

Part 3: SQL I

References:

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition, 1999. Chap. 8, "SQL — The Relational Database Standard" (only Section 8.2, 8.3.3, and part of 8.3.4.)
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Edition, McGraw-Hill, 1999: Chapter 4: "SQL".
- 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.
- van der Lans: SQL, Der ISO-Standard (in German). Hanser, 1990.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.

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.
- enumerate the main constructs of SQL, at least of the SQL-86 standard (e.g. all kinds of conditions).

All constructs from the SQL-86 standard are treated in the three chapters about SQL, plus some extensions from SQL-92 and the three DBMS (Oracle, DB2, SQL Server).

- check a given query for syntactical correctness.
- evaluate the portability of certain constructs.

Overview

1. Lexical Syntax

2. SELECT-FROM-WHERE, Tuple Variables

3. Terms and Conditions

4. A bit of Logic

5. Null Values

White Space and Comments

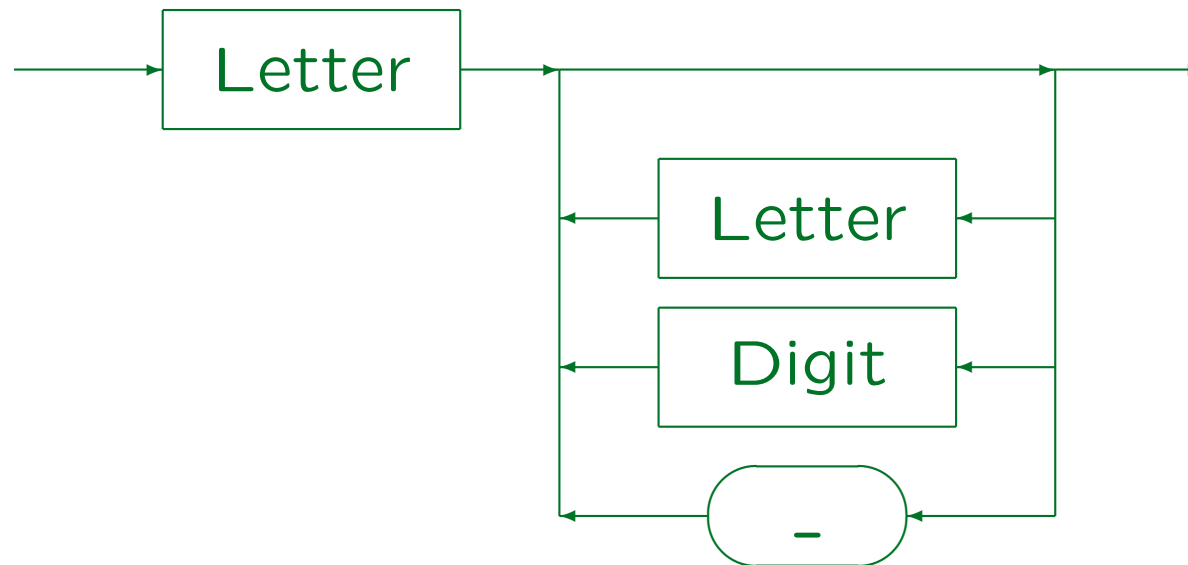
White space is allowed between words (tokens):

- Spaces, Tabulator characters
- Line breaks
- Comments:
 - ◇ From “--” to ⟨Line End⟩
(SQL-92 Standard, Oracle, SQL Server, IBM DB2)
 - ◇ From “/*” to “*/”
(Only Oracle and SQL Server, i.e. less portable)

SQL is a free-format language like Pascal, C, Java.

Identifiers (1)

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



- Letter (Letter|Digit|_)*.
- 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	≤ 18	A-Z	A-Z,0-9
SQL-92	≤ 128	A-Z,a-z	A-Z,a-z,0-9,_
Oracle	≤ 30	A-Z,a-z	A-Z,a-z,0-9,_,#,\$
SQL Server	≤ 128	A-Z,a-z,_,(@,#)	A-Z,a-z,0-9,_,@,#,\$
IBM DB2	≤ 18 (8)	A-Z,a-z	A-Z,a-z,0-9,_

- Identifiers (and keywords) are not case sensitive.

SQL*Plus converts all letters outside quotes to uppercase.

- Names must be different from all reserved words.

There are a lot of reserved words, see below.

- See also delimited identifiers below.

SQL Reserved Words (1)

1 = Oracle 8.0

2 = SQL-92

3 = SQL Server 7

— A —

ABSOLUTE²

ACCESS¹

ACTION²

ADD^{1,2,3}

ALL^{1,2,3}

ALLOCATE²

ALTER^{1,2,3}

AND^{1,2,3}

ANY^{1,2,3}

ARE²

AS^{1,2,3}

ASC^{1,2,3}

ASSERTION²

AT²

AUTHORIZATION^{2,3}

AUDIT¹

AVG^{2,3}

— B —

BACKUP³

BEGIN^{2,3}

BETWEEN^{1,2,3}

BIT²

BIT_LENGTH²

BOTH²

BREAK³

BROWSE³

BULK³

BY^{1,2,3}

— C —

CASCADE^{2,3}

CASCADE²

CASE^{2,3}

CATALOG²

CHAR^{1,2}

CHARACTER²

CHAR_LENGTH²

CHARACTER_LENGTH²

CHECK^{1,2,3}

CHECKPOINT³

CLOSE^{2,3}

CLUSTER¹

CLUSTERED³

COALESCE^{2,3}

COLLATE²

COLLATION²

COLUMN^{1,3}

COMMENT¹

COMMIT^{2,3}

COMMITTED³

COMPRESS¹

COMPUTE³

SQL Reserved Words (2)

CONFIRM ³	CURRENT ^{1,2,3}	DECLARE ^{2,3}	DOMAIN ²
CONNECT ^{1,2}	CURRENT_DATE ^{2,3}	DEFAULT ^{1,2,3}	DOUBLE ^{2,3}
CONNECTION ²	CURRENT_TIME ^{2,3}	DEFERRABLE ²	DROP ^{1,2,3}
CONSTRAINT ^{2,3}	CURRENT_TIMESTAMP ^{2,3}	DEFERRED ²	DUMMY ³
CONSTRAINTS ²	CURRENT_USER ^{2,3}	DELETE ^{1,2,3}	DUMP ³
CONTAINS ³	CURSOR ^{2,3}	DENY ³	— E —
CONTAINSTABLE ³	— D —	DESC ^{1,2}	ELSE ^{1,2,3}
CONTINUE ^{2,3}	DATABASE ³	DESCRIBE ²	END ^{2,3}
CONTROLROW ³	DATE ^{1,2}	DESCRIPTOR ²	END-EXEC ²
CONVERT ^{2,3}	DAY ²	DIAGNOSTICS ²	ERRLVL ³
CORRESPONDING ²	DBCC ³	DISCONNECT ²	ERROREXIT ³
COUNT ^{2,3}	DEALLOCATE ^{2,3}	DISK ³	ESCAPE ^{2,3}
CREATE ^{1,2,3}	DEC ²	DISTINCT ^{1,2,3}	EXCEPT ^{2,3}
CROSS ^{2,3}	DECIMAL ^{1,2}	DISTRIBUTED ³	EXCEPTION ²

SQL Reserved Words (3)

