**Data Queries: The Select Statement**

**General Form of Simple Queries**

- The general form of a simple query (*) looks as follows:
  ```sql
  SELECT [ALL | DISTINCT] fieldname1
            [,fieldname2 ,.. | *} ( * = all fields)
  FROM table
  [WHERE "conditions"]
  [ORDER BY "column-list" [ASC | DESC] ]
  [GROUP BY "column-list"]
  [HAVING "conditions"]
  (*) only one table involved
  ```
SQL Queries

• The SELECT clause lists the attributes or functions to be retrieved
• The FROM clause specifies all relations (tables) needed in the query, including joined relations.
• The WHERE clause specifies the conditions for selection of tuples from these relations, including joined conditions if needed.

• GROUP BY specifies grouping attributes to partition the set of tuples into a set of subgroups.
• HAVING specifies a condition on the groups being selected rather than on the individual tuples.
• Finally, ORDER BY specifies an order for displaying the result of a query.
• The built-in aggregate functions COUNT, SUM, MIN, MAX, and AVG are usually used in conjunction with grouping, but they can also be applied to all the selected tuples in a query without a GROUP BY clause.
Retrieving complete data from one Relation

• The most simple query retrieves complete data contained in one relation and looks like

```sql
SELECT *
FROM CarDB.Car;
```

• The `*` after the `Select` keyword defines that every column has to be selected
• The word 'Car' after the `From` keyword defines the data source, here it is the table 'Car'

Selecting Columns from a Relation

• Selecting columns from a relation is done by providing a comma separated list of attribute names after the `SELECT` keyword (the select list)

```sql
SELECT KeyNb, Brand, Model
FROM CarDB.Car;
```

• When specifying the `*` as the select list, the query returns all columns of the underlying table. In this case, degree (number of columns) and order of the result are not defined
DISTINCT keyword

- When specifying the keyword `DISTINCT` before the select list, the result contains only distinct tuples. Tuples containing the same values in all the attributes listed in the select list are skipped.

```sql
SELECT DISTINCT Brand
FROM CarDB.Car;
```

Calculated Columns

- Adding calculated data to a query is done by specifying an expression somewhere in the select list.
- Optionally, the calculated columns may be named (e.g. "AS Amount"). This name is only valid within the result of the query.

```sql
SELECT PricePerDay1, PricePerKm, Discount1,
       (PricePerDay1*2 + PricePerKm*725.6)
       * (100-Discount1) / 100 AS Amount
FROM CarDB.Price;
```
Calculated Columns: Expressions

- Expressions may contain
  - attributes
  - constants
  - arithmetic operations
  - parenthesis
  - functions

Calculated Columns: Functions

- Mathematical functions: Abs, Round, Sin, Cos, Power, Sqrt, Square, Exp, Log, ...
- String functions: Lower, Upper, Len, Substring, LTrim, RTrim, Replace, Reverse, ...
- Date and time functions: Day, Month, Year, Hour, Minute, Second, DateAdd, DateDiff, ...
- Cast and Convert functions: Cast, Convert, Format
- Aggregation functions (to be discussed later in the chapter)
- The set of functions may vary from one DBMS to another!
Calculated Columns: Functions ++


Selecting Tuples

- Selecting tuples from a relation may be done by adding a WHERE clause to the select command
- The WHERE clause may be seen as a filter condition: Tuples which satisfy the condition will be in the result, the other tuples will not

```sql
SELECT Name, FirstName, Phone
FROM CarDB.Person
WHERE FirstName = 'Alain';
```
Conditions

The where clauses may be
- simple boolean expressions, or
- complex boolean expressions combining other (simple or complex) boolean expressions using the operators
  - AND
  - OR
  - NOT
  - ( and )

Conditions: Value comparison

value1 { =, >, <, <=, >=, <> } value2

- value1 and value2 may be attributes, constants, or expressions
- Both values have to be defined on the same domain
- Examples:
  
  FirstName = 'Alain'
  Discount > 10
  StartDate = EndDate
  Duration * Price >= 120.00
Conditions: Range checking

\[ \text{value [NOT] BETWEEN start AND end} \]

• is semantically equivalent to

\[ \text{[NOT] (start} \leq \text{value AND value} \leq \text{end)} \]

• Examples:
  
  Date NOT BETWEEN StartDate AND EndeDate
  Discount BETWEEN 0 AND 10

Conditions: String pattern checking

\[ \text{string-value [NOT] LIKE string-pattern} \]

