

# Advanced Data Management Technologies

## Unit 9 — SQL Query Specification and Processing

J. Gamper

Free University of Bozen-Bolzano  
Faculty of Computer Science  
IDSE

*Acknowledgements: I am indebted to M. Böhlen for providing me the lecture notes.*

# Outline

- 1 **SQL Queries**
  - Query Specification
  - Processing SQL Queries

# Outline

- 1 **SQL Queries**
  - Query Specification
  - Processing SQL Queries

# Query Specifications

- **Query specifications** are the building block for most SQL statements.
- They determine much of the expressive power of SQL.

<code>&lt;select clause&gt;</code>	}	<b>table expression</b>	}	<b>query specification</b>
<code>&lt;from clause&gt;</code>				
<code>&lt;where clause&gt;</code>				
<code>&lt;group-by clause&gt;</code>				
<code>&lt;having clause&gt;</code>				

# Table Expressions

- A **table expression** is defined as follows:

```
from clause  
[where clause]  
[group-by clause]  
[having clause]
```

- The FROM clause is always required; the others are optional.
- The clauses are operators that take input and produce output.
- The output of each clause is a virtual table, i.e., a table that is not stored.

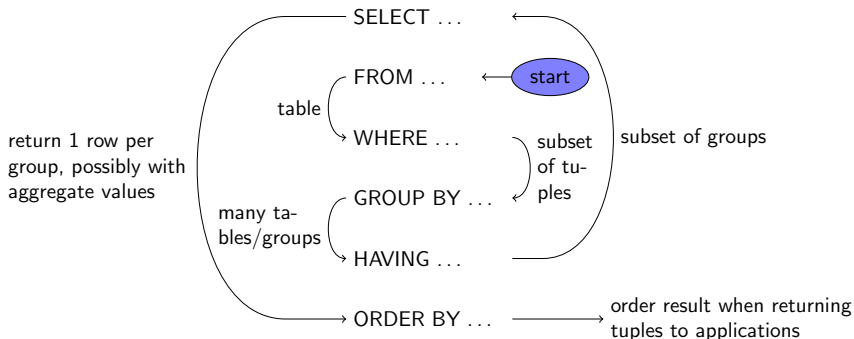
# Query Expressions

- Query specifications can be combined with set operations to form **query expressions**.
- The set operations from the Relational Algebra (RA) are almost directly available in SQL
  - UNION ( $\cup$ )
  - EXCEPT ( $\setminus$ )
  - INTERSECT ( $\cap$ )

```
SELECT X }
FROM r   } query specification
        }
UNION
SELECT Y }
FROM s   } query specification
        } query expression
```

# Processing of SQL Statement

- Logical order of processing an SQL statement



# The From Clause/1

- `FROM <table reference_1>,  
 ...,  
 <table reference_n>`
- Computes the Cartesian product of all input tables.
- A table reference is either a table name or a query expression that defines a derived table.

```
FROM r  
WHERE ...
```

```
FROM (  
    SELECT ...  
    WHERE ...  
    FROM ...  
    WHERE ...  
    ) AS r  
WHERE ...
```



# The From Clause/2

- SQL-92 supports many joins
  - `FROM t1 CROSS JOIN t2`
    - Cartesian product
  - `FROM t1 NATURAL JOIN t2`
    - Natural join (all identical columns)
  - `FROM t1 JOIN t2 ON <join condition>`
    - Theta join
  - `FROM t1 JOIN t2 USING (<columns>)`
    - Restricted natural join
  - `FROM t1 LEFT OUTER JOIN t2 ON <join cond>`
    - Left outer join (similar for RIGHT and FULL outer join)

# The WHERE Clause

- `WHERE <condition>`
- The `WHERE` clause takes the virtual table produced by the `FROM` clause and filters out those rows that do not meet the condition.
- The `WHERE` clause is used to specify join and selection conditions.
- Before SQL allowed query expressions in the `FROM` clause, most of its expressive power came from subqueries in the `WHERE` clause.

# The GROUP BY Clause

- `GROUP BY <grouping_column_1>,  
...  
<grouping_column_n>`
- The result of the `GROUP BY` clause is a grouped table.
- Every row in a group has the same value for the grouping columns.
- Apart from grouping, input and output table are identical.

# GROUP BY Example

movie\_titles

Title	Type	Price
Lethal Weapon	Action	3
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3
Rocky	Action	2

FROM movie\_titles  
GROUP BY Type

Title	Type	Price
Lethal Weapon	Action	3
Rocky	Action	2
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3

# HAVING Clause

- `HAVING <condition>`
- The `HAVING` clause takes a grouped table as input and returns a grouped table.
- The condition is applied to each group.
  - Only groups that fulfill the condition are returned.
- The condition can either reference
  - grouping columns (because they are constant within a group) or
  - aggregated columns (because an aggregate yields one value per group).

# HAVING Clause Example/1

movie\_titles

Title	Type	Price
Lethal Weapon	Action	3
Rocky	Action	2
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3

```
FROM movie_titles  
GROUP BY Type  
HAVING max(Price) <= 3
```

Title	Type	Price
Lethal Weapon	Action	3
Rocky	Action	2
Star Wars	Fiction	3

# HAVING Clause Example/2

movie\_titles

Title	Type	Price
Lethal Weapon	Action	3
Rocky	Action	2
Unforgiven	Western	4
Once upon a time	Western	3
Star Wars	Fiction	3

```
FROM movie_titles  
GROUP BY Type  
HAVING Price <= 3
```

↓  
Illegal SQL

# SELECT Clause/1

- `SELECT [<quantifier>] e_1, ..., e_n`
- The SELECT clause resembles a generalized projection.
- Each item in the SELECT clause is an expression.
- The SELECT clause can contain aggregates.
- If an item in a SELECT clause is an aggregate, then all items have to be aggregates, except the GROUP BY attributes.
- The quantifier enforces duplicate elimination (`DISTINCT`) or duplicate preservation (`ALL`).
  - The default is `ALL`.



# SELECT Clause/2

- `SELECT *` expands to all columns of all tables in the from clause (i.e., no projection in RA).
- `r.*` expands to all columns of table `r`.
- `SELECT *` and `r.*` should be avoided because they cause a statement to change along with schema modifications.
- Columns can be (re)named in the `SELECT` clause.
  - `SELECT r.A B, r.B+r.C*2 X`
  - `SELECT r.A AS B, r.B+r.C*2 AS X`

# Summary and Outlook

- The SQL fragment is fairly **powerful**.
- Over many years it developed into the “**intergalactic data speak**” [Stonebraker].
- At some point it was observed that SQL is not good for analytical queries:
  - too **difficult** to formulate;
  - too **slow** to execute.
- OLAP was born.
- A huge amount of activities during the last 1-2 decades.
- SQL provides several **extensions for OLAP**:
  - GROUP BY extension;
  - SQL for analysis and reporting.