EXCLUSIVE ¹	FLOPPY ³	GROUP ^{1,2,3}	INDEX ^{1,3}
EXEC ^{2,3}	FOR ^{1,2,3}	— H —	INDICATOR ²
EXECUTE ^{2,3}	FOREIGN ^{2,3}	HAVING ^{1,2,3}	INITIAL ¹
EXISTS ^{1,2,3}	FOUND ²	HOLDLOCK ³	INITIALLY ²
EXIT ³	FREETEXT ³	HOURL ²	INNER ^{2,3}
EXTERNAL ²	FREETEXTTABLE ³	— I —	INPUT ²
EXTRACT ²	FROM ^{1,2,3}	IDENTITY ^{2,3}	INSENSITIVE ²
— F —	FULL ^{2,3}	IDENTITY_INSERT ³	INSERT ^{1,2,3}
FALSE ²	— G —	IDENTITYCOL ³	INT ²
FETCH ^{2,3}	GET ²	IDENTIFIED ¹	INTEGER ^{1,2}
FILE ^{1,3}	GLOBAL ²	IF ³	INTERSECT ^{1,2,3}
FILLFACTOR ³	GO ²	IMMEDIATE ^{1,2}	INTERVAL ²
FIRST ²	GOTO ^{2,3}	IN ^{1,2,3}	INTO ^{1,2,3}
FLOAT ^{1,2}	GRANT ^{1,2,3}	INCREMENT ¹	IS ^{1,2,3}

SQL Reserved Words (4)

ISOLATION^{2,3}

— J —

JOIN^{2,3}

— K —

KEY^{2,3}KILL³

— L —

LANGUAGE²LAST²LEADING²LEFT^{2,3}LEVEL^{1,2,3}LIKE^{1,2,3}LINENO³LOAD³LOCAL²LOCK¹LONG¹LOWER²

— M —

MATCH²MAX^{2,3}MAXEXTENTS¹MIN^{2,3}MINUS¹MINUTE²MIRROREXIT³MODE¹MODIFY¹MODULE²MONTH²

— N —

NAMES²NATIONAL^{2,3}NATURAL²NCHAR²NETWORK¹NEXT²NO²NOAUDIT¹NOCHECK³NOCOMPRESS¹NONCLUSTERED³NOT^{1,2,3}NOWAIT¹NULL^{1,2,3}NULLIF^{2,3}NUMBER¹NUMERIC²

— O —

OCTET_LENGTH²OF^{1,2,3}OFF³OFFLINE¹OFFSETS³ON^{1,2,3}

SQL Reserved Words (5)

ONCE ³	— P —	PRIOR ^{1,2}	RELATIVE ²
ONLINE ¹	PARTIAL ²	PRIVILEGES ^{1,2,3}	RENAME ¹
ONLY ^{2,3}	PCTFREE ¹	PROC ³	REPEATABLE ³
OPEN ^{2,3}	PERCENT ³	PROCEDURE ^{2,3}	REPLICATION ³
OPENDATASOURCE ³	PERM ³	PROCESSEXIT ³	RESOURCE ¹
OPENQUERY ³	PERMANENT ³	PUBLIC ^{1,2,3}	RESTORE ³
OPENROWSET ³	PIPE ³	— R —	RESTRICT ^{2,3}
OPTION ^{1,2,3}	PLAN ³	RAISERROR ³	RETURN ³
OR ^{1,2,3}	POSITION ²	RAW ¹	REVOKE ^{1,2,3}
ORDER ^{1,2,3}	PRECISION ^{2,3}	READ ^{2,3}	RIGHT ^{2,3}
OUTER ^{2,3}	PREPARE ^{2,3}	READTEXT ³	ROLLBACK ^{2,3}
OUTPUT ²	PRESERVE ²	REAL ²	ROW ¹
OVER ³	PRIMARY ^{2,3}	RECONFIGURE ³	ROWCOUNT ³
OVERLAPS ²	PRINT ³	REFERENCES ^{2,3}	ROWGUIDCOL ³

SQL Reserved Words (6)

ROWID ¹	SET ^{1,2,3}	SUCCESSFUL ¹	TIMEZONE_HOUR ²
ROWNUM ¹	SETUSER ³	SUM ^{2,3}	TIMEZONE_MINUTE ²
ROWS ^{1,2}	SHARE ¹	SYNONYM ¹	TO ^{1,2,3}
RULE ³	SHUTDOWN ³	SYSDATE ¹	TOP ³
— S —	SIZE ^{1,2}	SYSTEM_USER ^{2,3}	TRAILING ²
SAVE ³	SMALLINT ^{1,2}	— T —	TRAN ³
SCHEMA ^{2,3}	SOME ^{2,3}	TABLE ^{1,2,3}	TRANSACTION ^{2,3}
SCROLL ²	SQL ²	TAPE ³	TRANSLATE ²
SECOND ²	SQLCODE ²	TEMP ³	TRANSLATION ²
SECTION ²	SQLERROR ²	TEMPORARY ^{2,3}	TRIGGER ^{1,3}
SELECT ^{1,2,3}	SQLSTATE ²	TEXTSIZE ³	TRIM ²
SERIALIZABLE ³	START ¹	THEN ^{1,2,3}	TRUE ²
SESSION ^{1,2}	STATISTICS ³	TIME ²	TRUNCATE ³
SESSION_USER ^{2,3}	SUBSTRING ²	TIMESTAMP ²	TSEQUAL ³

SQL Reserved Words (7)

— U —

UID¹

UNCOMMITTED³

UNION^{1,2,3}

UNIQUE^{1,2,3}

UNKNOWN²

UPDATE^{1,2,3}

UPDATETEXT³

UPPER²

USAGE²

USE³

USER^{1,2,3}

USING²

— V —

VALIDATE¹

VALUE²

VALUES^{1,2,3}

VARCHAR^{1,2}

VARCHAR2¹

VARYING^{2,3}

VIEW^{1,2,3}

— W —

WAITFOR³

WHEN^{2,3}

WHENEVER^{1,2}

WHERE^{1,2,3}

WHILE³

WITH^{1,2,3}

WORK^{2,3}

WRITE²

WRITETEXT³

— Y —

YEAR²

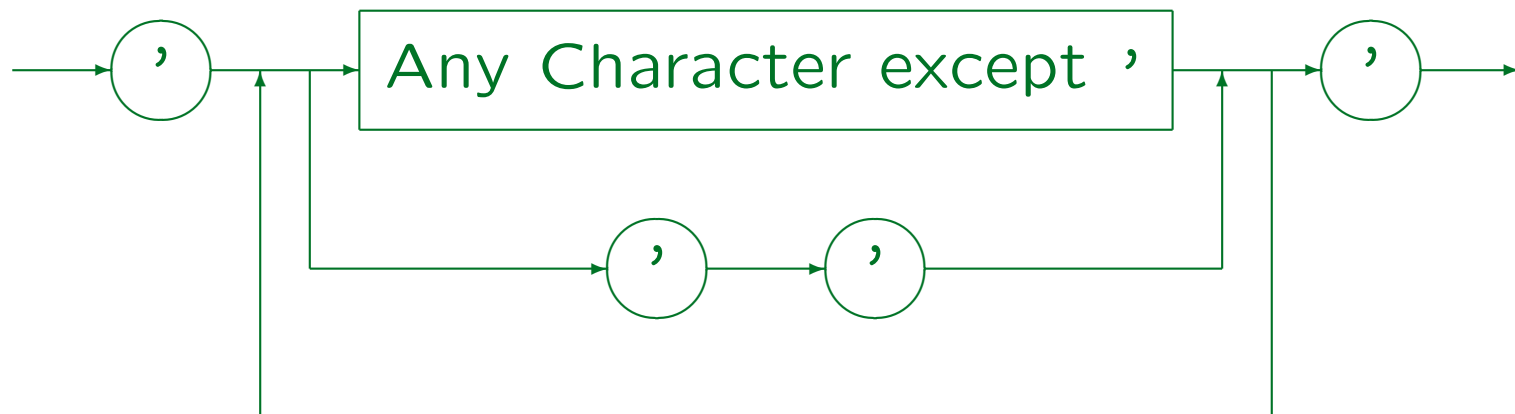
— Z —

ZONE²

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 ').

