NDBI049: **Query Languages**

http://www.ksi.mff.cuni.cz/~svoboda/courses/NDBI049/

Lecture 1

Advanced SQL

Martin Svoboda
martin.svoboda@matfyz.cuni.cz

30. 9. 2025

Charles University, Faculty of Mathematics and Physics

Outline

- **SQL**
  - **Data definition**
    - Definition of tables
    - Data types
    - Integrity constraints
    - Schema modification
  - **Data manipulation**
    - INSERT, UPDATE, and DELETE statements
    - SELECT statements

Structured Query Language (SQL)

Structured Query Language

- **SQL**
  - Standard language for accessing relational databases
    - **Data definition** (DDL)
      - Creation of table schemas and integrity constraints
    - **Data manipulation** (DML)
      - Querying
      - Data insertion, deletion, updates
    - Transaction management
    - Modules (programming language)
    - Database administration

Structured Query Language

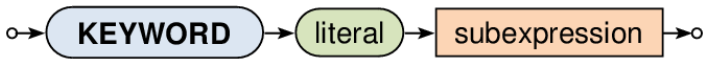
- **SQL standards**
  - Backwards compatible ANSI/ISO standards
    - **SQL-86** – intersection of IBM SQL implementations
    - SQL-89 – small revision, integrity constraints
    - **SQL-92** – schema modification, transactions, set operators, new data types, cursors, referential integrity actions, …
    - **SQL:1999** – recursive queries, triggers, object-relational features, regular expressions, types for full-text, images, spatial data, …
    - **SQL:2003** – SQL/XML, sequence generators
    - SQL:2006 – other extensions of XML, integration of XQuery
    - SQL:2008
    - SQL:2011 – temporal databases
    - SQL:2016, SQL:2019, SQL:2023

Structured Query Language

- **Commercial systems**
  - Current implementations at different standard levels
    - Most often SQL:2011, SQL:2016
  - However (and unfortunately)…
    - **Some extra proprietary features supported**
    - **Some standard features not supported**
    - Even syntax may differ
      - And so data migration is usually not straightforward
  - Specific extensions
    - Procedural, transactional and other functionality, e.g., TRANSACT-SQL (Microsoft SQL Server), PL/SQL (Oracle)

SQL Syntax Diagrams

- **Syntax** (railroad) **diagrams**
  - Graphical representation of context-free grammars
    - I.e. a practical approach how to describe languages (such as SQL) in a graphical and user-friendly way
  - Technically…
    - Directed graph representing an automaton accepting SQL
    - Terms in diagrams:
      - Capital letters on blue – keywords
      - Small letters on green – literals
      - Small letters on orange – subexpressions



SQL: Schema Definition

Table Creation

- **CREATE TABLE**
  - Construction of a table schema (and an empty table)
    - **Table name**
    - Definition of **table columns**
      - Together with their column-scope integrity constraints
    - Definition of **table-scope integrity constraints**

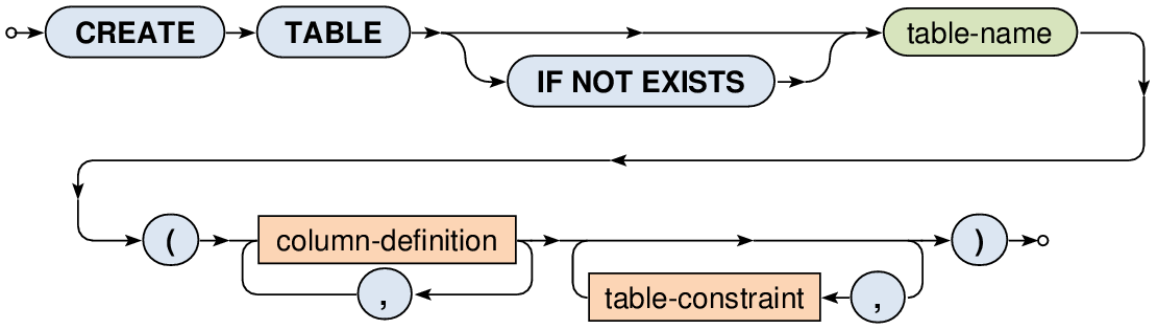


Table Creation

- **CREATE TABLE**
  - Definition of table columns
    - **Column name**
    - **Data type**
    - **Default value**
      - When a new row is about to be inserted and not all its values are specified, then the default values are used (if defined)
    - Definition of column-scope IC

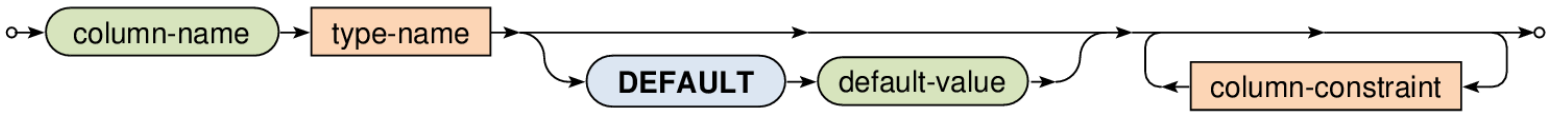


Table Creation

- **Example**
  - Simple table without integrity constraints

```
CREATE TABLE Product (
   Id INTEGER,
   Name VARCHAR(128),
   Price DECIMAL(6,2),
   Produced DATE,
   Available BOOLEAN DEFAULT TRUE,
   Weight FLOAT
);
```

Data Types

- Available data types
  - Precise numeric types
    - **INTEGER**, INT, SMALLINT, BIGINT
    - **DECIMAL**(precision, scale)
      - Precision = number of all digits (including decimal digits)
      - Scale = number of decimal digits
  - Approximate numeric types
    - FLOAT, REAL, DOUBLE PRECISION – real numbers
  - Logical values
    - **BOOLEAN**

Data Types

- Available data types
  - Character strings
    - **CHAR**(length), CHARACTER(length) – fixed-length strings
      - Shorter strings are automatically right-padded with spaces
    - **VARCHAR**(length), CHARACTER VARYING(length)
      - Strings of a variable length
  - Temporal types
    - **DATE**, **TIME**, **TIMESTAMP**
- **Type conversions**
  - Meaningful conversions are defined automatically
    - Otherwise see CAST…

