course:

Database Systems (NDBI025)

SS2011/12

lecture 3:

SQL – queries

doc. RNDr. Tomáš Skopal, Ph.D.

Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague

Today's lecture outline

- introduction to SQL
- queries
  - SELECT command
  - sorting
  - set operations

SQL

- structured query language
  - standard language for access to (relational) databases
  - originally ambitions to provide “natural language”
    (that’s why, e.g., SELECT is so complex – a single phrase)
- also language for
  - data definition (DDL)
    - creation and altering of relational (table) schemas
  - data manipulation (DML)
    - querying
    - data insertion, deletion, updating
  - transaction management
  - modules (programming language)
  - definition of integrity constraints
  - administration

SQL

- standards ANSI/ISO SQL 86, 89, 92, 1999, 2003 (backwards compatible)
- commercial systems implement SQL at different standard level (most often SQL 99, 2003)
  - unfortunately, not strict implementation – some extra nonstandard features supported while some standard ones not supported
  - specific extensions procedural, transactional and other functionality, e.g., TRANSACT-SQL (Microsoft SQL Server), PL/SQL (Oracle)

SQL evolution

- **SQL 86** – first „shot“, intersection of IBM SQL implementations
  - **SQL 89** – small revision triggered by industry, many details left for 3rd parties
- **SQL 92** – stronger language, specification 6x longer than for SQL 86/89
  - schema modification, tables with metadata, inner joins, cascade deletes/updates based on foreign keys, set operations, transactions, cursors, exceptions
  - four subversions – Entry, Transitional, Intermediate, Full
- **SQL 1999** – many new features, e.g.,
  - object-relational extensions
  - types STRING, BOOLEAN, REF, ARRAY, types for full-text, images, spatial data
  - triggers, roles, programming language, regular expressions, recursive queries, etc.
- **SQL 2003** – further extensions, e.g., XML management, autonumbers, std. sequences, but also type BIT removed

Queries in SQL

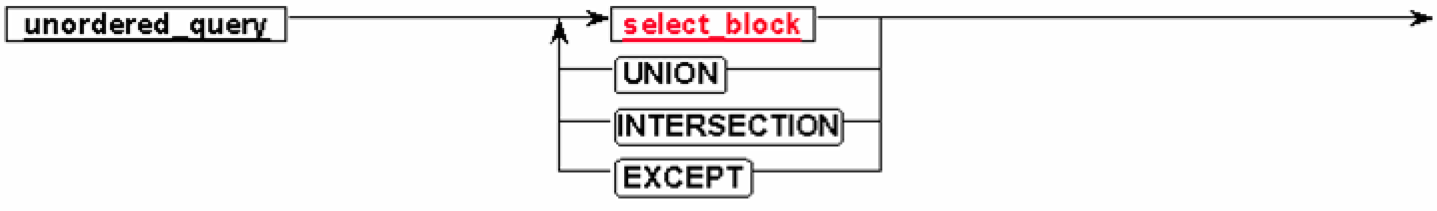
- query in SQL vs. relational calculus and algebra
  - command SELECT shares elements of both formalisms
    - extended domain relational calculus (usage of columns, quantifiers, aggregation functions)
    - algebra (some operations – projection, selection, join, cartesian product, set operations)
  - allowed duplicate rows and NULL attribute values
- syntax validators for SQL 92, 1999, 2003
  - allows to check a query (or other SQL command) based on the norm
  - http://developer.mimer.com/validator/index.htm

Queries in SQL

- for simplicity we consider SQL 86 syntax, and use syntax diagrams source: prof. H.J. Schek (ETH Zurich)
  - oriented graph (DFA automaton accepting SQL)
  - terms in diagrams
    - small letters, underlined – subexpression in give construction
    - capital letters – SQL keyword
    - small letters, italic – name (table/column/…)
  - we use usual terminology table/column instead of relation/attribute
  - diagrams do not include ANSI SQL 92 syntax for joins (clauses CROSS JOIN, NATURAL JOIN, INNER JOIN, LEFT | RIGHT | FULL OUTER JOIN, UNION JOIN) – shown later