Neither Oracle nor SQL Server support this. However, strings can be combined with the concatenation operator (|| in Oracle, + in SQL Server).

- Oracle, SQL Server, and DB2 allow line breaks inside string constants.

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

- Microsoft SQL Server accepts also string literals enclosed in double quotes. This does not conform to the standard.

Other Constants

Number Constants/Literals:

- $[+|-] [0-9]^* [.] [0-9]^* [E [+|-] [0-9]^*]$

If the decimal point is present, there must be a digit before or after it.

- E.g.: 1, +2., -34.5, -.67E-8
- Note that numbers are not enclosed in quotes.

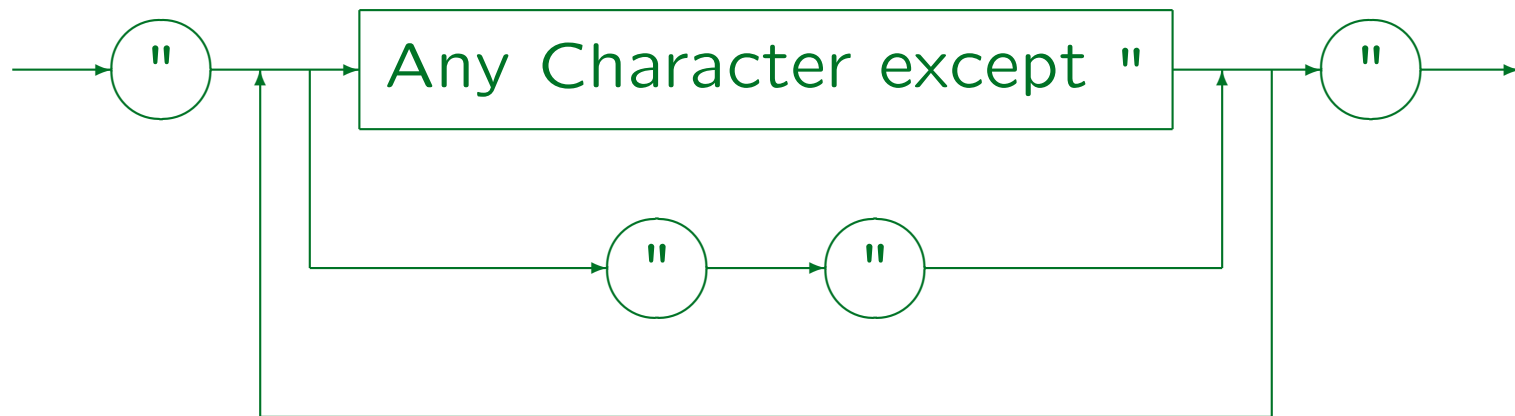
Date, Money, Binary Data etc.:

- Other data types are treated in a later chapter.

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).

- 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 old column names conflict with words reserved in a new DBMS version).

Overview

1. Lexical Syntax

2. SELECT-FROM-WHERE, Tuple Variables

3. Terms and Conditions

4. A bit of Logic

5. Null Values

Example Database (1)

STUDENTS		
<u>SSN</u>	FIRST	LAST
123-45-6789	John	Smith
111-22-3333	Ann	Miller
543-76-9821	David	Meyer
900-50-3000	Mary	Jones

ENROLLMENTS	
<u>SSN</u>	<u>CRN</u>
123-45-6789	41590
111-22-3333	41590
111-22-3333	31864
543-76-9821	22332

Example Database (2)

COURSES				
<u>CRN</u>	TOPIC	TITLE	INSTRUCTOR	SEATS
22332	2710	DB Management	Flynn, R.	50
24271	2610	Data Structures	Flynn, R.	50
31864	2550	Client-Server	Spring, M.	30
41590	2711	DB ... Design	Brass, S.	70
37329	2711	DB ... Design	Brass, S.	30
25688	2770	Document Proc.		35

Basic Query Syntax (1)

- The basic SQL-query (extensions follow) has the form:

```
SELECT  $A_1, \dots, A_n$   
FROM    $R_1, \dots, R_m$   
WHERE   $C$ 
```

- The **FROM** clause declares which tables are accessed in the query (actually, it declares variables ranging over the rows in each of the tables).
- If more than one table is listed under FROM, it will construct all possible combinations of tuples (rows) from these tables (cartesian product).

Basic Query Syntax (2)

- Next, **WHERE** filters those row combinations that satisfy the condition C .

Such a filter operation is called a selection in relational algebra.

- If more than one relation is listed under **FROM**, the **WHERE**-condition contains nearly always conditions of the form $R_i.A = R_j.B$.

Conditions that compare columns from different tables are called “join conditions”. The join is an important relational algebra operation that glues together selected combinations of rows from two tables.

- If all rows (row combinations) should be printed, the **WHERE**-clause can be omitted.

Basic Query Syntax (3)

- Finally, the **SELECT** part determines what is actually printed for each row combination that satisfies the **WHERE**-condition.

This corresponds to the operation “projection” in relational algebra. It projects the given row (combination) on certain attributes.

- If all attributes should be printed, one can write “*” instead explicitly listing all columns

All columns of all relations listed under **FROM** will be listed.

- To remove duplicate output rows, one must write **SELECT DISTINCT** instead of **SELECT**.

Basic Query Syntax (4)

- E.g., to list the complete table “COURSES” :

```
SELECT * FROM COURSES
```

- 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:

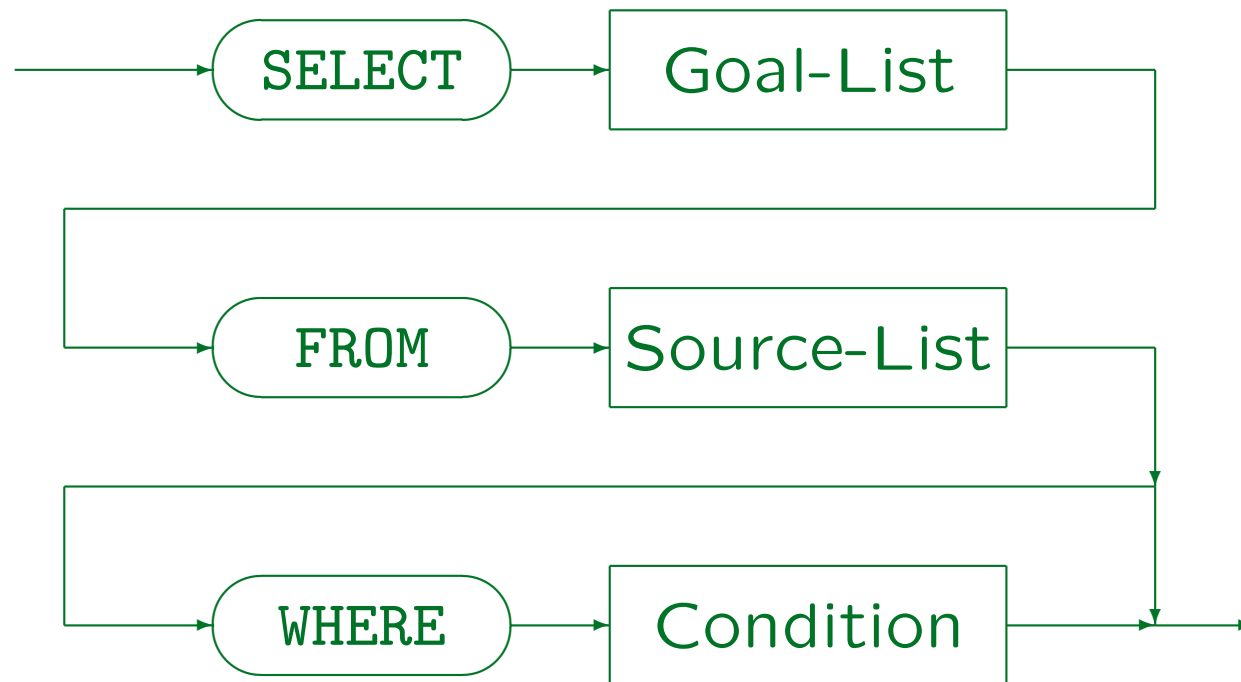
```
SELECT TO_CHAR(SQRT(2)) FROM DUAL.
```

