Part 9: 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 Publishing, 1995.
- Date/Darwen: A Guide to the SQL Standard, Fourth Edition, Addison-Wesley, 1997.
- van der Lans: SQL, Der ISO-Standard (in German, there is an English version), Hanser, 1990.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
- Don 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.
  Aggregations, GROUP BY, HAVING as well as
  NOT EXISTS / NOT IN are treated in Part 10.
- 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).
- evaluate the syntactical correctness of a given query.
- evaluate the portability of certain constructs.

Overview

1. General Remarks about SQL
2. Lexical Syntax
3. SELECT-FROM-WHERE, Tuple Variables
4. Terms and Conditions
5. Null Values

SQL

- Today, SQL is the only database language for relational database managagement systems (industry standard).
  RDBMS today must offer an SQL interface.
  There might be other interfaces in addition.
- SQL is used for:
  - Interactive "ad-hoc" commands and
  - application program development (embedded into other languages like C, Java, HTML, PL/SQL).
- SQL is based on tuple calculus.
  But includes elements from relational algebra, too.
  It tries to be relatively near to natural language.

History

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

  SEQUEL stands for "Structured English Query Language". Some people pronounce SQL this way. Others use "ess-cue-ell". The name was changed for legal reasons.
- Codd was also in San Jose when he invented the relational model in 1970.
- SQL was the language of System/R (IBM, 1976/77).

  System/R was a very influential research prototype.
- First commercial systems supporting SQL were Oracle (1979) and IBM SQL/DS (1981).

Standards

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

  This was very late as there were already several SQL systems on the market. The standard was the "smallest common denominator", i.e. only the common features of the existing implementations became part of the standard.
- Extension for foreign keys etc. in 1989 (SQL-1, SQL-89).

  All commercial implementations today support this standard, but have significant extensions of their own.
- Major Extension: SQL-2 or SQL-92 (1992).

  Upward compatible to SQL-1. The standard defines three levels: "entry", "intermediate", "full". Oracle 8.0 and SQL Server 7.0 have only entry level conformance, but many extensions. SQL-92 is still the yardstick for RDBMSs.

Future

- Current Standard: SQL-99 (Preliminary Version of SQL-3).

  Until now (Dec. 2000), the volumes 1–5 and 10 of the SQL-99 standard appeared. They have together 2355 pages. The SQL-2 standard, which is not yet completely implemented, had only 587 pages.
- Some Features of SQL-3:
  - User-defined data types, type constructors like "LIST", "SET", "ROW" for structured attribute values.
  - OO-Features: Inheritance (subtables), polymorphism, encapsulation.
  - Recursive queries.
  - Triggers, Persistent Stored Modules.

Overview

1. General Remarks about SQL
2. Lexical Syntax
3. SELECT-FROM-WHERE, Tuple Variables
4. Terms and Conditions
5. Null Values

White Space and Comments

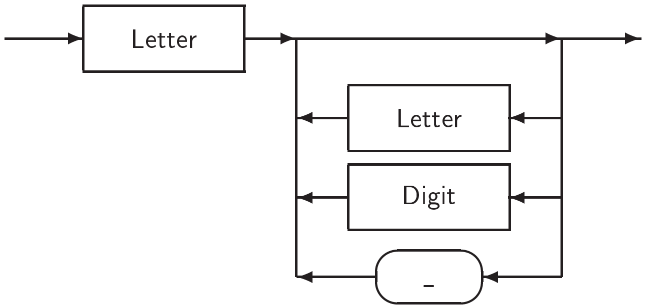
White space is allowed between words (tokens):

- 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)

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

Identifiers (1)

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



- Letter (Letter | Digit | _)*.
- E.g. `Instructor_Name`, `X27`, but not `_XYZ`, `12`, `2BE`.

Identifiers (2)

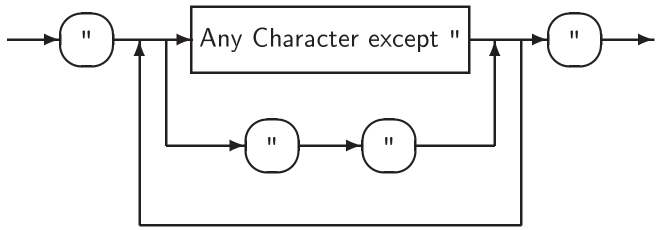
- Identifiers can have up to 18 characters (at least).