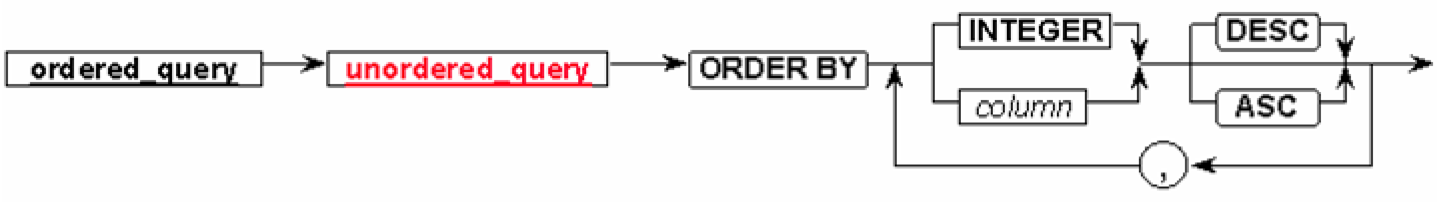
Basic query construction

- unordered query consists of
  - always command **SELECT** (main query logic)
  - optionally commands **UNION**, **INTERSECTION**, **EXCEPT** (union/intersection/subtraction of two or more results obtained by query defined by **SELECT** command)
  - elements in results have no defined order (their order is determined by the implementation of query evaluation, resp.)



Ordered query

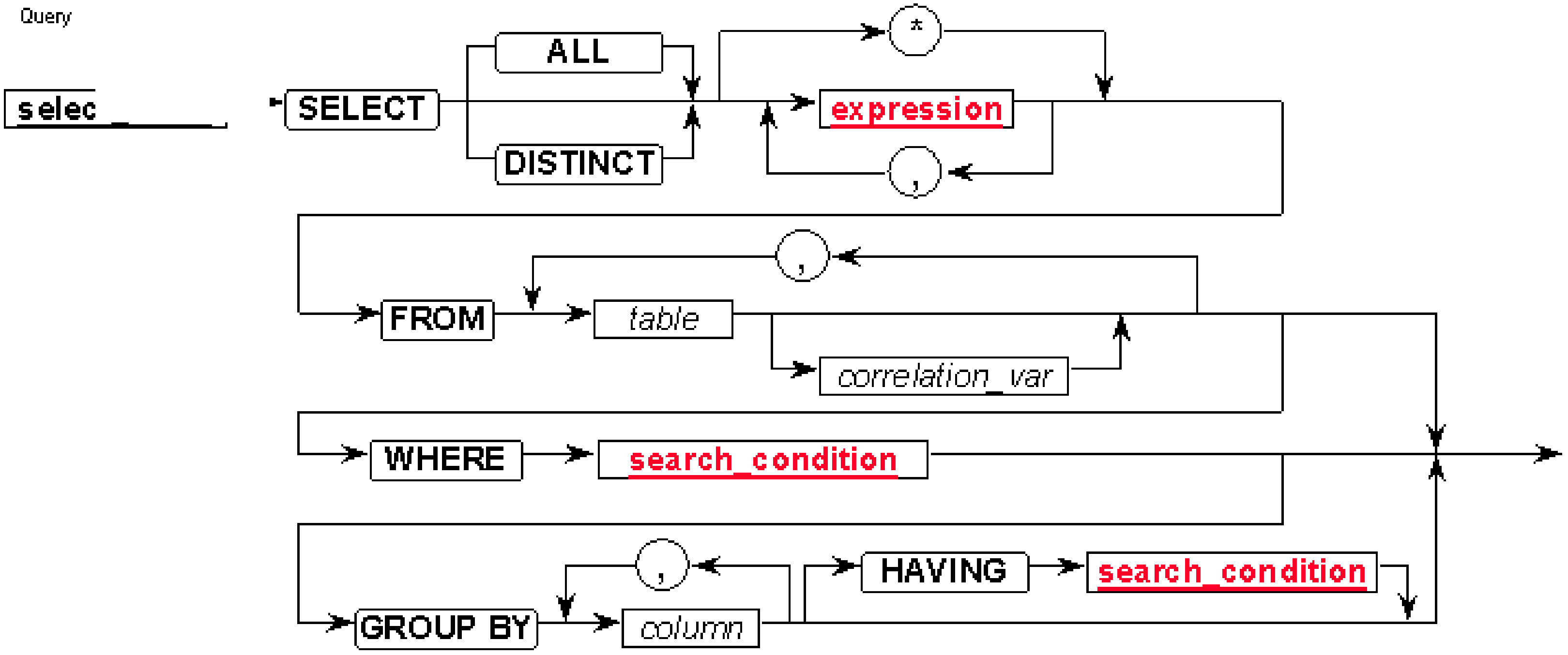
- the result of unorderd query can be ordered/sorted
  - clause **ORDER BY**, ordering according to column(s)
  - ascending (**ASC**) or descending (**DESC**) order
  - multiple secondary columns can be specified that apply in case of duplicate values in the primary (higher secondary) columns