In SQL Server, **FROM** can be omitted, e.g.

```
SELECT 1+1.
```

Basic Query Syntax (5)

SELECT-Expression (Simplified):



Tuple Variables (1)

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

```
SELECT C.INSTRUCTOR
FROM   COURSES C
WHERE  C.TOPIC = 2710
```

- This can be executed as:

```
for C in COURSES do
    if C.TOPIC = 2710 then
        print C.INSTRUCTOR
```

- **C** stands here for a single row in the table **COURSES** (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 COURSES.INSTRUCTOR  
FROM COURSES  
WHERE COURSES.TOPIC = 2710
```

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

```
FROM COURSES COURSES
```

(The tuple variable called “COURSES” ranges over the rows of the table “COURSES”.)

Tuple Variables (3)

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

`FROM COURSES C`

it is an error to try to access

`COURSES.INSTRUCTOR`

The tuple variable is now called “`C`”, not “`COURSES`”.

- 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 , see below.

Table Aliases

- Alternatively, and can understand “FROM COURSES C” as declaring another name (alias, abbreviation) “C” for the table COURSES.
- However, for self joins (which compare multiple rows of the same table) one then needs to consider this as copies of tables.

This might be less natural than thinking about two variables ranging over the rows of the same table.

- Use the view that helps you more!

Tuple variables and table aliases are the same thing.

Table Names

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

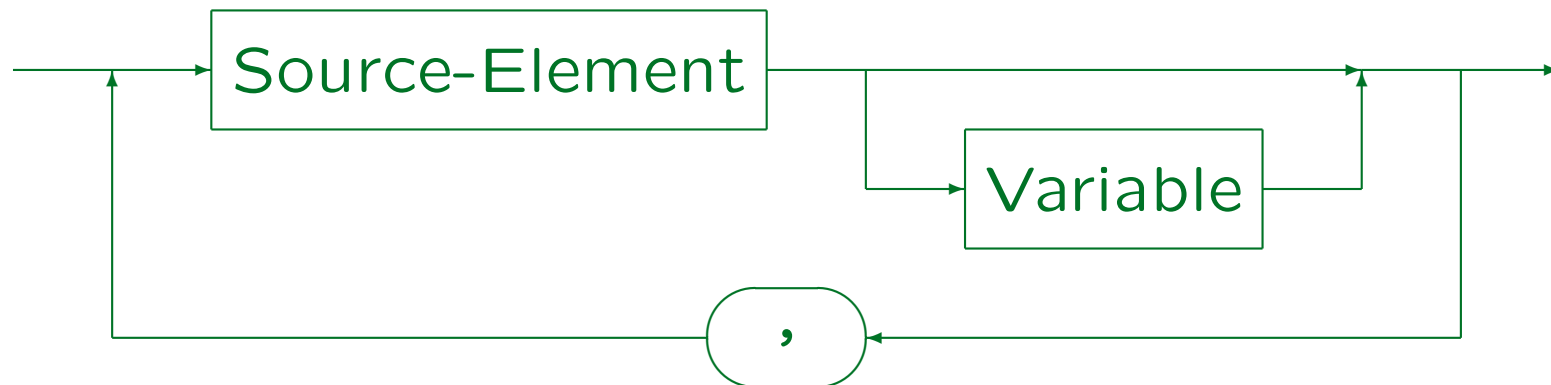
```
SELECT *  
FROM BRASS.DEPT
```

- 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 and you 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”.

FROM Syntax (1)

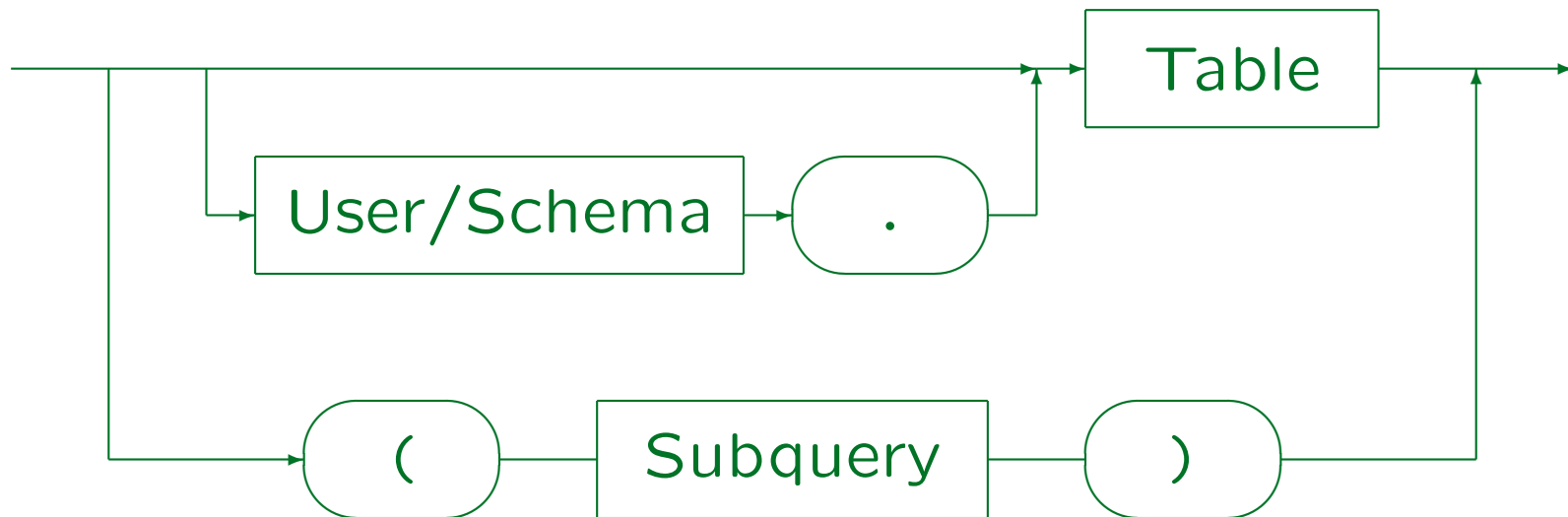
Source-List (after FROM):



- In SQL-92, SQL Server, and DB2 (but not in Oracle 8.0) one can write AS between Source-Element and Variable. In SQL-92 and DB2 (not Oracle, SQL Server) new column names can be defined:
`STUDENTS AS S(NO,FNAME,LLAME), ENROLLMENTS AS E(NO,C).`
- SQL-92, SQL Server, and DB2 support joins in the FROM-list (see Part 4).

FROM Syntax (2)

Source-Element:



- Subqueries are treated below.
- SQL-86 did not allow subqueries in the FROM-list.
- Basic (simplified) syntax of the FROM-clause:

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

Joins (1)

- It is possible to declare more than one tuple variable in the FROM clause:

```
SELECT  $A_1, \dots, A_n$   
FROM STUDENTS S, ENROLLMENTS E  
WHERE  $C$ 
```

- Then the tuple variable S will range over the 4 tuples in STUDENTS, and E will range over the 4 tuples in ENROLLMENTS.

Joins (2)

- In principle, all $4 * 4 = 16$ combinations are considered:

```
for S in STUDENTS do  
  for E in ENROLLMENTS do  
    if C then print  $A_1, \dots, A_n$ 
```

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

This is the task of the query optimizer.

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

Joins (3)

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

```
SELECT E.CRN
FROM   STUDENTS S, ENROLLMENTS E
WHERE  S.SSN = E.SSN      -- Join Condition
AND    S.FIRST = 'Ann' AND S.LAST = 'Miller'
```

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

```
SELECT S.FIRST, S.LAST      Wrong!
FROM   STUDENTS S, ENROLLMENTS E
WHERE  E.CRN = 41590
```

Joins (4)

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

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

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, ENROLLMENTS E, COURSES C
WHERE  S.SSN = E.SSN AND E.CRN = C.CRN
AND    C.INSTRUCTOR LIKE 'Brass%'
```

- The tuple variables are connected as follows:



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 CRN
FROM   STUDENTS S, ENROLLMENTS E
WHERE  S.SSN = E.SSN
AND    FIRST = 'Ann' AND LAST = 'Miller'
```

“FIRST” and “LAST” can only refer to “S”, “CRN” can only refer to “E”, but “SSN” alone would be ambiguous, since “S” and “E” both have an attribute with this name.

Attribute References (2)

- Consider this query:

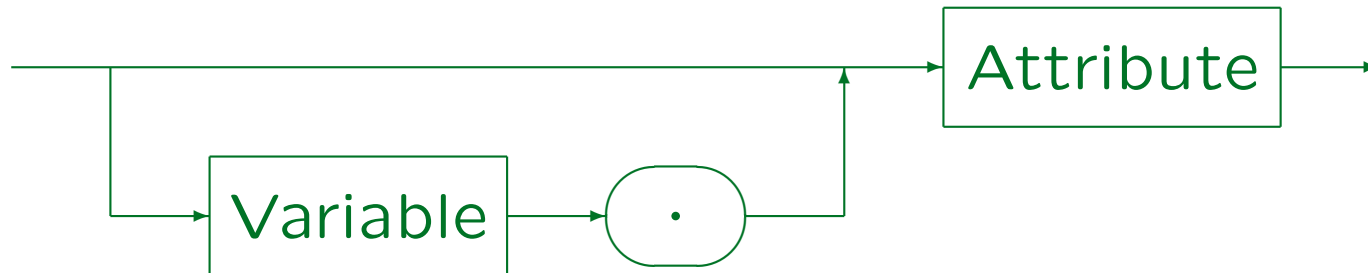
```
SELECT CRN, TITLE           Wrong!
FROM   STUDENTS S, ENROLLMENTS E, COURSES C
WHERE  S.SSN = E.SSN AND E.CRN = C.CRN
AND    FIRST = 'Ann' AND LAST = 'Miller'
```

- SQL requires that the user specifies whether he/she wants C.CRN or E.CRN 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 "CRN", the tuple variable cannot be left out, or the DBMS (e.g. Oracle) will print the error message "ORA-00918: column ambiguously defined".

Attribute References (3)

Attribute-Reference:



- The variable must be specified if there is more than one variable with this attribute.

Query Formulation (1)

- Task: Write an SQL query which prints the titles of all courses Ann Miller attends.
- First it must be understood that Ann Miller is a student, requiring a tuple variable **S** over **STUDENTS** and the condition **S.FIRST='Ann' AND S.LAST='Miller'**.
- Course titles are requested, so a tuple variable **C** over **COURSES** is needed, and the following piece can already be generated:

```
SELECT C.TITLE
```

Query Formulation (2)

- Finally, **S** and **C** 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 **E** over **ENROLLMENTS** is required, and yields the condition

$$S.SSN = E.SSN \text{ AND } E.CRN = C.CRN$$

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. suppose there are also GSAs (graduate student assistants) that are assigned to courses. They are 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. students registered for the course 41590:

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, ENROLLMENTS E, COURSES C
WHERE  S.SSN = E.SSN AND E.CRN = C.CRN
AND    C.CRN = 41590
```

- Can the following query ever give a different result?

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, ENROLLMENTS E
WHERE  S.SSN = E.SSN AND E.CRN = 41590
```

Unnecessary Joins (2)

- What will be the result of this query?

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, ENROLLMENTS E, COURSES C
WHERE  S.SSN = E.SSN AND E.CRN = 41590
```

- 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, ENROLLMENTS E
WHERE  S.SSN = E.SSN
```

Tuple-wise Evaluation (1)

- Suppose the FROM-clause declares two tuple variables: STUDENTS S, ENROLLMENTS E.
- The WHERE-condition is evaluated with a tuple from STUDENTS assigned to S, and one from ENROLLMENTS assigned to E.
- This is repeated for every possible combination of two such tuples.
- Therefore, when the WHERE-condition is evaluated, specific values for all attributes are known.

Tuple-wise Evaluation (2)

- Attribute references like `S.SSN` are conceptually replaced by the value of the attribute `SSN` in the current tuple `S` (from `STUDENTS`).

The same for references to attributes of `E`.

- After this substitution only values (i.e. constants), comparison operators (like `=`), and logical connectives (e.g. `AND`) remain.
- The `WHERE`-condition can then be evaluated to true or false.

Tuple-wise Evaluation (3)

- Suppose that S and E are assigned these tuples:

S:

SSN	FIRST	LAST
123-45-6789	John	Smith

E:

SSN	CRN
123-45-6789	41590

- Then the WHERE condition is evaluated as:

SELECT FIRST, LAST		'John', 'Smith'
FROM ...		
WHERE S.SSN=E.SSN		'123-45-6789'='123-45-6789'
AND CRN=41590		AND 41590=41590

- This gives "TRUE AND TRUE", i.e. TRUE.
- Thus "John Smith" appears in the output.

Overview

1. Lexical Syntax
2. SELECT-FROM-WHERE, Tuple Variables
3. Terms and Conditions
4. A bit of Logic
5. Null Values

Terms (1)

- A term denotes a data element.
Instead of term, one can also say “expression”.
- Terms are:
 - ◇ Attribute References, e.g. `STUDENT.SSN`.
 - ◇ Constants, e.g. `'John'`, `41590`.
 - ◇ Composed Terms, using datatype operators like `+`, `-`, `*`, `/` (for numbers), `||` (string concatenation), and datatype functions, e.g. `0.9 * SEATS`.
 - ◇ Aggregation terms (e.g. `MAX(SEATS)`): see below.

Terms (2)

- The SQL-86 standard contained only +, -, *, /.
- Current database management systems still differ in other data type operations.
- E.g. the operator || is contained in the SQL-92 standard, but does not work in SQL Server 7.0.

String concatenation is written "+" in SQL Server 7.0.

Other datatype functions (e.g. SUBSTR) are even less standardized.

- SQL knows the standard precedence rules, e.g. that $A+B*C$ means $A+(B*C)$. Parentheses may be used.

Terms (3)

- Terms are used in conditions, e.g.