• string-value is an attribute, constant or expression defined on a string domain
• The string-pattern may contain wildcards:
  \('_'\) representing a single unknown character
  \('%'\) representing an arbitrary sequence of unknown char's.
• Examples:
  
  Name LIKE 'Rose\%' 
  Name LIKE 'Me_er\%'
Conditions: List comparison

value \texttt{[NOT]} \texttt{IN} (value1, ..., valuen)

- is semantically equivalent to

\texttt{[NOT]} (value=value1 OR ... OR value=valuen)

- Examples:

```sql
SELECT *
FROM CarDB.Car
WHERE CatId \texttt{IN} ('C', 'SC', 'S');
```

Conditions: NULL values

- NULL values in value comparison result in undefined results, which will be interpreted to be false
  
e.g. \quad WHERE km < 1

- Use the \texttt{IS [NOT]} \texttt{NULL} operation to explicitly search for null values
  
e.g. \quad WHERE km < 1 OR km \texttt{IS NULL}

- Or use the \texttt{NVL} or \texttt{IsNULL} functions
  
e.g. \quad WHERE \texttt{NVL}(km, 0) < 1
Calculated Columns: NULL values in expressions

- If one value within an expression is NULL, the result is unknown and will be NULL
- The functions `NVL` (in Oracle) and `IsNull` (in MS SQL) may be used to replace NULL values by default values

```sql
SELECT IsNULL(CAST(PayDate-InvoiceDate-
    PayDelay AS NUMERIC),999999999)
FROM CarDB.Invoice;

SELECT NVL(CAST(PayDate-InvoiceDate-
    PayDelay AS NUMERIC),999999999)
FROM CarDB.Invoice;
```

Defining a Sort Order for the Result

- The result of a query has per default no defined order in tuples
- By adding an `ORDER BY` clause at the end of a select statement, the DBMS sorts the result before returning to the application
- The Order By clause takes a list of attributes and/or expressions and optionally the keywords `ASC` (default) or `DESC` for each element in the list

```sql
SELECT * FROM CarDB.Person
ORDER BY ZIP DESC, Name, FirstName;
```
SQL Queries

```sql
SELECT FNAME, EMPLOYEE.NAME
FROM EMPLOYEE, DEPARTEMENT
WHERE DEPARTEMENT.NAME='Research' AND
    DEPARTEMENT.DNUMBER=EMPLOYEE.DNUMBER;
```

- If a query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent ambiguity.

---

SQL Queries

```sql
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN;
```

- It is possible to declare alternative relation names, called aliases or tuple variables (here E and S for the EMPLOYEE relation).
SQL Queries

**SELECT** SSN, DNAME
**FROM** EMPLOYEE, DEPARTEMENT;

- This statement produces all the combinations of EMPLOYEE SSN and DEPARTEMENT DNAME.

**SELECT** *
**FROM** EMPLOYEE, DEPARTEMENT;

- This statement produces the cross product of both relations.

3.3 Data Aggregation
The Concept of Data Aggregation

- The aggregate (or summarize) operation aggregates a subset of tuples of a given table into one single tuple.

<table>
<thead>
<tr>
<th>Place</th>
<th>CarId</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE</td>
<td>A/0012</td>
</tr>
<tr>
<td>BE</td>
<td>B/0001</td>
</tr>
<tr>
<td>BE</td>
<td>B/0015</td>
</tr>
<tr>
<td>BE</td>
<td>C/0001</td>
</tr>
<tr>
<td>BE</td>
<td>C/0015</td>
</tr>
<tr>
<td>BE</td>
<td>D/0014</td>
</tr>
<tr>
<td>BE</td>
<td>D/0020</td>
</tr>
<tr>
<td>BE</td>
<td>MV/04</td>
</tr>
<tr>
<td>BE</td>
<td>T/07</td>
</tr>
<tr>
<td>BS</td>
<td>A/0014</td>
</tr>
<tr>
<td>BS</td>
<td>B/0008</td>
</tr>
<tr>
<td>BS</td>
<td>B/0011</td>
</tr>
<tr>
<td>BS</td>
<td>C/0009</td>
</tr>
<tr>
<td>BS</td>
<td>C/0016</td>
</tr>
<tr>
<td>BS</td>
<td>D/0021</td>
</tr>
<tr>
<td>BS</td>
<td>T/10</td>
</tr>
<tr>
<td>BS</td>
<td>T/20</td>
</tr>
<tr>
<td>GE</td>
<td>A/0008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Place</th>
<th>NbOfCars</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE</td>
<td>9</td>
</tr>
<tr>
<td>BS</td>
<td>8</td>
</tr>
<tr>
<td>GE</td>
<td>7</td>
</tr>
<tr>
<td>ZH</td>
<td>11</td>
</tr>
</tbody>
</table>