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

- 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 Appendix G.

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)

- Do not confuse delimited identifiers with string constants!
- E.g. if you write in Oracle:

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

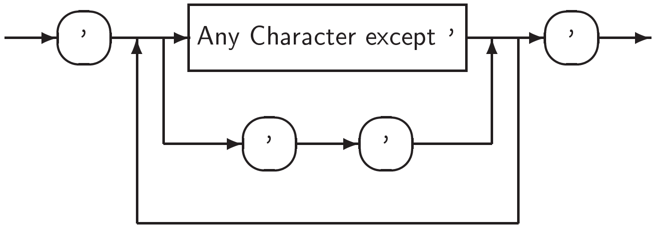
  Error: `"JONES"` is an invalid column name.
- SQL Server accepts by default `'` and `"` for string constants, and uses `[...]` for delimited identifiers.

  "`SET QUOTED_IDENTIFIER ON`" selects the SQL-92 standard behaviour (but quoted identifiers are not case sensitive).
- Quoted identifiers are normally used only to rename output columns (or if old column names conflict with words reserved in a new DBMS version).

Character String Constants (1)

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

  `SELECT 'John''s' FROM DUAL` prints `John's`.
- Instead of "constants" one also says "literals".



Character String Constants (2)

- The SQL-92 standard allows splitting strings between lines (each segment must be enclosed in single quotes).

  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.

  See above for "`SET QUOTED_IDENTIFIER ON`".

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.:

- Part 11 treats other data types, and also discusses their constants/literals.

Overview

1. General Remarks about SQL
2. Lexical Syntax
3. SELECT-FROM-WHERE, Tuple Variables
4. Terms and Conditions
5. Null Values

Example Database (1)

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

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

Example Database (2)

COURSES				
CRN	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 $A_1, \ldots, A_n$
  FROM $R_1, \ldots, R_m$
  WHERE φ

- It corresponds to the following algebra expression:

$$\pi_{A_1,\ldots,A_n}\big(\sigma_\varphi(R_1 \times \cdots \times R_m)\big)$$

- E.g., $\pi_{\texttt{INSTRUCTOR}}\big(\sigma_{\texttt{TOPIC} = \texttt{2710}}(\texttt{COURSES})\big)$ is written in SQL as:

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

Basic Query Syntax (2)

- A join must be done by including the join condition in φ.

  Forgetting the join condition is a common error.
- If no projection is needed: "*" instead of $A_1, \ldots, A_n$.
- If no selection is needed, the `WHERE`-clause can be omitted.
- E.g., to list the complete table "`COURSES`":

```
SELECT * FROM COURSES
```

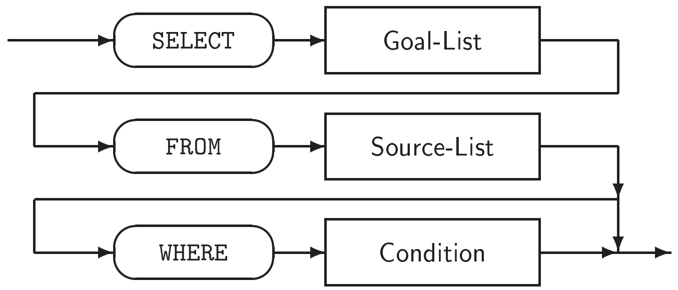
- Every SQL query must contain the keywords `SELECT` and `FROM`.

  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 (3)

SELECT-Expression (Simplified):



Tuple Variables (1)

- The `FROM` clause can be understood as declaring variables ranging 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
```

- So writing only `FROM COURSES` is understood as:

```
FROM   COURSES COURSES
```

  (The tuple variable called "`COURSES`" ranges over the rows of the table "`COURSES`".)
- So the `FROM` clause has the syntax (simplified):

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

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".

- Another way to understand "FROM COURSES C" is that C is defined as another name (alias) for the table COURSES.

  However, for self joins you need to consider copies of tables, which might be less natural than thinking about two variables ranging over the rows of the same table.

Table Names

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