SELECT

- **SELECT** command consists of 2-5 clauses (+ optionally **ORDER BY**)
  - clause **SELECT** – projection into output schema, or definition of new, derived, aggregated columns
  - clause **FROM** – which tables (in case of SQL ≥ 99 also nested queries, views) we query
  - clause **WHERE** – condition that must a row (record) satisfy in order to get into the result
  - clause **GROUP BY** – which attributes should serve for aggregation
  - clause **HAVING** – condition that must **an aggregated** row (record) satisfy in order to get into the result

SELECT – diagram



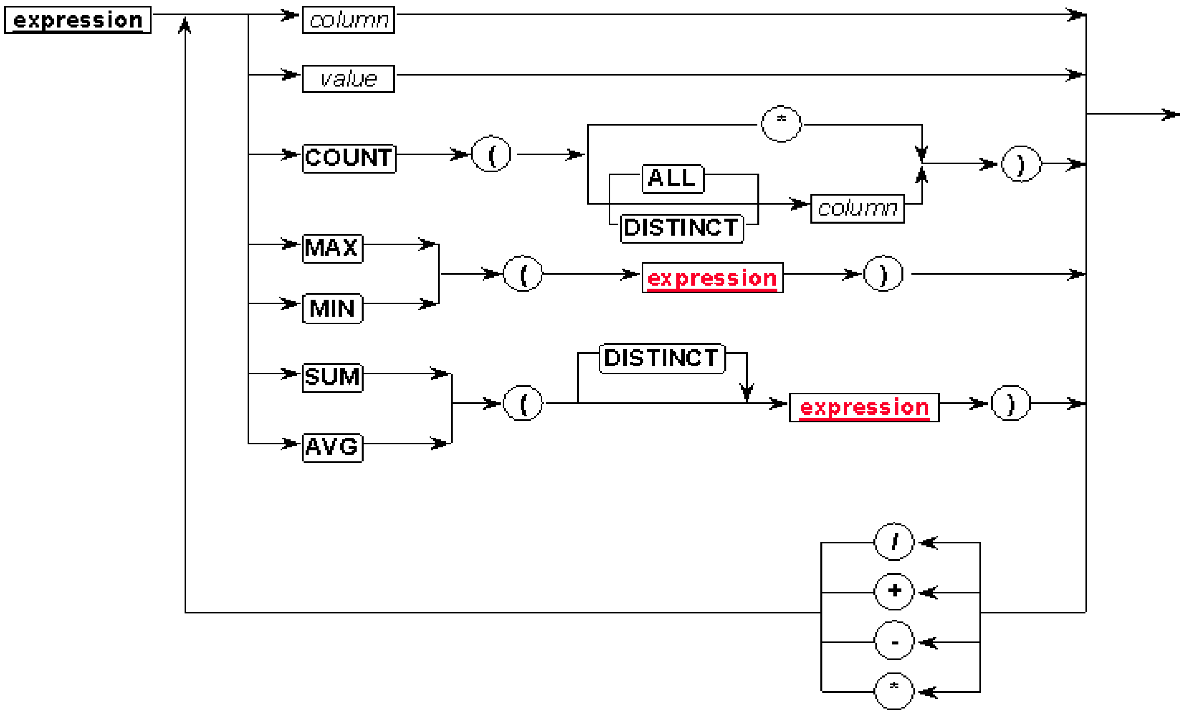
Logical order of evaluation (associativity of SELECT clauses, resp.):

