Part 9: SQL I

Elmasri/Navathe:

- Chap. 8, “SQL — The Relational Database Standard”

  In this part, we treat only Section 8.2, 8.3.3, and part of 8.3.4.

I have also used the following sources:

- 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

- 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.
- You should at least know all constructs from the SQL-86 standard. You should be able to evaluate the syntactical correctness of a given query.
- You should know a bit about the SQL-92 standard, and more or less be able to evaluate the portability of certain constructs.

Overview

SQL

- SQL is today “the” 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 both:
  - Interactive “ad-hoc” commands and
  - application program development (embedded into other languages like C, Java, HTML, PL/SQL).
- SQL is based on the tuple calculus.

  But has also ingredients from relational algebra.
  SQL 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, there were already several SQL systems on the market. The standard was only 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 quite extensive extensions of their own.

- Current standard: 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.

Future

- There is a draft for an SQL-3 standard, but it has not yet been finished/finalized.

  SQL-3 products will probably appear only in 5–10 years. The draft has 1500 pages, whereas the SQL-2 standard is only one third of the size.

- Some Features of SQL-3:
  - User-defined data types, type constructors like "LIST", "SET", "ROW" for structured attribute values.
  - Supertables/subtables (inheritance).
  - Recursive queries.
  - Triggers.

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

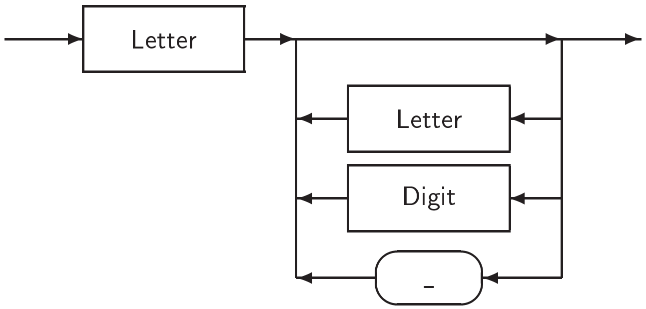
White space is allowed between words (tokens):

- 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 e.g. Pascal.

Identifiers (1)

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



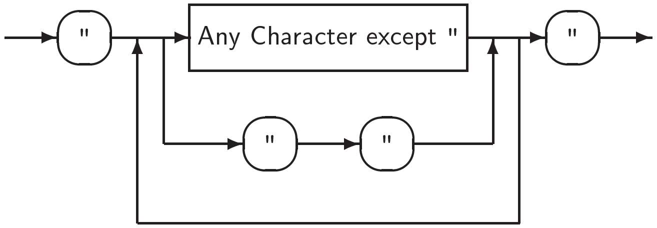
- Letter (Letter | Digit | _)*.
- E.g. `Instructor_Name`, `X27`, but not `_XYZ`, `12`, `2BE`.

Identifiers (2)

- Identifiers can be up to 18 characters long, some systems allow longer identifiers and additional characters in them.
  SQL-86: Up to 18 characters length, no "_", only uppercase.
  SQL-92: Up to 128 characters length.
  Oracle: Length ≤ 30, "$" and "#" allowed after first letter.
  SQL Server: Up to 128 characters, "_", "@", #" are allowed also as first character, "$" allowed later in ID.
  IBM DB2: Up to 18 characters (usernames etc. up to 8).
- Identifiers (and keywords) are not case sensitive.
  SQL*Plus converts all letters outside quotes to uppercase.
- Names must be different from all reserved words.
  And 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:

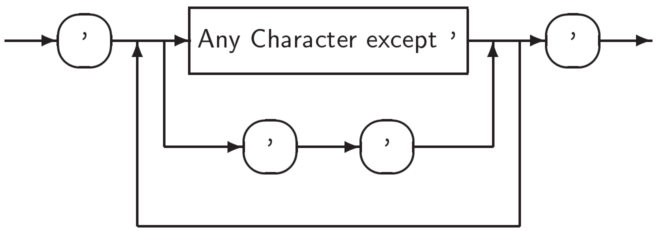
  `SELECT * FROM EMP WHERE ENAME = "JONES"`

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

  “`SET QUOTED_IDENTIFIER ON`” selects the SQL-92 standard behaviour (but quoted identifiers are still 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'`.



