Part 7: SQL I

References:

- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999.
  Chap. 8, "SQL — The Relational Database Standard" (Sect. 8.2, 8.3.3, part of 8.3.4.)
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Ed., 1999. Ch. 4.
- Kemper/Eickler: Datenbanksysteme (in German), Ch. 4, Oldenbourg, 1997.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Heuer/Saake: Datenbanken, Konzepte und Sprachen (in German), Thomson, 1995.
- Date/Darwen: A Guide to the SQL Standard, Fourth Edition, Addison-Wesley, 1997.
- Date: A Guide to the SQL Standard, First Edition, Addison-Wesley, 1987.
- van der Lans: SQL, Der ISO-Standard (in German). Hanser, 1990.
- Oracle 8i SQL Reference, Release 2 (8.1.6), Dec. 1999, Part No. A76989-01.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.
- Microsoft Jet Database Engine Programmer's Guide, 2nd Ed. (Part of MSDN Library).
- DuBois: MySQL. New Riders Publishing, 2000, ISBN 0-7357-0921-1, 756 pages.
- Boyce/Chamberlin: SEQUEL: A structured English query language.
  In ACM SIGMOD Conf. on the Management of Data, 1974.
- Astrahan et al: System R: A relational approach to database management.
  ACM Transactions on Database Systems 1(2), 97–137, 1976.

Objectives

After completing this chapter, you should be able to:

- write advanced queries in SQL including, e.g., several tuple variables over the same relation.

  Subqueries, Aggregations, `UNION`, outer joins and sorting are treated in Chapter 8.

- Avoid errors and unnecessary complications.

  E.g. you should be able to explain the concept of an inconsistent condition. You should also be able to check whether a query can possibly produce duplicates (at least in simple cases).

- Check given queries for errors or equivalence.
- Evaluate the portability of certain constructs.

Overview

SQL

- Today, SQL is the only database language for relational DBMSs (industry standard).

  Other languages such as QUEL have died (the language QBE has at least inspired some graphical interfaces to databases, e.g. in Access, and the language Datalog has influenced SQL:1999 in certain ways and is still studied and further developed in research). Any commercial RDBMS must today offer an SQL interface. There might be other (typically graphical) interfaces in addition.

- SQL is used for:
  - ◇ Interactive "ad-hoc" commands and
  - ◇ application program development (embedded into other languages like C, Java, HTML).

Example Database

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

Basic SQL Queries (1)

Simple One-Table Queries (SQL for Beginners):

- A simple SQL query has the form

```
SELECT Colums
FROM   Table
WHERE  Condition
```

- For example, the following query lists all homework results for student 101:

```
SELECT ENO, POINTS
FROM   RESULTS
WHERE  CAT = 'H' AND SID = 101
```

ENO	POINTS
1	10
2	8

Basic SQL Queries (2)

From Relational Algebra to SQL:

- A relational algebra expression

$$\pi_{A_1,\ldots,A_n}(\sigma_F(R_1 \times \cdots \times R_m))$$

is written in SQL as

```
SELECT A1, ..., An
FROM   R1, ..., Rm
WHERE  F
```

- If different R_i have attributes with the same name A, write $R_i.A$ to make the reference unique.
- Duplicate answers might occur ($\rightarrow$ `SELECT DISTINCT`).

Basic SQL Queries (3)

From Relational Algebra to SQL, continued:

- However, one can also use an explicit renaming:

$$\pi_{A_1,\ldots,A_n}(\sigma_F(\rho_{X_1}(R_1) \times \cdots \times \rho_{X_m}(R_m)))$$

  is written in SQL as

```
SELECT  A1, ..., An
FROM    R1 X1, ..., Rm Xm
WHERE   F
```

- SQL has a `UNION` operator for combining the results of `SELECT`-expressions.

- While there is also `EXCEPT` for set difference, one normally would use subqueries for this (see Part 8).

Basic SQL Queries (4)

From Logic to SQL:

- SQL is closely related to tuple relational calculus.

  This is a variant of first order logic where the relations are treated as sorts, and the attributes as functions of arity one, written in dot notation. (At least, this is the subset of tuple calculus used in SQL.)

- E.g. all homework results of student 101 (again):

$$\{\texttt{X.eno},\ \texttt{X.points}\ [\texttt{results X}] \mid \texttt{X.cat} = \texttt{'H'} \land \texttt{X.sid} = 101\}$$

- Only a small syntactic variant of the SQL query:

```
SELECT X.ENO, X.POINTS
FROM   RESULTS X
WHERE  X.CAT = 'H' AND X.SID = 101
```

Basic SQL Queries (5)

From Logic to SQL, continued:

- If the condition F does not contain quantifiers,

  $\{t_1, \ldots, t_n\ [R_1\ X_1, \ldots, R_m\ X_m] \mid \exists S_1\ Y_1 \ldots \exists S_k\ Y_k\colon F\}$

  is written in SQL as

  ```
  SELECT $t_1$, ..., $t_n$
  FROM   $R_1$ $X_1$, ..., $R_m$ $X_m$, $S_1$ $Y_1$, ..., $S_k$ $Y_k$
  WHERE  $F$
  ```

- There might be small differences with duplicates.

  If necessary use `SELECT DISTINCT` to remove duplicates.

- For quantifiers inside F, subqueries must be used.

Relation to Theory

- SQL is based on logic (tuple calculus).

  The translation of nested quantifiers into subqueries is still very direct (if the variable is bound to a relation). SQL has many extensions, e.g., aggregations and the possibility to use subqueries as terms.

- SQL also contains an increasing number of constructs from relational algebra.

- SQL was designed to be relatively near to natural language.

  SQL often has many different ways to express the same condition. Many syntactic variants were introduced because they “read better”.

- In SQL, duplicates are significant.

History

- SEQUEL, an earlier version of SQL, was designed by Chamberlin, Boyce et al. at IBM Research, San Jose (1974).

  SEQUEL stands for "Structured English Query Language". Some people pronounce SQL this way. Others use "ess-cue-ell". The name was changed for legal reasons (SEQUEL was a registered trademark). Codd was also in San Jose when he invented the relational model.

- SQL was the language of System/R (1976/77).

  System/R was a very influential research prototype.

- First commercial systems supporting SQL were Oracle (1979) and IBM SQL/DS (1981).

Standards (1)

- First Standard 1986/87 (ANSI/ISO).

  This was very late as there were already several SQL systems on the market. The standard was the "smallest common denominator". It contains only the common features of the existing implementations.

- Extension for foreign keys etc. in 1989 (SQL-89).

  This version is called also SQL-1. All commercial implementations today support this standard, but each have significant extensions. The standard had 120 pages.

- Major Extension: SQL2 or SQL-92 (1992).

  626 pages, upward compatible to SQL-1. The standard defines three levels: "entry", "intermediate", "full" (later a transitional level was added between entry and intermediate). Oracle 8.0 and SQL Server 7.0 have only entry level conformance, but many extensions.

Standards (2)

- Another major extension: SQL:1999.

  First SQL:1999 was announced as a preliminary version of the SQL3 standard, but it seems that now it is SQL3. Until 12/2000, the volumes 1–5 and 10 of the SQL:1999 standard appeared. They have together 2355 pages. Instead of language levels, the standard defines now "Core SQL" and a number of packages of features, for which vendors can claim conformance.

- Important new features in SQL:1999:
  - ◇ User-defined data types, type constructors.

    One can now define "distinct types", which are similar to domains, but now comparisons between different distinct types yield an error. There are also type constructors "`ARRAY`" and "`ROW`" for structured attribute values, and "`REF`" for pointers to rows.

Standards (3)

- Important new features in SQL:1999, continued:
  - ◇ OO-Features (e.g. inheritance/subtables).
  - ◇ Recursive queries.
  - ◇ Triggers, Persistent Stored Modules.
- Minor update: SQL:2003.

  The biggest addition are support for XML and the management of external data. Furthermore, there are now sequence generators and a "`MULTISET`" type constructor. Otherwise it is simply the second editi-on of the SQL:1999 standard. Also the OLAP constructs (On-Line Analytical Processing) that were published as an amendment to the SQL:1999 standard are now integrated to the SQL:2003 standard.

Standards (4)

- Volumes of the SQL:2003 Standard:
  - ◇ Part 1: Framework (SQL/Framework) [84 pp]
  - ◇ Part 2: Foundation (SQL/Foundation) [1268 pp]
  - ◇ Part 3: Call-Level Interface (SQL/CLI) [406 pp]

    This specifies a set of procedures which can be used to execute SQL statements and get the results (similar to ODBC).
  - ◇ Part 4: Persistant Stored Modules (SQL/PSM) [186 pp]

    This is a programming language for stored procedures. It makes SQL computationally complete and is similar to PL/SQL in Oracle, or Transact SQL in Microsoft SQL Server and Sybase.

Standards (5)

- Volumes of the SQL:2003 standard, continued:
  - ◇ Part 9: Management of External Data (SQL/MED)
  - ◇ Part 10: Object Language Bindings (SQL/OLB) [404 pages]

    This defines how SQL statements can be embedded into Java programs.
  - ◇ Part 11: Information and Definition Schemas (SQL/Schemas) [298 pages]

    This defines a standard data dictionary (system catalog).

Standards (6)

- Volumes of the SQL:2003 standard, continued:
  - ◇ Part 13: SQL Routines and Types using the Java Programming Language (SQL/JRT) [206 pp]

    This defines how Java static methods can be used inside SQL statements, and how Java classes can be used as SQL structured types.
  - ◇ Part 14: XML-Related Specifications (SQL/XML) [268 pp]
- The official name of Part n of the standard is [INCITS/] ISO/IEC 9075-n-2003: Information technology — Database Languages — SQL.

Standards (7)

- The missing part numbers of the SQL:2003 standard will probably never appear.

  Part 5 described in SQL:1999 “object language bindings”. This was integrated into Part 2. Part 6 was intended to cover the X/OPEN XA-specification (transactions). This part was deleted. Part 7 was intended to specify SQL/Temporal. This effort was stopped, because the committee had too big differences in their opionion. Part 8 was intended to cover extended objects/abstract data types, and was integrated into Part 2. Part 12 is intended to cover replication.

- There is a related standard ISO/IEC 13249: “SQL multimedia and application packages”.

  Part 1: Framework, Part 2: Full-Text, Part 3: Spatial, Part 5: Still Image, Part 6: Data Mining.

Syntax Formalism

- In this course, the syntax of SQL queries is defined with "syntax graphs" (Example on next slide).

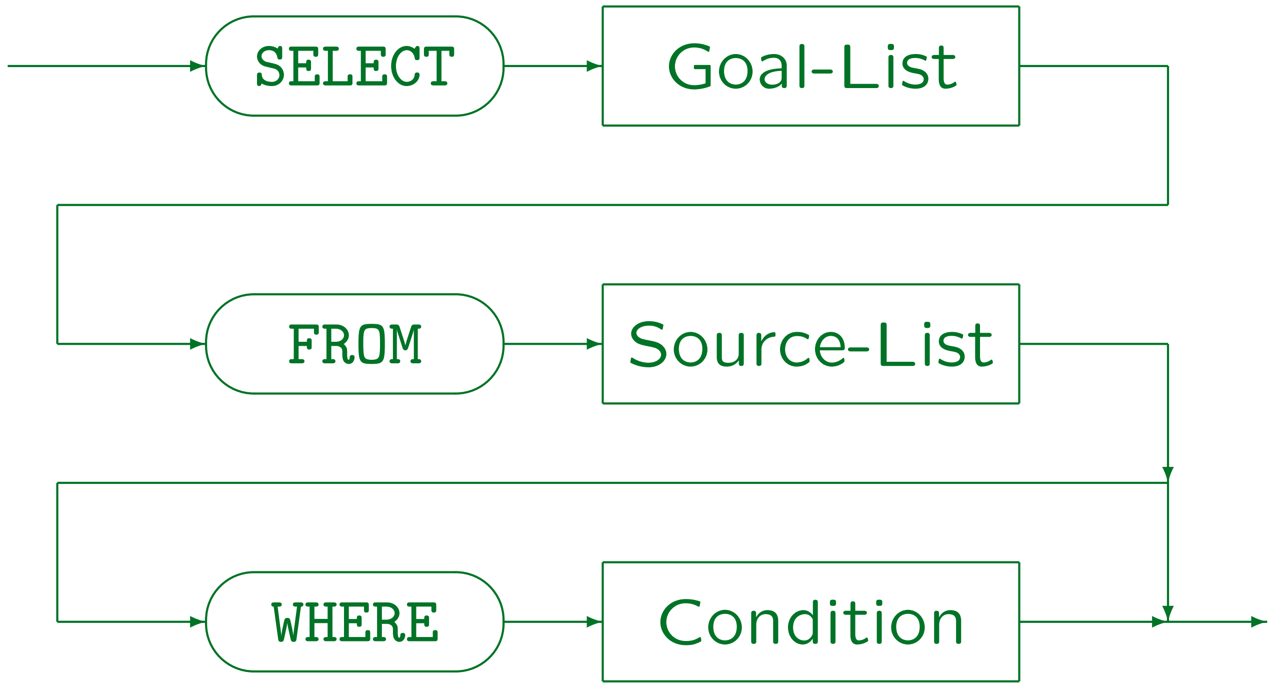
  This is an alternative to context-free grammars (it defines the same class of languages). Appendix A explains syntax graphs in more detail.

- Basically, in order to generate a syntactically correct character sequence, one has to follow a path from start to goal through the graph.

  Words in ovals are written directly to the output, boxes are "calls" to other graphs: At this point one must find a path to the graph with the name written in the box, and then return to the edge that leaves the box. The Oracle SQL Reference Manual also contains syntax graphs, but there the meaning of oval and box is inversed.

Basic Query Syntax (1)

SELECT-Expression (Simplified):



Basic Query Syntax (2)

- Every SQL query must contain the keywords `SELECT` and `FROM`.

  Oracle provides a relation "`DUAL`" which has only one row. It can be used if only a computation is done without access to the database: "`SELECT TO_CHAR(SQRT(2)) FROM DUAL`" computes $\sqrt{2}$.

