Part 5: 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 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.
- Date: A Guide to the SQL Standard, First Edition, Addison-Wesley, 1987.
- van der Lans: SQL, Der ISO-Standard (in German). Hanser, 1990.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- 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 Edition (Part of MSDN Library Visual Studio 6.0).
- DuBois: MySQL. New Riders Publishing, 2000, ISBN 0-7357-0921-1, 756 pages.

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 6.

- 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

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

Example Database

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

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⟩

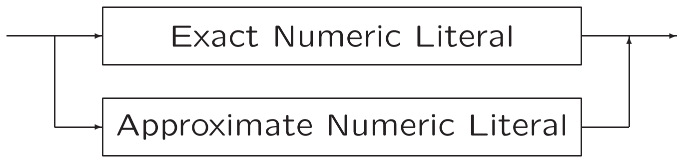
    Supported in SQL-92, Oracle, SQL Server, IBM DB2, MySQL. MySQL requires a space after the "--", SQL-92 does not. Access does not support this comment, and also not `/* ...*/`.
  - ◇ From "/*" to "*/"

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

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

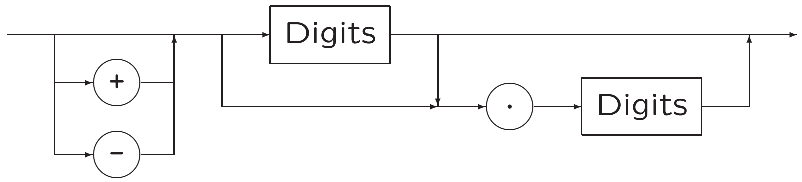
Numbers (1)

- 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)

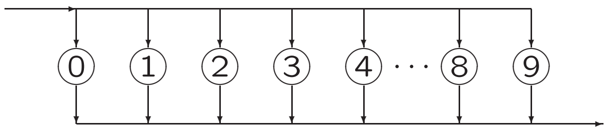
- Exact Numeric Literal:



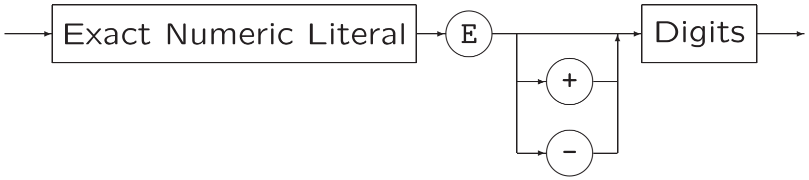
- Digits (Unsigned Integer):

Numbers (3)

- Digit:



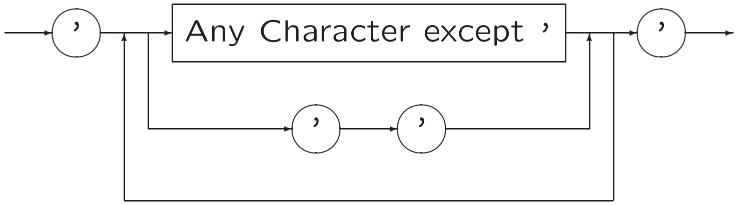
- Approximate Numeric Literal:



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 7):
  - ◇ 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.

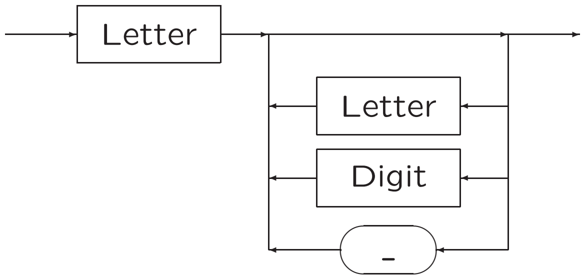
Other Constants (2)

- 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	≤ 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,_
Access	≤ 64	A-Z,a-z	A-Z,a-z,0-9,_
MySQL	≤ 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)

- 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. However, keywords and column names are not case sensitive.

- 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.

