IT 775
Database Technology

SQL

DML

SQL: Relational Model
DML

Data Manipulation Language (DML) - is a language used for adding (inserting), deleting, and modifying (updating) data in a database. A DML is often a sublanguage of a broader database language such as SQL.

SQL Commands

https://mariadb.com/kb/en/basic-sql-statements

DML

- SELECT is used when you want to read (or select) your data.
- INSERT is used when you want to add (or insert) new data.
- UPDATE is used when you want to change (or update) existing data.
- DELETE is used when you want to remove (or delete) existing data.

SQL: Relational Model DML

INSERT, DELETE, UPDATE modify table contents

SELECT statement
- retrieves data
- implements all RA & RC operations–relationally complete

IT 775
Database Technology

SQL-DML

Select Statement

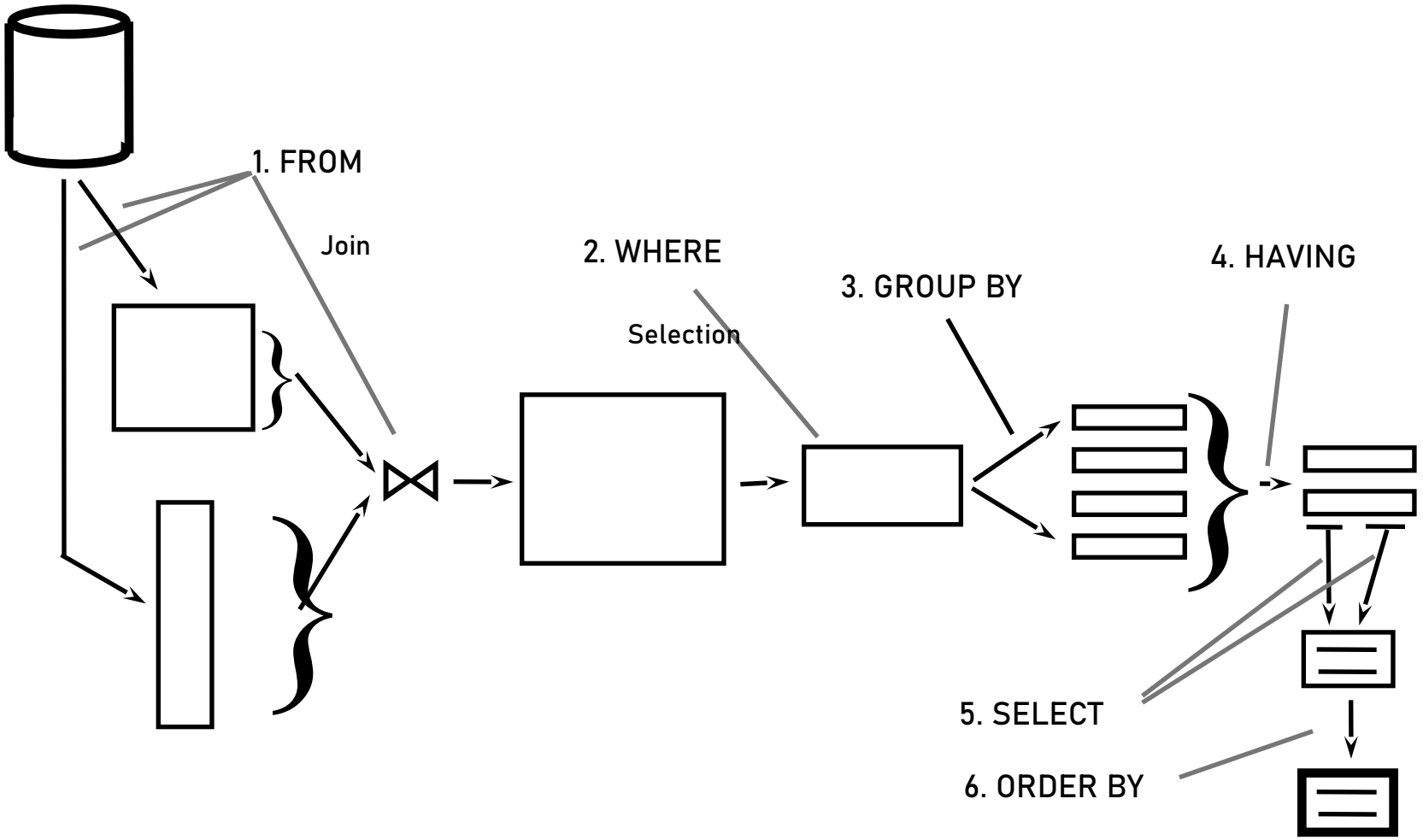
SELECT Syntax

SELECT <attribute-list>
FROM (<table-list> | <join-expression>)
[WHERE <conditional-expression>]
[GROUP BY <attribute-list>]
[HAVING <conditional-expression>]
[ORDER BY <attribute-list>]

SQL literals shown all caps but SQL is case insensitive
[ ] clauses are optional
< > sections are user specified

SELECT –– specifies attributes in the result (including calculated attr.)
FROM –– specifies the source relations for the data
WHERE –– specifies conditions retrieved data must meet
GROUP BY –– specifies grouping for statistical operators
HAVING –– specifies criteria for groups
ORDER BY –– specifies ordering of result

Select Processing –– SQL–92



Retrieval

Retrieve entire table

absent or empty WHERE clause means all rows qualify

```
SELECT name, id, major, gpa, advisor FROM student
```

asterisk is shorthand for all columns–equivalent to 1st

```
SELECT * FROM student
```

Retrieve some attributes

```
SELECT   major, gpa FROM student
```

duplicates can result when retrieval list excludes unique attribute

- e.g., student.id
- duplicates may arise in result (major, gpa)

