B0B36DBS, BD6B36DBS: **Database Systems**

http://www.ksi.mff.cuni.cz/~svoboda/courses/172-B0B36DBS/

Lecture 4

SQL: Data Querying

Martin Svoboda

martin.svoboda@fel.cvut.cz

20. 3. 2018

Czech Technical University in Prague, Faculty of Electrical Engineering

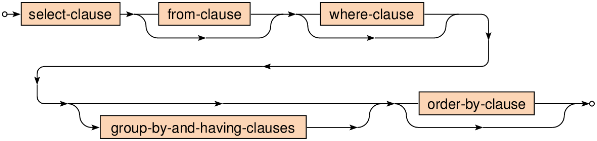
Outline

- **SQL**
  - Data manipulation
    - **SELECT queries**
  - **Views**

SQL: Select Queries

Select Queries

- **SELECT statements** in a nutshell
  - Consist of 1-5 clauses and optionally also ORDER BY clause
    - **SELECT** clause: which columns should be included in the result table
    - **FROM** clause: which source tables should provide data we want to query
    - **WHERE** clause: condition a row must satisfy to be included in the result
    - **GROUP BY** clause: which attributes should be used for the aggregation
    - **HAVING** clause: condition an aggregated row must satisfy to be in the result
    - **ORDER BY** clause: attributes that are used to sort rows of the final result



Sample Tables

- Database of flights and aircrafts

Flights:

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
OK321	CSA	London	156
AC906	Air Canada	Toronto	116
KL7621	KLM	Rotterdam	75
KL1245	KLM	Amsterdam	130

Aircrafts:

Aircraft	Company	Capacity
Boeing 717	CSA	106
Airbus A380	KLM	555
Airbus A350	KLM	253

Select Queries: Example

- Which aircrafts can be used for the scheduled flights?
  - Only aircrafts of a given company and sufficient capacity can be used

```
SELECT Flights.*, Aircraft
FROM Flights NATURAL JOIN Aircrafts
WHERE (Passengers <= Capacity)
ORDER BY Flight
```

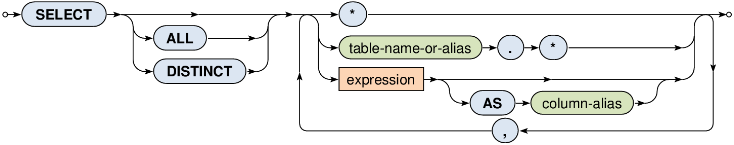
Aircraft	Company	Capacity
Boeing 717	CSA	106
Airbus A380	KLM	555
Airbus A350	KLM	253

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	**CSA**	**Milano**	**37**
OK321	CSA	London	156
AC906	Air Canada	Toronto	116
KL7621	**KLM**	**Rotterdam**	**75**
KL1245	**KLM**	**Amsterdam**	**130**

Flight	Company	Destination	Passengers	Aircraft
KL1245	KLM	Amsterdam	130	Airbus A380
KL1245	KLM	Amsterdam	130	Airbus A350
KL7621	KLM	Rotterdam	75	Airbus A380
KL7621	KLM	Rotterdam	75	Airbus A350
OK012	CSA	Milano	37	Boeing 717

Select Clause

- **SELECT …** FROM … WHERE … ORDER BY …
  - **List of columns** to be included in the result
    - Projection of input columns
      - **Column name**
      - ***** (all columns), **table.*** (all from a given table)
    - Definition of new, derived and aggregated columns
      - Using expressions based on literals, functions, subqueries, …
    - Columns can also be assigned (new) names using **AS**

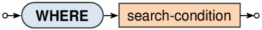


Select Clause

- **SELECT**
  - **Output modifiers**
    - **ALL** (default) – all the rows are included in the output
    - **DISTINCT** – duplicities are removed
  - Examples
    - **SELECT ALL** * …
    - **SELECT** Flights.*, Aircraft …
    - **SELECT DISTINCT** Company **AS** Carrier …
    - **SELECT** ((3*5) + 5) **AS** MyNumber, 'Hello' AS MyString …
    - **SELECT** SUM(Capacity) …
    - **SELECT** (SELECT COUNT(*) FROM Table) **AS** Result …