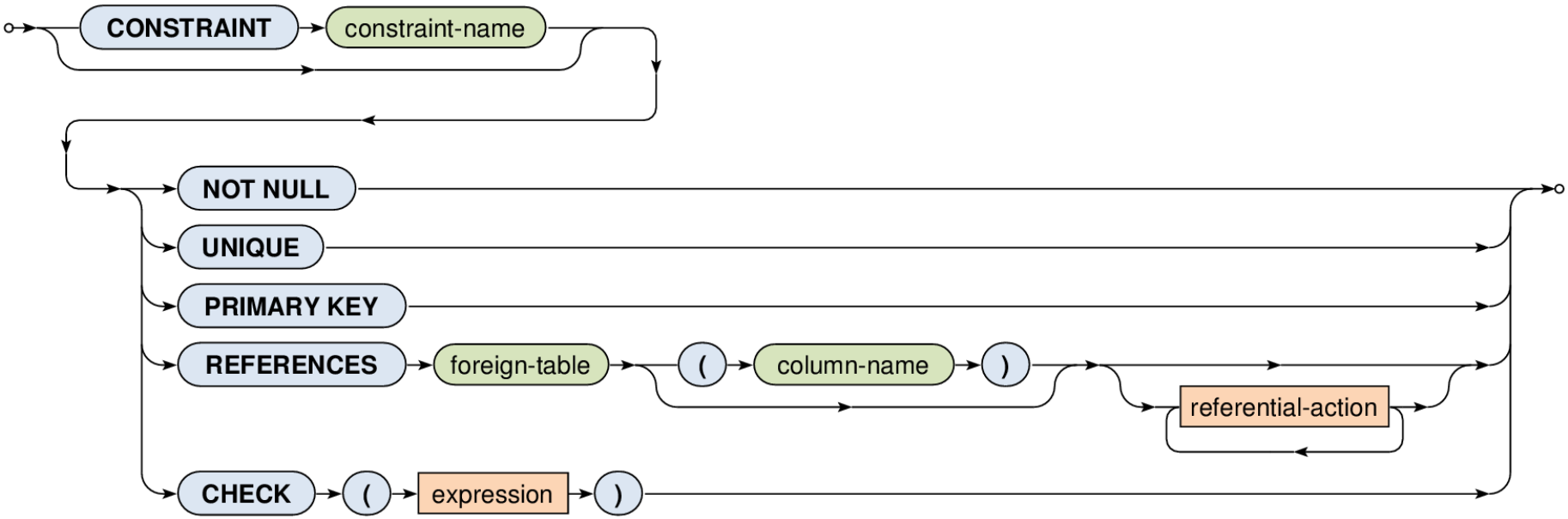
Data Types

- **Example**
  - Simple table without integrity constraints

```
CREATE TABLE Product (
   Id INTEGER,
   Name VARCHAR(128),
   Price DECIMAL(6,2),
   Produced DATE,
   Available BOOLEAN DEFAULT TRUE,
   Weight FLOAT
);
```

Integrity Constraints

- **Column integrity constraints**
  - Allow us to limit domains of the allowed values



Integrity Constraints

- Column integrity constraints
  - **NOT NULL**
    - Values must not be `NULL`
  - **UNIQUE**
    - All values must be distinct
      - But can there be just one or multiple `NULL` values?
  - **PRIMARY KEY**
    - Only one primary key is allowed in a table!
    - Equivalent to NOT NULL + UNIQUE

Integrity Constraints

- Column integrity constraints
  - **FOREIGN KEY**
    - Referential integrity
      - Values from the referencing table must also exist in the referenced table
      - `NULL` values are ignored
      - Only unique / primary keys can be referenced
  - **CHECK**
    - Generic condition that must be satisfied
      - However, only values within a given row may be tested

Integrity Constraints: Example

```
CREATE TABLE Producer (
   Id INTEGER PRIMARY KEY,
   Name VARCHAR(128),
   Country VARCHAR(64)
);

CREATE TABLE Product (
   Id INTEGER CONSTRAINT IC_Product_PK PRIMARY KEY,
   Name VARCHAR(128) UNIQUE,
   Price DECIMAL(6,2) CONSTRAINT IC_Product_Price NOT NULL,
   Produced DATE CHECK (Produced >= '2015-01-01'),
   Available BOOLEAN DEFAULT TRUE NOT NULL,
   Weight FLOAT,
   Producer INTEGER REFERENCES Producer (Id)
);
```