SQL Reserved Words (1)

[1] = Oracle 8.0	ARE[2]	BOTH[2]	CHECK[1,2,3]
[2] = SQL-92	AS[1,2,3]	BREAK[3]	CHECKPOINT[3]
[3] = SQL Server 7	ASC[1,2,3]	BROWSE[3]	CLOSE[2,3]
	ASSERTION[2]	BULK[3]	CLUSTER[1]
— **A** —	AT[2]	BY[1,2,3]	CLUSTERED[3]
ABSOLUTE[2]	AUTHORIZATION[2,3]	— **C** —	COALESCE[2,3]
ACCESS[1]	AUDIT[1]	CASCADE[2,3]	COLLATE[2]
ACTION[2]	AVG[2,3]	CASCADED[2]	COLLATION[2]
ADD[1,2,3]	— **B** —	CASE[2,3]	COLUMN[1,3]
ALL[1,2,3]	BACKUP[3]	CATALOG[2]	COMMENT[1]
ALLOCATE[2]	BEGIN[2,3]	CHAR[1,2]	COMMIT[2,3]
ALTER[1,2,3]	BETWEEN[1,2,3]	CHARACTER[2]	COMMITTED[3]
AND[1,2,3]	BIT[2]	CHAR_LENGTH[2]	COMPRESS[1]
ANY[1,2,3]	BIT_LENGTH[2]	CHARACTER_LENGTH[2]	COMPUTE[3]

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

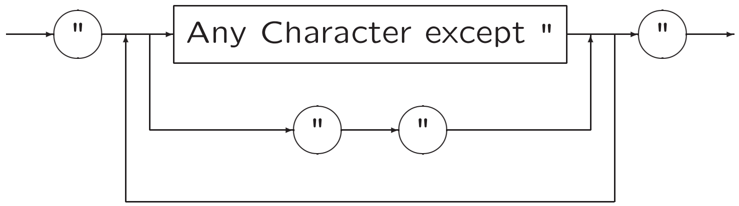
SQL Reserved Words (7)

— 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, no 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. Lexical Syntax
2. SELECT-FROM-WHERE, Tuple Variables
3. Terms and Conditions
4. A bit of Logic
5. Null Values

Basic Query Syntax

- Basic SQL query (extensions follow):

  SELECT $A_1, \ldots, A_n$
  FROM $R_1, \ldots, R_m$
  WHERE C

- The FROM clause declares which table(s) are accessed in the query.
- The WHERE clause specifies a condition for the rows (row combinations) in these tables that are considered in the query.
- The SELECT clause specifies what to print.

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?

Wrong!

```
SELECT S.FIRST, S.LAST
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.

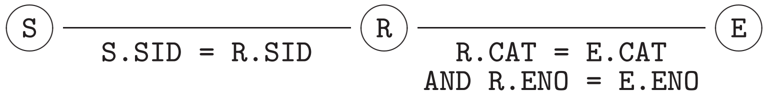
  However, it is also possible that constant values are required for the join attributes instead. In seldom cases a connection might also be done in a subquery.

- 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.

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.

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 (several exercises can have the same topic):

```
SELECT DISTINCT E.TOPIC
```


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):

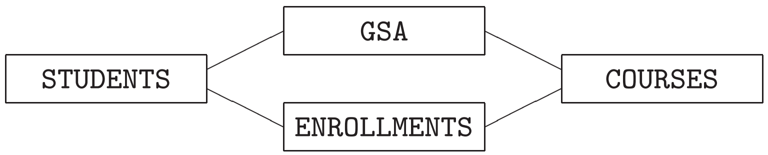
- This shows that a tuple variable `R` over `RESULTS` is required, and yields the condition

```
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 SID, 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, Homwork 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:

Wrong!

```
SELECT S.FIRST, S.LAST
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.

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)

- If the query might contain duplicates, and there is no specific reason why they should be shown, use "SELECT DISTINCT" (DISTINCT applies to rows, not columns):

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

CAT	ENO
H	1
H	2
M	1

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

  However, "ALL" is the default.

Duplicate Elimination (3)

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).

Duplicate Elimination (4)

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 $\mathcal{K}$.

Duplicate Elimination (5)

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'.

Duplicate Elimination (6)

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").

Duplicate Elimination (7)

- 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.

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 (see below).
- Duplicates are often an indication for errors: One should understand the source of the duplicates and not simply specify DISTINCT to avoid the problem.
- An unnecessary DISTINCT should be avoided.

Renaming of Output Colums

- To rename the output columns:

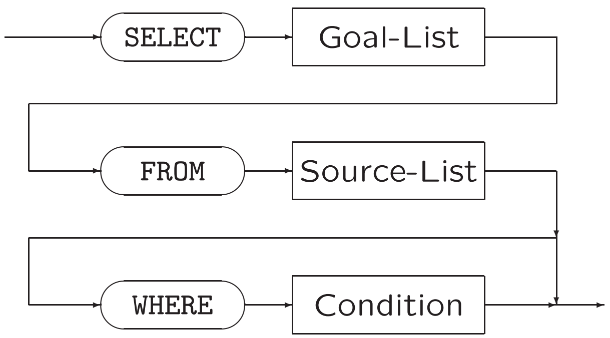
```
SELECT FIRST AS First_Name, LAST AS "Last Name"
FROM   STUDENTS
```

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.

Basic Query Syntax (1)

SELECT-Expression (Simplified):



Basic Query Syntax (2)

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

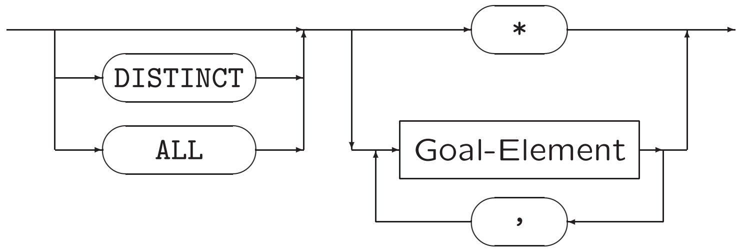
  Oracle provides a relation "DUAL" which has only one row. It can be used if only a computation is done without access to the database: `SELECT TO_CHAR(SQRT(2)) FROM DUAL`.

- 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.

SELECT Syntax (1)

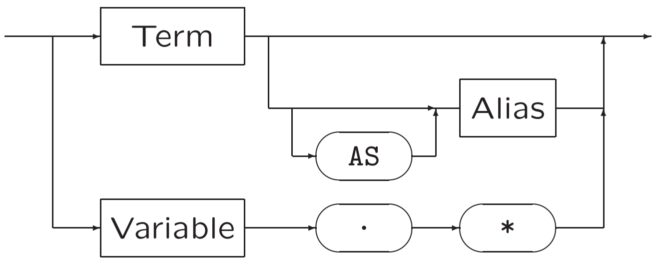
Goal-List (after SELECT):



- ALL (no duplicate elimination) is the default.

SELECT Syntax (2)

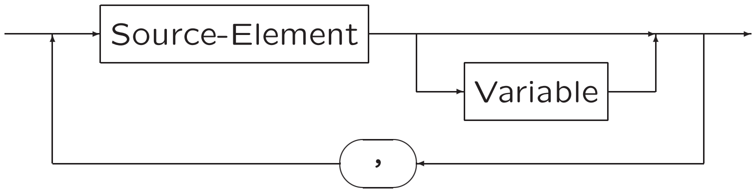
Goal-Element:



- "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.

FROM Syntax (1)

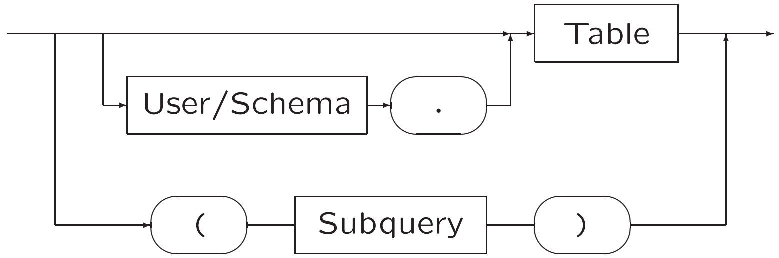
Source-List (after FROM):



- 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 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". In MySQL, one can write "database.table".