```
C.SEATS > 50
```

contains the two terms "C.SEATS" and "50".

- Also the SELECT-list can contain arbitrary terms:

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

Name
Smith, John
Miller, Ann
Meyer, David
Jones, Mary

Conditions (1)

- Conditions consist of atomic formulas, e.g.

`SEATS > 50,`

connected by “AND”, “OR”, “NOT”.

- AND binds more strongly than OR, thus

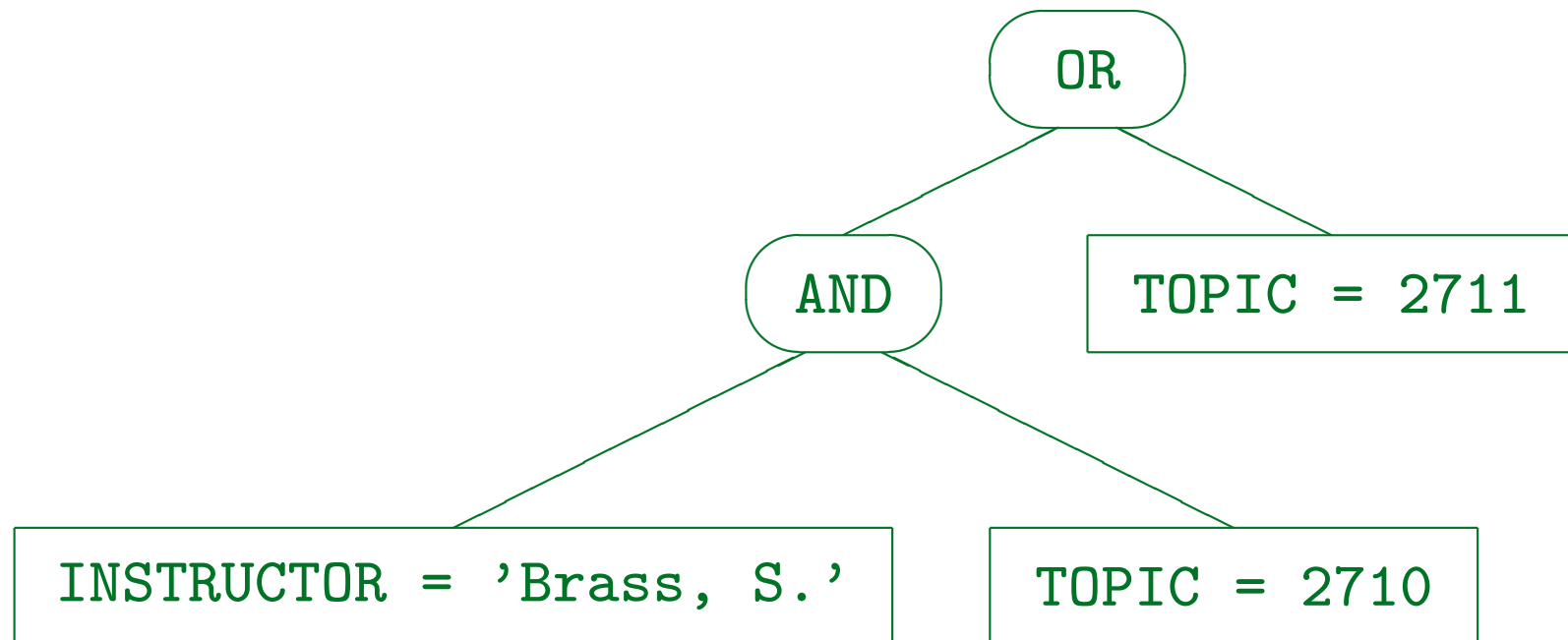
```
INSTRUCTOR = 'Brass, S.' AND
TOPIC = 2710 OR TOPIC = 2711
```

is implicitly parenthesized as

```
(INSTRUCTOR = 'Brass, S.' AND TOPIC = 2710)
OR TOPIC = 2711
```

Conditions (2)

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



Conditions (3)

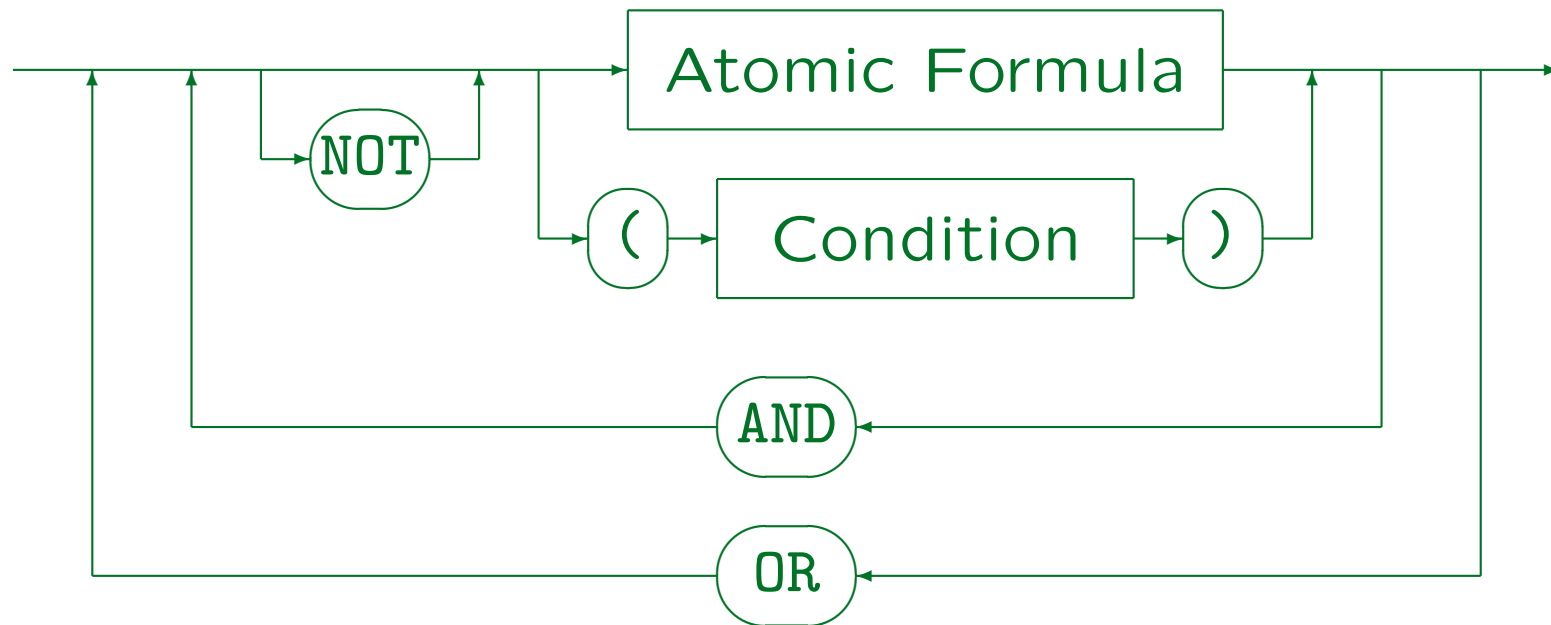
- NOT binds most strongly, i.e. unless parentheses are used, it is applied only to the immediately following atomic formula.
- E.g. NOT TOPIC = 2710 OR TOPIC = 2711
means (NOT TOPIC = 2710) OR TOPIC = 2711
i.e. TOPIC <> 2710 OR TOPIC = 2711
i.e. TOPIC <> 2710

The reason for the last step is that the condition TOPIC = 2711 logically implies the condition TOPIC <> 2710. Therefore, whenever TOPIC = 2711 is satisfied, TOPIC <> 2710 is also satisfied.

- Parentheses (...) can be used if necessary.

Conditions (4)

Condition:



- SQL-92 allows “IS NOT TRUE”, “IS FALSE” etc. after formulas (not supported in Oracle 8.0, SQL Server, DB2).

Conditions (5)

- 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 S.FIRST, S.LAST
FROM   STUDENTS S, ENROLLMENTS E
WHERE  S.SSN = E.SSN
AND    E.CRN = 41590 OR 31864
```

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

Here the word AND does not denote the logical connective.