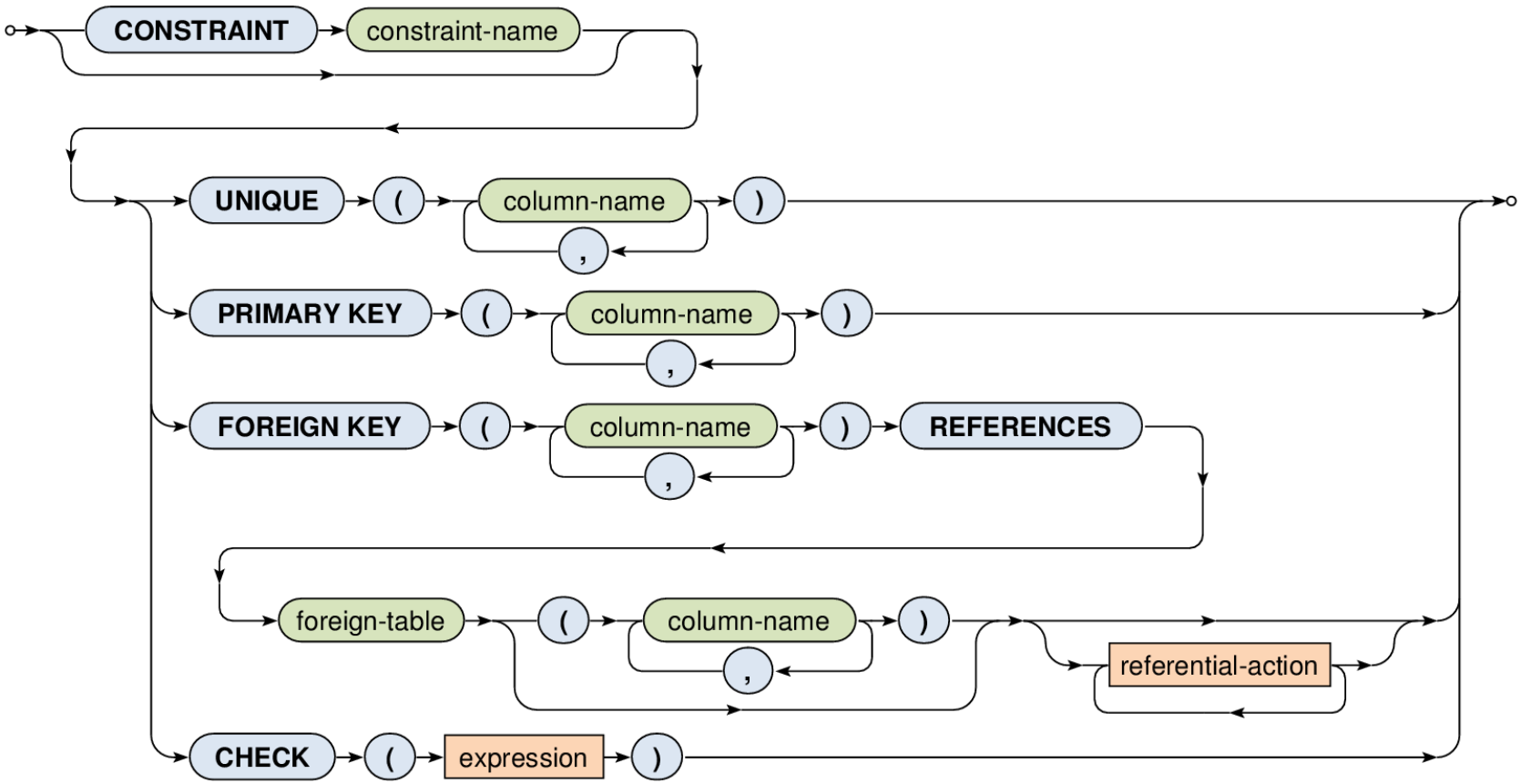
Integrity Constraints: Example

- **Example**
  - Referential integrity within a single table

```
CREATE TABLE Employee (
   Id INTEGER PRIMARY KEY,
   Name VARCHAR(128),
   Boss INTEGER REFERENCES Employee (Id)
);
```

Integrity Constraints

- **Table integrity constraints**



Integrity Constraints

- **Table integrity constraints**
  - Analogous to column IC, just for multiple columns, i.e. for tuples of values
  - **UNIQUE**
  - **PRIMARY KEY**
  - **FOREIGN KEY**
    - Tuples containing at least one `NULL` value are ignored
  - **CHECK**
    - Even with more complex conditions testing the entire tables
      - However, table integrity constraints are considered to be satisfied on empty tables (by definition, without evaluation)
      - See CREATE ASSERTION…

Integrity Constraints: Example

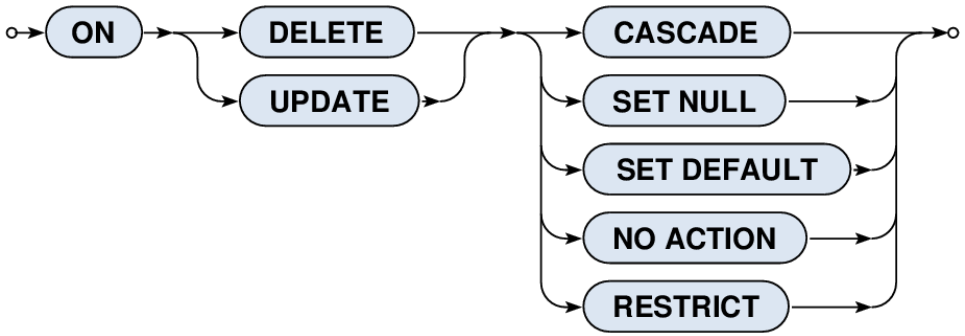
```
CREATE TABLE Producer (
   Name VARCHAR(128),
   Country VARCHAR(3),
   CONSTRAINT IC_Producer_PK PRIMARY KEY (Name, Country)
);

CREATE TABLE Product (
   Id INTEGER PRIMARY KEY,
   …
   ProducerName VARCHAR(128),
   ProducerCountry VARCHAR(3),
   CONSTRAINT IC_Product_Producer_FK
      FOREIGN KEY (ProducerName, ProducerCountry)
      REFERENCES Producer (Name, Country)
);
```

Referential Integrity

- **Referential actions**
  - When an operation on the referenced table would cause violation of the foreign key in the referencing table…
    - I.e. value of the foreign key of at least one row in the referencing table would become invalid as a result
  - … then…
    - this operation is blocked and an error message is generated
    - but if a referential action is defined, it is triggered…

Referential Integrity

- **Referential actions**



  - Triggering situations
    - **ON UPDATE, ON DELETE**
      - When the action is triggered
      - Once again, these are considered to be operations on the referenced table