- However, in SQL Server, Access, and MySQL, the `FROM`-clause can be omitted, e.g. `SELECT 1+1`.

  In Oracle, DB2, and the SQL-92 Standard, this is a syntax error.

SQL Syntax in this Course

- The complete SQL:2003 is too large to be treated in this course. Furthermore, a large part of the standard is not yet implemented in current DBMS.
- The complete SQL/89 ($\sim$ Entry Level SQL/92) will be treated, and that part of SQL:2003 which is implemented in most of the major DBMS.

  Sometimes, details in the SQL syntax of specific systems will be explained, usually in the small print. These are not relevant for exams. They are intended to give an impression of the portability of these constructs (and to help when you actually have to use that DBMS). In the exams, points might be taken off for extremely non-portable constructs (e.g., queries that work only in MySQL).

Overview

Lexical Syntax Overview

- The lexical syntax of a language defines how word symbols ("tokens") are composed from single characters. E.g. it defines the exact syntax of
  - ◇ Identifiers (names for e.g. tables, columns),
  - ◇ Literals (datatype constants, e.g. numbers),
  - ◇ Keywords, Operators, Punctation marks.
- Thereafter, the syntax of queries and other commands is defined in terms of these word symbols.

White Space and Comments

White space is allowed between words (tokens):

- Spaces (normally also tabulator characters)
- Line breaks
- Comments:
  - ◇ From "--" to ⟨Line End⟩

    Supported in SQL-92, Oracle, SQL Server, IBM DB2, MySQL. MySQL requires a space after the "--", SQL-92 does not. Access does not support this comment, and also not `/* ...*/`.
  - ◇ From "/*" to "*/"

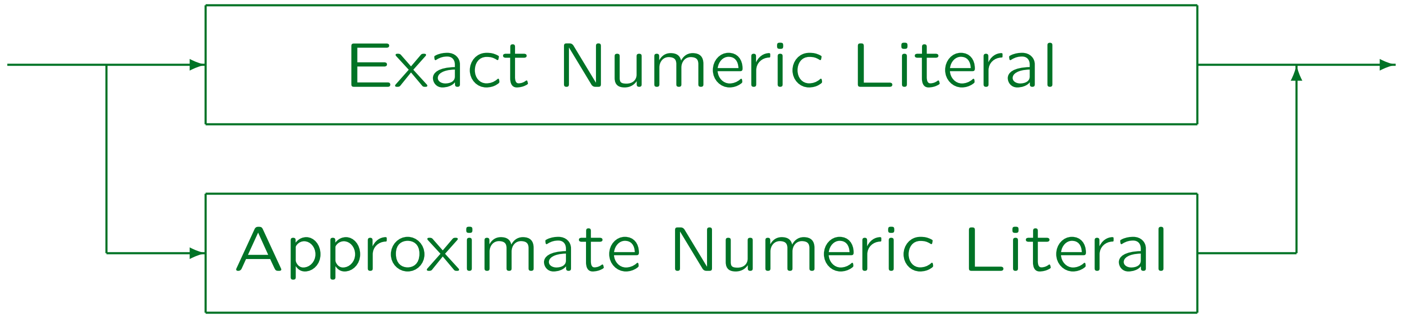
    Supported only in Oracle, SQL Server, MySQL: Less portable.

Free-Format Language

- The above rule (any whitespace between tokens) imply that SQL is a free-format language like Pascal, C, Java:
  - ◇ It is not required that "`SELECT`", "`FROM`", "`WHERE`" are written at the beginning of a new line, one could write the entire query on a single line.
  - ◇ One could distribute, e.g., a complicated condition over several lines, and use indentation to make the structure clear.

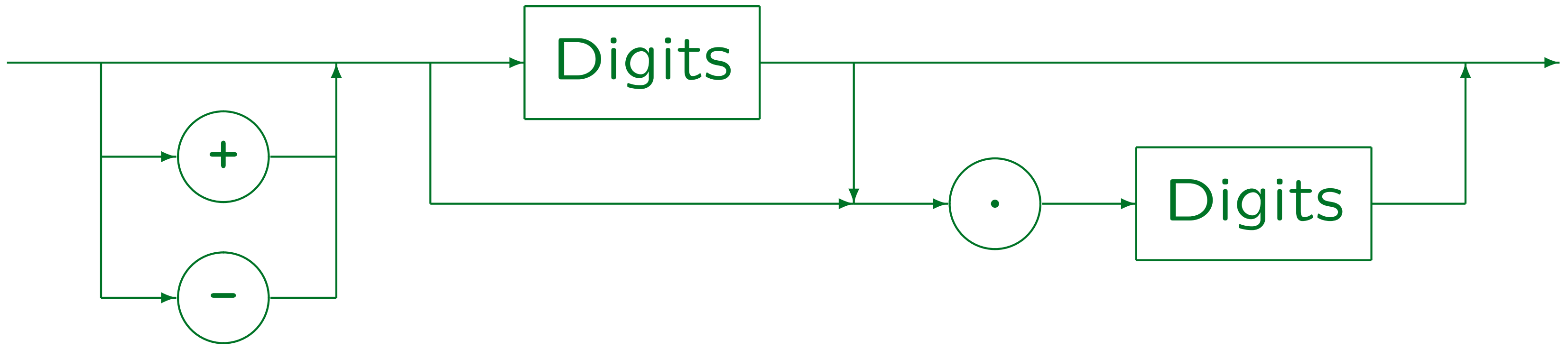
Numbers (1)

- Numeric literals are constants of numeric data types (fixed point and floating point numbers).
- E.g.: `1`, `+2.`, `-34.5`, `-.67E-8`
- Note that numbers are not enclosed in quotes.
- Numeric Literal:

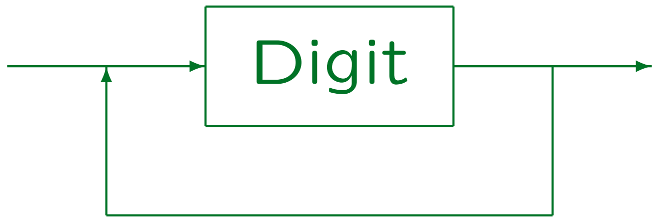


Numbers (2)

- Exact Numeric Literal:

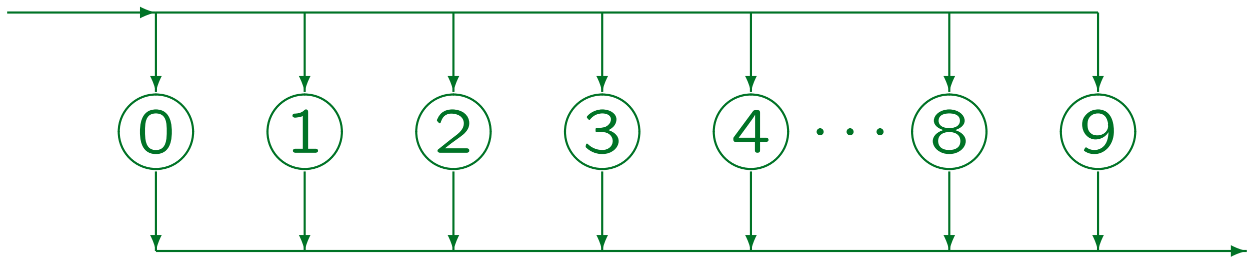


- Digits (Unsigned Integer):

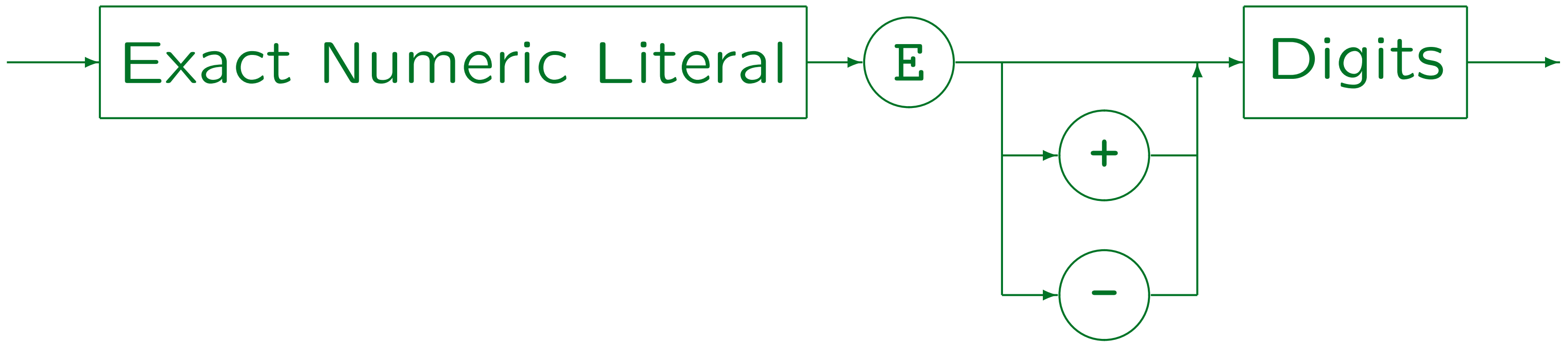


Numbers (3)

- Digit:



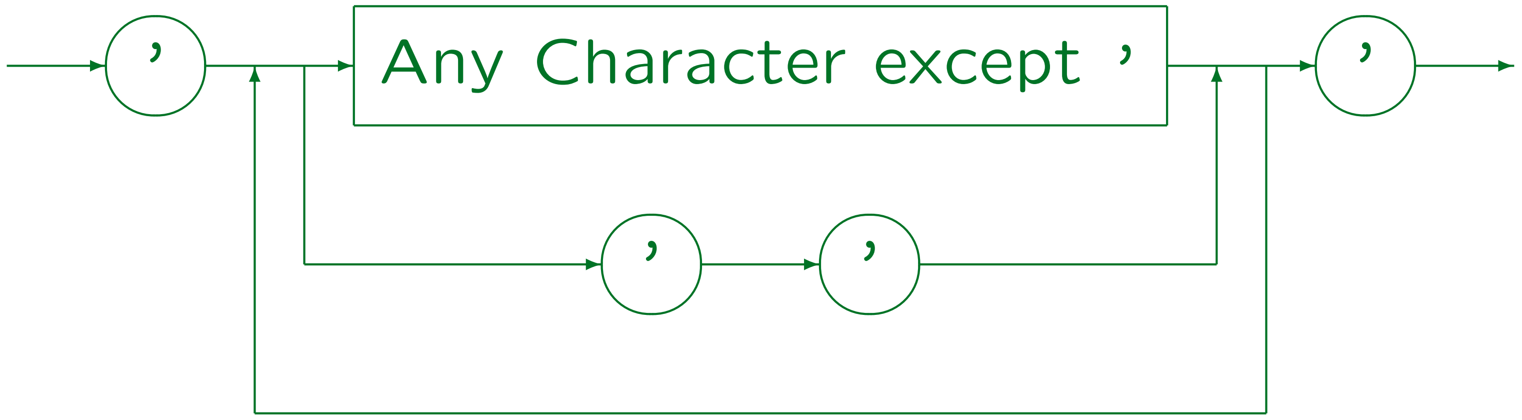
- Approximate Numeric Literal:



Character Strings (1)

- A character string constant/literal is a sequence of characters enclosed in single quotes, e.g. `'abc'`.
- Single quotes in a string must be doubled, e.g. `'John''s Book'`.

  The real value of the string is `John's Book` (with a single quote). The doubling is only a way to input it.



Character Strings (2)

- The SQL-92 standard allows splitting strings between lines (with each segment enclosed in ').

  MySQL does support this syntax. Oracle, SQL Server, and Access do not support it. However, strings can be combined with the concatenation operator (|| in Oracle, + in SQL Server and Access).

- SQL-92 and all five DBMS allow line breaks inside string constants.

  I.e. the quote can be closed on a subsequent line.

- Microsoft SQL Server, MS Access, and MySQL accept also string literals enclosed in double quotes. This does not conform to the standard.

Other Constants (1)

- There are more data types besides numbers and strings, e.g. (see Chapter 9):
  - ◇ Character strings in a national character set
  - ◇ Date, Time, Timestamp, Date/Time Interval
  - ◇ Bit strings, binary data
  - ◇ Large Objects
- The syntax of constants of these types is generally very system-dependent.

  Often, there are no constants of these types, but there is an automatic type conversion ("coercion") from strings.

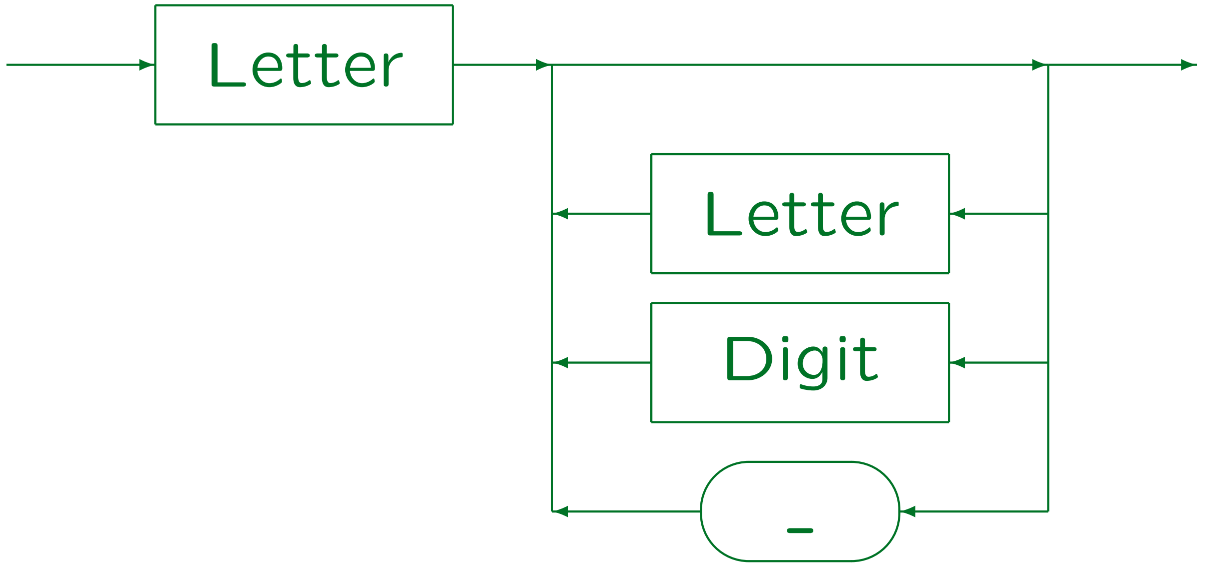
Other Constants (2)

- E.g. date values are written as follows:
  - ◇ Oracle: `'31-OCT-02'` (US), `'31.10.2002'` (Ger.).