FROM → **WHERE** → **GROUP BY** → **HAVING** → projection **SELECT** (→ **ORDER BY**)

SELECT – expression

Context: SELECT ALL | DISTINCT **expression** FROM …

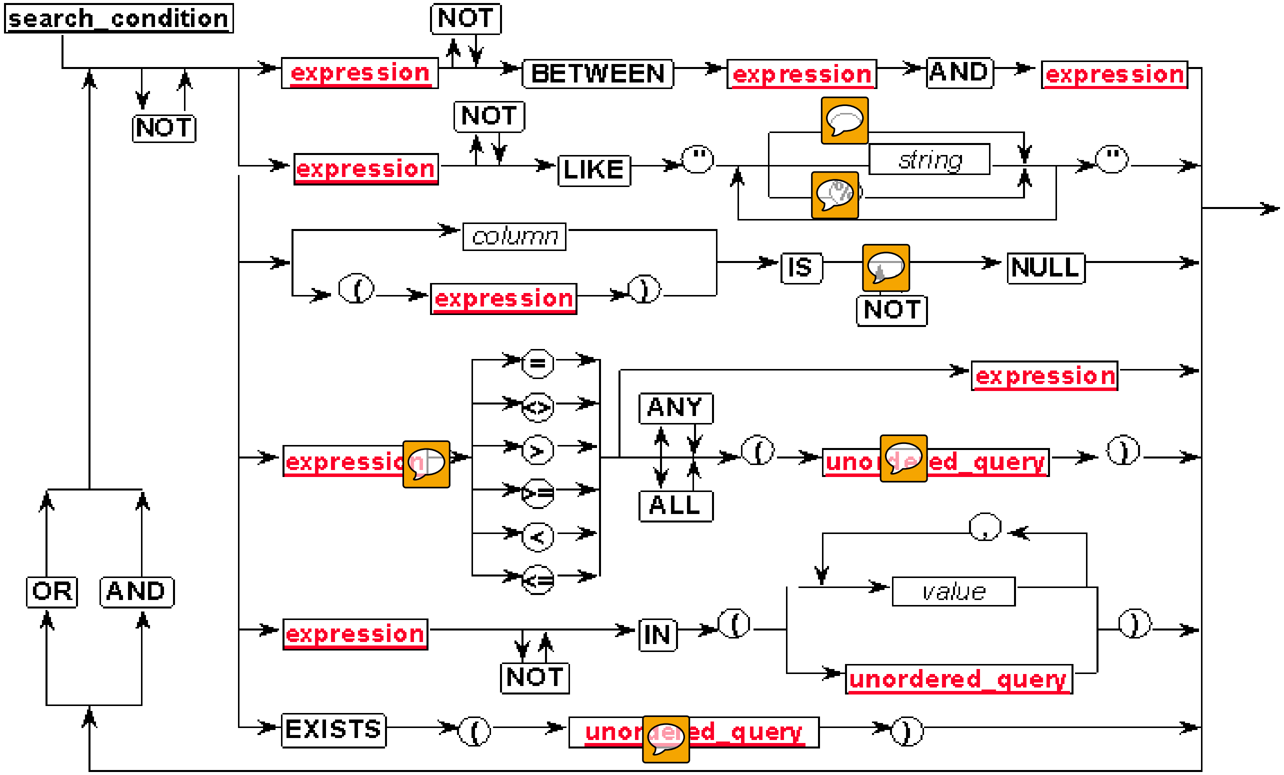
multiple times inside **search_condition**