Referential Integrity

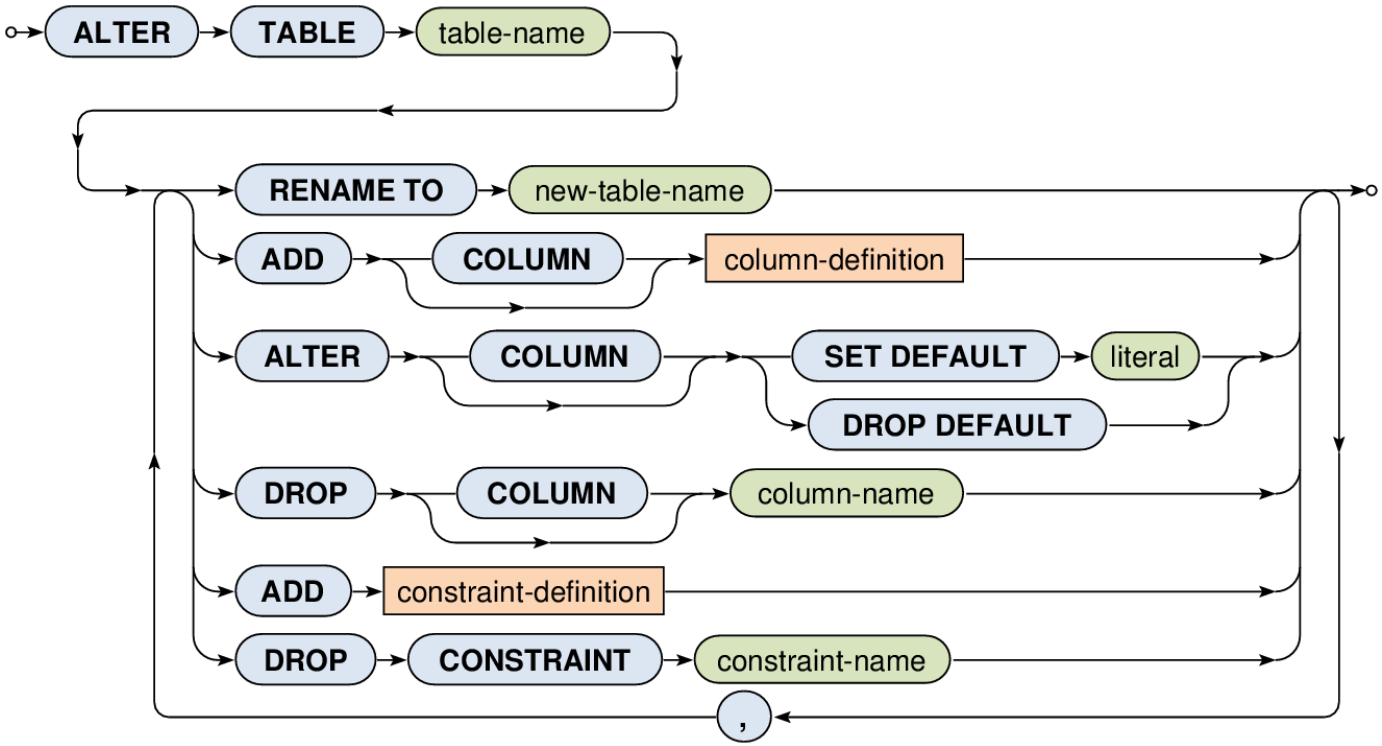
- Referential actions
  - **CASCADE**
    - Row with the referencing value is updated / deleted as well
  - **SET NULL** – referencing value is set to `NULL`
  - **SET DEFAULT** – referencing value is set to its default
  - **NO ACTION** – default – no action takes place
    - I.e. as if no referential action would be defined at all
  - **RESTRICT** – no action takes place as well…
    - However, the integrity check is performed at the beginning, i.e. before the operation is even tried to be executed
      - ... and so triggers or the operation itself have no chance to remedy the situation even if they could be able to achieve such a state (and so RESTRICT is different to NO ACTION)

Referential Integrity: Example

```
CREATE TABLE Producer (
   Id INTEGER PRIMARY KEY,
   Name VARCHAR(128),
   Country VARCHAR(64)
);

CREATE TABLE Product (
   Id INTEGER PRIMARY KEY,
   …
   Producer INTEGER
      REFERENCES Producer (Id) ON DELETE CASCADE
);
```

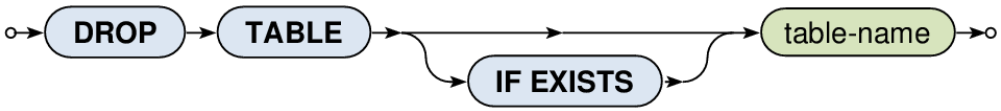
Schema Modification

- **ALTER TABLE**
  - Addition/change/removal of table columns/IC



Schema Modification

- **DROP TABLE**
  - Complementary to the table creation
    - I.e. table definition as well as table content are deleted



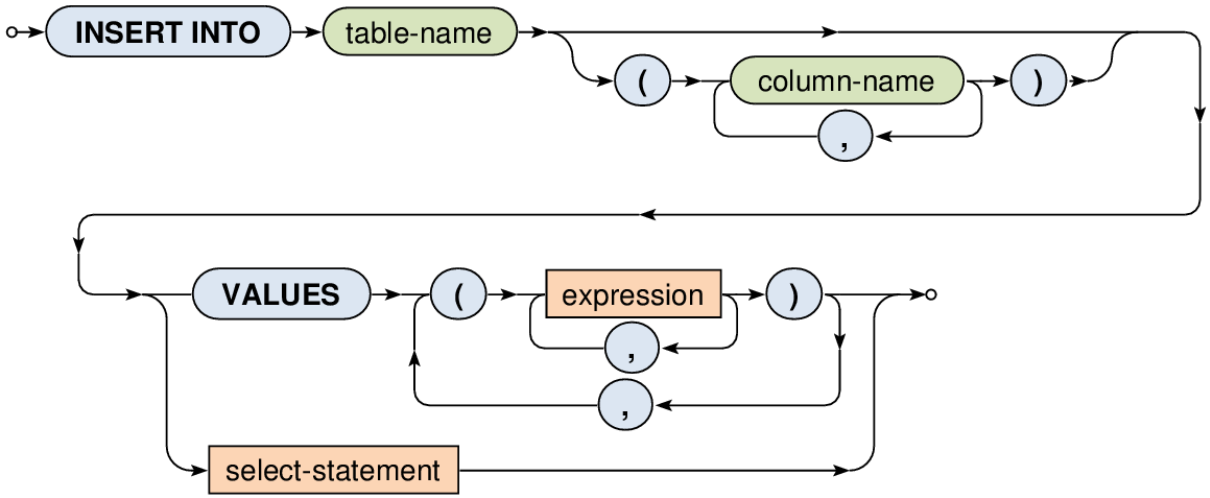
SQL: Data Manipulation

SQL Data Manipulation

- Data manipulation language
  - Data modification
    - **INSERT INTO** – insertion of rows
    - **DELETE FROM** – deletion of rows
    - **UPDATE** – modification of rows
  - Data querying
    - **SELECT** – retrieval of rows

Data Insertion

- **INSERT INTO**
  - Insertion of new rows into a table
    - …by an explicit enumeration / from a result of a selection
    - Default values are assumed for the omitted columns



Data Insertion: Example

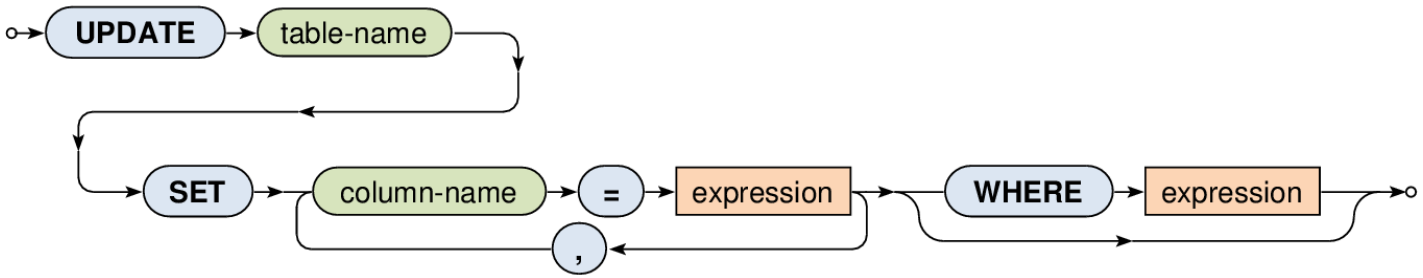
```
CREATE TABLE Product (
    Id INTEGER PRIMARY KEY,
    Name VARCHAR(128) UNIQUE,
    Price DECIMAL(6,2) NOT NULL,
    Produced DATE,
    Available BOOLEAN DEFAULT TRUE,
    Weight FLOAT,
    Producer INTEGER
);
```

```
INSERT INTO Product
   VALUES (0, 'Chair1', 2000, '2015-05-06', TRUE, 3.5, 11);
```

```
INSERT INTO Product
   (Id, Name, Price, Produced, Weight, Producer)
   VALUES (1, 'Chair2', 1500, '2015-05-06', 4.5, 11);
```