Character String Constants (2)

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

  Neither Oracle nor SQL Server support this. However, you can combine strings 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, then there must be a digit before it or after it.
- E.g.: `1`, `+2.`, `-34.5`, `-.67E-8`
- Note that numbers are not inclosed in quotes.

Date, Money, Binary Data etc.:

- Constants for other data types are discussed together with the data types in Part 11.

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)

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

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

Example Database (2)

COURSE				
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

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

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

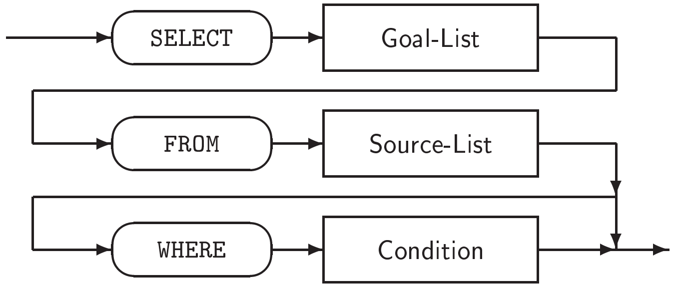
Basic Query Syntax (2)

- A join has to 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 missing.
- But the keywords `SELECT` and `FROM` must always be present.
  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 missing, e.g. `SELECT 1+1`.
- E.g., to list the complete table “`COURSE`”:

```
SELECT * FROM COURSE
```


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   COURSE C
WHERE  C.TOPIC = 2710
```

- This can be executed as:

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

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

Tuple Variables (2)

- A tuple variable is always created: If you do not give it a name, it will have the name of the relation:

```
SELECT COURSE.INSTRUCTOR
FROM   COURSE
WHERE  COURSE.TOPIC = 2710
```

- So if you write only `FROM COURSE`, this is understood as:

```
FROM   COURSE COURSE
```

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

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

Tuple Variables (3)

- Note that if you declared explicitly a tuple variable name,

  `FROM COURSE C`

  it is an error to try to access

  `COURSE.INSTRUCTOR`

  The tuple variable is now called “C”, not “COURSE”.
- Another way to understand “FROM COURSE C” is that C is defined as another name (alias) for the table COURSE.

  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

- If other users have given you read permission on their tables, you can refer to them in the FROM-list:

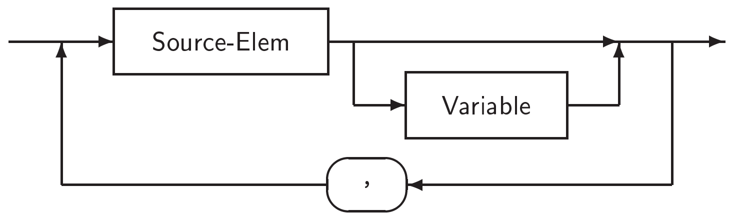
```
SELECT *
FROM   sbrass.DEPT
```

- 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):

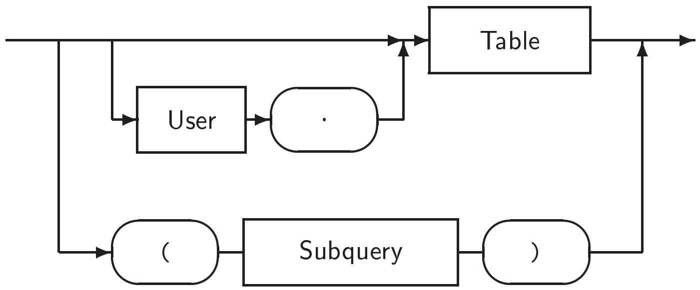


- In SQL-92, SQL Server, and DB2 (but not in Oracle 8.0) one can write AS between Source-Elem and Variable. In SQL-92 and DB2 (not Oracle, SQL Server) one can also define new column names:

  `STUDENT AS S(NO,FNAME,LLAME), ENROLLENT AS E(NO,C).`
- For joins in the FROM-list (not in Oracle8) see Part 10.

FROM Syntax (2)

Source-Elem:



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

Joins (1)

- You can declare more than one tuple variable in the `FROM` clause:

```
SELECT ...
FROM   STUDENT S, ENROLLMENT E
WHERE  φ
```

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

for `S` **in** `STUDENT` **do**
    for `E` **in** `ENROLLMENT` **do**
        if φ **then print** ...

Joins (2)

Query Optimization:

- Of course, 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:

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

- 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   STUDENT S, ENROLLMENT 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   STUDENT S, ENROLLMENT E
WHERE  E.CRN = 41590
```

Wrong!

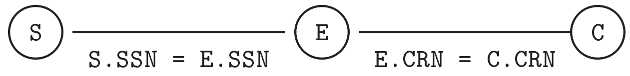
- Exercise: Write a query that prints the SSN of all students enrolled in Client-Server course.

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   STUDENT S, ENROLLMENT E, COURSE 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)

- You can access attributes in the Form

  `Variable.Attribute`

- But if only one variable has this attribute, you can leave out the variable name, e.g. write "LAST" instead of "C.LAST".
- E.g., the following query is legal:

```
SELECT CRN
FROM   STUDENT S, ENROLLMENT 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:

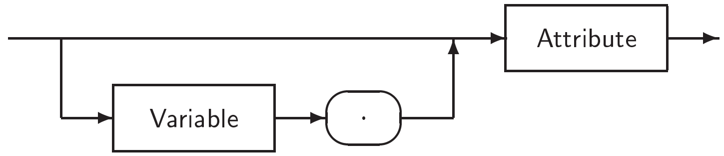
```
SELECT CRN, TITLE          Wrong!
FROM   STUDENT S, ENROLLMENT E, COURSE C
WHERE  S.SSN = E.SSN AND E.CRN = C.CRN
AND    FIRST = 'Ann' AND LAST = 'Miller'
```

- It is pedantic, but you must specify in the SELECT-clause whether you want C.CRN or E.CRN, 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", you must specify which tuple variable you mean, or Oracle will print the error "ORA-00918: column ambiguously defined".

Attribute References (3)

Attr-Ref:



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

Query Formulation (1)

- Suppose your task is to write an SQL query which prints the titles of all courses which `Ann Miller` attends.
- First you must understand that `Ann Miller` is a student, thus you need a tuple variable `S` over `STUDENT` and the condition `S.FIRST='Ann' AND S.LAST='Miller'`.
- Then you see that titles of courses are requested, so you also need a tuple variable `C` over `COURSE` and will have

  `SELECT C.TITLE`

- Finally, you see that `S` and `C` are not connected.

Query Formulation (2)

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

  STUDENT — ENROLLMENT — COURSE

- From this we see that we need a tuple variable `E` over `ENROLLMENT` and the condition

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

- Note that it is not always so simple. The connection graph may contain cycles, which makes the selection of the right path more difficult and error-prone.

Unnecessary Joins (1)

- Do not join more tables than needed. Queries will run slower.
  And you will lose points in the exam.
- E.g. students registered for the course with `CRN 41590`:

```
SELECT S.FIRST, S.LAST
FROM   STUDENT S, ENROLLMENT E, COURSE C
WHERE  S.SSN = E.SSN AND E.CRN = C.CRN
AND    C.CRN = 41590
```

- Can this ever give a different result than this query?