Where Clause

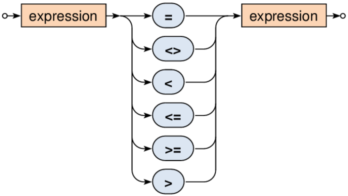
- SELECT … FROM … **WHERE …** ORDER BY …
  - **Selection condition**
    - I.e. condition that a row must satisfy to get into the result
    - Simple expressions may be combined using conjunctions
      - **AND, OR, NOT**



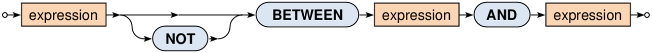
  - Examples
    - … **WHERE** (Capacity > 200) AND (Aircraft LIKE 'Airbus%') …
    - … **WHERE** (Company IN ('KLM', 'Emirates')) …
    - … **WHERE** NOT (Passengers BETWEEN 100 AND 200) …

Search Conditions

- **Comparison predicates**
  - Standard comparison
  - Works even for tuples
    - Example: (1,2,3) <= (1,2,5)

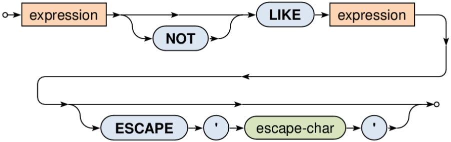


- **Interval predicate**
  - Value BETWEEN Min AND Max
    is equivalent to
    (Min <= Value) AND (Value <= Max)



Search Conditions

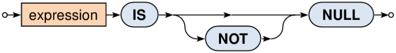
- **String matching predicate**
  - Tests whether a string value matches a given pattern
    - This pattern may contain special characters:
      - % matches an arbitrary substring (even empty)
      - _ matches an arbitrary character
    - Optional escaping character can also be set



- Example
  - Company LIKE '%Airlines%'

Search Conditions

- **NULL values detection predicate**
  - Tests whether a given value is / is not `NULL`
    - Note that, e.g., (expression = NULL) cannot be used!



NULL Values

- **Impact of NULL values**
  - NULL values were introduced to handle **missing information**
  - But how such values should act in functions a predicates?
  - **When a function (or operator) cannot be evaluated**, `NULL` is returned
    - For example: 3 + NULL is evaluated as NULL
  - **When a predicate cannot be evaluated**, special logical value `UNKNOWN` is returned
    - For example: 3 < NULL is evaluated to UNKNOWN
    - This means we need to work with a **three-value logic**
      - TRUE, FALSE, UNKNOWN

Three-Value Logic

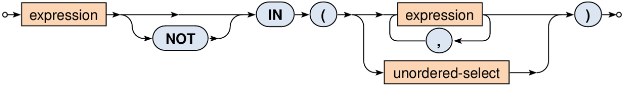
- **Truth tables**

p	q	p AND q	p OR q
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	TRUE
UNKNOWN	FALSE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

NOT q
FALSE
TRUE
UNKNOWN

Search Conditions

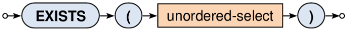
- **Set membership predicate**
  - Tests whether a value exists in a given set of values
    - Example: Company IN ('KLM', 'Emirates')



  - Note that…
    - … IN (∅) = `FALSE`
      - ∅ represents an empty table
    - … IN (ℵ) = `UNKNOWN`
      - ℵ represents any table having rows with only `NULL` values

Search Conditions

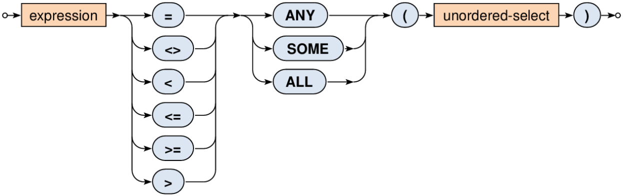
- **Existential quantifier predicate**
  - Tests whether a given set is not empty
  - Can be used to simulate the universal quantifier too
    - ∀ corresponds to ¬∃¬



  - Note that…
    - EXISTS (∅) = FALSE
    - EXISTS (ℵ) = TRUE

Search Conditions

- **Set comparison predicates**
  - **ALL**
    - All the rows from the nested query must satisfy the operator
    - ALL (∅) = `TRUE`
    - ALL (ℵ) = `UNKNOWN`