SELECT – search condition

Context: SELECT … FROM … WHERE **search_condition**

SELECT … FROM … WHERE … GROUP BY … HAVING **search_condition**



Tables used in the examples

tabule
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	Torronto	116
KL7621	KLM	Rotterdam	75
KL1245	KLM	Amsterdam	130

table
Planes

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

SELECT … FROM …

- simplest query:

  SELECT [ALL] | DISTINCT expression **FROM** table1, table2, …
- the expression might contain
  - columns (* is surrogate for all columns)
  - constants
  - aggregation functions on expressions
    - if there is at least one aggregation, in the expression we can use only the aggregated columns (those specified in clause **GROUP BY**), while other columns must be „wrapped“ into aggregation functions
    - if the clause **GROUP BY** is not defined, the aggregation results in a single group (all the sources defined in the clause FROM is aggregated into single row)
  - **DISTINCT** eliminates suplicate rows in the output, **ALL** (default) allows duplicate rows in the output (note that this affects aggregations – with **DISTINCT** there is less values entering the aggregation)
- **FROM** contains one or more tables that serve as the source for the query
  - if there is multiple tables specified, the cartesian product is created

Examples – SELECT … FROM …

Which companies are in service?

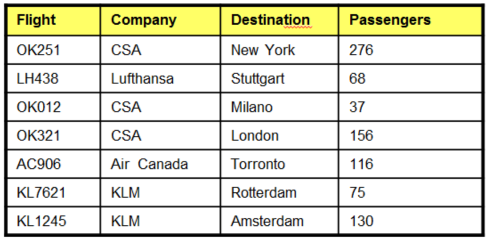
SELECT DISTINCT ‘Comp.:’, Company **FROM** Flights

‘Comp.:’	Company
Comp.:	CSA
Comp.:	Lufthansa
Comp.:	Air Canada
Comp.:	KLM

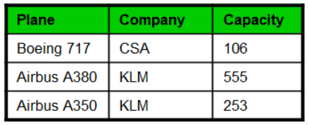
What plane pairs can be created (regardless the owner) and what is their capacity:

SELECT L1.Plane, L2.Plane,
L1.Capacity + L2.Capacity
FROM Planes **AS L1**, Planes **AS L2**

L1.Plane	L2.Plane	L1.Capacity + L2.Capacity
Boeing 717	Boeing 717	212
Airbus A380	Boeing 717	661
Airbus A350	Boeing 717	359
Boeing 717	Airbus A380	661
Airbus A380	Airbus A380	1110
Airbus A350	Airbus A380	803
Boeing 717	Airbus A350	359
Airbus A380	Airbus A350	803
Airbus A350	Airbus A350	506



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



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

Examples – SELECT ... FROM ...

How many companies are in service?

SELECT COUNT(DISTINCT Company) FROM Flights

COUNT(Company)
4

How many flights were accomplished?

SELECT COUNT(Company) FROM Flights

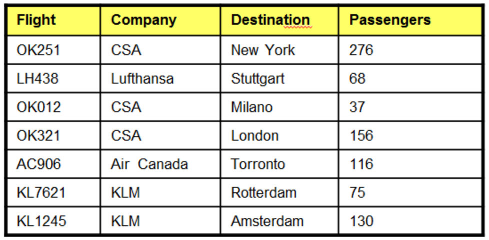
SELECT COUNT(*) FROM Flights

COUNT(Company), resp. COUNT(*)
7

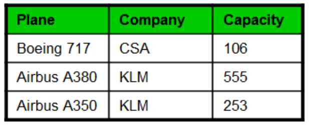
How many planes is in the fleet,what is their max, min, average and total capacity?