```
SELECT S.FIRST, S.LAST
FROM   STUDENT S, ENROLLMENT 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   STUDENT S, ENROLLMENT E, COURSE C
WHERE  S.SSN = E.SSN AND E.CRN = 41590
```

- Is there any difference between these two queries?

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

SELECT DISTINCT S.FIRST, S.LAST
FROM   STUDENT S, ENROLLMENT E
WHERE  S.SSN = E.SSN
```


Repetition: Tuple-wise Evaluation (1)

- Suppose the `FROM`-clause declares two tuple variables: `STUDENT S, ENROLLMENT E`.
- The `WHERE`-condition is evaluated with one tuple from `STUDENT` assigned to `S`, and one from `ENROLLMENT` 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 `STUDENT` 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)

- Suppose we need 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   STUDENT S, ENROLLMENT E
WHERE  S.SSN = E.SSN
AND    E.CRN = 41590 AND E.CRN = 31864
```

- There are only 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).
- If you want students which are registered for both courses, you need two tuple variables over the relation ENROLLMENT (since you need to refer to two distinct tuples of this relation at the same time):

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


Self Joins (3)

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

```
SELECT S.FIRST, S.LAST        Wrong!
FROM   STUDENT S, ENROLLMENT E,
       COURSE C1, COURSE 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)

- Note that 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   STUDENT S, ENROLLMENT E1, ENROLLMENT E2
WHERE  S.SSN = E1.SSN AND S.SSN = E2.SSN
```

- E1 and E2 can refer to the same row in the ENROLLMENT 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   COURSE
```

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.

- If you want to make very clear that there are duplicates and you want them, you can write “SELECT ALL”. However, “ALL” is the default.

Renaming of Output Colums

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

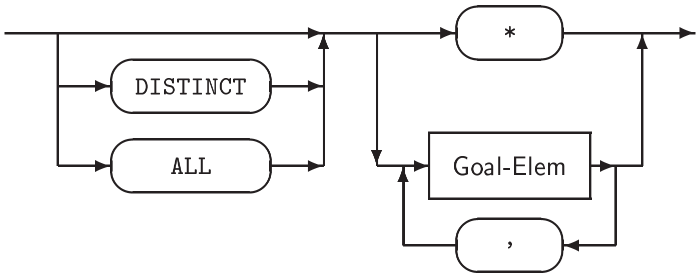
```
SELECT FIRST First_Name, LAST "Last Name"
FROM   STUDENT
```

FIRST_NAME	Last Name
John	Smith
Ann	Miller
David	Meyer
Mary	Jones

- You can write “AS” between term and new column name.
  In SQL-92, Oracle, SQL Server, DB2.

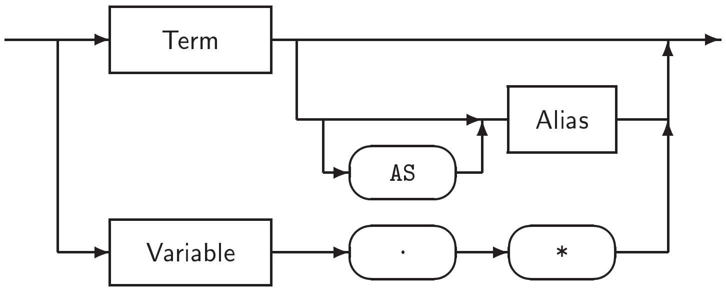
SELECT SYNTAX (1)

Goal-List (after SELECT):



SELECT Syntax (2)

Goal-Elem:



“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)`. You may use parentheses.

Terms (2)

- You use terms in conditions, e.g. `C.SEATS > 50` contains the two terms "`C.SEATS`" and "`50`".

  Often, one says also "expression" instead of "term".

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

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

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 stronger than `OR`, so

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

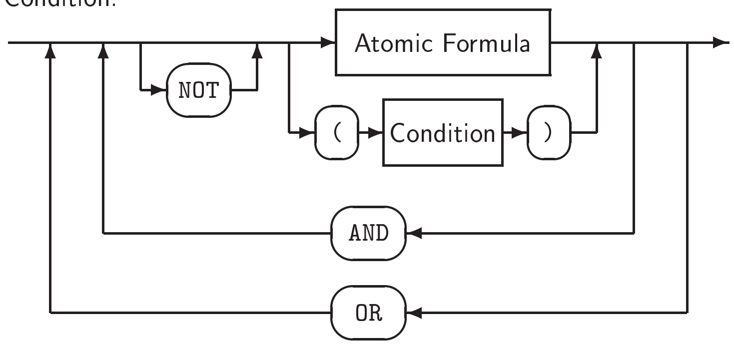
  is implicitly parenthesized as

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

- `NOT` binds strongest, i.e. unless you use parentheses, it is applied only to the immediately following atomic formula.
- Use parentheses (...) if necessary.