```
SELECT *
FROM   sbrass.DEPT
```

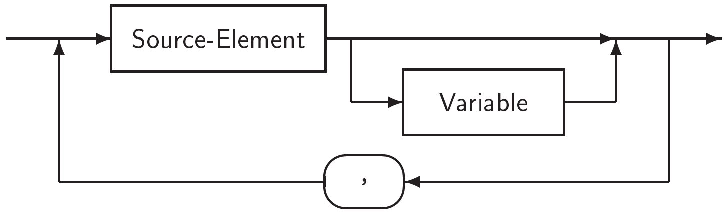
- The username is here really a name of a database schema (one database instance can manage several schemas).

  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):



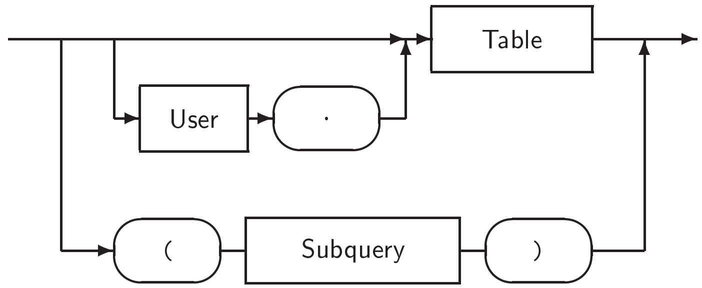
- In SQL-92, SQL Server, and DB2 (but not in Oracle 8.0) one can write AS between Source-Element and Variable. In SQL-92 and DB2 (not Oracle, SQL Server) new column names can be defined:

  `STUDENTS AS S(NO,FNAME,LLAME), ENROLLMENTS AS E(NO,C).`

- For joins in the FROM-list (not in Oracle8) see Part 10.

FROM Syntax (2)

Source-Element:



- Subqueries are treated below.
- SQL-86 did not allow subqueries in the FROM-list.

Joins (1)

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

```
SELECT ...
FROM   STUDENTS S, ENROLLMENTS E
WHERE  φ
```

- Then the tuple variable S will range over the 4 tuples in STUDENTS, and E will range over the 4 tuples in ENROLLMENTS.
- In principle, all $4 * 4 = 16$ combinations are considered:

for S **in** STUDENTS **do**
  for E **in** ENROLLMENTS **do**
    if φ **then print** ...

Joins (2)

Query Optimization:

- Depending on the condition φ, a good DBMS might use a better evaluation algorithm.
- But in order to understand the meaning of a query, it suffices to consider this simple algorithm.

Explanation via Relational Algebra:

- Simple Translation into relational algebra:

$$\pi_{\ldots}\Big(\sigma_{\varphi}\big(\rho_{\mathtt{S}}(\mathtt{STUDENTS}) \times \rho_{\mathtt{E}}(\mathtt{ENROLLMENTS})\big)\Big)$$

- The operator $\rho_{\mathsf{X}}(\mathsf{R})$ renames every attribute A of R into X.A.

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
FROM   STUDENTS S, ENROLLMENTS E
WHERE  E.CRN = 41590
```

Wrong!

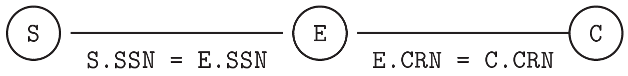
- Exercise: Write a query that prints the SSN of all students enrolled in Client-Server course.

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 varibles 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. "LAST" can be written instead of "C.LAST".
- So the following 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 "S" and "E" both have an attribute "SSN".

Attribute References (2)

- Consider this query:

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

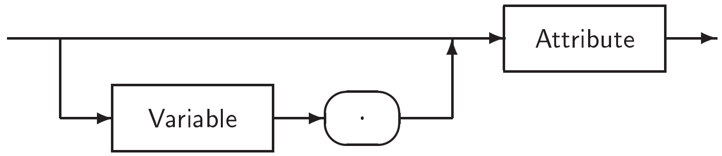
Wrong!

- 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 which one.

  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 Oracle will print the error "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 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`

- Finally, S and C are not connected.

Query Formulation (2)

- When trying to understand a relational database schema, it helps to draw a connection graph of the tables based on the foreign keys:

```
STUDENTS ---- ENROLLMENTS ---- COURSES
```

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

```
S.SSN=E.SSN AND E.CRN=C.CRN
```

- 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. there is also a `GSA` table between `STUDENTS` and `COURSES`.)