    The default format (part of national language settings) is automatically converted, otherwise: `TO_DATE('31.10.2002','DD.MM.YYYY')`.
  - ◇ SQL-92 Syntax: `DATE '2002-10-31'`.
  - ◇ MySQL uses this syntax (also without "`DATE`").
  - ◇ DB2: `'2002-10-31'`, `'10/31/2002'`, `'31.10.2002'`.
  - ◇ SQL Server: e.g. `'20021031'`, `'10/31/2002'`, `'October 31, 2002'` (depends on language).
  - ◇ Access: `#10/31/2002#` (US), `#31.10.2002#` (Ger.).

Identifiers (1)

- Identifiers are used e.g. as table and column names.



- E.g. `Instructor_Name`, `X27`, but not `_XYZ`, `12`, `2BE`.

Identifiers (2)

- Identifiers can have up to 18 characters (at least).

System	Length	First Character	Other Characters
SQL-86	$\leq$ 18	A-Z	A-Z,0-9
SQL-92	$\leq$ 128	A-Z,a-z	A-Z,a-z,0-9,_
Oracle	$\leq$ 30	A-Z,a-z	A-Z,a-z,0-9,_,#,$
SQL Server	$\leq$ 128	A-Z,a-z,_,(@,#)	A-Z,a-z,0-9,_,@,#,$
IBM DB2	$\leq$ 18 (8)	A-Z,a-z	A-Z,a-z,0-9,_
Access	$\leq$ 64	A-Z,a-z	A-Z,a-z,0-9,_
MySQL	$\leq$ 64	A-Z,a-z,0-9,_,$	A-Z,a-z,0-9,_,$

Intermediate SQL-92: "_" at the end forbidden. Entry Level: Like SQL-86 (plus "_"). In MySQL, identifiers can start with digits, but must contain at least one letter. Access might permit more characters, depending on the context.

- Names must be different from all reserved words.

There are a lot of reserved words, see below. Embeddings in a programming language (PL/SQL, Visual Basic) add reserved words.

Identifiers (3)

- It is possible to use also national characters.

  This is implementation dependent. E.g. in Oracle, one chooses a database character set when the database is installed. Alphanumeric characters from this character set can be used in identifiers.

- Identifiers (and keywords) are not case sensitive.

  It seems that this is what the SQL-92 standard says (the book by Date/Darwen about the Standard states this clearly). Oracle SQL*Plus converts all letters outside quotes to uppercase. In SQL Server, case sensitivity can be chosen at installation time. In MySQL, case sensitivity of table names depends on the case sensitivity of file names in the underlying operating system (tables are stored as files). Within a query, one must use consistent case in MySQL. However, keywords and column names are not case sensitive.

Identifiers (4)

- For instance, consider the query

```
SELECT X.FIRST, X.LAST
FROM   STUDENTS X
```

- The following query is completely equivalent:

```
select X . First ,
       x.LaSt
  From Students X
```

(This also demonstrates that SQL is free format.)

But note that string comparisons are normally case sensitive:

```
select x.last from students x where x.first = 'ann'
```

will return 0 answers (although there is `'Ann'`).

SQL Reserved Words (1)

^{1} = Oracle 8.0
^{2} = SQL-92
^{3} = SQL Server 7

— A —

ABSOLUTE^{2}
ACCESS^{1}
ACTION^{2}
ADD^{1,2,3}
ALL^{1,2,3}
ALLOCATE^{2}
ALTER^{1,2,3}
AND^{1,2,3}
ANY^{1,2,3}
ARE^{2}
AS^{1,2,3}
ASC^{1,2,3}
ASSERTION^{2}
AT^{2}
AUTHORIZATION^{2,3}
AUDIT^{1}
AVG^{2,3}

— B —

BACKUP^{3}
BEGIN^{2,3}
BETWEEN^{1,2,3}
BIT^{2}
BIT_LENGTH^{2}
BOTH^{2}
BREAK^{3}
BROWSE^{3}
BULK^{3}
BY^{1,2,3}

— C —

CASCADE^{2,3}
CASCADED^{2}
CASE^{2,3}
CATALOG^{2}
CHAR^{1,2}
CHARACTER^{2}
CHAR_LENGTH^{2}
CHARACTER_LENGTH^{2}
CHECK^{1,2,3}
CHECKPOINT^{3}
CLOSE^{2,3}
CLUSTER^{1}
CLUSTERED^{3}
COALESCE^{2,3}
COLLATE^{2}
COLLATION^{2}
COLUMN^{1,3}
COMMENT^{1}
COMMIT^{2,3}
COMMITTED^{3}
COMPRESS^{1}
COMPUTE^{3}

SQL Reserved Words (2)

CONFIRM[3]
CONNECT[1,2]
CONNECTION[2]
CONSTRAINT[2,3]
CONSTRAINTS[2]
CONTAINS[3]
CONTAINSTABLE[3]
CONTINUE[2,3]
CONTROLROW[3]
CONVERT[2,3]
CORRESPONDING[2]
COUNT[2,3]
CREATE[1,2,3]
CROSS[2,3]
CURRENT[1,2,3]
CURRENT_DATE[2,3]
CURRENT_TIME[2,3]
CURRENT_TIMESTAMP[2,3]
CURRENT_USER[2,3]
CURSOR[2,3]

— **D** —

DATABASE[3]
DATE[1,2]
DAY[2]
DBCC[3]
DEALLOCATE[2,3]
DEC[2]
DECIMAL[1,2]
DECLARE[2,3]
DEFAULT[1,2,3]
DEFERRABLE[2]
DEFERRED[2]
DELETE[1,2,3]
DENY[3]
DESC[1,2]
DESCRIBE[2]
DESCRIPTOR[2]
DIAGNOSTICS[2]
DISCONNECT[2]
DISK[3]
DISTINCT[1,2,3]
DISTRIBUTED[3]
DOMAIN[2]
DOUBLE[2,3]
DROP[1,2,3]
DUMMY[3]
DUMP[3]

— **E** —

ELSE[1,2,3]
END[2,3]
END-EXEC[2]
ERRLVL[3]
ERROREXIT[3]
ESCAPE[2,3]
EXCEPT[2,3]
EXCEPTION[2]

SQL Reserved Words (3)

EXCLUSIVE[1]
EXEC[2,3]
EXECUTE[2,3]
EXISTS[1,2,3]
EXIT[3]
EXTERNAL[2]
EXTRACT[2]

— **F** —

FALSE[2]
FETCH[2,3]
FILE[1,3]
FILLFACTOR[3]
FIRST[2]
FLOAT[1,2]
FLOPPY[3]
FOR[1,2,3]
FOREIGN[2,3]
FOUND[2]
FREETEXT[3]
FREETEXTTABLE[3]
FROM[1,2,3]
FULL[2,3]

— **G** —

GET[2]
GLOBAL[2]
GO[2]
GOTO[2,3]
GRANT[1,2,3]
GROUP[1,2,3]

— **H** —

HAVING[1,2,3]
HOLDLOCK[3]
HOUR[2]

— **I** —

IDENTITY[2,3]
IDENTITY_INSERT[3]
IDENTITYCOL[3]
IDENTIFIED[1]
IF[3]
IMMEDIATE[1,2]
IN[1,2,3]
INCREMENT[1]
INDEX[1,3]
INDICATOR[2]
INITIAL[1]
INITIALLY[2]
INNER[2,3]
INPUT[2]
INSENSITIVE[2]
INSERT[1,2,3]
INT[2]
INTEGER[1,2]
INTERSECT[1,2,3]
INTERVAL[2]
INTO[1,2,3]
IS[1,2,3]

SQL Reserved Words (4)

ISOLATION[2,3]
— J —
JOIN[2,3]
— K —
KEY[2,3]
KILL[3]
— L —
LANGUAGE[2]
LAST[2]
LEADING[2]
LEFT[2,3]
LEVEL[1,2,3]
LIKE[1,2,3]
LINENO[3]
LOAD[3]
LOCAL[2]
LOCK[1]
LONG[1]
LOWER[2]
— M —
MATCH[2]
MAX[2,3]
MAXEXTENTS[1]
MIN[2,3]
MINUS[1]
MINUTE[2]
MIRROREXIT[3]
MODE[1]
MODIFY[1]
MODULE[2]
MONTH[2]
— N —
NAMES[2]
NATIONAL[2,3]
NATURAL[2]
NCHAR[2]
NETWORK[1]
NEXT[2]
NO[2]
NOAUDIT[1]
NOCHECK[3]
NOCOMPRESS[1]
NONCLUSTERED[3]
NOT[1,2,3]
NOWAIT[1]
NULL[1,2,3]
NULLIF[2,3]
NUMBER[1]
NUMERIC[2]
— O —
OCTET_LENGTH[2]
OF[1,2,3]
OFF[3]
OFFLINE[1]
OFFSETS[3]
ON[1,2,3]

SQL Reserved Words (5)

ONCE[3]
ONLINE[1]
ONLY[2,3]
OPEN[2,3]
OPENDATASOURCE[3]
OPENQUERY[3]
OPENROWSET[3]
OPTION[1,2,3]
OR[1,2,3]
ORDER[1,2,3]
OUTER[2,3]
OUTPUT[2]
OVER[3]
OVERLAPS[2]

— **P** —

PARTIAL[2]
PCTFREE[1]
PERCENT[3]
PERM[3]
PERMANENT[3]
PIPE[3]
PLAN[3]
POSITION[2]
PRECISION[2,3]
PREPARE[2,3]
PRESERVE[2]
PRIMARY[2,3]
PRINT[3]
PRIOR[1,2]
PRIVILEGES[1,2,3]
PROC[3]
PROCEDURE[2,3]
PROCESSEXIT[3]
PUBLIC[1,2,3]

— **R** —

RAISERROR[3]
RAW[1]
READ[2,3]
READTEXT[3]
REAL[2]
RECONFIGURE[3]
REFERENCES[2,3]
RELATIVE[2]
RENAME[1]
REPEATABLE[3]
REPLICATION[3]
RESOURCE[1]
RESTORE[3]
RESTRICT[2,3]
RETURN[3]
REVOKE[1,2,3]
RIGHT[2,3]
ROLLBACK[2,3]
ROW[1]
ROWCOUNT[3]
ROWGUIDCOL[3]

SQL Reserved Words (6)

ROWID[1]
ROWNUM[1]
ROWS[1,2]
RULE[3]

— **S** —

SAVE[3]
SCHEMA[2,3]
SCROLL[2]
SECOND[2]
SECTION[2]
SELECT[1,2,3]
SERIALIZABLE[3]
SESSION[1,2]
SESSION_USER[2,3]
SET[1,2,3]
SETUSER[3]
SHARE[1]
SHUTDOWN[3]
SIZE[1,2]
SMALLINT[1,2]
SOME[2,3]
SQL[2]
SQLCODE[2]
SQLERROR[2]
SQLSTATE[2]
START[1]
STATISTICS[3]
SUBSTRING[2]
SUCCESSFUL[1]
SUM[2,3]
SYNONYM[1]
SYSDATE[1]
SYSTEM_USER[2,3]

— **T** —

TABLE[1,2,3]
TAPE[3]
TEMP[3]
TEMPORARY[2,3]
TEXTSIZE[3]
THEN[1,2,3]
TIME[2]
TIMESTAMP[2]
TIMEZONE_HOUR[2]
TIMEZONE_MINUTE[2]
TO[1,2,3]
TOP[3]
TRAILING[2]
TRAN[3]
TRANSACTION[2,3]
TRANSLATE[2]
TRANSLATION[2]
TRIGGER[1,3]
TRIM[2]
TRUE[2]
TRUNCATE[3]
TSEQUAL[3]

SQL Reserved Words (7)

— **U** —

UID[1]
UNCOMMITTED[3]
UNION[1,2,3]
UNIQUE[1,2,3]
UNKNOWN[2]
UPDATE[1,2,3]
UPDATETEXT[3]
UPPER[2]
USAGE[2]
USE[3]
USER[1,2,3]
USING[2]

— **V** —

VALIDATE[1]
VALUE[2]
VALUES[1,2,3]
VARCHAR[1,2]
VARCHAR2[1]
VARYING[2,3]
VIEW[1,2,3]

— **W** —

WAITFOR[3]
WHEN[2,3]
WHENEVER[1,2]
WHERE[1,2,3]
WHILE[3]
WITH[1,2,3]
WORK[2,3]
WRITE[2]
WRITETEXT[3]

— **Y** —

YEAR[2]

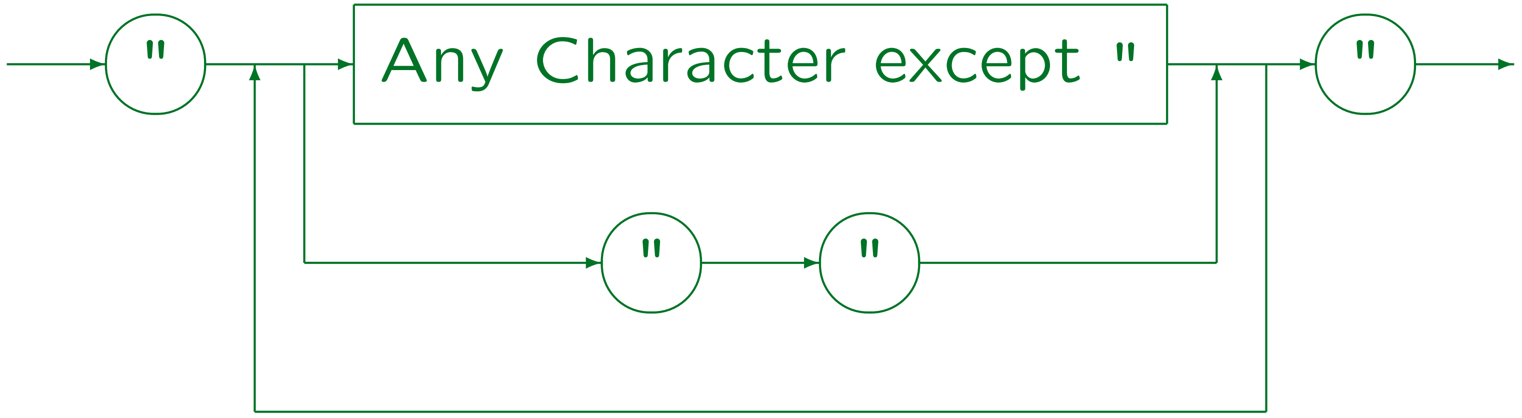
— **Z** —

ZONE[2]

Delimited Identifiers (1)

- It is possible to use any sequence of characters in double quotes as identifiers, e.g. "id, 2!".