Conditions (2)

Condition:



- SQL-92 allows one to write "`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   STUDENT S, ENROLLMENT 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.

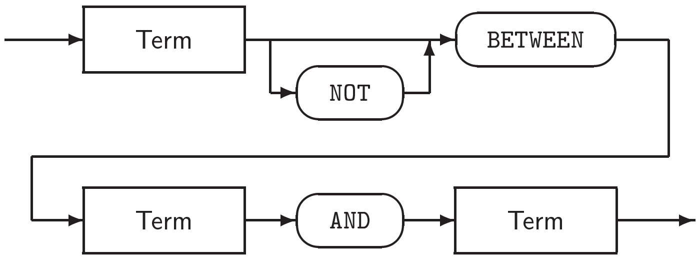
Conditions (4)

Atomic Formula (Form 1):

- Comparison operators are: =, <>, <, >, <=, >=.
- 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)

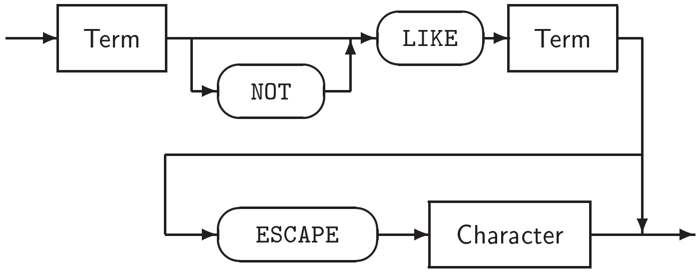
Atomic Formula (Form 2):



- `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):



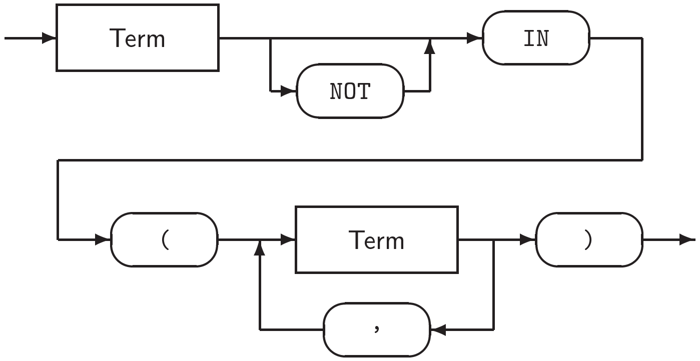
- E.g.: `INSTRUCTOR LIKE '%Brass%'`
  Is true for all instructors containing the substring `'Brass'`.

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]`.
- If you need the characters "%" and "_" without their special meaning in the second argument, you must first declare an "escape" character. This character removes the special meaning of the following character.
- E.g.: `ProcName LIKE '\_%' ESCAPE '\'`
  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 completely 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 might be better style to use `OR` if you don't have an enumeration of constants.

Overview

Null Values (1)

- Consider the following query:

```
SELECT CRN, TITLE
FROM   COURSE
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   COURSE
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 not existing, 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`”, a result row is printed for the SELECT-list.
- For this reason, the following query gives the empty result:

```
SELECT CRN, TITLE
FROM   COURSE
WHERE  INSTRUCTOR = null
```

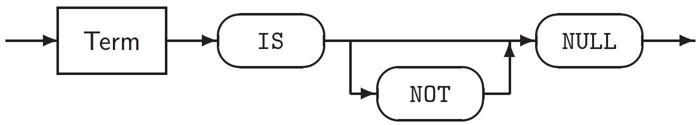
  Actually, the query is illegal in SQL-92, and DB2 also refuses it.
- “AND” / “OR” forward the truth value “`unknown`”, unless the result is clear: E.g. “`true OR unknown = true`”.

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

- Since we are used 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. if we count courses taught by Brass and courses not taught by Brass, we would 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` is by itself 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.