most DBMS do not automatically eliminate duplicates

- specify DISTINCT to force it

```
SELECT DISTINCT name, major, gpa FROM student
```

duplicate removal takes time and is often unnecessary when

- subquery result is an unseen intermediate result table
- number of likely duplicates are too small to affect performance

WHERE Clause Returns Selected Rows

```
SELECT name, major, gpa FROM student WHERE advisor = 'wise';
```

comparison expressions can include
attribute names from relation(s) in the FROM clause and/or literals (numbers, strings, time, date) separated by relational operators

=, <> or !=, <, >, <=, >=

LIKE	implements pattern match
BETWEEN	tests upper & lower bounds in one operation
IN	tests set membership
IS NULL	tests for NULL value (more about NULL later)
IS NOT NULL	tests for absence of NULL
EXISTS	Boolean test (more later)

logical expressions
comparison expressions separated by Boolean operators AND, OR, NOT

Literals

Numbers are written normally

periods, sign optional, no comma

compare numerically

Strings in single quotes –– compare alphabetically

- ← 'Joe College' = 'Joe College'
- ←'joe college' > 'Joe College' (strings are case sensitive)
- ← 'Joe College' > 'Joe Collage'
- ← 'Joe''s College' or 'Joe\'s College'

\n \b \t … also allowed in strings

Dates and times compare chronologically

Sample SP Queries

– Comparison of Attribute & Literal

```
SELECT  *                  SELECT  *
  FROM  student              FROM  student
WHERE   gpa >= 3.0;        WHERE   major = 'cs';
```

– Comparison of Attributes

```
SELECT  *
  FROM  student
WHERE   name = advisor;
```

Compound Conditions

usual precedence NOT AND OR applies

parentheses override as usual

```
SELECT  *                  SELECT  name
  FROM  student              FROM  student
WHERE   major = 'cs'       WHERE   gpa >= 2.0
  AND   gpa > 3;             AND   gpa < 3.0;
```

```
SELECT  name
  FROM  student
WHERE   NOT ( gpa < 3.0 )
  AND   ( major = 'cs'
    OR    major = 'math' );
```