Unnecessary Joins (1)

- Do not join more tables than needed. Queries will run more slowly (most optimizers do not discover this).
- E.g. students registered for the course with `CRN 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 above query ever give a different result than this one?

```
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
```


Repetition: Tuple-wise Evaluation (1)

- Suppose the `FROM`-clause declares two tuple variables: `STUDENTS S, ENROLLMENTS E`.
- The `WHERE`-condition is evaluated with one tuple from `STUDENTS` assigned to `S`, and one from `ENROLLMENTS` assigned to `E`. This is repeated for every such combination of tuples.
- Attribute References like `S.SSN` in the `WHERE`-condition are replaced by the value of the attribute `SSN` in the current `STUDENTS` tuple. The same for references to attributes of `E`.
- After this substitution only values (constants), comparison operators (like =), and logical connectives (e.g. `AND`) remain.
- The `WHERE`-condition can then be evaluated to true or false.

Repetition: Tuple-wise Evaluation (2)

- 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, and thus "John Smith" appears in the output.

Self Joins (1)

- Task: Print a list of students enrolled in both the course with CRN 31864 and the course with CRN 41590.
- The condition in this query can never be true:

```
SELECT S.FIRST, S.LAST        Wrong!
FROM   STUDENTS S, ENROLLMENTS E
WHERE  S.SSN = E.SSN
AND    E.CRN = 41590 AND E.CRN = 31864
```

- There are three possibilities:
  - E.CRN is 41590. Then E.CRN=31864 is false.
  - E.CRN is 31864. Then E.CRN=41590 is false.
  - E.CRN is something else. Then both conditions are false.

Self Joins (2)

- So E.CRN = 41590 AND E.CRN = 31864 can never be satisfied together (at the same time, i.e. for one specific assignment of a tuple to E).
- To find students registered for both courses, two tuple variables over the relation ENROLLMENTS are required (since two distinct tuples of this relation must be referenced at the same time):

```
SELECT S.FIRST, S.LAST
FROM   STUDENTS S, ENROLLMENTS E1, ENROLLMENTS E2
WHERE  S.SSN = E1.SSN AND S.SSN = E2.SSN
AND    E1.CRN = 41590 AND E2.CRN = 31864
```


Self Joins (3)

- To formulate a query in SQL, one must not only think about which relations to access, but also whether two or more tuples from the same relation must be considered at the same time (i.e. to produce a single output tuple).
- Exercise: What is wrong with this query?

```
SELECT S.FIRST, S.LAST        Wrong!
FROM   STUDENTS S, ENROLLMENTS E,
       COURSES C1, COURSES C2
WHERE  S.SSN = E.SSN
AND    E.CRN = C1.CRN AND E.CRN = C2.CRN
AND    C1.TOPIC = 2550 AND C2.TOPIC = 2711
```

Self Joins (4)

- Different tuple variables can refer to the same tuple.
  E.g. this query would not print all students who are enrolled in at least two courses:

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

- E1 and E2 can refer to the same row in the ENROLLMENTS table. So you need to add:

```
AND   E1.CRN <> E2.CRN
```

  DISTINCT is also needed, see below.

- Exercise: Which courses have two sessions?
  (I.e. the same topic number, but different CRNs.)

Duplicate Elimination (1)

- One difference of SQL to relational algebra is that duplicates have to be explicitly eliminated in SQL.
- E.g. instructors who teach courses this term:

```
SELECT INSTRUCTOR
FROM   COURSES
```

INSTRUCTOR
Flynn, R.
Flynn, R.
Spring, M.
Brass, S.
Brass, S.

Duplicate Elimination (2)

- If you suspect that the query may contain duplicates (because your projection list does not contain a key), use the keyword "DISTINCT" after "SELECT":

```
SELECT DISTINCT INSTRUCTOR
FROM   Course
```

INSTRUCTOR
Brass, S.
Flynn, R.
Spring, M.