  Such identifiers are case-sensitive, and there are no conflicts with reserved words. SQL-86 does not contain them.



Delimited Identifiers (2)

- Delimited identifiers are not character string constants! Character strings have the form '...'.

  SQL Server accepts ' and " for string constants, and uses [...] for delimited identifiers. "SET QUOTED_IDENTIFIER ON" selects the SQL-92 standard behaviour (but quoted identifiers are not case sensitive). Access understands [...] and `...` for delimited identifiers and excludes the characters !.`[]" and leading spaces in delimited identifiers.

- E.g. if you write in Oracle:

```
SELECT * FROM EMP WHERE ENAME = "JONES"
```

  Error: "JONES" is an invalid column name.

  Quoted identifiers are normally used only to rename output columns (or if column names become reserved words in a new DBMS version).

Delimited Identifiers (3)

- Delimited identifiers are often used when output columns are renamed, e.g.

  ```
  SELECT FIRST AS "First Name", LAST "Last Name"
  FROM   STUDENTS
  ```

  Note that “AS” is optional (except in MS Access).

- But if the new column name is a legal identifier, the double quotes are not necessary:

  ```
  SELECT FIRST AS FIRST_NAME, LAST Last_Name
  FROM   STUDENTS
  ```

- At least in Oracle, it will be printed all-uppercase.

Summary: Lexical Errors

- Using double quotes, e.g. `"Smith"`, for string constants. This is a delimited identifier, not a string.

  Some systems accept `"..."`, but that is a violation of the standard.

- Using quotes for numbers, e.g. `'123'`.

  This should give a type error. However, the DBMS may simply convert the type of one of the operands. Since < and so on are differently defined for strings and for numbers, this might be dangerous and should be avoided. E.g. `'12' < '3'`.

- Using reserved words as table, column, or tuple variable names.

  The error message might be strange (not understandable). Therefore, one should keep this possibility in mind.

Delimiting SQL Queries

- In Oracle SQL*Plus, every SQL statement must be terminated with a semicolon ";".

  Since SQL statements can extend over several lines, this is necessary so that SQL*Plus can see where the SQL statement is complete. Also when SQL is embedded into C programs, the semicolon is used as delimiter.

- But strictly speaking the semicolon is not part of the SQL statement.

  E.g. in the query analyzer window of MS SQL Server no semicolon is necessary. It might even be an error, as in the command line interface of DB2. Also, when SQL statements are passed to interface procedures as strings, as e.g. in ODBC, no semicolon is necessary.

Overview

1. Introduction: SELECT-FROM-WHERE
2. Lexical Syntax
3. Tuple Variables, FROM-Clause, Joins
4. Terms (Scalar Expressions)
5. Conditions, WHERE-Clause
6. SELECT-Clause, Duplicates

Example Database (again)

STUDENTS			
SID	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

EXERCISES			
CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Tuple Variables (1)

- The `FROM` clause can be understood as declaring variables that range over all tuples of a relation:

```
SELECT E.ENO, E.TOPIC
FROM   EXERCISES E
WHERE  E.CAT = 'H'
```

- This can be executed as:

```
for E in EXERCISES do
    if E.CAT = 'H' then
        print E.ENO, E.TOPIC
```

- `E` stands here for a single row in the table `EXERCISES` (the loop assigns each row in succession).

Tuple Variables (2)

- A tuple variable is always created: If not given a name explicitly, it will have the name of the relation:

```
SELECT EXERCISES.ENO, EXERCISES.TOPIC
FROM   EXERCISES
WHERE  EXERCISES.CAT = 'H'
```

- I.e. writing only `FROM EXERCISES` is understood as:

```
FROM   EXERCISES EXERCISES
```

(The tuple variable called "`EXERCISES`" ranges over the rows of the table "`EXERCISES`".)

Tuple Variables (3)

- If a tuple variable name is explicitly declared, e.g.,

```
FROM  EXERCISES E
```

  it is an error to try to access "`EXERCISES.ENO`".

  The tuple variable is now called "`E`", not "`EXERCISES`".

- When one refers to an attribute A of a tuple variable R, it is possible to write simply A instead of $R.A$ if R is the only tuple variable that has attribute A.

  This is explained further below. In the example, one can write "`ENO`" for the attribute, no matter whether one explicitly introduces a tuple variable or not.

Joins (1)

- Consider a query with two tuple variables:

```
SELECT  $A_1, \ldots, A_n$
FROM    STUDENTS S, RESULTS R
WHERE   $C$
```

- Then S will range over the 4 tuples in STUDENTS, and R will range over the 8 tuples in RESULTS. In principle, all $4 * 8 = 32$ combinations are considered:

```
for S in STUDENTS do
    for R in RESULTS do
        if $C$ then print $A_1, \ldots, A_n$
```

Joins (2)

- A good DBMS might use a better evaluation algorithm (depending on the condition C).

  This is the task of the query optimizer. E.g. if C contains the join condition `S.SID = R.SID`, the DBMS might loop over all tuples in `RESULTS`, and find the corresponding `STUDENTS` tuple by using an index over `STUDENTS.SID` (most systems automatically create an index over the key attributes).

- But in order to understand the meaning of a query, it suffices to consider this simple algorithm.

  The query optimizer can use any algorithm that produces the same output, possibly in a different sequence (SQL does not define the sequence of the result tuples).

Joins (3)

- The join must be explicitly specified in the WHERE-condition:

```
SELECT R.CAT, R.ENO, R.POINTS
FROM   STUDENTS S, RESULTS R
WHERE  S.SID = R.SID     -- Join Condition
AND    S.FIRST = 'Ann' AND S.LAST = 'Smith'
```

- Exercise: What will be the output of this query?

```
SELECT S.FIRST, S.LAST                Wrong!
FROM   STUDENTS S, RESULTS R
WHERE  R.CAT = 'H' AND R.ENO = 1
```

Joins (4)

- It is almost always an error if there are two tuple variables which are not linked (maybe indirectly) via join conditions.

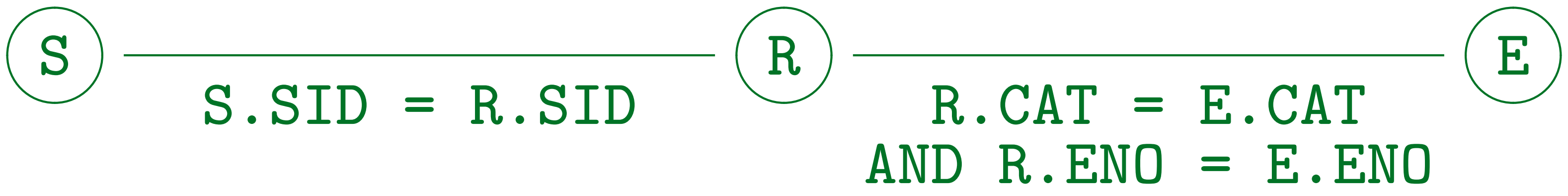
  However, it is also possible that constant values are required for the join attributes instead. In seldom cases a connection might also be done in a subquery.

- In this query, all three tuple varibles are connected:

```
SELECT E.CAT, E.ENO, R.POINTS, E.MAXPT
FROM   STUDENTS S, RESULTS R, EXERCISES E
WHERE  S.SID = R.SID
AND    R.CAT = E.CAT AND R.ENO = E.ENO
AND    S.FIRST = 'Ann' AND S.LAST = 'Smith'
```

Joins (5)

- The tuple variables are connected as follows:



- This corresponds to the key-foreign key relationships between the tables.
- If one forgets a join condition, one will often get many duplicates.

  Then it would be wrong to specify `DISTINCT` without thinking about the reason for the duplicates.

Query Formulation (1)

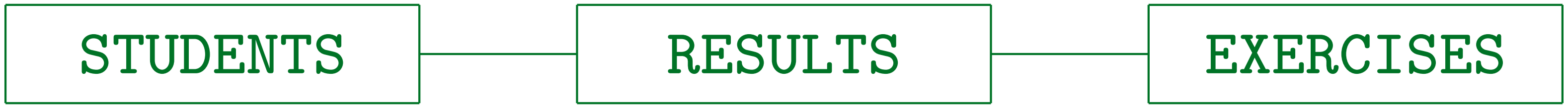
- Task: Write an SQL query which prints the topics of all exercises solved by Ann Smith.
- First it must understood that Ann Smith is a student, requiring a tuple variable `S` over `STUDENTS` and the condition `S.FIRST='Ann' AND S.LAST='Smith'`.
- Exercise topics are requested, so a tuple variable `E` over `EXERCISES` is needed, and the following piece can already be generated:

```
SELECT DISTINCT E.TOPIC
```

Several exercises can have the same topic, therefore "`SELECT DISTINCT`".

Query Formulation (2)

- Finally, `S` and `E` are not connected.
- When trying to understand a relational database schema, it helps to draw a connection graph of the tables based on common columns (foreign keys):



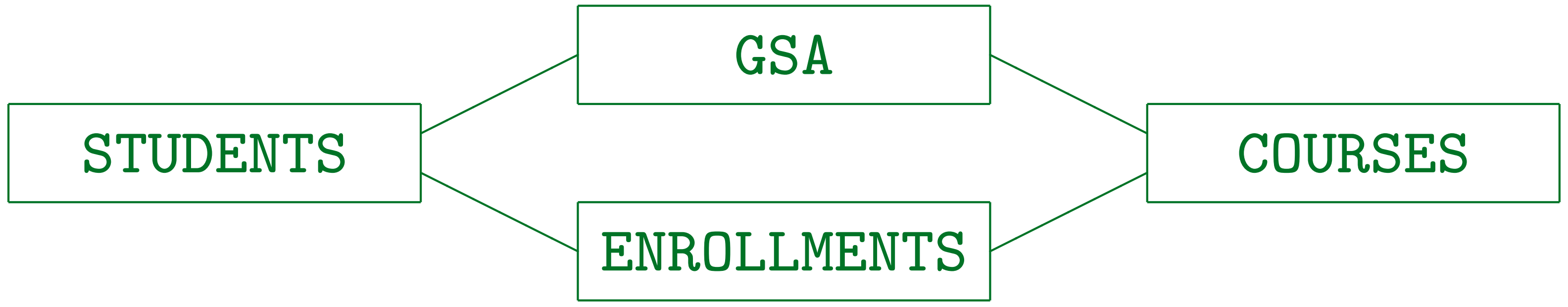
- This shows that a tuple variable `R` over `RESULTS` is required, and yields the condition

```
S.SID = R.SID AND R.CAT = E.CAT AND R.ENO = E.ENO
```

Query Formulation (3)

- It is not always that simple. The connection graph may contain cycles, which makes the selection of the right path more difficult and error-prone.
- E.g. consider a course registration database that also contains GSA assignments.

  Graduate student assistants are advanced students (often PhD students) who help correcting homeworks etc.



Unnecessary Joins (1)

- Do not join more tables than needed.

  Queries will run more slowly: Most optimizers do not remove joins.

- E.g. results for Homework 1:

```
SELECT R.SID, R.POINTS
FROM   RESULTS R, EXERCISES E
WHERE  R.CAT = E.CAT AND R.ENO = E.ENO
AND    E.CAT = 'H' AND E.ENO = 1
```

- Can the following query ever give a different result?

```
SELECT R.SID, R.POINTS
FROM   RESULTS R
WHERE  R.CAT = 'H' AND R.ENO = 1
```

Unnecessary Joins (2)

- What will be the result of this query?

```
SELECT R.SID, R.POINTS
FROM   RESULTS R, EXERCISES E
WHERE  R.CAT = 'H' AND R.ENO = 1
```

- Is there any difference between these two queries?

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S
```

```
SELECT DISTINCT S.FIRST, S.LAST
FROM   STUDENTS S, RESULTS R
WHERE  S.SID = R.SID
```

Self Joins (1)

- It might be possible that in order to generate a result tuple, more than one tuple must be considered from the same relation.
- Task: Is there a student who got 10 points for both, Homework 1 and Homework 2?

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, RESULTS H1, RESULTS H2
WHERE  S.SID = H1.SID AND S.SID = H2.SID
AND    H1.CAT = 'H' AND H1.ENO = 1
AND    H2.CAT = 'H' AND H2.ENO = 2
AND    H1.POINTS = 10 AND H2.POINTS = 10
```

Self Joins (2)

- Find students who solved at least two exercises:

```
SELECT S.FIRST, S.LAST                    Wrong!
FROM   STUDENTS S, RESULTS E1, RESULTS E2
WHERE  S.SID = E1.SID AND S.SID = E2.SID
```

- The tuple variables `E1` and `E2` can point to the same input tuple.
- One must explicitly request that they are different:

```
WHERE S.SID = E1.SID AND S.SID = E2.SID
AND   (E1.CAT <> E2.CAT OR E1.ENO <> E2.ENO)
```

- This task can also be solved with aggregations.

Exercise

- Is there any problem with this query? The task is to list all students who solved an exercise about SQL and an exercise about relational algebra.