Data Updates

- **UPDATE**
  - Modification of existing rows in a table
    - Only rows matching the given condition are considered
  - Newly assigned values can be…
    - `NULL`, literal, value given by an expression, result of a scalar subquery



Data Updates: Example

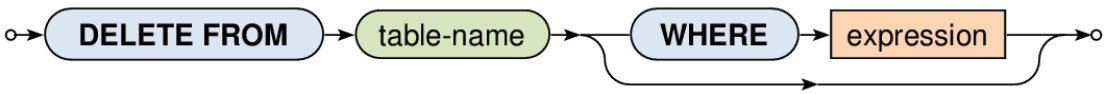
```
CREATE TABLE Product (
    Id INTEGER PRIMARY KEY,
    Name VARCHAR(128) UNIQUE,
    Price DECIMAL(6,2) NOT NULL,
    Produced DATE,
    Available BOOLEAN DEFAULT TRUE,
    Weight FLOAT,
    Producer INTEGER
);
```

```
UPDATE Product
    SET Name = 'Notebook'
    WHERE (Name = 'Laptop');
```

```
UPDATE Product
    SET Price = Price * 0.9
    WHERE (Produced < '2015-01-01');
```

Data Deletion

- **DELETE FROM**
  - Deletion of existing rows from a table
    - Only rows matching the given condition are considered



Data Deletion: Example

```
CREATE TABLE Product (
    Id INTEGER PRIMARY KEY,
    Name VARCHAR(128) UNIQUE,
    Price DECIMAL(6,2) NOT NULL,
    Produced DATE,
    Available BOOLEAN DEFAULT TRUE,
    Weight FLOAT,
    Producer INTEGER
);
```

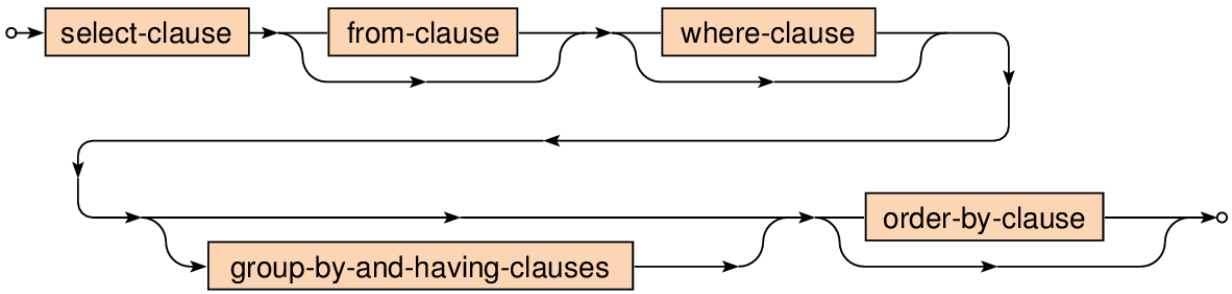
```
DELETE FROM Product
    WHERE (Price > 2000);
```

```
DELETE FROM Product;
```