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

Renaming of Output Colums

- To rename the output columns
  (in SQL-92, Oracle, SQL Server, DB2, not in SQL-86):

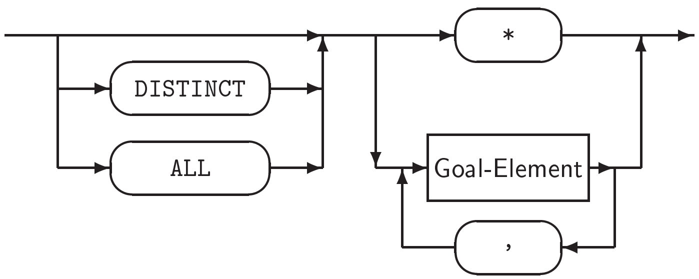
```
SELECT FIRST First_Name, LAST "Last Name"
FROM   STUDENTS
```

FIRST_NAME	Last Name
John	Smith
Ann	Miller
David	Meyer
Mary	Jones

- "AS" can be written between the term and new column name.
  In SQL-92, Oracle, SQL Server, DB2.

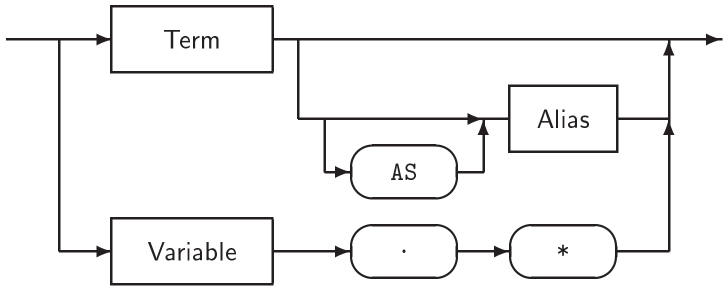
SELECT SYNTAX (1)

Goal-List (after SELECT):



SELECT Syntax (2)

Goal-Element:



"Variable.*" and "[AS] Alias" work in SQL-92, Oracle, SQL Server, and DB2. They are not contained in the old SQL-86 standard.

Overview

1. General Remarks about SQL
2. Lexical Syntax
3. SELECT-FROM-WHERE, Tuple Variables
4. Terms and Conditions
5. Null Values

Terms (1)

- A term denotes a data element.
- Terms are (for aggregations see below):
  - Attribute References,
  - Constants,
  - Composed Terms, using datatype operators like +, −, ∗, /, || (string concatenation), and datatype functions.

    SQL-86 contained only +, −, ∗, /. The operator || is contained in the SQL-92 standard, but does not work in SQL Server 7.0 (use "+"). Datatype functions (such as `COS`, `SUBSTR`, see below) are less standardized.
    SQL knows that standard precedence rules, e.g. that `A+B*C` means `A+(B*C)`. Parentheses may be used.

Terms (2)

- Terms are used in conditions, e.g. `C.SEATS > 50` contains the two terms "`C.SEATS`" and "`50`".

  The word "expression" is frequently used instead of "term".

- 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 the logical connectives "`AND`", "`OR`", "`NOT`".
- `AND` binds more strongly than `OR`, so

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

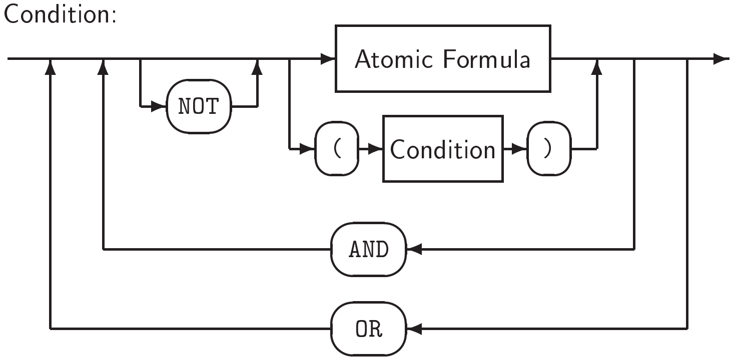
  is implicitly parenthesized as

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

- `NOT` binds most strongly, i.e. unless parentheses are used, it is applied only to the immediately following atomic formula.
- Parentheses `( ... )` can be used if necessary.

Conditions (2)

Condition:



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

Conditions (3)

- `AND` and `OR` must take complete logical conditions — something that evaluates to true or false (or null) — 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.