Search Conditions

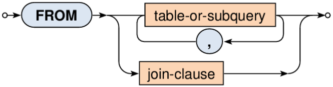
- **Set comparison predicates**
  - **ANY** and **SOME** (synonyms)
    - At least one row from the nested query must satisfy the given comparison operator
    - ANY (∅) = `FALSE`
    - ANY (ℵ) = `UNKNOWN`

From Clause

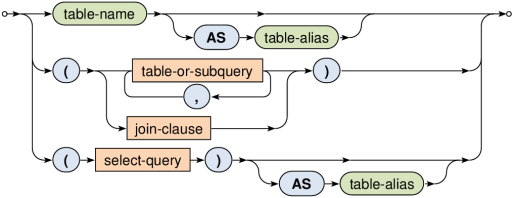
- SELECT … **FROM …** WHERE … ORDER BY …
  - **Description of tables to be queried**
    - Actually not only tables, but also **nested queries or views**
  - **Old way**
    - Comma separated **list of tables** (…)
    - **Cartesian product** of their rows is assumed
    - Required join conditions are specified in the WHERE clause
    - Example: SELECT … FROM Flights, Aircrafts WHERE …
  - **New way**
    - Usage of **join operators** with optional conditions
    - Example: SELECT … FROM Flights JOIN Aircrafts WHERE …

From Clause

- SELECT … **<u>FROM …</u>** WHERE … ORDER BY …
  - **Description of tables to be queried**
    - Overall diagram
      - Both old and new ways

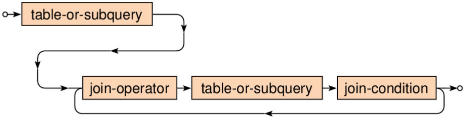


    - Tables and subqueries
      - Table name, auxiliary parentheses, direct select statement



From Clause

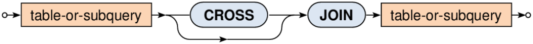
- SELECT … **<u>FROM …</u>** WHERE … ORDER BY …
  - **Description of tables to be queried**
    - Basic structure of joins



      - Examples
        - » Flights NATURAL JOIN Aircrafts
        - » Flights JOIN Aircrafts USING (Company)
        - » …
    - **What types of joins are we provided?**

Table Joins

- **Cross join**
  - Cartesian product of all the rows from both the tables



  - SELECT * FROM T1 **CROSS JOIN** T2

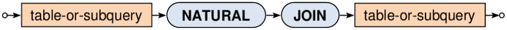
A	T1.*
1	…
2	…
3	…

A	T2.*
1	…
4	…

T1.A	T1.*	T2.A	T2.*
1	…	1	…
1	…	4	…
2	…	1	…
2	…	4	…
3	…	1	…
3	…	4	…

Table Joins

- **Natural join**
  - Pairs of rows are combined only when they have equal values in all the columns they share
    - I.e. columns of the same name



  - SELECT * FROM T1 **NATURAL JOIN** T2

A	T1.*
1	…
2	…
3	…

A	T2.*
1	…
4	…

⇨

A	T1.*	T2.*
1	…	…

Table Joins

- **Inner join**
  - Pairs of rows are combined only when…
    - **ON**: … they satisfy the given join condition
    - **USING**: … they have equal values in the listed columns
  - Note that inner join is a subset of the cross join

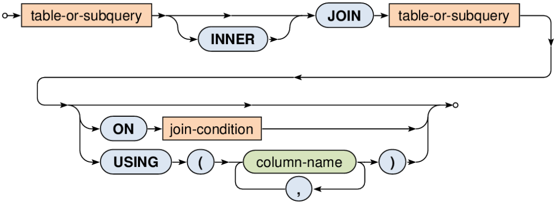


Table Joins

- **Inner join**
  - SELECT * FROM T1 **JOIN** T2 **ON** (T1.A <= T2.A)