Conditions (6)

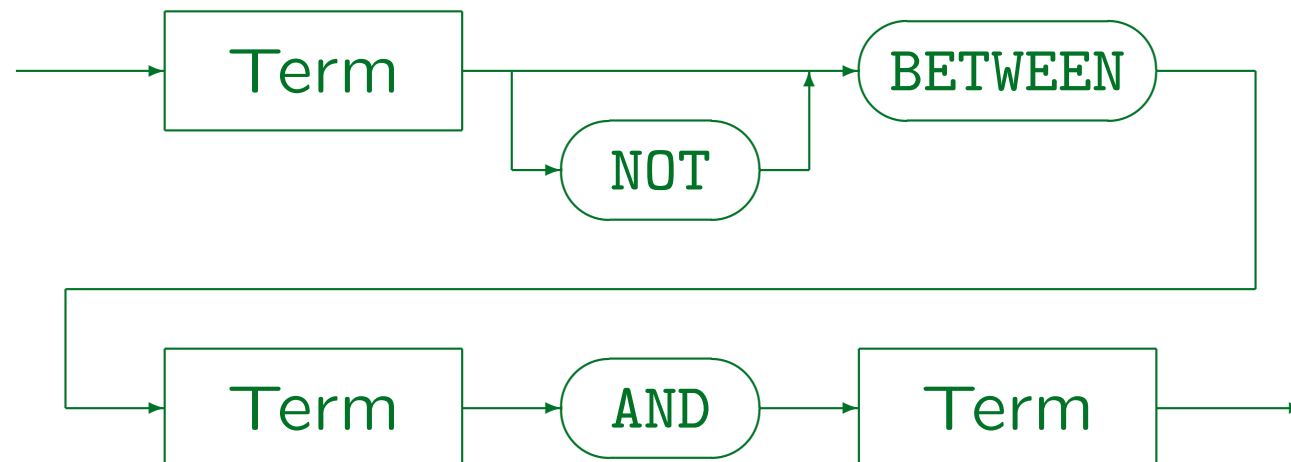
Atomic Formula (Form 1):



- Comparison operators: =, <>, <, >, <=, >=.
- Comparison operators can be used for numbers as well as for strings.
- “Not equals” is written in standard SQL as “<>”.
Oracle, SQL Server, and DB2 understand also “!=”.
“^=” works in Oracle and DB2, but not in SQL Server.
- E.g.: SEATS > 50, LAST < 'M'.

Conditions (7)

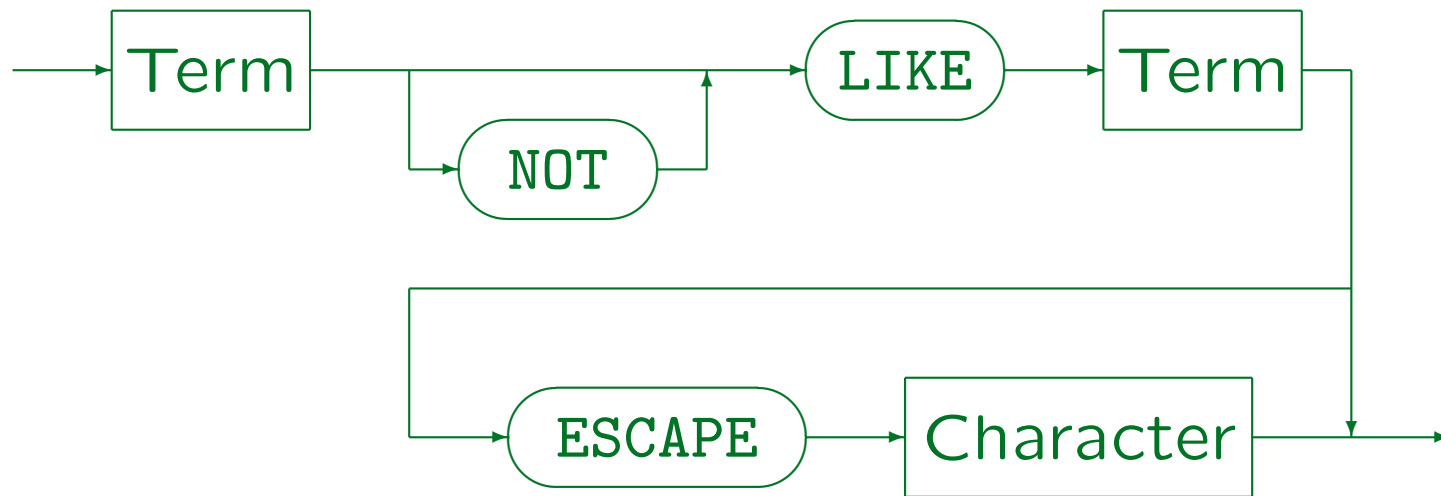
Atomic Formula (Form 2):



- x BETWEEN y AND z is equivalent to $x \geq y$ AND $x \leq z$.
- E.g.: SEATS BETWEEN 20 AND 40

Conditions (8)

Atomic Formula (Form 3):



- E.g.: `INSTRUCTOR LIKE '%Brass%'`

Is true for all instructors containing the substring 'Brass'.

Conditions (9)

LIKE Conditions, Continued:

- “%” in the second argument to LIKE matches any sequence of arbitrary characters in the first argument.

- “_” matches any single character.

SQL Server supports also character ranges, e.g. [a-zA-Z].

- In SQL-86 and DB2 the pattern must be a string constant.

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

Conditions (10)

LIKE Conditions, Continued:

- To use the characters “%” and “_” without their special meaning in the second argument, an “escape” character is used. It removes the special meaning of the following character. It must be explicitly declared, e.g.:

```
ProcName LIKE '\_%' ESCAPE '\'
```

This gives all procedure names starting with an “_”.

Conditions (12)

IN Conditions, Continued:

- E.g. `INSTRUCTOR IN ('Brass, S.', 'Spring, M.')`
- This is equivalent to

```
INSTRUCTOR = 'Brass, S.'  
OR INSTRUCTOR = 'Spring, M.'
```

- 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.

Overview

1. Lexical Syntax
2. SELECT-FROM-WHERE, Tuple Variables
3. Terms and Conditions
4. A bit of Logic
5. Null Values

A bit of Logic (1)

- Conditions used in the `WHERE`-clause are formulas of tuple calculus, which is a variant of predicate logic.
- Predicate logic is studied for about 100 years in mathematics and philosophy.
- Some basic knowledge of logic can actually help in query formulation.
- Here, the notions “inconsistent”, “tautology”, “implied”, and “equivalent” are introduced, as well as some concrete equivalences for the propositional connectives `AND`, `OR`, `NOT`.

A bit of Logic (2)

- A condition is inconsistent if it can never be satisfied, i.e. is always false, no matter what the database state is and no matter which tuples are assigned to the tuple variables.
- E.g., no matter what row stands **C** for, **C.TOPIC** cannot be two different values at the same time:

C.TOPIC = 2710 AND C.TOPIC = 2711

- An inconsistent condition as **WHERE**-clause means that the query will never return any result rows.