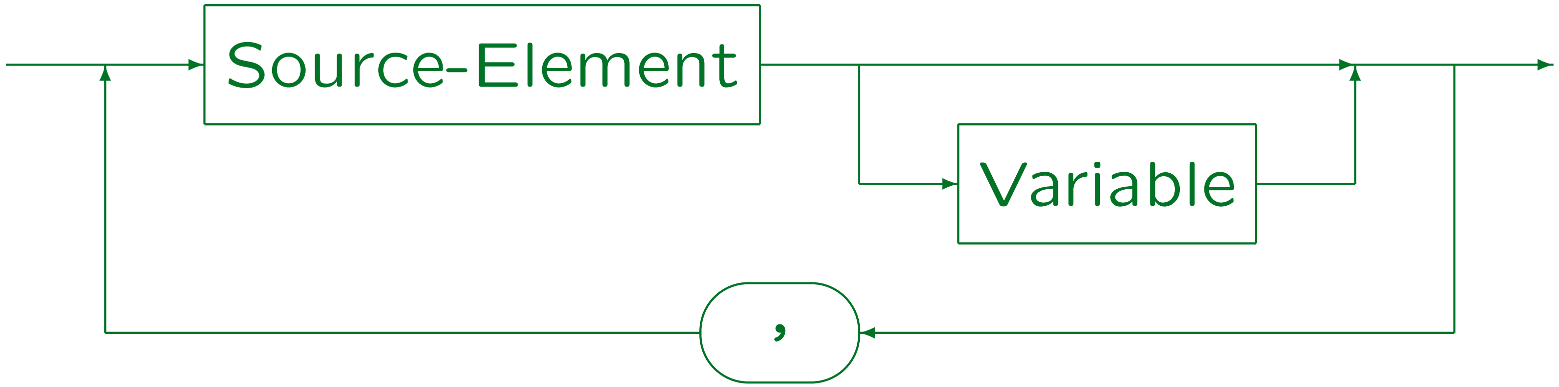
```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, RESULTS R,
       EXERCISES E1, EXERCISES E2
WHERE  S.SID = R.SID
AND    R.CAT = E1.CAT AND R.ENO = E1.ENO
AND    R.CAT = E2.CAT AND R.ENO = E2.ENO
AND    E1.TOPIC = 'SQL'
AND    E2.TOPIC = 'Rel. Alg.'
```

Summary: Join Errors

- Missing join conditions (very common)
- Unnecessary joins (make query slower)
- Problems when several tuple variables over the same relation are required: If these are "merged", one often gets an inconsistent condition.
- Duplicates are often an indication for errors: One should understand the source of the duplicates and not simply specify `DISTINCT` to avoid the problem.

FROM Syntax (1)

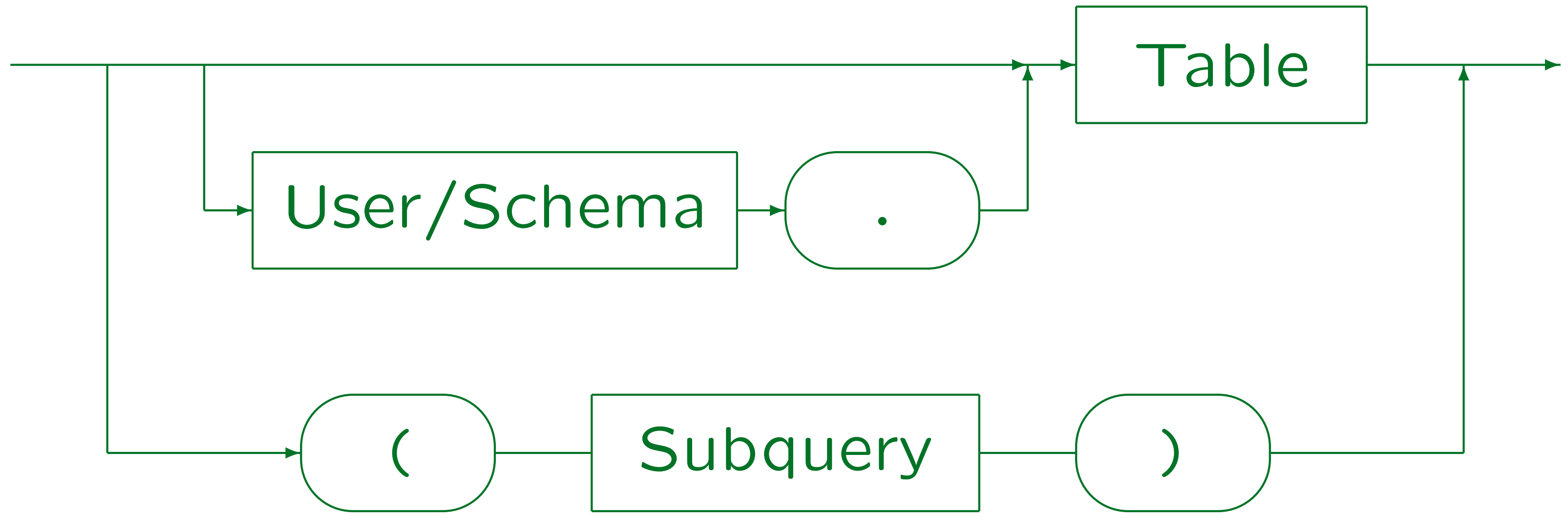
Source-List (after FROM):



- In SQL-92, SQL Server, Access, DB2, and MySQL (but not in Oracle 8i) one can write "`AS`" between Source-Element and Variable.
- In SQL-92 and DB2 (but not Oracle, SQL Server, Access, MySQL) new column names can be defined: "`STUDENTS AS S(NO,FNAME,LNAME, EMAIL)`".
- If the "Source-Element" is a subquery, the tuple variable is required in SQL-92, SQL Server, and DB2, but not in Oracle and Access. In this case the above column renaming syntax suddenly works in SQL Server.
- SQL-92, SQL Server, Access, DB2 support joins under `FROM` (see below).

FROM Syntax (2)

Source-Element:



- SQL-86 did not allow subqueries in the `FROM`-list.
- MySQL does not support subqueries at all.
- Basic (simplified) syntax of the `FROM`-clause:

```
FROM Table [Variable], ..., Table [Variable]
```

FROM Syntax (3)

Table Names:

- Tables of other users can be referenced in the FROM-list (if read permission was granted):

```
SELECT * FROM BRASS.EXERCISES
```

- The username is here really a name of a DB schema (one DBMS server can manage several schemas).

  In Oracle, schema and user are more or less the same: Every user has his/her own schema, every schema belongs to exactly one user. In DB2, there can be multiple schemas per user (one can write "schema.table" as in Oracle). In SQL Server, a fully qualified name has the form "server.database.owner.table", but there are various abbreviations including "owner.table" or simply "table". In MySQL, one can write "database.table".

Overview

Terms (1)

- A term denotes a data element.

  The word "term" is used in logic. In programming languages, one usually says "expression". The SQL standard uses "scalar expression", because it also has "table expressions".

- Terms are:
  - ◇ Attribute References, e.g. `STUDENT.SID`.
  - ◇ Constants ("literals"), e.g. `'Ann'`, `1`.
  - ◇ Composed Terms, using datatype operators like `+`, `-`, `*`, `/` (for numbers), `||` (string concatenation), and datatype functions, e.g. `0.9 * MAXPT`.
  - ◇ Aggregation terms, e.g. `MAX(POINTS)`: see Part 8.

Terms (2)

- Terms are used in conditions, e.g.

```
R.POINTS > E.MAXPT * 0.8
```

  contains the terms "`R.POINTS`" and "`E.MAXPT * 0.8`".

- Also the `SELECT`-list can contain arbitrary terms:

```
SELECT LAST || ', ' || FIRST
FROM   STUDENTS
```

...
Smith, Ann
Jones, Michael
Turner, Richard
Brown, Maria

Attribute References (1)

- Attributes can be accessed in the form

  `Variable.Attribute`

- If only one variable has this attribute, the variable name can be left out. E.g. this query is legal:

```
SELECT CAT, ENO, POINTS
FROM   STUDENTS S, RESULTS R
WHERE  S.SID = R.SID
AND    FIRST = 'Ann' AND LAST = 'Smith'
```

"`FIRST`" and "`LAST`" can only refer to "`S`". The attributes "`CAT`", "`ENO`", and "`POINTS`" can only refer to "`R`". However, "`SID`" alone would be ambiguous, since "`S`" and "`R`" both have an attribute with this name.

Attribute References (2)

- Consider this query:

```
SELECT ENO, SID, POINTS, MAXPT          Wrong!
FROM   RESULTS R, EXERCISES E
WHERE  R.ENO = E.ENO
AND    R.CAT = 'H' AND E.CAT = 'H'
```

- SQL requires that the user specifies whether he/she wants `R.ENO` or `E.ENO` in the `SELECT`-clause, although both are equal, so it actually does not matter.

  The rule is purely syntactic: If more than one tuple variable in the `FROM` clause has the attribute "`ENO`", the tuple variable cannot be left out, or the DBMS (e.g. Oracle) will print the error message "ORA-00918: column ambiguously defined". DB2, SQL Server, Access, MySQL are equally pedantic.

Composed Terms (1)

- The SQL-86 standard contained only `+`, `-`, `*`, `/`.
- Current database management systems still differ in other data type operations.

  But they typically have a large collection of data type operations, e.g. `sin`, `cos`, `substr`. In Chapter 9, lists of data type operations in serveral systems are given.

- E.g. the operator `||` is contained in the SQL-92 standard, but does not work e.g. in SQL Server.

  String concatenation is written "`+`" in SQL Server and Access.
  In MySQL, one must write "`concat(`s_1`,` s_2`)`" (but there is "`--ansi`").
  Other datatype functions (e.g. `SUBSTR`) are even less standardized.

Composed Terms (2)

- SQL knows the standard precedence rules, e.g. that `A+B*C` means

  `A+(B*C)`,

  and not

  `(A+B)*C`.

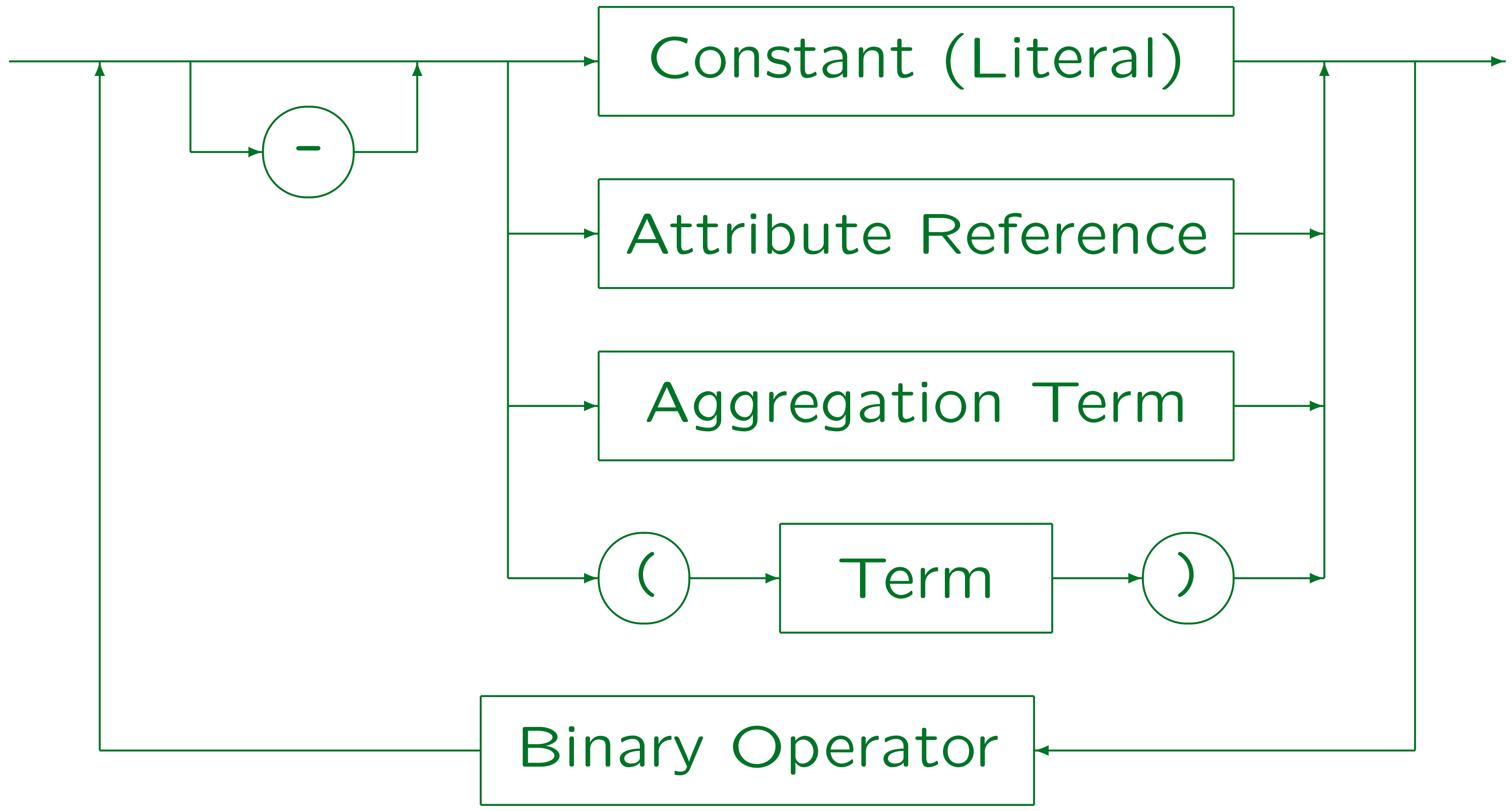
- Parentheses `(...)` may be used to enforce a particular structure.

- Exercise: What is the result of `7+3*2-4-1`?

  It might be useful to draw an operator tree. "-" is left associative (evaluated from the left).