SELECT COUNT(*), MAX(Capacity), MIN(Capacity), AVG(Capacity), SUM(Capacity) FROM Planes

COUNT(*)	MAX(Capacity)	MIN(Capacity)	AVG(Capacity)	SUM(Capacity)
3	555	106	304,666	914



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



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

SELECT … FROM … WHERE …

- selection condition, i.e., table row (or row of a join) for which the condition is true gets into the result
  - simple conditions can be combined by **AND, OR, NOT**
- predicates
  - (in)equation predicate (=, <>, <, >, <=, >=) on values of two columns
  - interval predicate – expr1 **[NOT] BETWEEN** (expr2 **AND** expr3)
  - string matching predicate **[NOT] LIKE** "*mask*", where **%** in the mask represents an arbitrary substring, and **_** represents an arbitrary character
  - NULL value predicate – (**expr1**) **IS [NOT] NULL**
  - set membership predicate – **expr1** **[NOT] IN** (**unordered_query**)
  - empty set predicate – **EXISTS** (**unordered_query**)
  - extended quantifiers
    - existence quantifier **expr1** = | <> | < | > | <= | >= **ANY** (**unordered_query**)
      - i.e., **at least one** row in **unordered_query** results in true when compared with **expr1**
    - universal quantifier **expr1** = | <> | < | > | <= | >= **ALL** (**unordered_query**)
      - i.e., **all** rows in **unordered_query** result in true when compared with **expr1**

Examples – SELECT … FROM … WHERE …

What flights are occupied by more than 100 passengers?

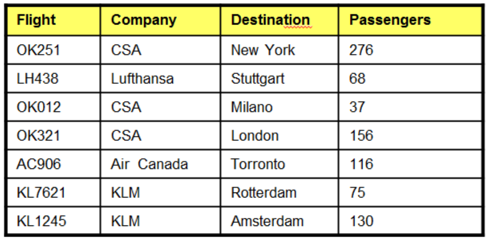
```
SELECT Flight, Passengers FROM Flights
    WHERE Passengers > 100
```

Flight	Passengers
OK251	276
OK321	156
AC906	116
KL1245	130

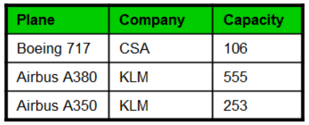
Which planes are ready to fly if we require utilization of a plane at least 30%?

```
SELECT Flight, Plane,
    (100 * Flights.Passengers / Planes.Capacity) AS Utilization  FROM Flights, Planes
    WHERE          Flights.Company = Planes.Company AND
                   Flights.Passengers <= Planes.Capacity AND
                   Utilization >= 30
```

Flight	Plane	Utilization
OK012	Boeing 717	35
KL1245	Airbus A350	51
KL7621	Airbus A350	30



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



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

Examples – SELECT … FROM … WHERE …

Which destinations can fly Airbus of any company (regardless utilization)?

```
SELECT DISTINCT Destinations FROM Flights
     WHERE Flights.Company IN (SELECT Company FROM Planes
                                          WHERE Plane LIKE “Airbus%”)
```

or

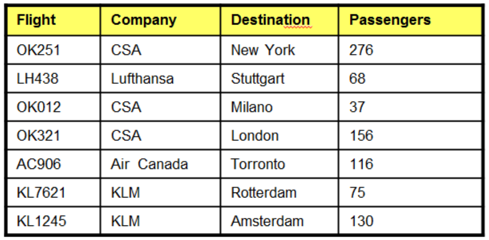
```
SELECT DISTINCT Destination FROM Flights, Planes
     WHERE Flights.Company = Planes.Company AND Plane LIKE “Airbus%”
```

Destination
Rotterdam
Amsterdam

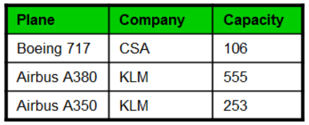
Passengers of which flights fit any plane (regardless plane owner)?

```
SELECT * FROM Flights WHERE Passengers <= ALL (SELECT Capacity FROM Planes)
```