LIKE
String Pattern Matching

Special pattern characters

% represents arbitrary string of 0 or more characters

_ represents arbitrary single character

\ escape character

- treats next character as a char literal
- allows % and _ to be regular character literals

```
SELECT * FROM student WHERE name LIKE 'Joe %';

SELECT * FROM student WHERE major LIKE '%science %';

SELECT * FROM student WHERE name LIKE '%smythe\_jones%';

SELECT name FROM student WHERE name NOT LIKE 'Joe %';
```

BETWEEN performs an interval test

Equivalent to a compound AND expression with <= & >= applied to the same attribute

```
SELECT      name
  FROM      student
WHERE       gpa BETWEEN 2.0 AND 3.0;
```

- NOT BETWEEN -- tests for values outside an interval equivalent to a compound OR expression with > | <

```
SELECT      name
  FROM      student
WHERE       gpa NOT BETWEEN 2.0 AND 3.0;
```

IN – tests for membership in a list of elements

Equivalent to sequence of ORed tests

```
SELECT    name
  FROM    student
WHERE     major IN ( 'cs', 'math', 'ee' );
```

- NOT + IN -- excludes rows that satisfy membership test; equivalent to negated sequence of ANDed tests

```
SELECT    name
  FROM    student
WHERE     major NOT IN ( 'cs', 'math', 'ee' );
```

NULL is Not a Value

NULL represents the absence of a value

- NULLs behave non-intuitively in some queries

Testing NULL with comparison operators always fails

These two queries retrieve nothing

```
SELECT name FROM student WHERE major = NULL
```

fails even for students with NULL major field

```
SELECT name FROM student WHERE major <> NULL
```

the second fails even for students with non NULL major

These queries together should retrieve all students, but they miss students with NULL gpas

```
SELECT name FROM student WHERE gpa <= 3.0
SELECT name FROM student WHERE gpa > 3.0
```

IS [NOT] NULL Predicates

NULL is special, SQL has special test predicates for it

IS NULL retrieves rows with NULL in the tested attribute

IS NOT NULL excludes rows with NULL in the tested attribute

Retrieve undeclared students

```
SELECT name FROM student WHERE major IS NULL;
```

Retrieve students who have declared a major

```
SELECT name FROM student WHERE major IS NOT NULL;
```

Retrieve all students

```
SELECT name
  FROM student
WHERE major IS NULL
      OR major IS NOT NULL;
```

Ordering Result ORDER BY

```
SELECT name, major, gpa FROM student
 ORDER BY   gpa [DESC];
```

ascending [ASC] is the default ordering

Mixed ordering is possible

```
SELECT name, major, gpa FROM student
 ORDER BY major, gpa DESC;
```

Specifies:

- students listed by major in alphabetical order
- within each major, students appear in descending gpa order
- major is called the major sort field
- gpa is called the minor sort field

Statistical Queries

perform statistical summaries of a table

- aggregate/summarize values in a column of qualifying rows
- form: fn(columnName) where fn can be:
  - count any attribute
  - max any ordinal attribute
  - min any ordinal attribute
  - sum any numeric attribute
  - average any numeric attribute
- JOIN & WHERE clauses evaluate before statistical functions
- if no rows qualify, then
  - COUNT returns 0
  - SUM AVG MAX MIN return NULL

Count & Maximum/Minimum

enumerates the number of rows in a table
or subtable of selected tuples

count(*) is valid
doesn't look at the values in any column
it returns the number of qualifying tuples

count(attr) returns the number of rows with non–NULL values for attr

count(DISTINCT attr) returns the number of different values for attr (not how many times each value occurs)

max(attr)/min(attr) returns the largest/smallest non-value in a ordinal column of a table (or subtable of selected tuples)

returns NULL if no tuples qualify

Sum

sum(attr) totals values in the numeric attribute column of a table (or subtable of selected tuples)

```
SELECT SUM( gpa ) FROM students
```

includes duplicate values (unless DISTINCT specified)

sum(DISTINCT attr) totals distinct values of attr

NULLs are not included in the summation

if no tuples qualify, sum returns NULL

Average

avg(attr) computes arithmetic mean of numeric column of a table (or subtable) of qualifying tuples

```
SELECT AVG( gpa ) FROM students
SELECT AVG( snbr ) FROM students
```

average includes duplicates by default

computed as

sum of non-null values / count of non-null values

returns NULL if no tuples qualify

avg(DISTINCT attr) excludes duplicate values

```
SELECT AVG( DISTINCT gpa ) FROM students
```