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.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 6.

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 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.
- 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.

  R.POINTS > E.MAXPT * 0.8

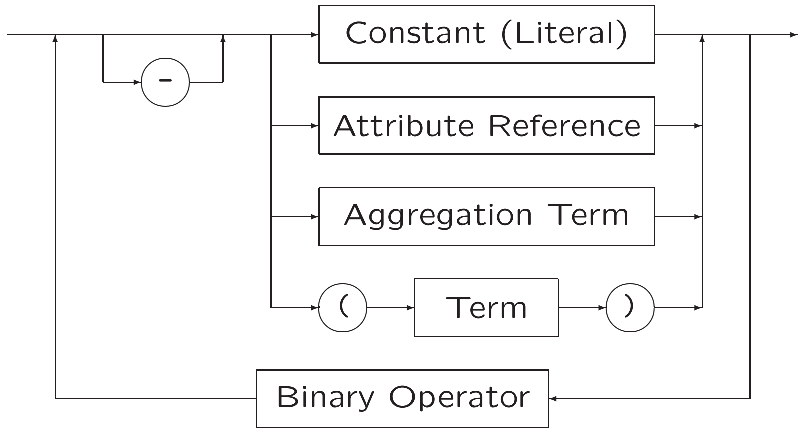
  contains the terms "R.POINTS" and "E.MAXPT * 0.8".
- Also the SELECT-list can contain arbitrary terms:

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

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

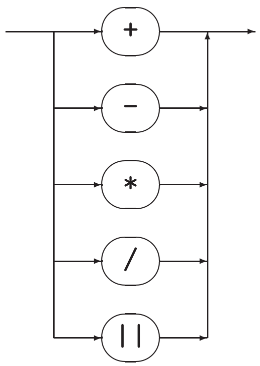
Terms (4)

Term (Scalar Expression, Value Expression):



Terms (5)

Binary Operator:



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

Conditions (1)

- Conditions consist of atomic formulas, e.g.

  `POINTS >= 8`,

  connected by "`AND`", "`OR`", "`NOT`".
- `AND` binds more strongly than `OR`, thus

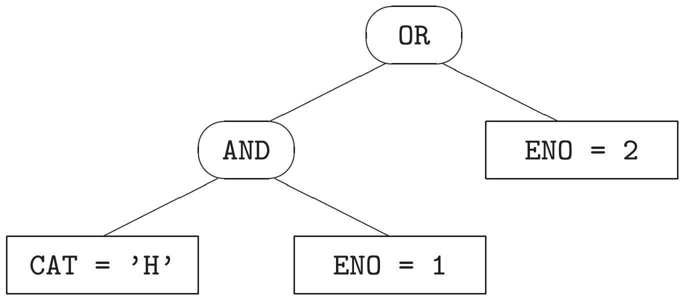
  `CAT = 'H' AND ENO = 1 OR ENO = 2`

  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":



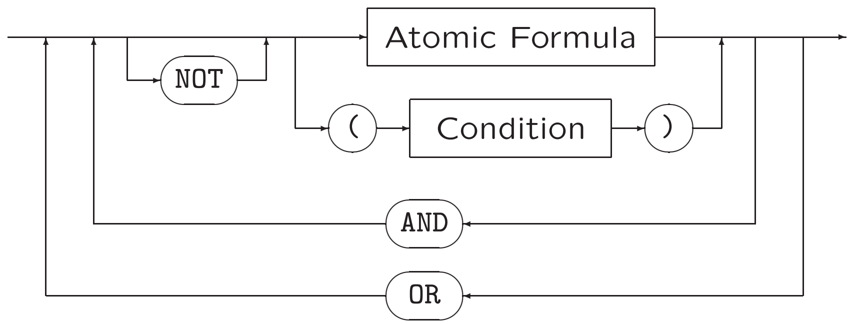
Conditions (3)

- 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)

Condition:



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

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 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:

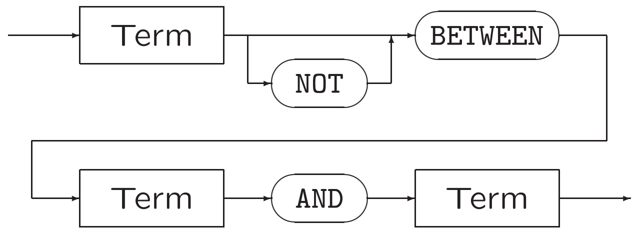
```
SELECT '"' || LAST || '"' AS LAST_NAME
FROM   STUDENTS
```

- 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

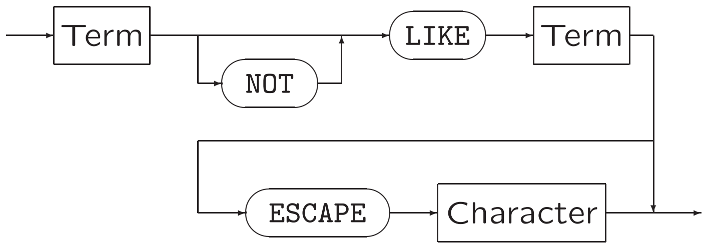
Atomic Formula (Form 2):



- `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'`

  Thius 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).
- "_" matches any single character.

  SQL Server and Access support also character ranges, e.g. `[a-zA-Z]`. MySQL has an additional operator "`RLIKE`" (or "`REGEXP`") that accepts arbitrary regular expressions as patterns.

LIKE Conditions (3)

- 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 (4)

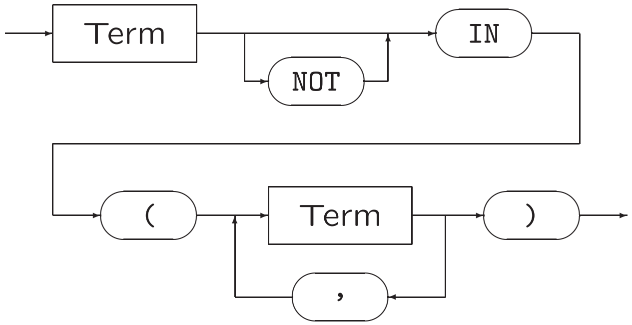
- `LIKE` uses the non-padded semantics.

  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.

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".

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 `R` for, `R.ENO` cannot be two different values at the same time:

  `R.ENO = 1 AND R.ENO = 2` **Wrong!**

- 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.:

```
R.ENO < 3  OR  R.ENO > 2
```

- 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. A set of conditions $\{A_1, \ldots, A_n\}$ implies a condition B if, whenever A_1 to A_n are all true, also B is true.

- E.g. "`R.ENO = 2`" implies "`R.ENO <> 1`".
- Therefore, the condition

```
R.ENO = 2  AND  R.ENO <> 1
```

  can be safely simplified to `R.ENO = 2`.

  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

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

  or vice versa

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


A bit of Logic (6)

- For the correctness of a query, it is not important which one out of several logically equivalent formulation 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 `AND` B vs. B `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

  `ENO - 2 = 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

  `ENO = 2`

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

Some Equivalences (1)

- A AND B $\equiv$ B AND A

  This is called commutativity. It holds also for OR.

- A AND (B AND C) $\equiv$ (A AND B) 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 AND (B OR C) $\equiv$ (A OR B) AND (A OR C)

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

- NOT (NOT A) $\equiv$ A

  This means that double negation cancels out.

Some Equivalences (2)

- NOT(A AND B) $\equiv$ (NOT A) OR (NOT B)

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

- A AND A $\equiv$ A

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

- NOT X < Y $\equiv$ X >= 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.

Some Equivalences (3)

- X = Y $\equiv$ Y = X (symmetry)
- X < Y $\equiv$ Y > X

  And the same for <= and >=.
  Also, X <= Y is equivalent to X < Y OR X = Y.

- X = Y AND Y = Z implies X = Z (transitivity)

  And the same for <, <=, >, >=. When A implies B, the formulas A and A AND B are equivalent. Thus, X = Y AND Y = Z is equivalent to X = Y AND Y = Z AND X = Z Since certain equality conditions can be evaluated by using an index, it makes sense for a query optimizer to compute implied such conditions.

- X = X is a tautology if X cannot be null.

  SQL uses a three-valued logic, see below.

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.'
```


Overview

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

Three-Valued Logic (1)

- Consider the following query:

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

- What happens if a course 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.

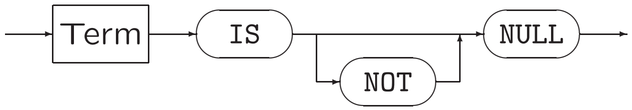
- "`AND`" / "`OR`" forward the truth value "`unknown`", unless the result is clear:
  E.g. "`true OR unknown = true`".

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):



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

"`EMAIL = 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, "`EMAIL = NULL`" works after the command "`SET ANSI_NULLS OFF`" (then a two-valued logic is used).

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.

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, Access, and MySQL "`NULL`" is handled like a normal term (with arbitrary type).