Flight	Company	Destination	Passengers
LH438	Lufthansa	Stuttgart	68
OK012	CSA	Milano	37
KL7621	KLM	Rotterdam	75



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



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

Joins

How to join without special language constructions (SQL 86):

- (inner) join on condition and natural join can be realized as restricted cartesian product, i.e.,
  SELECT … **FROM** table1, table2 **WHERE** table1.A = table2.B
- left/right semi-join is specified in clause **SELECT**, i.e., by projection

New SQL 92 constructions:

- **cartesian product** – **SELECT** … **FROM** table1 **CROSS JOIN** table2 …
- **natural join** – **SELECT** … **FROM** table1 **NATURAL JOIN** table2 **WHERE** …
- **inner join** –
  SELECT … **FROM** table1 **INNER JOIN** table2 **ON search_condition WHERE** …
- **union join** – returns rows of the first table having NULLs in attributes of the second one
  + the same with second table
  SELECT … **FROM** table1 **UNION JOIN** table2 **WHERE** …
- **left, right, full outer join** –
  SELECT … **FROM** table1 **LEFT** | **RIGHT** | **FULL OUTER JOIN** table2 **ON search_condition**
  … **WHERE** …

Examples – joins, ORDER BY

Get pairs flight-plane sorted in ascendant order w.r.t. unoccupied space in plane when the passengers board (consider just the planes the passengers can fit in)?

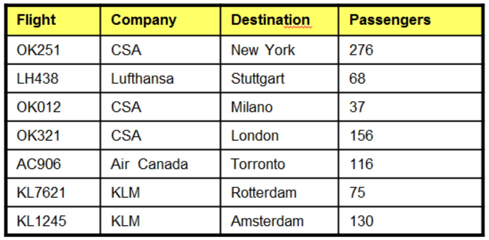
```
SELECT Flight, Plane, (Capacity – Passengers) AS Unoccupied FROM Flights INNER JOIN Planes
    ON (Flights.Company = Planes.Company AND Passengers <= Capacity)
    ORDER BY Unoccupied
```

Flight	Plane	Unoccupied
OK012	Boeing 717	69
KL1245	Airbus A350	123
KL7621	Airbus A350	178
KL1245	Airbus A380	425
KL7621	Airbus A380	480

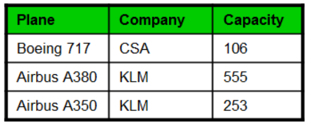
Which flights cannot be operated (because the company does not own any/suitable plane)?

```
SELECT Flight, Destination FROM Flights LEFT OUTER JOIN Planes
    ON (Flights.Company = Planes.Company AND
            Passengers <= Capacity)
    WHERE Planes.Company IS NULL
```

Flight	Destination
OK251	New York
LH438	Stuttgart
OK321	London
AC906	Torronto



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



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

SELECT /…/ GROUP BY … HAVING …

- aggregation clauses that group the rows of the result table after **FROM** and **WHERE** based on identical values in defined columns into columns groups

- the result is table of „superrows“, where the aggregating columns have defined values (as in the original rows), while the other columns must be created by aggregation (otherwise the values would be ambiguous)

- generalization of aggregating function shown earlier (**COUNT**, **MAX**, **MIN**, **AVG**, **SUM**), where the result is not single-row table (as in case of not using **GROUP BY**), but a table of as many rows as the number of superrows is (after **GROUP BY**)

- from this „superrow“ table we can filter out row based on **HAVING** clause, similarly as using **WHERE** after **FROM**
  - note that **HAVING** expression can only use aggregated columns as in **SELECT**

Examples – SELECT /…/ GROUP BY … HAVING …

What (positive) transport capacity have the companies?

```
SELECT Company, SUM(Capacity) FROM Planes GROUP BY Company
```

Company	SUM(Capacity)
CSA	106
KLM	803

What companies can fit all their passengers into their planes (regardless of their destinations)?

```
SELECT Company, SUM(Passengers) FROM Flights
GROUP BY Company HAVING SUM(Passengers) <=
    (SELECT SUM(Capacity) FROM Planes WHERE Flights.Company = Planes.Company)
```