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
```

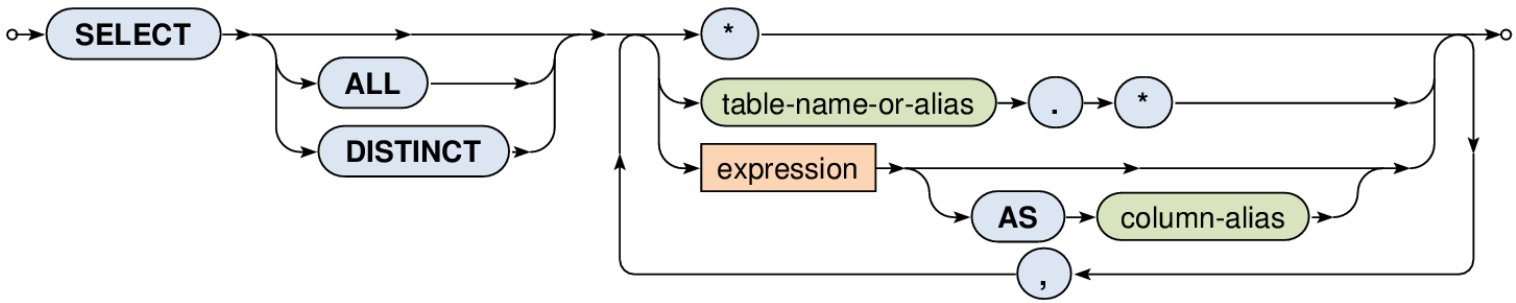
Aircraft	Company	Capacity
Boeing 717	CSA	106
Airbus A380	KLM	555
Airbus A350	KLM	253

Flight	Company	Destination	Passengers	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

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**

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**

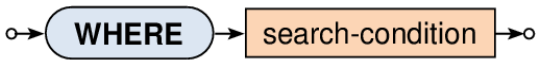


Select Clause

- **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

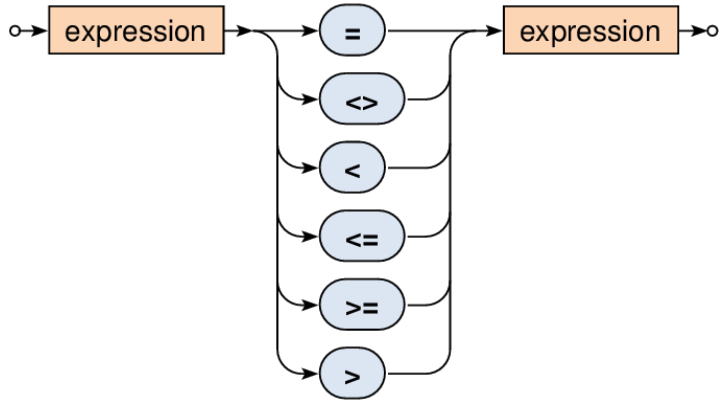
- SELECT … FROM … **WHERE …** ORDER BY …
  - **Selection condition**
    - I.e. condition that a row must satisfy to get into the result
    - Simple expressions may be combined using conjunctions
      - **AND, OR, NOT**



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

Search Conditions

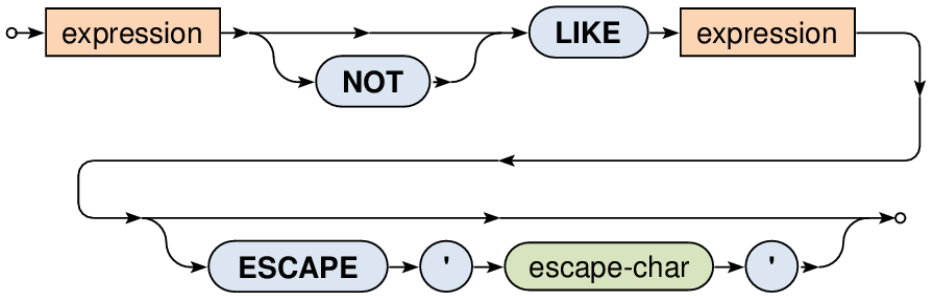
- **Comparison predicates**
  - Standard comparison
  - Works even for tuples
    - Example: (1,2,3) <= (1,2,5)



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

Search Conditions

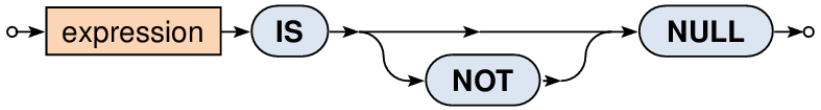
- **String matching predicate**
  - Tests whether a string value matches a given pattern
    - This pattern may contain special characters:
      - % matches an arbitrary substring (even empty)
      - _ matches an arbitrary character
    - Optional escaping character can also be set



  - 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

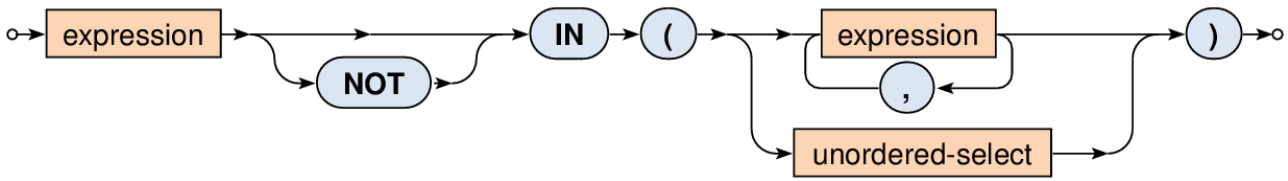
Three-Value Logic

- **Truth tables**