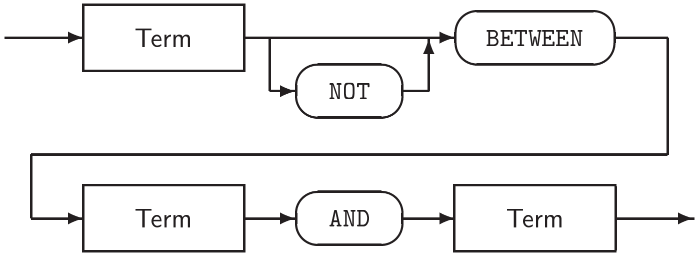
Conditions (4)

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 (5)

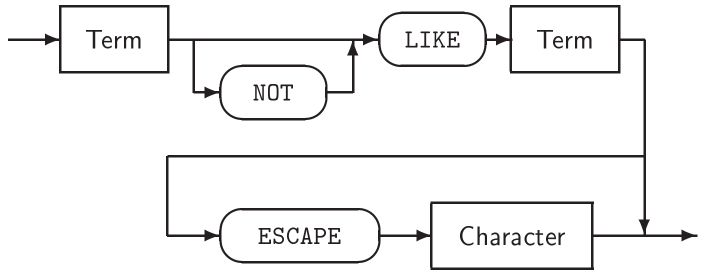
Atomic Formula (Form 2):



- `x BETWEEN y AND z` is equivalent to `x >= y AND x <= z`.
- E.g.: `SEATS BETWEEN 20 AND 40`

Conditions (6)

Atomic Formula (Form 3):



- E.g.: `INSTRUCTOR LIKE '%Brass%'`
  Is true for all instructors containing the substring `'Brass'`.

Conditions (7)

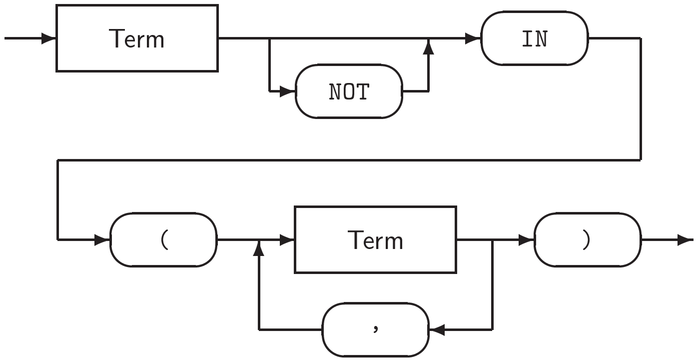
- "%" in the second argument to `LIKE` matches any sequence of arbitrary characters in the first argument.
- "_" matches any single character.
  SQL Server supports also character ranges, e.g. `[a-zA-Z]`.
- To use the characters "%" and "_" without their special meaning in the second argument, an "escape" 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 "_".
- In SQL-86 and DB2 the pattern must be a string constant.

Conditions (8)

Atomic Formula (Form 4):



Conditions (9)

`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

Null Values (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.')
```

Null Values (2)

- So the condition does not evaluate to false if `INSTRUCTOR` is null, since in this case 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: `true`, `false`, and `unknown`.
- 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.

Null Values (3)

- In SQL, a comparison with a null value always yields the truth value "`unknown`".
- Only if the `WHERE`-condition evaluates to "`true`", is a result row printed for the `SELECT`-list.
- For this reason, 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 also refuses it.

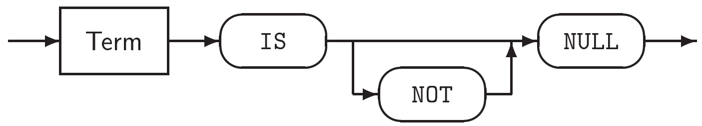
- "`AND`" / "`OR`" forward the truth value "`unknown`", unless the result is clear: E.g. "`true OR unknown = true`".

Null Values (4)

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. The result of = is neither true nor false when one or both operands are null, but instead "unknown".
- In SQL Server 7, "`INSTRUCTOR = NULL`" works after "`SET ANSI_NULLS OFF`" (then a two-valued logic is used).

Problems of Null Values

- For those accustomed to working with a two-valued logic, null values can sometimes lead to surprises: Some standard logical equivalences do not hold in SQL.
- E.g. 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.
- 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.

Expressions with Null Values

- Data type functions will normally return null if one of their arguments is null. E.g. if `A` is null, `A+B` will return 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 which otherwise require an expression.

  `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`" You need to write `TO_NUMBER(null)`. In SQL Server "`NULL`" seems to be handled like a normal expression.