Aggregation Criteria and Functions

- The aggregation is based on normal queries and needs two more things to know:
  - Which tuples have to be aggregated into a single one? → **Aggregation criteria**
  - How should multiple attribute values from multiple tuples be aggregated into a single value? → **Aggregation function**
Data Aggregation: GROUP BY

- In SQL, the specification of the aggregation criteria takes the form of an attribute list given in the **GROUP BY** clause of the select statement. Tuples containing identical values for all of the listed attributes are aggregated into one single tuple.

```sql
SELECT Place, Count(KeyNb) NbOfCars
FROM CarDB.Car
GROUP BY Place;
```

Data Aggregation: Functions

- The specification of aggregation functions is similar to calculated columns in the select list.

```sql
SELECT Place, Count(KeyNb) NbOfCars
FROM CarDB.Car
GROUP BY Place;
```

- The argument of an aggregate function is an attribute, either an existing one or a calculated one (expression).
- Aggregate functions consider all values of the specified attribute not being NULL.
Data Aggregation: Functions

- The following aggregation functions are available in standard SQL:
  - **COUNT**: Counts the aggregated tuples or values,
  - **MIN**: Is the minimum of the values,
  - **MAX**: Is the maximum of the values,
  - **SUM**: Is the sum of the values,
  - **AVG**: Is the arithmetic average.

Data Aggregation: Valid Group By Statements

- All expressions in the select-list must either be an aggregation expression or occur in the **GROUP BY** clause of the select statement.
- If a select statement contains aggregate functions in its select list, but no **GROUP BY** clause, all tuples of the underlying relation will be aggregated into a single one.
- If a select statement contains a **GROUP BY** clause, but no aggregate function is used, the result is a list of unique tuples only (since tuples containing all identical values will be aggregated into one).
Data Aggregation: Conditions

- Aggregation functions may not occur in the WHERE clause. To specify restrictions on aggregated data, we have to specify a HAVING clause after the grouping list.

```sql
SELECT Place, Count(KeyNb) NbOfCars
FROM CarDB.Car
GROUP BY Place
HAVING Count(KeyNb) > 10;
```

Data Aggregation: DISTINCT

- If we specify the DISTINCT option in the argument of an aggregation function, the function considers only the first of identical attribute values.

```sql
SELECT Count(Brand) AS NbOfCars
FROM CarDB.Car
WHERE Place='BE';

NbOfCars
---------
13

SELECT Count(DISTINCT Brand) AS NbOfBrands
FROM CarDB.Car
WHERE Place='BE';

NbOfBrands
-----------
4
```
Data Aggregation:  \texttt{COUNT (\ast)}

- The argument of the \texttt{COUNT} function may also be a '\texttt{\ast}' . This means that not attribute values must be counted, but rather whole tuples

\begin{verbatim}
SELECT Count(*)  AS NbOfInvoices
FROM CarDB.Invoice;
\end{verbatim}

\begin{verbatim}
  NbOfInvoices
  ---------
    3
\end{verbatim}

\begin{verbatim}
SELECT Count(PayDate)  AS NbOfInvoices
FROM CarDB.Invoice;
\end{verbatim}

\begin{verbatim}
  NbOfInvoices
  ----------
    2
\end{verbatim}

3.4 Sub-Queries
Characteristics of Sub-Queries

• We speak of a sub-query whenever a complete select statement is used as operand within another select statement

• In SQL, sub-queries may occur in
  – the select list,
  – the from clause, or
  – in conditions

Sub-Queries in the FROM list

• Results from almost any query may be used as underlying relations in other queries

SELECT C.NbOfCars, P.ShortName, P.City 
FROM (SELECT Place, Count(*) NbOfCars 
    FROM CarDB.Car 
    GROUP BY Place) C 
RIGHT JOIN CarDB.Place P 
ON P.ShortName = C.Place 
ORDER BY C.NbOfCars DESC;
Conditions with dynamic lists

- The value list to be used in the \textit{IN} or \textit{NOT IN} operation may be dynamically generated by a sub-query returning exactly one column

\begin{verbatim}
SELECT Id, Name, FirstName
FROM CarDB.Person
WHERE Id NOT IN
  (SELECT CustomerId
   FROM CarDB.Reservation
   WHERE StartDate > '2003-01-01');
\end{verbatim}