p	q	p AND q	p OR q	NOT q
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

Search Conditions

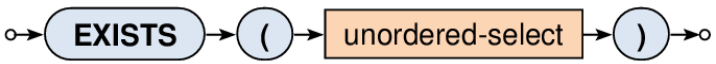
- **Set membership predicate**
  - Tests whether a value exists in a given set of values
    - Example: Company IN ('KLM', 'Emirates')



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

Search Conditions

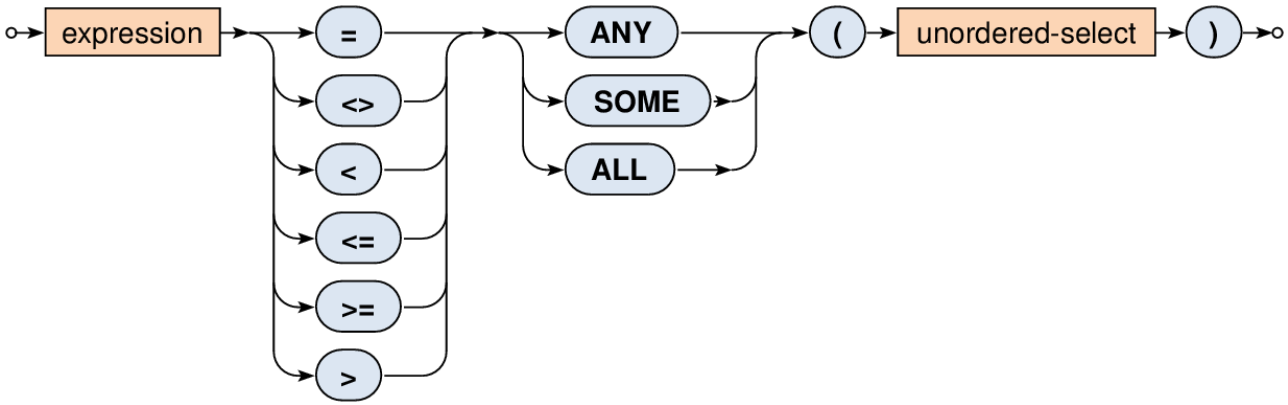
- **Existential quantifier predicate**
  - Tests whether a given set is not empty
  - Can be used to simulate the universal quantifier too
    - ∀ corresponds to ¬∃¬



  - 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

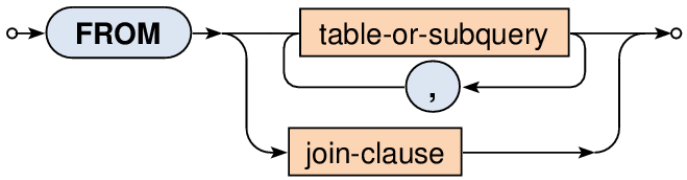
- **Set comparison predicates**
  - **ANY** and **SOME** (synonyms)
    - At least one row from the nested query must satisfy the given comparison operator
    - ANY (∅) = `FALSE`
    - ANY (א) = `UNKNOWN`

From Clause

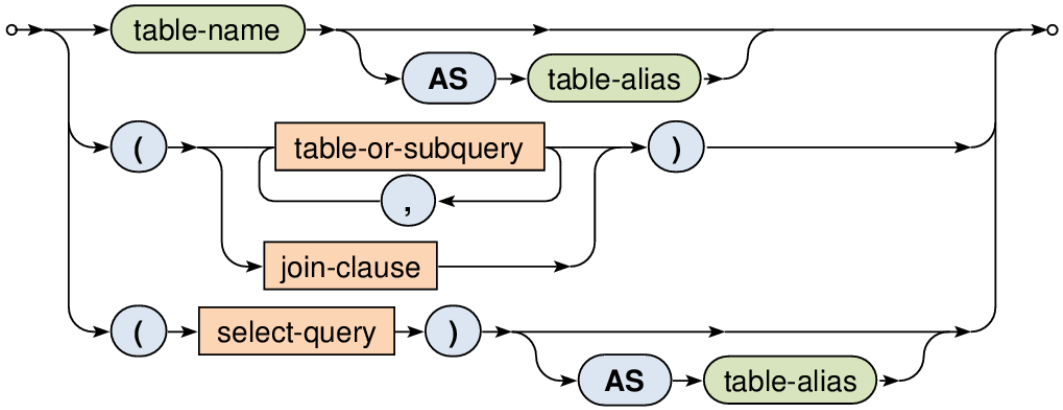
- SELECT … **FROM …** WHERE … ORDER BY …
  - **Description of tables to be queried**
    - Actually not only tables, but also **nested queries or views**
  - **Old way**
    - Comma separated **list of tables** (…)
    - **Cartesian product** of their rows is assumed
    - Required join conditions are 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 … **FROM …** WHERE … ORDER BY …
  - **Description of tables to be queried**
    - Overall diagram
      - Both old and new ways

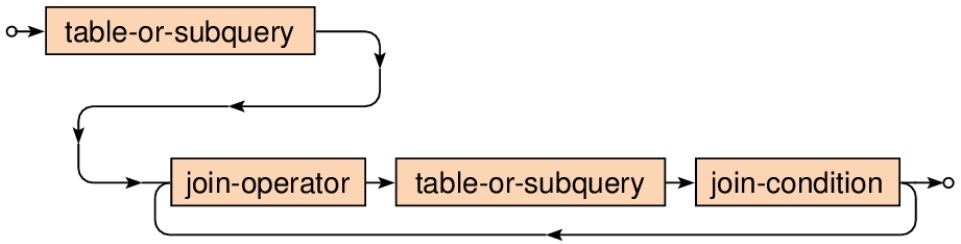


    - Tables and subqueries
      - Table name, auxiliary parentheses, direct select statement