A	T1.*
1	…
2	…
3	…

A	T2.*
1	…
4	…

⇨

T1.A	T1.*	T2.A	T2.*
1	…	1	…
1	…	4	…
2	…	4	…
3	…	4	…

  - SELECT * FROM T1 **JOIN** T2 **USING** (A)
    - Equals to the corresponding natural join
  - SELECT * FROM T1 **JOIN** T2
    - Equals to the corresponding cross join

Table Joins

- **Outer join**
  - Pairs of rows from the standard inner join + rows that cannot be combined, in particular, …
    - **LEFT** / **RIGHT**: … rows from the left / right table only
    - **FULL** (default): … rows from both the tables

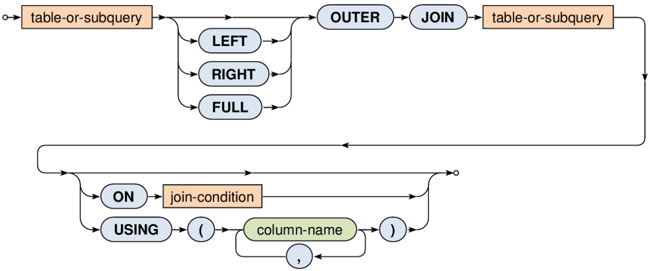


Table Joins

- **Outer join**
  - Note that…
    - NULL values are used to fill missing information in rows that could not be combined

  -
    ```
    SELECT *
    FROM T1 LEFT OUTER JOIN T2 ON (T1.A = T2.A)
    ```

A	T1.*
1	…
2	…
3	…

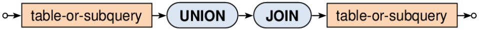
A	T2.*
1	…
4	…

⇨

T1.A	T1.*	T2.A	T2.*
1	…	1	…
2	…	NULL	NULL
3	…	NULL	NULL

Table Joins

- **Union join**
  - Rows of both tables are integrated into one table, no pairs of rows are combined together at all



  - SELECT * FROM T1 **UNION JOIN** T2

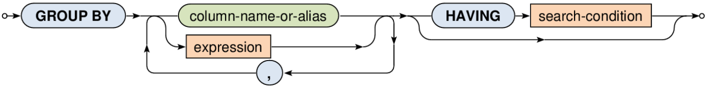
A	T1.*
1	…
2	…
3	…

A	T2.*
1	…
4	…

T1.A	T1.*	T2.A	T2.*
1	…	NULL	NULL
2	…	NULL	NULL
3	…	NULL	NULL
NULL	NULL	1	…
NULL	NULL	4	…

Aggregations

- **Basic idea of table aggregation**
  - First…
    - **FROM** and **WHERE** clauses are evaluated in a standard way
      - This results into an intermediate table
  - Then…
    - **GROUP BY**: rows of this table are divided into groups according to equal values over all the specified columns
    - **HAVING**: and, finally, these aggregated rows (superrows) can be filtered out using a provided search condition



Aggregations: Example

- How many flights does each company have scheduled?
  - However, we are not interested in flights to Stuttgart and Munich
  - As well as we do not want companies with just one flight or less