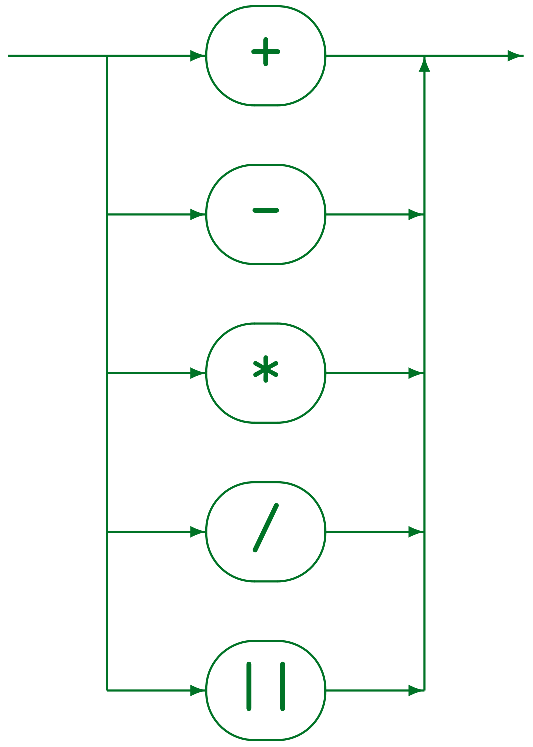
Terms: Syntax (1)

Term (Scalar Expression, Value Expression):



Terms: Syntax (2)

Binary Operator:



- SQL Server, Access, and MySQL do not use "||" for string concatenation.

Terms with Null Values (1)

- As explained in Chapter 4, an attribute value can be “null” (unless excluded with `NOT NULL` in the table declaration).
- The null value is different from all normal values of the data type, especially it is different from the number 0 and the empty string.

  In Oracle, still in version 9i, the null value and the empty string are identified. This is a severe violation of the SQL standard (Oracle lists this as “non-conforming” in an appendix to its SQL reference manual). However, since it might be used in application programs, and since existing database files do not differ between the two, it is difficult to change.

Terms with Null Values (2)

- Data type functions will normally return null if one of their arguments is null. E.g. if `A` is null, `A+B` will be null.
- The keyword `NULL` by itself is not a term (expression), although it can be used in many contexts that otherwise require a term.

Terms with Null Values (3)

- `NULL` has no type, so at least we need a context in which the type is clear:
  - ◇ In SQL-92 and DB2, `CAST(NULL AS type)` gives a null value of the specified type.
  - ◇ In Oracle, `NULL` often can be used as a term, but e.g. this gives an error:

    ```
    select 1 from dual union select null from dual
    ```

    One must write `TO_NUMBER(null)`.
  - ◇ In SQL Server, Access, and MySQL "`NULL`" is handled like a normal term (with arbitrary type).

Overview

Example Database (again)

STUDENTS			
SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES			
CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Conditions (1)

- Conditions consist of atomic formulas, e.g.

  `POINTS >= 8,`

  connected by "`AND`", "`OR`", "`NOT`".

- `AND` binds more strongly than `OR`, thus

  ```
  CAT = 'H' AND ENO = 1 OR ENO = 2
  ```

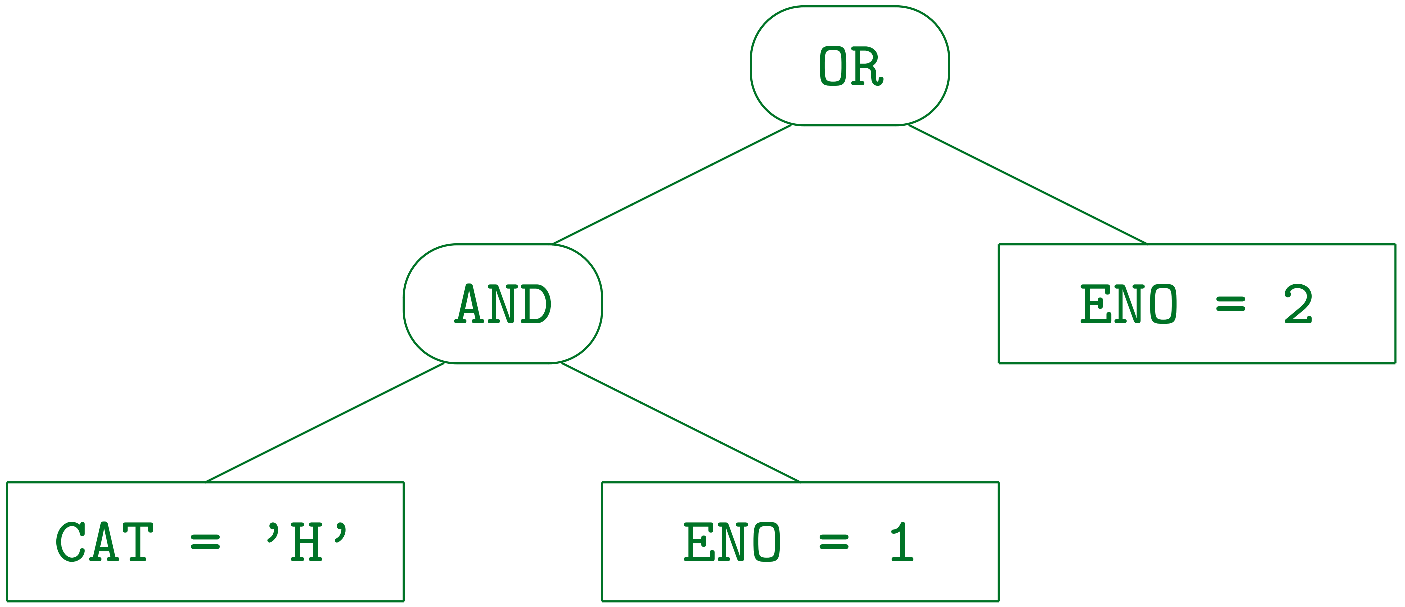
  is implicitly parenthesized as

  ```
  (CAT = 'H' AND ENO = 1) OR ENO = 2
  ```

- In this example, this is probably not intended.

Conditions (2)

- It might help to draw a complex condition (or complex term) as an "operator tree":



Conditions (3)

- `NOT` binds most strongly, i.e. it is applied only to the immediately following condition (atomic formula).
- Parentheses `( ... )` can be used to override the operator priorities (precedences, binding strengths).
- Sometimes, it might be clearer to use parentheses even if they are not necessary to enforce the right structure of the formula.

  However, beginners tend to use a lot of parentheses (probably because they are unsure about the operator priorities). This does not make the formula easier to understand.

Conditions (4)

- The WHERE-condition is conceptionally evaluated for every combination of rows of the tables listed under FROM. If it is true, the SELECT-list is printed.
- An AND-condition is true if both parts are true, an OR-condition is already true if one part is true:

C1	C2	C1 AND C2	C1 OR C2	NOT C1
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Conditions (5)

- E.g., list the SIDs of Ann and Maria:

```
SELECT SID
FROM   STUDENTS          wrong!
WHERE  FIRST = 'Ann' AND FIRST = 'Maria'
```

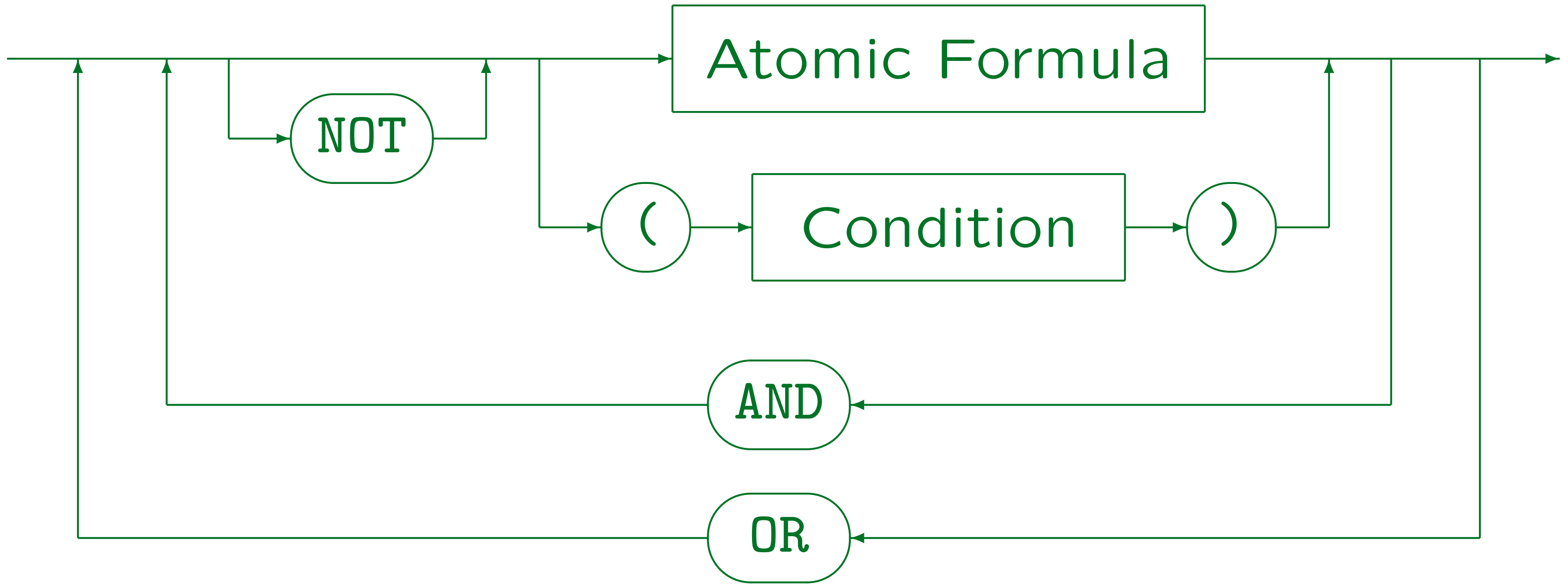
STUDENTS		
SID	FIRST	LAST
101	Ann	Smith
102	Michael	Jones
103	Richard	Turner
104	Maria	Brown

FIRST='Ann'	FIRST='Maria'	WHERE
True	False	False
False	False	False
False	False	False
False	True	False

- The above condition is inconsistent. OR must be used in this query.

Conditions (6)

Condition:



- SQL-92 allows "`IS NOT TRUE`", "`IS FALSE`" etc. after formulas (not supported in Oracle 8.0, SQL Server, DB2, MySQL, Access).

Conditions (7)

- AND and OR must take complete logical conditions (something that is true or false) on both sides.
- So the following is a syntax error although it is similar to natural language:

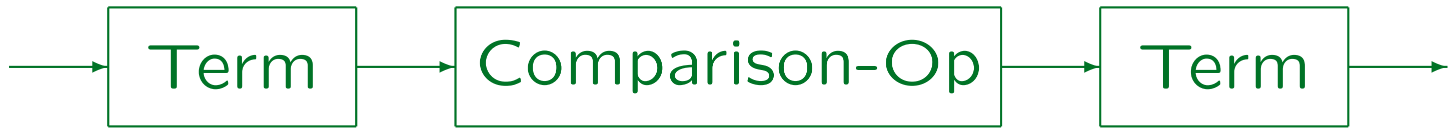
```
SELECT DISTINCT SID          Wrong!
FROM   RESULTS
WHERE  CAT = 'H' AND POINTS >= 9
AND    ENO = 1 OR 2
```

- Exception: ... BETWEEN ... AND ...

  Here the word AND does not denote the logical connective.

Comparisons (1)

Atomic Formula (Form 1):



- Comparison operators: `=`, `<>`, `<`, `>`, `<=`, `>=`.
- Comparison operators can be used for numbers as well as for strings, e.g.: `POINTS >= 8`, `LAST < 'M'`.
- "Not equals" is written in standard SQL as "`<>`".

  Oracle, SQL Server, DB2, and MySQL understand also "`!=`" (Access does not accept this notation). "`^=`" works in Oracle and DB2, but not in SQL Server, Access, or MySQL.

Comparisons (2)

- Numbers are compared differently than strings, e.g. `3 < 20`, but `'3' > '20'`.

  String comparison is done character by character until the outcome is clear. In this case, "3" comes alphabetically after "2", therefore the rest of the string is not important.

- According to the SQL-92 standard, it is an error to compare strings with numbers, e.g. `3 > '20'`.

  The two compared values must be of compatible types: All numeric types are compatible, and all string types are compatible, but numeric types are not compatible with string types.

Comparisons (3)

- Comparing a string with a number should be avoided, since the outcome is very system dependent:
  - ◇ SQL-92, DB2, and Access produce a type error.
  - ◇ Oracle, MySQL, and SQL Server convert the string to a number and do a numeric comparison.

    If the string is not of numeric format, MySQL simply converts it to 0. E.g. `0 = 'abc'` is true in MySQL. In Oracle and SQL Server, one gets an error if the string is not of numeric format. This might be a runtime error if the string is a column value.
  - ◇ However, if a column is compared with a constant, SQL server uses the column type.

    Aggregate functions have still higher priority than columns.

String Comparisons (1)

- The outcome of comparing (`=`, `<>`, `<`, `<=`, `>`, `>=`) two character strings may depend on the DBMS.

  Or settings within the DBMS.

- The SQL-92 standard defines the notion of "collation sequences" (or "collations") which determine
  - ◇ for any pair X and Y of characters, whether $X < Y$, $X = Y$, or $X > Y$, and
  - ◇ whether the blank-padded semantics (`PAD SPACE`) or the non-padded semantics (`NO PAD`) is used.

String Comparisons (2)

- `'a' < 'b'` etc., and `'A' < 'B'` etc. can be expected.
- But the systems differ in the comparison of uppercase and lowercase characters. The defaults are:
  - ◇ In Oracle all uppercase characters come before all lowercase characters (ASCII), e.g. `'Z' < 'a'`.
  - ◇ In DB2, uppercase and lowercase characters are interleaved, e.g.: `'a' < 'A'`, `'A' < 'b'`.
  - ◇ SQL Server, MS Access, and MySQL are case-insensitive, e.g.: `'a' = 'A'`.

String Comparisons (3)

- It might be possible to change this, but e.g. only during installation (SQL Server), or during database creation (Oracle, DB2).
- When the order (<, =, >) of every two characters is known, the comparison of strings of the same length is clear:
  - ◇ The system compares character by character, the first comparison which does not give "=" determines the result.

