`, `VARIANCE(`*column*`)`, `STD(`*column*`)`
- `COUNT(`*column*`)`, `COUNT(`*column, column2*`)`, `COUNT(*)`
- `GROUP_CONCAT(`*column*`)`
- bit operations (logical and shifts), file operations, ...

Applying to the base schema

Schema of a longer query

Select	*columns*
From	*table alias* Join *table alias2*
Where	*conditions*
Group by	*columns*
Order by	*columns*
Having	*condition*

Applying to the base schema

The order

1. We establish syntactical correctness of the query.
2. We check whether all entities exist and do we have access to them.
3. `FROM` clause is considered (also creating aliases and derived tables)
4. `WHERE` clause is verified.
5. `GROUP`ing takes place.
6. Aggregate functions are calculated at the `SELECT` clause.
7. `ORDER`ing of the result.
8. Filtering with `HAVING` clause.
9. Projection (selecting only the stated columns) at the `SELECT` is made.

It is important to note that independent parts of queries may be considered in a different order due to optimization, most often when subqueries may be modified.