Count & Sum Examples

- count the number of students

```
SELECT count( snbr ) FROM student;
SELECT count( * ) FROM student;
```

- count the number of cs majors

```
SELECT count( snbr ) FROM student WHERE major = 'cs';
SELECT count( * ) FROM student WHERE major = 'cs';
```

DISTINCT unneeded because snbr is unique

- total credits a student has earned

```
SELECT sum( credits )
  FROM student NATURAL JOIN transcript NATURAL JOIN course
WHERE name = 'Joe College';
```

- compute gpa for a student

```
SELECT sum( cr * grade ) / sum( cr ) AS newgpa
  FROM student NATURAL JOIN transcript NATURAL JOIN course
WHERE name = 'Joe College';
```

Average & Max/Min Examples

- compute average gpa of all students

  SELECT avg(gpa) FROM students;

- compute average gpa of cs students

  SELECT avg(gpa) FROM students WHERE major = 'cs';

  note: duplicates are significant here –– want to keep them therefore DISTINCT is not appropriate

- find highest gpa among cs students

  SELECT max(gpa) FROM students WHERE major = 'cs';

- find lowest (lexically smallest) major e.g., accounting

  SELECT min(major) FROM students;

Grouping in Statistical Queries

- SQL can
  - group rows with matching values in designated columns
  - apply statistical functions separately to each group
  - select groups that meet specified qualifications & reject others
- GROUP BY clause forms groups
- HAVING clause restricts groups
- rows with NULL in the grouping column form a single separate group
- SELECT fields restricted to
  - the summary values
  - the grouping column values

Grouping Example

- list each major, its count, and its average gpa in descending gpa order

```
SELECT major, count( * ), avg( gpa )
  FROM student
GROUP BY major
ORDER BY avg( gpa ) DESC;
```

- attribute participation
  - major is the grouping attribute
    - each distinct major forms a separate group
  - gpa is the summarization attribute
    - each group is summarized separately
  - selecting any other attribute (e.g. sname) nonsensical

Having Example

- list depts with at least 5 majors whose students' gpas average at least 3.0

```
SELECT major, count( * ), avg( gpa )
  FROM student
GROUP BY          major
HAVING avg( gpa ) >= 3.0 AND count( * ) >= 5;
```

- this query
  - groups students by major
  - counts the number of students by major
  - computes the average gpa by major
  - discards majors averaging < 3.0 or majors with < 5 students
- result

major	count(*)	avg(gpa)
cs	45	2.85

Statistical Subquery Scalar-Valued

- students with above average gpa values
  - stat subquery version

```
SELECT sname, gpa FROM student
WHERE gpa > ( SELECT avg( gpa ) FROM student );
```

- students with above average in-major gpa values

```
SELECT sname, major, gpa FROM student S
WHERE gpa >
  ( SELECT avg( gpa ) FROM student WHERE major = S.major );
```

note: horrible efficiency –– this correlated subquery reevaluates the average gpa of a major for each student in that major

(unless optimizer rescues us)

Statistical Subquery Set Valued

- compute students' in–major gpas

```
SELECT sname, student.snbr,
          SUM( cr * grade ) / sum( cr ) AS majgpa
  FROM ( student JOIN transcript
           ON student.snbr = transcript.snbr AND major = dept
         )
         NATURAL JOIN course
GROUP BY sname, student.snbr;
```

- department with highest average gpa
  - multi-valued subquery

```
SELECT major, avg( gpa ) FROM student
 GROUP BY major
HAVING avg( gpa ) >= ALL
  ( SELECT AVG(gpa) FROM student
    GROUP BY major
  );
```

Numeric Functions

ROUND(nbr, length)

length > 0: round to length fractional digits

- ROUND(123.499, 1) rounds to tenths position = 123.4

length < 0: round to length whole nbr position

- ROUND(123.499, -1) rounds to tens position = 120

ABS(nbr)