    Actually, DB2 makes two passes: It first compares the character "weights", and if there is no difference, also the character codes.

String Comparisons (4)

- For strings of different lengths, there are
  - ◇ Non-Padded Comparison Semantics:

    E.g. `'a' < 'a '`.

    Strings are compared character by character. When one string ends and no difference was found, the shorter string is considered less than the longer one.
  - ◇ Blank-Padded Comparison Semantics:

    E.g. `'a' = 'a '`.

    The shorter string is filled with `' '` before the comparison.

String Comparisons (5)

- DB2, SQL Server, Access, and MySQL use the blank-padded semantics (at least by default).
- Oracle uses the nonpadded semantics if at least one operand of a comparison has type `VARCHAR2`.

  Oracle has introduced a type `VARCHAR2(`n`)`. It is currently synonymous to `VARCHAR(`n`)`, but Oracle intends to change the comparison semantics for `VARCHAR`, while the semantics for `VARCHAR2` will remain stable. String literals (constants) in the query have type `CHAR(`n`)`. E.g. a comparison of `CHAR(10)` and `CHAR(20)` columns can possibly yield "true" as can a comparison of these columns with, e.g., `'abc'`. But `CHAR(10)` and `VARCHAR(20)` can only be equal if the `VARCHAR` happens to be of length 10. Trailing spaces in `VARCHAR2`-columns can be quite annoying: They are not visible in the output, but the comparison does not work.

String Comparisons (6)

- If the system uses a case-sensitive semantics, one can get a case-insentive comparison by converting both sides e.g. to uppercase:

```
SELECT FIRST, LAST
FROM   STUDENTS
WHERE  UPPER(EMAIL) = UPPER('xyz@hotmail.com')
```

- UPPER works in SQL-92, Oracle, SQL Server, DB2, MySQL. In Access, use UCASE.

  UCASE works also in DB2 and MySQL. The book by Chamberlin about DB2 lists only UCASE.

String Comparisons (7)

- The opposite case (case-sensitive comparison with a case-insensitive system) is more difficult.

  But also much more seldom required.

- E.g. in MySQL, one can convert a string to a binary string in order to get case-sensitive comparison:

```
BINARY EMAIL = 'xyz@hotmail.com'
```

- The same trick works in SQL Server:

```
  CAST(EMAIL AS VARBINARY(255))
= CAST('...' AS VARBINARY(255))
```

String Comparisons (8)

- If one suspects that trailing spaces make a comparison fail, one can make them visible in this way:

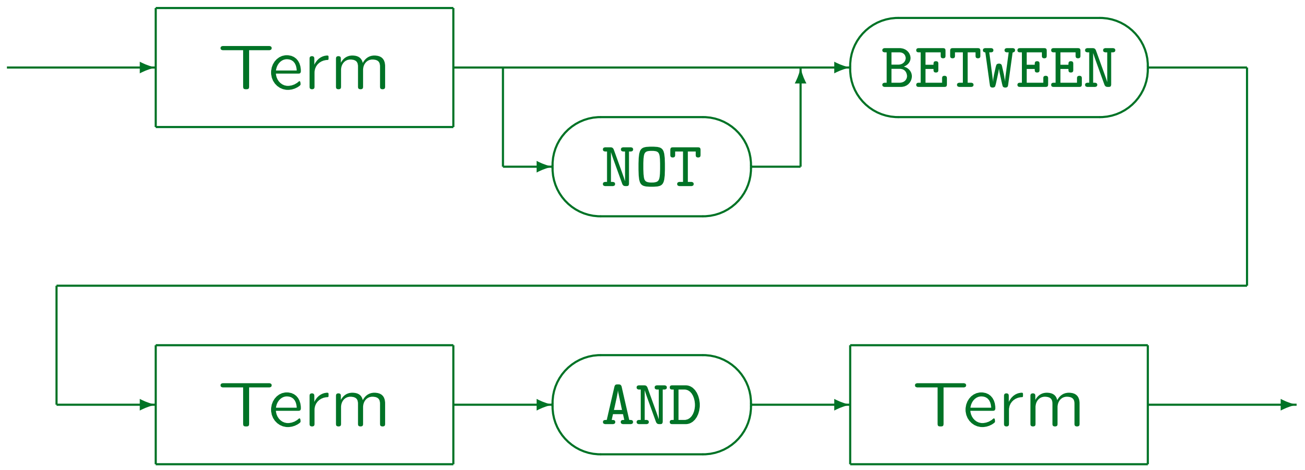
```
SELECT '"' || LAST || '"' AS LAST_NAME
FROM   STUDENTS
```

- One can also remove trailing spaces:
  - ◇ `TRIM(TRAILING ' ' FROM LAST)`
    in SQL-92 (works in MySQL)

    This syntax is not supported in Oracle, DB2, SQL Server, Access.
  - ◇ `RTRIM(LAST)`
    in Oracle, DB2, SQL Server, MySQL, Access.

BETWEEN Conditions

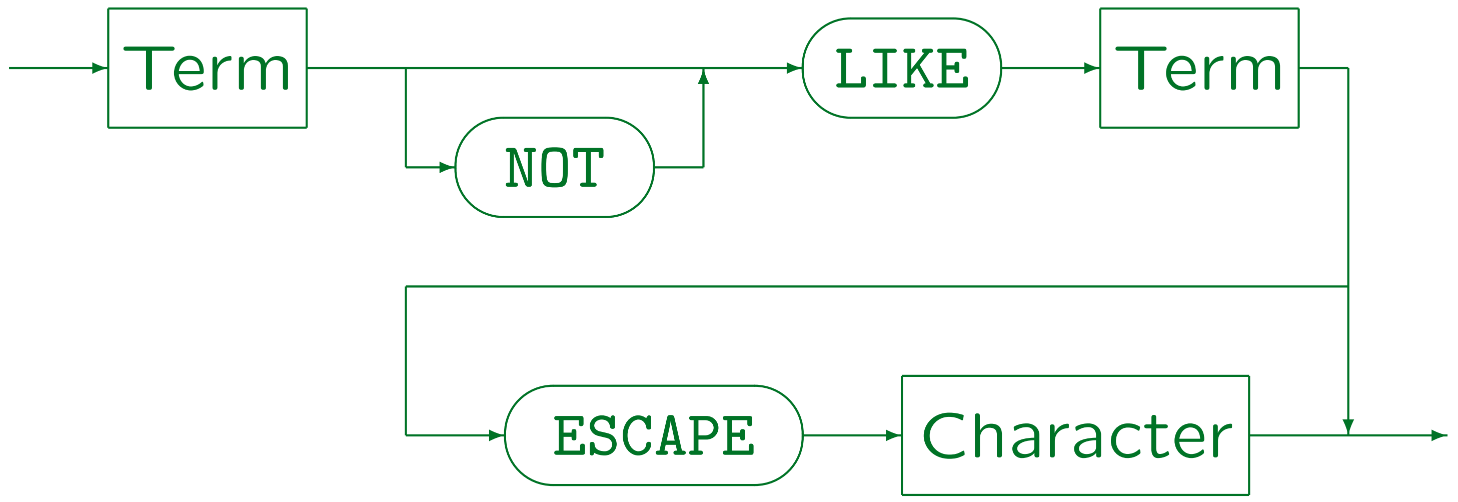
Atomic Formula (Form 2):



- `x BETWEEN y AND z` is equivalent to `x >= y AND x <= z`.
- E.g.: `POINTS BETWEEN 5 AND 8`

LIKE Conditions (1)

Atomic Formula (Form 3):



- E.g.: `EMAIL LIKE '%.pitt.edu'`

  This is true for all email addresses that end in "`.pitt.edu`".

LIKE Conditions (2)

- The right argument is interpreted as pattern.

  In SQL-86 and DB2, it must be a string constant.

  In Oracle, SQL Server, Access, and MySQL, one can use any string valued term as pattern (especially also another column).

- "%" in the pattern matches any sequence of arbitrary characters (including the empty string).

  In the UNIX shell (command interpreter), "*" is used instead of "%".

- "_" matches any single character.

  This corresponds to "?" in the shell.

LIKE Conditions (3)

- LIKE must be used for pattern matching.

  The equals sign only tests for literal equality.

  Even if the comparison string contains "%" or "_".

- E.g. the following is legal SQL, but will return the empty result (not Ann Smith):

```
SELECT FIRST, LAST
FROM   STUDENTS
WHERE  LAST = 'S%'      wrong!
```

LIKE Conditions (4)

- To use the characters "%" and "_" without their special meaning in the pattern, an "escape" character is used.

  The escape character removes the special meaning of the following character. E.g. if "\" is the escape character, then "\%" matches only a percent sign, not an arbitrary string.

- The escape character must be declared, e.g.:

  ```
  PROCNAME LIKE '\_%' ESCAPE '\'
  ```

  This gives all procedure names starting with an "_".

  In MySQL, if no escape character is explicitly declared, "\" is the default escape character. However, this violates the SQL-92 standard.

LIKE Conditions (5)

- `LIKE` uses the non-padded semantics.

  Oracle, DB2, MySQL, and Access use the non-padded semantics as required by the SQL-92 standard. Note that MySQL removes trailing spaces when strings are stored. All systems fill values with blanks if the column is declared as fixed-length character string.
  In SQL Server, if the stored string contains more spaces at the end than the pattern, it might still match. If the pattern contains more spaces, the match fails. With the Unicode national character set types, the strict non-padded semantics is used.

- E.g. `'a' = 'a '` might be true (in some DBMS), but `'a' LIKE 'a '` is surely false.
- The case sensitivity is the same as for ordinary comparisons.

Regular Expressions

- SQL Server and Access support also character ranges, e.g. `[a-zA-Z]` in `LIKE`-conditions.

  This violates the standard.

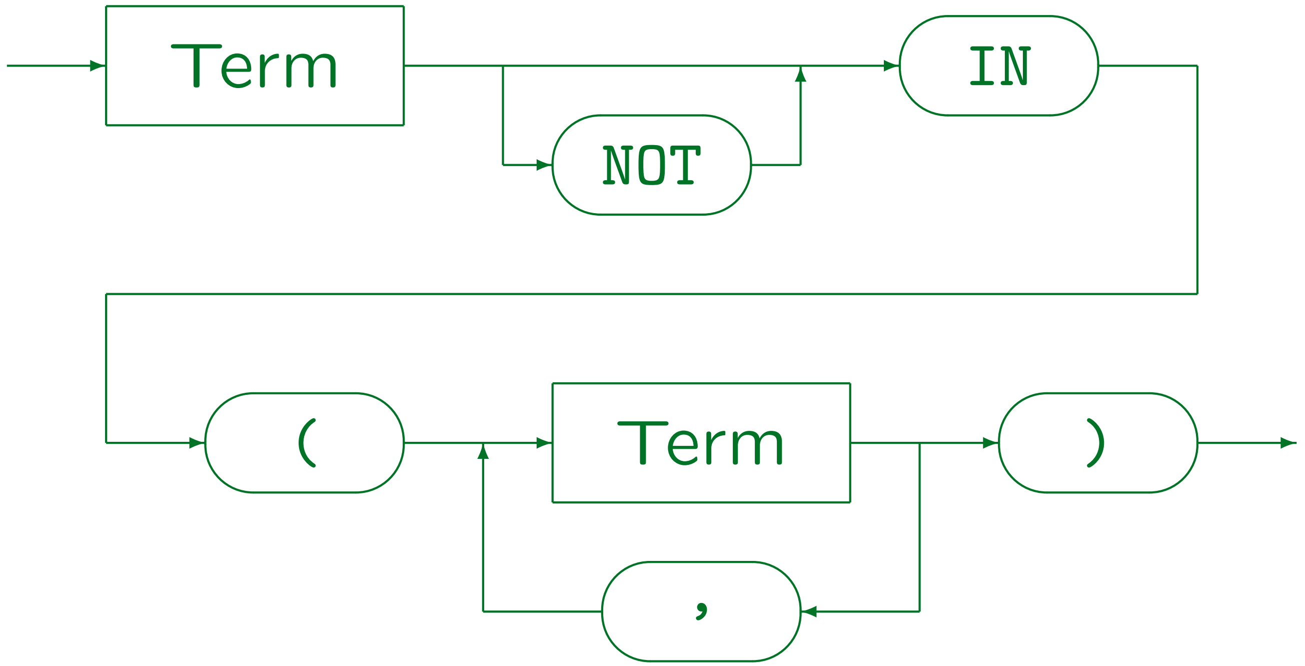
- MySQL has an additional operator "`RLIKE/REGEXP`" that accepts arbitrary regular expressions as patterns.

- The SQL:1999 standard has introduced a predicate "`SIMILAR TO`" that does comparisons with regular expressions.

  In most current systems, e.g. Oracle 9i, it is not yet implemented.

IN Conditions (1)

Atomic Formula (Form 4):



IN Conditions (2)

- E.g. `CAT IN ('M', 'F')`
- This is equivalent to

```
CAT = 'M' OR CAT = 'F'
```

- The SQL-86 standard allowed only constants in the enumeration of values.

  SQL-92, Oracle, SQL Server, and DB2 allow arbitrary terms, but it is normally better style to use `OR` if the set is not an enumeration of constants.

- Note that although in mathematics, "`(...)`" are used for intervals, here they mean "set".

Three-Valued Logic (1)

- Consider the following query:

```
SELECT FIRST, LAST
FROM   STUDENTS
WHERE  EMAIL = 'xyz@acm.org'
```

- What happens if a student has a null value in the column `EMAIL`? It is not printed.
- But it also does not appear in the result of this query (because the value is unknown):

```
SELECT FIRST, LAST
FROM   STUDENTS
WHERE  NOT (EMAIL = 'xyz@acm.org')
```

Three-Valued Logic (2)

- The condition

  ```
  EMAIL = 'xyz@acm.org'
  ```

  does not evaluate to false if `EMAIL` is null, since then the row would appear in the negated query.

  Of course, it also does not yield true.

- SQL uses a three-valued logic for treating null values. The three truth values are **`true`**, **`false`**, and **`unknown`**.

  Instead of "`unknown`", one also often reads "`null`".