```
SELECT Company, COUNT(*) AS Flights FROM Flights
WHERE (Destination NOT IN ('Stuttgart', 'Munich'))
GROUP BY Company HAVING (Flights > 1)
```

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	**Lufthansa**	**Stuttgart**	**68**
OK012	CSA	Milano	37
OK321	CSA	London	156
AC906	Air Canada	Toronto	116
KL7621	KLM	Rotterdam	75
KL1245	KLM	Amsterdam	130

⇨

Flight	Company	Destination	Passengers
OK251	**CSA**	New York	276
OK012		Milano	37
OK321		London	156
AC906	**Air Canada**	Toronto	116
KL7621	**KLM**	Rotterdam	75
KL1245		Amsterdam	130

⇨

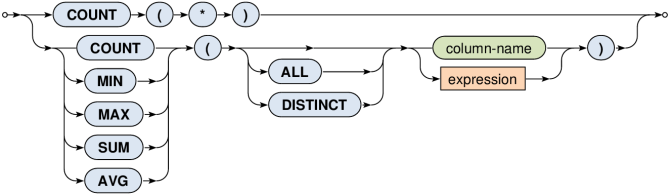
Company	Flights
CSA	3
Air Canada	**1**
KLM	2

⇩

Company	Flights
CSA	3
KLM	2

Aggregations

- **What columns can be used…**
  - in the SELECT clause as well as in the HAVING clause
  - **… when table aggregation takes place?**
  - Answer (for both the cases): only…
    - **Aggregating columns** (i.e. those from the GROUP BY clause)
    - Columns newly derived using **aggregation functions**



Aggregations

- **Aggregate functions**
    - Allow to produce values from the rows within a group
  - **COUNT**(*)
    - Number of all the rows including duplicities and `NULL` values
  - **COUNT** / **SUM** / **AVG** / **MIN** / **MAX**
    - Number of values / sum of values / average / min / max
    - `NULL` values are always and automatically ignored
    - Modifier **ALL** (default) includes duplicities, **DISTINCT** not
    - COUNT(∅) = 0
    - SUM(∅) = `NULL` (which is strange!)
    - AVG(∅) = `NULL`, MIN(∅) = `NULL`, MAX(∅) = `NULL`

Aggregations: Example

- Find basic characteristics for all the scheduled flights
  - I.e. return the overall number of flights, the overall number of the involved companies, the sum of all the passengers, the average / minimal / maximal number of passengers

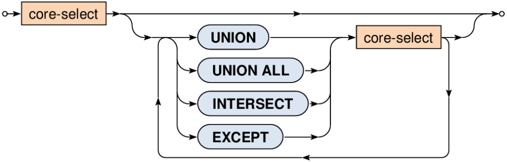
```
SELECT
   COUNT(*) AS Flights,
   COUNT(DISTINCT Company) AS Companies,
   SUM(Passengers) AS PSum,
   AVG(Passengers) AS PAvg,
   MIN(Passengers) AS PMin,
   MAX(Passengers) AS PMax
FROM Flights
```

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
OK321	CSA	London	156
AC906	Air Canada	Toronto	116
KL7621	KLM	Rotterdam	75
KL1245	KLM	Amsterdam	130

Flights	Companies	PSum	PAvg	PMin	PMax
7	4	858	123	37	276

Set Operations

- Available **set operations**
  - **UNION** – union of two tables (without duplicities)
  - **UNION ALL** – union of two tables (with duplicities)
  - **INTERSECT** – intersection of two tables
  - **EXCEPT** – difference of two tables



Set Operations: Example

- Merge available companies from tables of flights and aircrafts

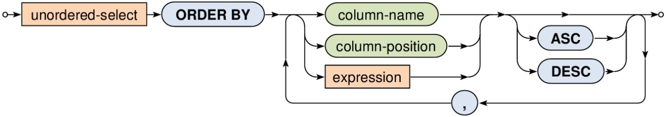
```
SELECT Company FROM Flights
UNION
SELECT Company FROM Aircrafts
```

Company
CSA
Lufthansa
Air Canada
KLM

- Note that…
  - **Both the operands must be compatible**
    - I.e. they have the same number of columns
    - And these columns must be of the same types

Ordered Queries

- **ORDER BY**
  - Note that **rows in the result have no defined order!**
    - … unless this order is explicitly specified
  - Multiple columns (…) can be used for such order
  - `NULL` values precede any other values
  - Directions
    - **ASC** (default) – ascending
    - **DESC** – descending



Ordered Queries: Example

- Return an ordered list of all the scheduled destinations

```
SELECT DISTINCT Destination
FROM Flights
ORDER BY Destination ASC
```

Flight	Company	Destination	Passengers
OK251	CSA	**New York**	276
LH438	Lufthansa	**Stuttgart**	68
OK012	CSA	**Milano**	37
OK321	CSA	**London**	156
AC906	Air Canada	**Toronto**	116
KL7621	KLM	**Rotterdam**	75
KL1245	KLM	**Amsterdam**	130

⇨

Destination
Amsterdam
London
Milano
New York
Rotterdam
Stuttgart
Toronto

Nested Queries

- **Where the nested queries can be used?**
  - In predicates…
    - **ANY, SOME, ALL**
    - **IN**
    - **EXISTS**
  - For definition of **tables** in the **FROM clause**
  - Almost in any expression if scalar values are produced

Nested Queries: Example

- Find all the scheduled flights which have higher than average number of passengers.