CEILING(nbr)

FLOOR(nbr)

SQRT(float-nbr)

RAND(int) random nbr in (0, 1)

int is seed, omit it to return same number each time

Numeric Data Types

- integer

Type	Storage (Bytes)	Minimum Value (Signed/Unsigned)	Maximum Value Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

Numeric Data Types

- decimal
  - decimal[(p [, s])] p digits precision, s of them fractional (scale)
  - numeric[(p [, s])] p digits precision, s of them fractional (scale)
    - » same as decimal
- real (approximate values)
  - float 24, single precision
  - double 53, double precision

  - NON-STANDARD:
  - float(p, s) p digits precision, s of them fractional (scale)
  - double(p, s) p digits precision, s of them fractional (scale)

- bit(m) m digits binary

String Data Types

- fixed length,
  - char[acter][(n)] 1 – 8000 chars, 1 is the default
    n bytes allocated
- variable length
  - varchar[acter][(n)] 1 – 8000 chars, 1 is the default
    bytes allocated to hold actual string

Value	CHAR(4)	Storage Required	VARCHAR (4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Temporal Data Types

- date — date only, no time of day

  3 bytes: 0001-01-01 – 9999-12-31
  constants ' YYYY-MM-DD'

- time — time only, no date

  '-838:59:59' to '838:59:59‘ shows time of day or elapsed time

  constants ' HH:MM:SS' ‘HHH:MM:SS'

- datetime — date and time 6 – 8 bytes

  '1000-01-01 00:00:00' to '9999-12-31 23:59:59‘
  constants YYYY-MM-DD HH:MM:SS'
  ANSI standard name is TIMESTAMP

Date Comparison

date types compare with usual comparison operators

all forms of literals are equivalent

for date comparisons of different types inferior type cast to superior type

Examples:

'10-01-2010' < '10-02-2010'

'10-01-2010' < '10-01-2011'

'01-oct-2010' = '10-01-2010'

'10-01-2010' < '10-01-2010 01:01:01'

- date promoted to datetime for comparison

Data Type Conversion

- implicit conversion
  - SQL hides from end user
    - assigning value to column –
      - converts value to column type
    - expression with differently typed arguments
      - result has higher type
  - implicit conversions
    - from lower precedence type to higher
    - within type
      - each precision/length is a different type
      - more precision/length is higher
- explicit conversion
  - when implicit conversion isn't available
  - CAST and CONVERT do explicit conversion

highest
---
datetime
(timestamp)
smalldatetime
date time
float
real
decimal
int
smallint
tinyint
bit
varchar
char
---
lowest

CAST, CONVERT and String Functions

cast is SQL standard type

CAST(expr AS type)

convert is unique to MySQL, not portable – avoid

LTRIM(string), RTRIM(string)

returns string with leading/trailing spaces removed
ANSI SQL specifies TRIM which does both (in MySQL)

SUBSTR(string, start, length)

returns portion from start position for length characters

LOWER(string), UPPER(string)

modifies case of characters in the string

Date and Time Functions

GETDATE( ) returns today's date

SYSDATETIME( ) reurns current local date and time

SYSDATETIMEOFFSET( ) returns date, time, & offset

DAY(date) returns day of month as int

MONTH(date) returns month as int

YEAR(date) returns year as 4-digit int

DATEPART(datepart, date)

DATEPART(month, '2010-10-01') => 10

DATENAME(month, '2010-10-01') => october

day, month, year, hour, minute, second, quarter, dayofyear, week, weekday, millisecond, microsecond, nanosecond, tzoffset

DATEADD(datepart, number, date)

DATEADD(hour, 2, '2013-10-01 13:05:44') => '2013-10-01 15:05:44'

DATEDIFF(datepart, startdate, enddate)

DATEDIFF(hour, 2, '2013-10-01 13:05:44', '2013-10-01 15:05:44') => 2

searching for date values

SELECT … WHERE MONTH(date) = 9 AND YEAR(date) = 2013