From Clause

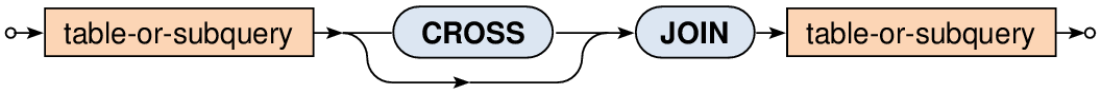
- SELECT … **FROM …** WHERE … ORDER BY …
  - **Description of tables to be queried**
    - Basic structure of joins



      - 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

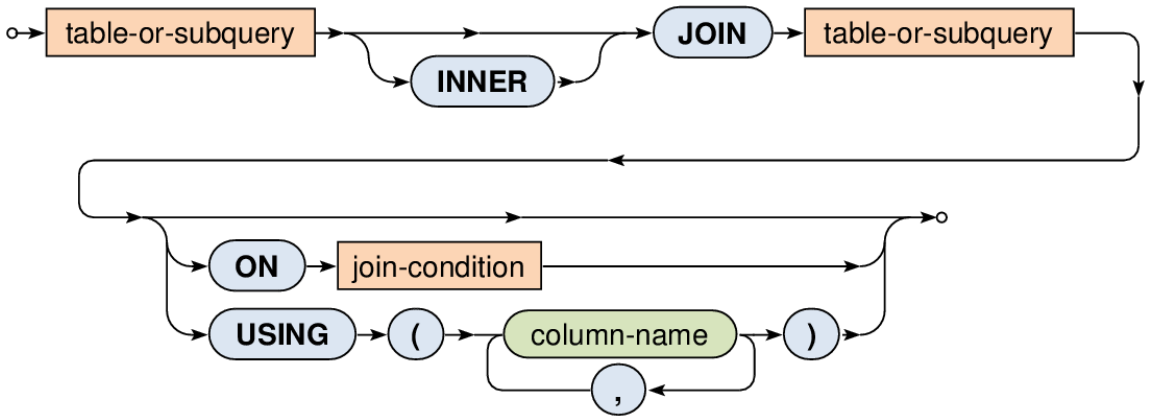


Table Joins

- **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

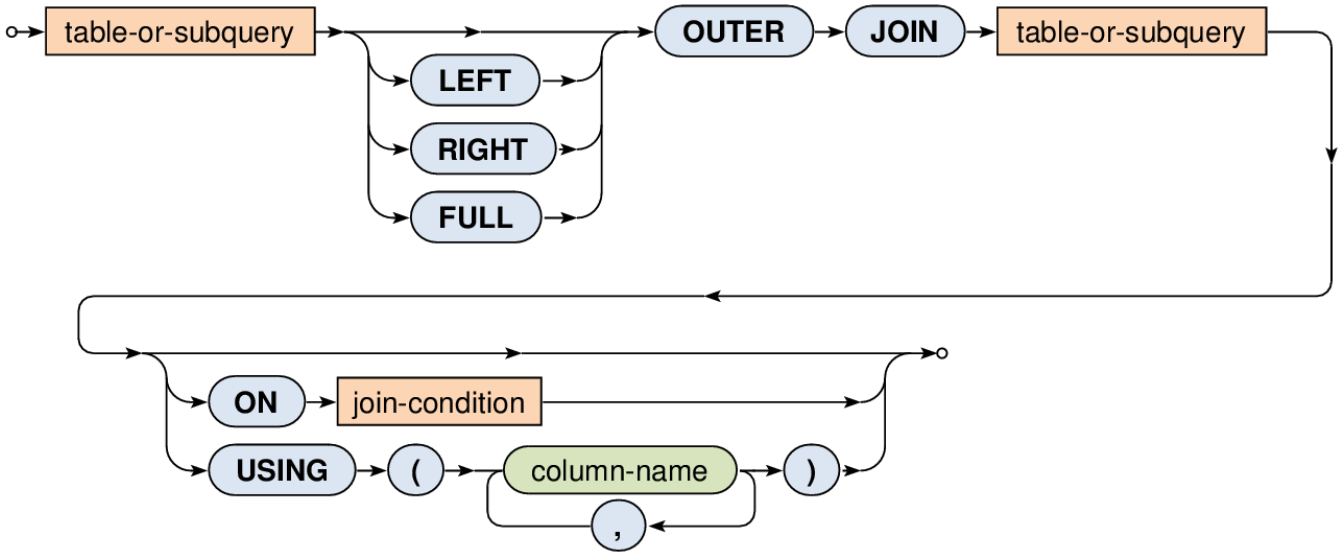


Table Joins

- **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)

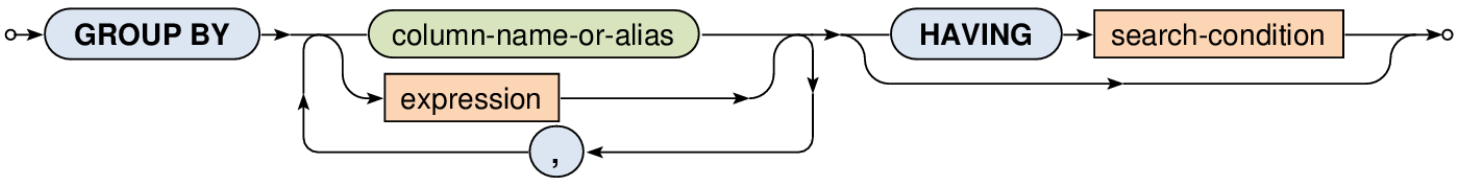
A	T1.*
1	…
2	…
3	…

A	T2.*
1	…
4	…

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

Aggregation

- **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



Aggregation: 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

⇨

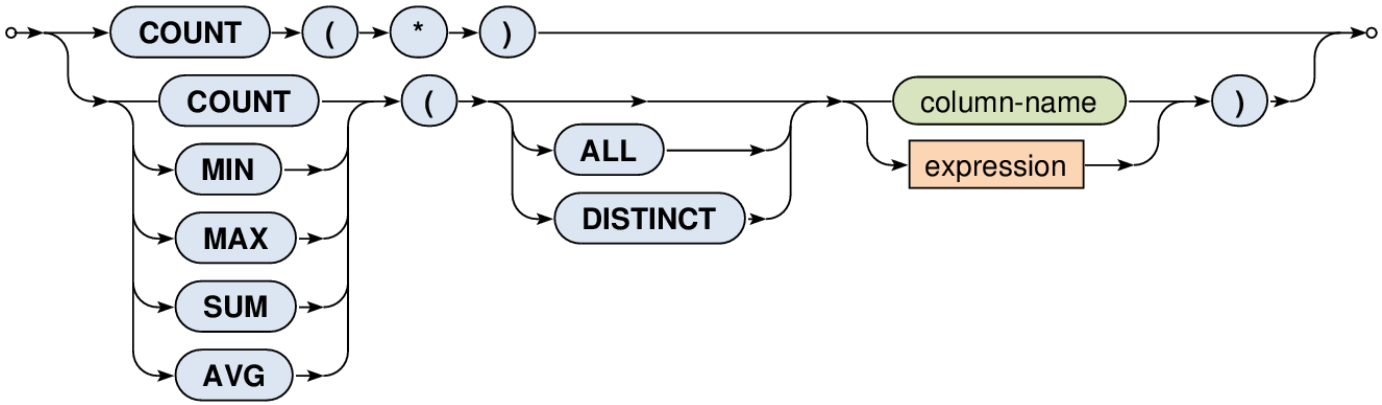
Company	Flights
CSA	3
Air Canada	**1**
KLM	2

⇩

Company	Flights
CSA	3
KLM	2

Aggregation

- **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**



Aggregation

- **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`

Aggregation: 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

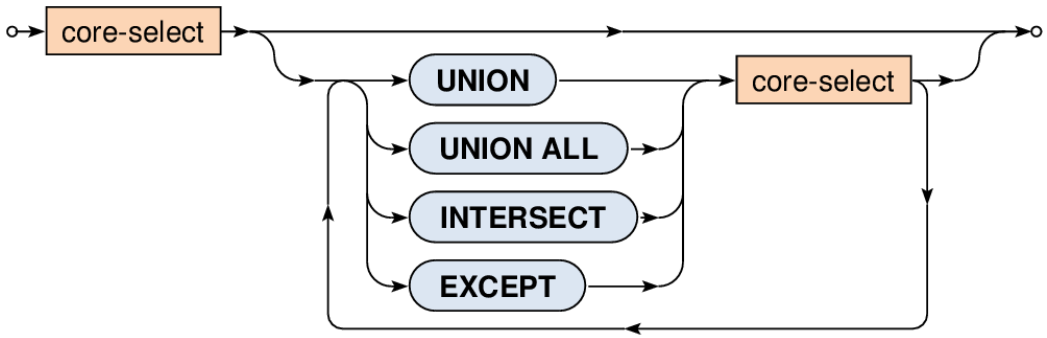
```
SELECT
    COUNT(*) AS Flights,
    COUNT(DISTINCT Company) AS Companies,
    SUM(Passengers) AS PSum,
    AVG(Passengers) AS PAvg,
    MIN(Passengers) AS PMin,
    MAX(Passengers) AS PMax
FROM Flights
```

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

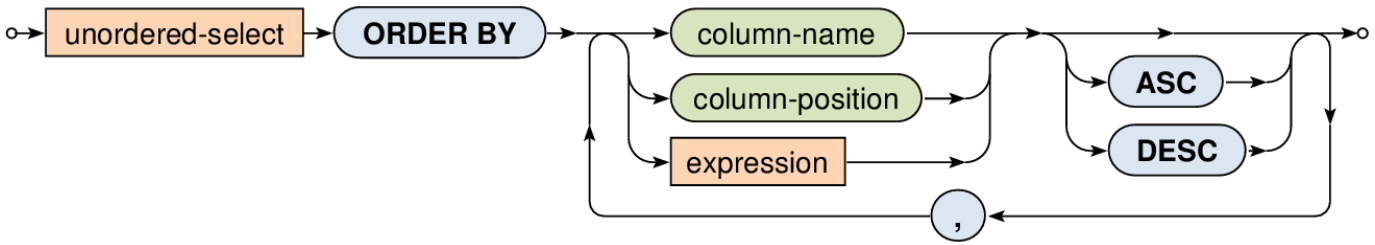
```
SELECT Company FROM Flights
UNION
SELECT Company FROM Aircrafts
```

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