Company	SUM(Passengers)
KLM	205

Nested queries

- standard SQL92 (full) allows to use nested queries also in the **FROM** clause
  - while in SQL 86 the nested queries were allowed only in **ANY**, **ALL**, **IN**, **EXISTS**
- two ways of application
  - **SELECT** … **FROM** (**unordered_query**) **AS** **q1** **WHERE** …
    - allows selection right from the subquery result (instead of table/join)
    - subquery result is called **q1** while this identifier is used in other clauses as it would be a table
  - **SELECT** … **FROM** ((**unordered_query**) **AS** q1 **CROSS** | **NATURAL** | **INNER** | **OUTER** | **LEFT** | **RIGHT JOIN** (**unordered_query**) **AS** q2 **ON** (**expression**)
    - usage in joins of all kinds

Examples – nested queries

Get sums of passengers and plane capacities for all companies that own a plane.

```
SELECT Flights.Company, SUM(Flights.Passengers), MIN(Q1.TotalCapacity)
FROM Flights, (SELECT SUM(Capacity) AS TotalCapacity, Company  FROM Planes GROUP BY Company) AS Q1
      WHERE Q1.Company = Flights.Company
      GROUP BY Flights.Company;
```

Company	SUM(Passengers)	TotalCapacity
CSA	469	106
KLM	205	808

Get triplets of different flights for which the numbers of passengers differ by max 50.

```
SELECT Flights1.Flight, Flights1.Passengers, Flights2.Flight, Flights2.Passengers,
               Flights3.Flight, Flights3.Passengers
      FROM Flights AS Flights1
      INNER JOIN (Flights AS Flights2 INNER JOIN Flights AS Flights3 ON Flights2.Flight < Flights3.Flight)
               ON Flights1.Flight < Flights2.Flight
      WHERE                  abs(Flights1.Passengers – Flights2. Passengers) <=50 AND
                             abs(Flights2.Passengers – Flights3. Passengers) <=50 AND
                             abs(Flights1. Passengers – Flights3. Passengers) <=50
      ORDER BY Flights1. Flight, Flights2. Flight, Flights3. Flight;
```

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

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

Flights1. Flight	Flights1. Passengers	Flights2. Flight	Flights2. Passengers	Flights3. Flight	Flights3. Passengers
AC906	116	KL1245	130	OK321	156
AC906	116	KL7621	75	LH438	68
KL7621	75	LH438	68	OK012	37

Limited number of rows

- often we want to return only first k rows (based on ORDER BY)
- slow version (SQL 92)

  -
    ```
    SELECT … FROM tab1 AS a
    WHERE (      SELECT COUNT(*)
                 FROM tab1 AS b
                 WHERE a.<ordering attribute> < b.<ordering attribute>) < k;
    ```
- fast version (SQL:1999 non-core, implements Oracle, DB2)

  -
    ```
    SELECT … FROM (
                 SELECT ROW_NUMBER() OVER (ORDER BY <ordering attribute(s)>
                              ASC | DESC) AS rownumber
                 FROM tablename)      WHERE rownumber <= k
    ```
- proprietary implementation (nonstandard)
  - `SELECT TOP k … FROM … ORDER BY <ordering attribute(s)> ASC | DESC`
    - MS SQL Server
  - `SELECT … FROM … ORDER BY <ordering attribute(s)> ASC | DESC LIMIT k`
    - MySQL, PostgreSQL

Example (MS-SQL notation)

What is the largest plane?

```
SELECT TOP 1 Plane FROM Planes ORDER BY Capacity DESC
```

Plane
Airbus A380

What are the two largest companies (w.r.t. sums of their passengers)?

```
SELECT TOP 2 Company, SUM(Passengers) AS Number FROM Flights GROUP BY
    Company ORDER BY Number DESC
```

Company	Number
CSA	469
KLM	205

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

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