Three-Valued Logic (3)

- The idea is that tuples which have a null value in an attribute which is important for the query should be "filtered out" — they should not influence the query result.
- The real attribute value is unknown or does not exist, so saying that the result of a comparison with a null value is true or false is equally wrong.
- In SQL, a comparison with a null value always yields the third truth value "`unknown`".

Three-Valued Logic (4)

- A result row is printed only if the WHERE-condition evaluates to "true".
- Thus, the following query gives the empty result:

```
SELECT FIRST, LAST
FROM   STUDENTS
WHERE  EMAIL = null
```

  Actually, the query is illegal in SQL-92, and DB2 refuses it. Oracle, SQL Server, Access, and MySQL accept it and print the empty result.

- "AND" / "OR" forward the truth value "unknown", unless the result is clear:

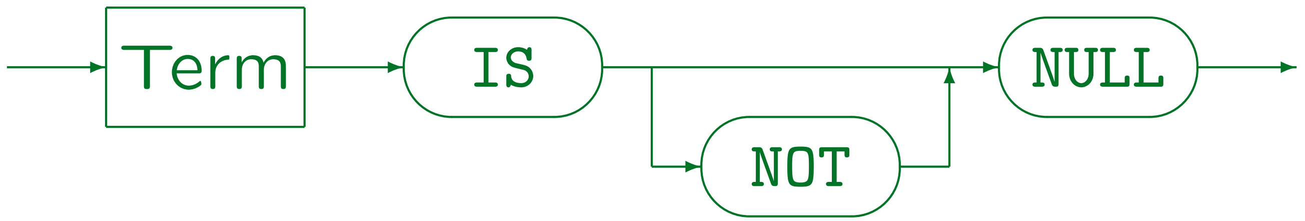
  E.g. "true OR unknown = true".

Three-Valued Logic (5)

P	Q	NOT P	P AND Q	P OR Q
false	false	true	false	false
false	unknown	true	false	unknown
false	true	true	false	true
unknown	false	unknown	false	unknown
unknown	unknown	unknown	unknown	unknown
unknown	true	unknown	unknown	true
true	false	false	false	true
true	unknown	false	unknown	true
true	true	false	true	true

Test for Null (1)

Atomic Formula (Form 5):



- Example: `EMAIL IS NULL`
- Note that `EMAIL = NULL` does not work.

  In Oracle and SQL Server, it evaluates always to "unknown" (not "true" or "false"), and in SQL-92 and DB2, it is a syntax error. In SQL Server 7, "`EMAIL = NULL`" works after the command "`SET ANSI_NULLS OFF`" (then a two-valued logic is used).

- `EMAIL NOT NULL` is a syntax error ("`IS`" is missing).

Test for Null (2)

- Exercise: The following query prints all students with an email address in the domain "`.pitt.edu`":

```
SELECT FIRST, LAST
FROM   STUDENTS
WHERE  EMAIL IS NOT NULL
AND    EMAIL LIKE '%.pitt.edu'
```

  Is the test for null necessary?

- `CHECK`-integrity constraints are satisfied if the condition evaluates to the third truth value "unknown".

  They are only violated if the condition evaluates to false.

Problems of Null Values (1)

- For those accustomed to working with a two-valued logic (all of us), null values can sometimes lead to surprises: Some standard logical equivalences do not hold in SQL.
- E.g. if students with an email address in the domain "`.pitt.edu`" are counted, and students with an outside email address, one would normally assume to get all students.
- But this is not true in SQL — those with a null value in the `EMAIL` column are not counted.

Problems of Null Values (2)

- E.g. `X = X` evaluates to “unknown”, not to “true” if `X` is null.
- Since the null value is used with different meanings, there can be no satisfying semantics for a query language.

  E.g. the meaning “value exists, but unknown” ($\exists X: \ldots$) would allow to use standard logical equivalences.

Overview

SELECT Clause, *

- The SELECT-clause lists the terms to be printed if the WHERE-condition is true (output columns).
- The abbreviation SELECT * can be used to print all columns of the table(s) under FROM, e.g.

```
SELECT *
FROM   STUDENTS
```

  is equivalent to

```
SELECT SID, FIRST, LAST, EMAIL
FROM   STUDENTS
```

- In programs, it might be clever to avoid *, since sometimes columns are later added to a table.

Duplicate Elimination (1)

- One difference of SQL to relational algebra is that duplicates have to be explicitly eliminated in SQL.
- E.g. which exercises have already been solved by at least one student?

```
SELECT CAT, ENO
FROM   RESULTS
```

CAT	ENO
H	1
H	2
M	1
H	1
H	2
M	1
H	1
M	1

Duplicate Elimination (2)

- The duplicates occur because the query is executed by a loop over the rows in RESULTS.
- If the query might contain duplicates, and there is no specific reason why they should be shown, use "SELECT DISTINCT":

```
SELECT DISTINCT CAT, ENO
FROM   RESULTS
```

CAT	ENO
H	1
H	2
M	1

Duplicate Elimination (3)

- To emphasize that there are duplicates and that they are really wanted, one can write "`SELECT ALL`".

  However, "`ALL`" is the default.

- Note that `DISTINCT` always applies to entire rows, not to single columns.

  Else NF^2 tables were needed. With classical relations, one cannot get e.g. one row for every student with all of his/her results. But output formatting can produce something similar: One can tell SQL*Plus to print a column value only if it changed from the preceding row.

- For instance, the following is a syntax error:

```
SELECT CAT, ENO, DISTINCT TOPIC     Wrong!
FROM   EXERCISES
```

IS DISTINCT Needed? (1)

Sufficient condition for unnecessary `DISTINCT`:

- Let $\mathcal{K}$ be the set of attributes that appear as output columns under `SELECT`.

  The elements of $\mathcal{K}$ are of the form "Tuplevariable.Attribute". $\mathcal{K}$ is the set of attributes that have a unique value for a given output row.

- Add to $\mathcal{K}$ attributes A such that $A = c$ with a constant c appears in the `WHERE`-condition.

  This test assumes that the condition is a conjunction. Of course, a condition $c = A$ is treated in the same way. Conditions in subqueries do not count (subqueries are simply removed before the test is done).

Is DISTINCT Needed? (2)

Test for unnecessary `DISTINCT`, continued:

- As long as something changes, do the following:
  - ◇ Add to $\mathcal{K}$ attributes A such that $A = B$ appears in the `WHERE`-condition and $B \in \mathcal{K}$.
  - ◇ If $\mathcal{K}$ contains a key of a tuple variable, add all other attributes of this tuple variable.
- If $\mathcal{K}$ contains a key of every tuple variable listed under `FROM`, `DISTINCT` is superfluous.

  If the query contains `GROUP BY`, one checks instead whether all `GROUP BY` columns are contained in $\mathcal{K}$.

Is DISTINCT Needed? (3)

Example:

- Consider the following query:

```
SELECT DISTINCT S.FIRST, S.LAST, R.ENO, R.POINTS
FROM   STUDENTS S, RESULTS R
WHERE  R.CAT = 'H' AND R.SID = S.SID
```

- Let us assume that FIRST, LAST is declared as an alternative key for STUDENTS.
- $\mathcal{K}$ is initialized with S.FIRST, S.LAST, R.ENO, R.POINTS.
- R.CAT is added because of the condition R.CAT = 'H'.

Is DISTINCT Needed? (4)

Example, continued:

- `S.SID` and `S.EMAIL` are added, because $\mathcal{K}$ contains a key of `STUDENTS S` (`S.FIRST` and `S.LAST`).
- `R.SID` is added because of `R.SID = S.SID`.
- Now $\mathcal{K}$ contains also a key of `RESULTS R` (`R.SID`, `R.CAT`, `R.ENO`), thus `DISTINCT` is superfluous.
- If `FIRST`, `LAST` were not a key of `STUDENTS`, this test would not succeed.

  However, this case it might be useful to print duplicates since in the real world, students are identified by name ("soft key").

DISTINCT vs. GROUP BY

- Duplicates should be eliminated with `DISTINCT`, although it works also with `GROUP BY`:

```
SELECT   CAT, ENO      Bad Style!
FROM     RESULTS
GROUP BY CAT, ENO
```

  This splits the table into groups of tuples: each group contains tuples that agree in the values for the grouping attributes `CAT`, `ENO`. For each group, only one output tuple is produced. Normally this is used to compute aggregation functions (`SUM`, `COUNT`) for each group.

- I would consider this as an abuse of `GROUP BY`.

  However, `GROUP BY` is more flexibe than `DISTINCT` if one wants to eliminate only some duplicates. Also old versions of MySQL did not support `DISTINCT`. Then one had to use `GROUP BY`.

Renaming of Output Colums

- To rename the output columns:

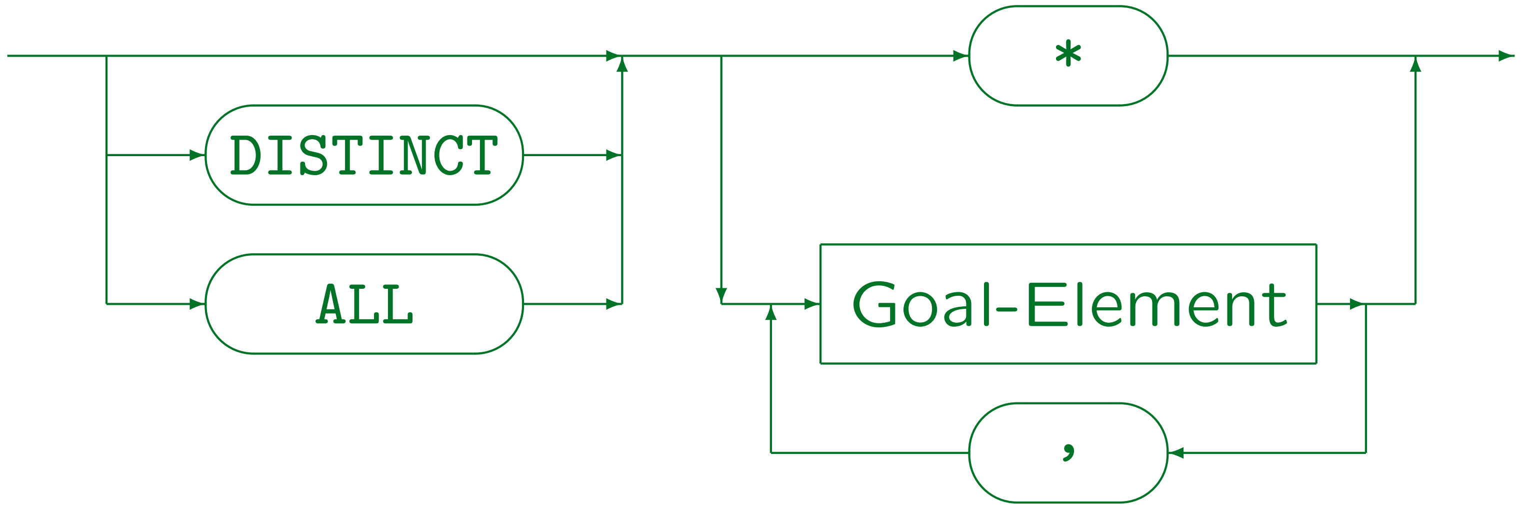
```
SELECT FIRST AS First_Name, LAST AS "Last Name"
FROM   STUDENTS
```

FIRST_NAME	Last Name
Ann	Smith
Michael	Jones
Richard	Turner
Maria	Brown

- This works in SQL-92, Oracle, SQL Server, DB2, MySQL, Access, but not in SQL-86.
- "`AS`" can be left out in SQL-92 and all of the above systems except Access.

SELECT Syntax (1)

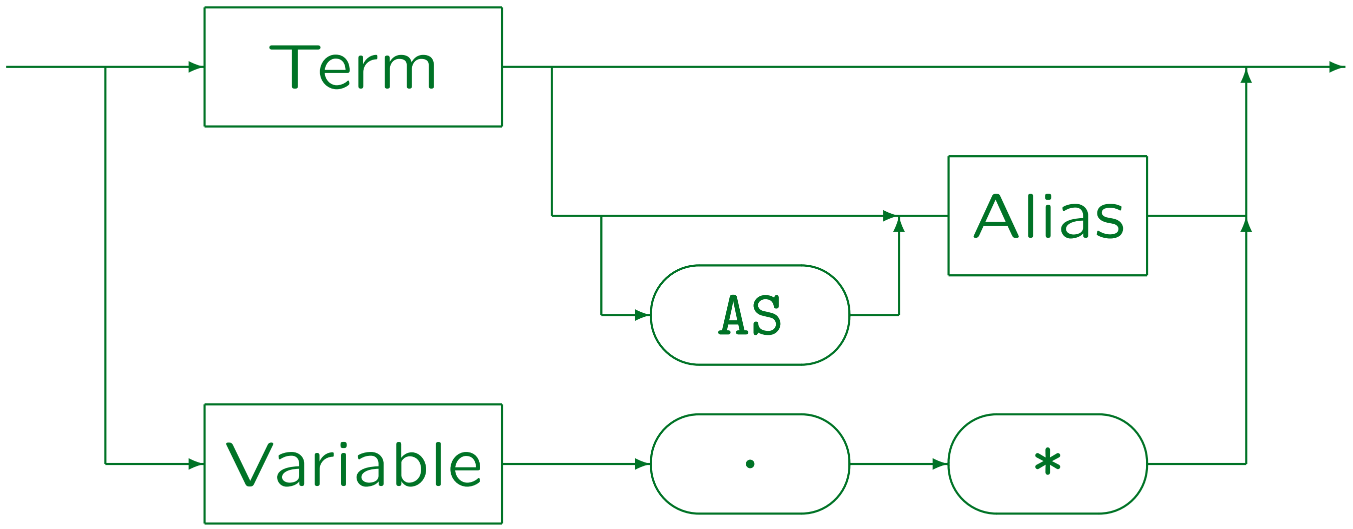
Goal-List (after `SELECT`):



- `ALL` (no duplicate elimination) is the default.

SELECT Syntax (2)

Goal-Element:



- "Variable.*" and "[AS] Alias" work in SQL-92, Oracle, SQL Server, and DB2, MySQL and Access (in Access "AS" is required). These constructs are not contained in the old SQL-86 standard.