Correlated Sub-Queries

- A sub-query is called correlated if it accesses attributes from the outer query

\begin{verbatim}
SELECT Id, Name, FirstName
FROM CarDB.Person P
WHERE NOT EXISTS
  (SELECT *
   FROM CarDB.Reservation
   WHERE StartDate > '2003-01-01'
   AND CustomerId = P.Id);
\end{verbatim}
Table Lookup

- Using a sub-query we may directly look up for values in other tables

```sql
SELECT P.Id, Name, FirstName,
    (SELECT City FROM CarDB.Place
     WHERE ShortName=E.worksAt) WorksIn
FROM CarDB.Person P, CarDB.Employee E
WHERE P.Id = E.Id;
```

- The same result may be achieved with a Left Outer Join, but queries are much more readable this way

3.5 Set Operations
Characteristics of Set Operations

- Set operations operate on two mutually independent select statements
- The results of the two statements must have compatible headings (also called "Union Compatibility")
  - same number of attributes
  - pair wise attributes with same or compatible domains
- Only the second of the two statements may have an ORDER BY clause

**UNION**

- Builds the union of the two result sets

```sql
SELECT fromDate, toDate, Text
FROM CarDB.History
WHERE CarKeyNb = 5088
UNION
SELECT StartDate, EndDate,
    'Rental (CustId='+CAST(R.CustomerId AS VARCHAR)+')'
FROM CarDB.Rental RT, CarDB.Reservation R
WHERE CarKeyNb = 5088
    AND RT.ResId = R.ResId
ORDER BY fromDate;
```
UNION ALL

- When adding the ALL option, the UNION operation does not remove identical tuples from the result (as it does by default)

```
SELECT ZIP, Name, FirstName
FROM CarDB.Person
WHERE FirstName = 'Peter'
UNION ALL
SELECT ZIP, Name, FirstName
FROM CarDB.Person
WHERE ZIP BETWEEN 3000 AND 3999
ORDER BY ZIP, Name, FirstName;
```

INTERSECT

- Builds the intersection of the two result sets (not available in MS SQL and many other DBMS)

```
SELECT ZIP, Name, FirstName
FROM CarDB.Person
WHERE FirstName = 'Peter'
INTERSECT
SELECT ZIP, Name, FirstName
FROM CarDB.Person
WHERE ZIP BETWEEN 3000 AND 3999
ORDER BY ZIP, Name, FirstName;
```
MINUS

- Builds the difference of the two result sets (not available in MS SQL and many other DBMS)

```sql
SELECT ZIP, Name, FirstName
FROM CarDB.Person
WHERE FirstName = 'Peter'
MINUS
SELECT ZIP, Name, FirstName
FROM CarDB.Person
WHERE ZIP BETWEEN 3000 AND 3999
ORDER BY ZIP, Name, FirstName;
```

4. Data Modifications

- Insert
- Update
- Delete
INSERT

• Inserting a single tuple

    INSERT INTO CarDB.Car
    (KeyNb, CatId, Brand, Model, Color, Year, SN, Place)
    VALUES (9999, 'C', 'Fiat', 'Punto', 'rot', 1997, '02-319 637', 'ZH');

    • The system assigns the values (or expressions) in the value list (after the VALUES keyword) to the corresponding attributes in the attribute list (after the table name)
    • The attribute list is optional. When omitted, the system assumes all attributes of the table in the order of declaration

• Inserting a query result

    INSERT INTO CarDB.Stat
    (SDate, NbOfReservations)
    (SELECT ResDate, COUNT(*)
     FROM CarDB.Reservation
     GROUP BY ResDate);
**UPDATE**

- An **UPDATE** command always updates a set of tuples: the tuples that meet the **WHERE** clause!
- An assignment in the **SET** list has an attribute on the left and a value or expression on the right side
- For the evaluation of expressions, the system uses the values of the respective tuples

```
UPDATE CarDB.Car
SET     Color = 'blau', Place = 'BE'
WHERE   KeyNb = 9999;
```

**DELETE**

- A **DELETE** command always removes a set of tuples: the tuples that meet the **WHERE** clause!

```
DELETE FROM CarDB.Car
WHERE   KeyNb = 9999;

DELETE FROM CarDB.Stat;
```
5. Conclusion
Lessons learned

• SQL is a structured language for querying and manipulating data in Relational Database Systems
• SQL is set oriented
• SQL is declarative → What not How!
• The powerful concepts and combination possibilities in SQL allow for the specification of very complex queries, calculations, and data analyses

6. Bibliography
Bibliography