```
SELECT *
FROM Flights
WHERE (Passengers > (SELECT AVG(Passengers) FROM Flights))
```

Flight	Company	Destination	Passengers
OK251	**CSA**	**New York**	**276**
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
OK321	**CSA**	**London**	**156**
AC906	Air Canada	Toronto	116
KL7621	KLM	Rotterdam	75
KL1245	**KLM**	**Amsterdam**	**130**

⇨

Flight	Company	Destination	Passengers
OK251	CSA	New York	276
OK321	CSA	London	156
KL1245	KLM	Amsterdam	130

Nested Queries: Example

- Return the number of suitable aircrafts for each flight.
  - Only aircrafts of a given company and sufficient capacity can be used
  - Note how values from the outer query are bound with the inner one

```
SELECT
   Flights.*,
   (
      SELECT COUNT(*)
      FROM Aircrafts AS A
      WHERE
         (A.Company = F.Company) AND
         (A.Capacity >= F.Passengers)
   ) AS Aircrafts
FROM Flights AS F
```

Flight	Company	Destination	Passengers	Aircrafts
OK251	CSA	New York	276	0
LH438	Lufthansa	Stuttgart	68	0
OK012	CSA	Milano	37	1
OK321	CSA	London	156	0
AC906	Air Canada	Toronto	116	0
KL7621	KLM	Rotterdam	75	2
KL1245	KLM	Amsterdam	130	2

Aircraft	Company	Capacity
Boeing 717	CSA	106
Airbus A380	KLM	555
Airbus A350	KLM	253

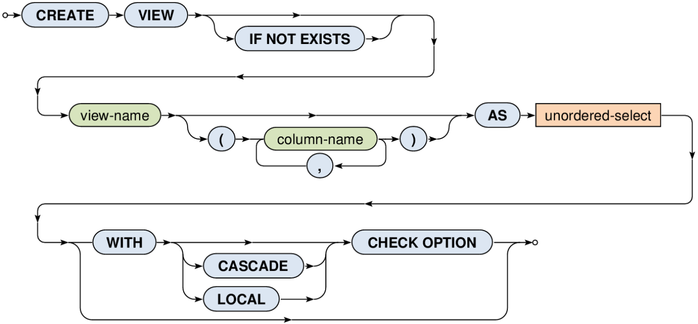
SQL: Database Views

Database Views

- **What are views?**
  - **Named SELECT queries**
    - They can be used similarly as tables
      - E.g. in the FROM clause of the SELECT statements
    - **Evaluated dynamically**
  - Motivation for views
    - Creation of virtual tables, security reasons (hiding tables and their content from particular users), repeated usage of the same complicated statements, …
  - Content of views can be updatable
    - But only when explicitly allowed and only sometimes!

Database Views

- **CREATE VIEW**
  - View name and optionally names of its columns
  - Select query and check option



Database Views

- **View updatability**
  - I.e. can rows be inserted / updated in a view?
  - Yes, but only when…
    - It is permitted, i.e. **WITH CHECK OPTION** is specified
    - And, at the same time, it makes sense…
      - I.e. the given view is based on a **simple SELECT query** (without aggregations, subqueries, …) with only projections (without derived values, …) and selections **over right one table** (without joins, …)
      - I.e. we are deterministically able to reconstruct the entire tuples to be inserted / updated in the original table(s)
    - And, moreover, …

Database Views

- **View updatability**
  - I.e. can rows be inserted / updated in a view?
  - Yes, but only when…
    - …
    - Newly inserted / updated tuples will be visible…
      - **LOCAL** – in the given view
      - **CASCADE** (default) – in the given view as well as all the other views this given one is derived from (depends on)

Database Views

- **Examples**
  - View creation

```
CREATE VIEW BigPlanes AS
SELECT * FROM Aircrafts WHERE (Capacity > 200)
WITH LOCAL CHECK OPTION
```

  - Successful insertion

```
INSERT INTO BigPlanes
VALUES ('Boeing 737', 'CSA', 201);
```

  - Denied insertion

```
INSERT INTO BigPlanes
VALUES ('Boeing 727', 'CSA', 100);
```

    - This aircraft is only too small (will not be visible in the view)