A bit of Logic (3)

- Database management systems like Oracle do not give warnings for inconsistent conditions.

Actually, it can be proven that it is impossible to develop an algorithm that detects all inconsistent conditions (if also subqueries or arithmetic operations are allowed).

- The other extreme is a tautology, i.e. a condition that is always true, e.g.:

`C.TOPIC < 2000 OR C.TOPIC > 1000`

- Obviously, such conditions are not useful.

A bit of Logic (4)

- A condition A implies a condition B if, whenever A is true, also B is true.

The implied condition B is weaker than condition A that implies it.

- E.g. “ $C.TOPIC = 2710$ ” implies “ $C.TOPIC \neq 2711$ ”.
- Therefore, the condition

$C.TOPIC = 2710 \text{ AND } C.TOPIC \neq 2711$

can be safely simplified to

$C.TOPIC = 2710.$

The second part gives nothing new.

A bit of Logic (5)

- Two conditions are called (logically) equivalent if they always yield the same truth value.

I.e. A and B are equivalent if for all database states and all assignments of rows to the tuple variables, if A is true, then B is true, and if A is false, then B is false. Equivalence means implication in both directions.

- E.g. it is not important whether one writes

```
INSTRUCTOR = 'Brass, S.' AND TITLE LIKE '%DB%'
```

or vice versa

```
TITLE LIKE '%DB%' AND INSTRUCTOR = 'Brass, S.'
```

A bit of Logic (6)

- For the correctness of a query, it is not important which one out of several logically equivalent formulations one chooses.
- Of course, some formulations are more complicated than others, and one should choose a simple one.

For instance, although adding an implied condition as shown above does not change the correctness of the query, points might be taken off in the exam for unnecessary complications.
- Modern DBMSs have good optimizers, such that simple equivalences like $A \text{ AND } B$ vs. $B \text{ AND } A$ are not important for the runtime of a query.

A bit of Logic (7)

- More complicated equivalences might not be detected by the query optimizer, e.g. writing

$$\text{TOPIC} - 2710 = 0$$

might prevent that a special access structure for finding rows quickly (B-tree index) is used, which would have been used for the logically equivalent condition

$$\text{TOPIC} = 2710$$

- However, one gets the same answer in both cases, only the first query might run slightly longer.

Some Equivalences (1)

- $A \text{ AND } B \equiv B \text{ AND } A$

This is called commutativity. It holds also for OR.

- $A \text{ AND } (B \text{ AND } C) \equiv (A \text{ AND } B) \text{ AND } C$

This is called associativity. It means that no parentheses are necessary if one has a sequence of conditions all connected with AND. The associative law also holds for OR.

- $A \text{ AND } (B \text{ OR } C) \equiv (A \text{ OR } B) \text{ AND } (A \text{ OR } C)$

This is the distribution law. It holds also for AND and OR exchanged.

- $\text{NOT } (\text{NOT } A) \equiv A$

This means that double negation cancels out.

Some Equivalences (2)

- $\text{NOT}(A \text{ AND } B) \equiv (\text{NOT } A) \text{ OR } (\text{NOT } B)$

This is De Morgan's Law. It holds also with `AND` and `OR` exchanged.

- $A \text{ AND } A \equiv A$

It makes no sense to repeat a condition. This holds also for `OR`.

- $\text{NOT } X < Y \equiv X \geq Y$

The comparison operators always come in complementary pairs, and it is not necessary to use `NOT` directly in front of such a condition. Together with De Morgan's law and the double negation rule, one can eliminate `NOT` from conditions (that use only the six comparison operators). But this might not always make the condition simpler.

Overview

1. Lexical Syntax
2. SELECT-FROM-WHERE, Tuple Variables
3. Terms and Conditions
4. A bit of Logic
5. Null Values

Null Values (1)

- The relational model permits that attribute values are missing.

So a tuple does not necessarily define values for all attributes of its schema. Table entries can be empty.

- Formally, all domains (data types) are extended by a new value “null”.

“Null” is not the number 0 or the empty string.
It is different from all values of the data type.

- On the one hand, null values are often used in practice, on the other hand, they lead to quite severe problems.

Null Values (2)

Null values are used e.g. in the following situations:

- A value exists, but is not known.

E.g., every customer's phone number may not be known, although probably most have a phone.

- No value exists.

Not every customer has a fax.

- The attribute is not applicable to this tuple.

For instance, storing the known programming languages only for the subclass of employees who are programmers.

A committee once found 13 different meanings for a null value.

Null Values (3)

Advantages of Null Values:

- Without null values, it would be necessary to split most relations in many relations (corresponding to subclasses).

E.g. `Customer_with_Phone`, `Customer_without_Phone`.

Or extra relation: `Cust_Phone(CustNo, Phone)`.

Null values can help to avoid joins and unions.

- If null values are not allowed, users will invent fake values to fill the missing columns.

This makes the database structure even more unclear.

Three-Valued Logic (1)

- Consider the following query:

```
SELECT CRN, TITLE
FROM   COURSES
WHERE  INSTRUCTOR = 'Brass, S.'
```

- What happens if a course has a null value in the column INSTRUCTOR? It is not printed.
- But it also does not appear in the result of this query:

```
SELECT CRN, TITLE
FROM   COURSES
WHERE  NOT (INSTRUCTOR = 'Brass, S.')
```

Three-Valued Logic (2)

- The condition

`INSTRUCTOR = 'Brass, S'`

does not evaluate to false if `INSTRUCTOR` 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 CRN, TITLE
FROM COURSES
WHERE INSTRUCTOR = null
```

Actually, the query is illegal in SQL-92, and DB2 refuses it.

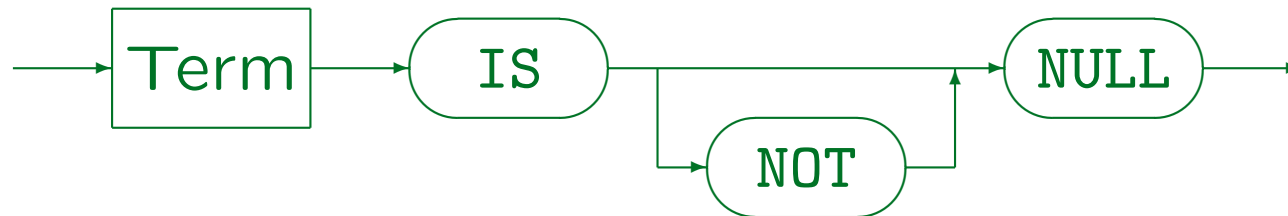
- “AND” / “OR” forward the truth value “unknown”, unless the result is clear:
E.g. “true OR unknown = true”.

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

Atomic Formula (Form 5):



- E.g. `INSTRUCTOR IS NULL`
- The test for a null value can only be done in this way.

“`INSTRUCTOR = NULL`” does not give the expected result in Oracle and SQL Server, it is a syntax error in SQL-92 and DB2.
In SQL Server 7, “`INSTRUCTOR = NULL`” works after the command “`SET ANSI_NULLS OFF`” (then a two-valued logic is used).

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 courses taught by Brass and courses not taught by Brass are counted, one would normally assume to get all courses. But this is not true in SQL — those with a null value in the `INSTRUCTOR` 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” ($\exists X: \dots$) would allow to use standard logical equivalences.

Terms with Null Values (1)

- 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.

In Oracle, A || B (the concatenation of strings A and B) returns B if A is null (violates the SQL-92 standard).

- 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 (2)

- 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 “NULL” seems to be handled like a normal expression.