Simple SQL Queries

SQL – a Query Language

- Declarative query language:
  - User states which information is required, not how to reach it.
- The query optimizer translates the queries to an internal procedural language of the DBMS.
- The programmer focuses on the readability of the query, not on its execution efficiency.
- This is the most qualifying aspect of relational database systems.

SQL Queries

- SQL queries have a shape like `select-from-where`
- Syntax:

```
select [distinct] AttrExpr {, AttrExpr}
from Table {, Table}
[ where Condition ]
```

- Three clauses:
  - `select` / target list
  - `from`
  - `where`
- The query evaluates a Cartesian product of the tables whose names are in the `from` clause, and returns the tuples which fulfill the `where` clause, only. Returned attributes are those of the `select` clause.
- The `where` clause is evaluated tuple by tuple.
- The clause `Distinct` removes duplicate tuples.

Algebraic Interpretation of an SQL Query

- A generic query:

```
select T1.Attribute1, …, Th.Attributeth
from Table1, …, Tablen
where Condition
```

Corresponds to the relational agebra query:

$$\pi_{T1.Attribute1,Th.Attributeth}(\sigma_{Condition}(Table1 \times \ldots \times Tablen))$$

Example:
Student Examinations

Student

ID	NAME	CITY	DEPT
123	Carlo	Bologna	CS
415	Paola	Torino	CS
702	Antonio	Roma	Log

Exam

ID	CODE	DATE	MARK
123	1	7-9-03	10
123	2	8-1-03	8
702	2	7-9-03	5

Course

CODE	NAME	TEACHER
1	Maths	Barozzi
2	Databases	Meo

(Very Simple) Query

```
select  *
from Student
```

ID	NAME	CITY	DEPT
123	Carlo	Bologna	CS
415	Paola	Torino	CS
702	Antonio	Roma	Log

Simple Query

Student

Id	Name	City	Dept

```
select Name
from Student
where Dept = ’Log’
```

Algebraic interpetation (disregarding the duplicates)

$$\Pi_{\text{Name}}\ \sigma_{\text{Dept='Log'}}\ \text{Student}$$

Syntax for the `select` clause

```
select  *
select  Name, City
select distinct  City
select  City as Residence
select CadastralIncome * 0.05
          as IMUtax
select  sum(Salary)
```

Syntax for the `from` clause

```
from   Student
from   Student as X
from   Student, Exam
from   Student join Exam
     on Student.Id=Exam.Id
```

Syntax for the `where` clause

- Boolean expression of simple predicates (just like in RA).
- Some additional predicates:
  - **between:**

    ```
    Date between 1-1-90 and 31-12-99
    ```

  - **like:**

    ```
    Dept like ’Lo%’
    PlateNumber like ’MI_777_8%'
    ```

Conjunction of Predicate

- Find the students of CS from Bologna

```
select *
from Student
where DEPT = “CS”’ and
      City = “Bologna”
```

- Result:

ID	NAME	CITY	DEPT
123	Carlo	Bologna	CS

Disjunction of Predicates

- Find the students from Bologna or from Torino:

```
select *
from Student
where City = “Bologna” or
      City = “Torino”
```

- Result:

ID	NAME	CITY	DEPT
123	Carlo	Bologna	CS
415	Paola	Torino	CS

Boolean Expressions

- Find the students from Rome attending the course of CS or of Log:

```
select *
from Student
where City = ’Roma’ and
     (Dept = ’CS’ or Dept = ’Log’)
```

- Result:

Id	Name	City	Dept
702	Antonio	Roma	Log

The Operator `like`

- Find the students whose name has an ‘a’ as second char and ends by ‘o’ :

```
select *
from Student
where Name like ’_a%o’
```

- Result:

Id	Name	City	Dept
123	Carlo	Bologna	Inf

Duplicates

- RA queries do NOT include duplicates.
- SQL may return tables with identical rows.
- Duplicates can be explicitly removed by the command `distinct`.

Duplicates

```
select
distinct Dept
from Student
```

Dept
CS
Log

```
select Dept
from Student
```

Dept
CS
CS
Log

Null values

- Null values may depict different situations:
  - the values does not apply;
  - the value applies but remains unknown;
  - the value may/may not apply.
- SQL-89 uses a two-value logic:
  - a comparison with *null* returns FALSE
- SQL-2 uses a three-value logic:
  - a comparison with *null* returns UNKNOWN
- To check if an attribute has the null value:

  Attribute **`is`** [**`not`**] **`null`**

Predicates and Null Values

- **three value logic (T,F,U)**

T and U = U
T or U = T

F and U = F
F or U = U

U and U = U
U or U = U
not U = U

- **P =**

```
(City is not null) and
(Dept like 'C%')
```

City	Dept
Milano	CS
Milano	NULL
NULL	CS
Milano	Log

P	Selected tuple
T	yes
U	no
F	no
F	no

Queries Over Null Value

```
select *
from Student
where City  is  [not]  null
```

if `City` has the value *null*

`(City = ’Milano’)` returns Unknown

Queries Over Null Value

```
select *
from Student
where Dept = ’CS’ or
    Dept <> ’CS’
```

is equivalent to:

```
select *
from Student
where Dept is not null
```

The Complete Syntax

`select` *AttrExpr* [[`as`] *Alias*] {, *AttrExpr* [[`as`] *Alias*]}
`from` *Table* [[`as`] *Alias*] {, *Table* [[`as`] *Alias*] }
[`where` *Condition*]

Query

Find the names of the students from the Logistics Dept whose mark was 5.

```
select Name
from Student, Exam
where Student.Id = Exam.Id
    and Dept like ’Log%’ and Mark = 5
```

NAME
Antonio

Join in SQL-2

- SQL-2 introduced the `join` clause within the `from` clause:

  `select` *AttrExpr* [[`as`] *Alias*] {, *AttrExpr* [[`as`] *Alias*] }
  `from` *Table* [[`as`] *Alias*]
  { [*JoinType*] `join` *Table* [[`as`] *Alias*] `on` *Condition* }
  [`where` *OtherConditions*]

- *JoinType* can be `inner`, `right` [`outer`], `left` [`outer`] or `full` [`outer`].

Join in SQL-2

```
select Name
from Student, Exam
where  Student.Id = Exam.Id
    and Dept like ’Lo%’ and Mark = 5

select Name
from Student join Exam
     on Student.Id = Exam.Id
where Dept like ’Lo%’and Mark = 5
```

Example: Car and Driver

DRIVER

FirstName	Surname	DriverID
Mary	Brown	VR 2030020Y
Charles	White	PZ 1012436B
Marco	Neri	AP 4544442R

AUTOMOBILE

CarRegNo	Make	Model	DriverID
ABC 123	BMW	323	VR 2030020Y
DEF 456	BMW	Z3	VR 2030020Y
GHI 789	Lancia	Delta	PZ 1012436B
BBB 421	BMW	316	MI 2020030U

Left join

- Find the drivers with their respective cars, also including drivers with no car:

```
select FirstName, Surname, Driver.DriverID,
       CarRegNo, Make, Model
from Driver left join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL

Right join

- Find the drivers with their respective cars, also including cars with no driver:

```
select FirstName, Surname, Driver.DriverID,
       CarRegNo, Make, Model
from Driver right join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
NULL	NULL	NULL	BBB 421	BMW	316

Full join

- Find the drivers with their respective cars, also showing all the possible relationships among them:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver full join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	NULL	BBB 421	BMW	316

A 3-Table Query

- Find the name of students with a mark of “10” in “Math”.

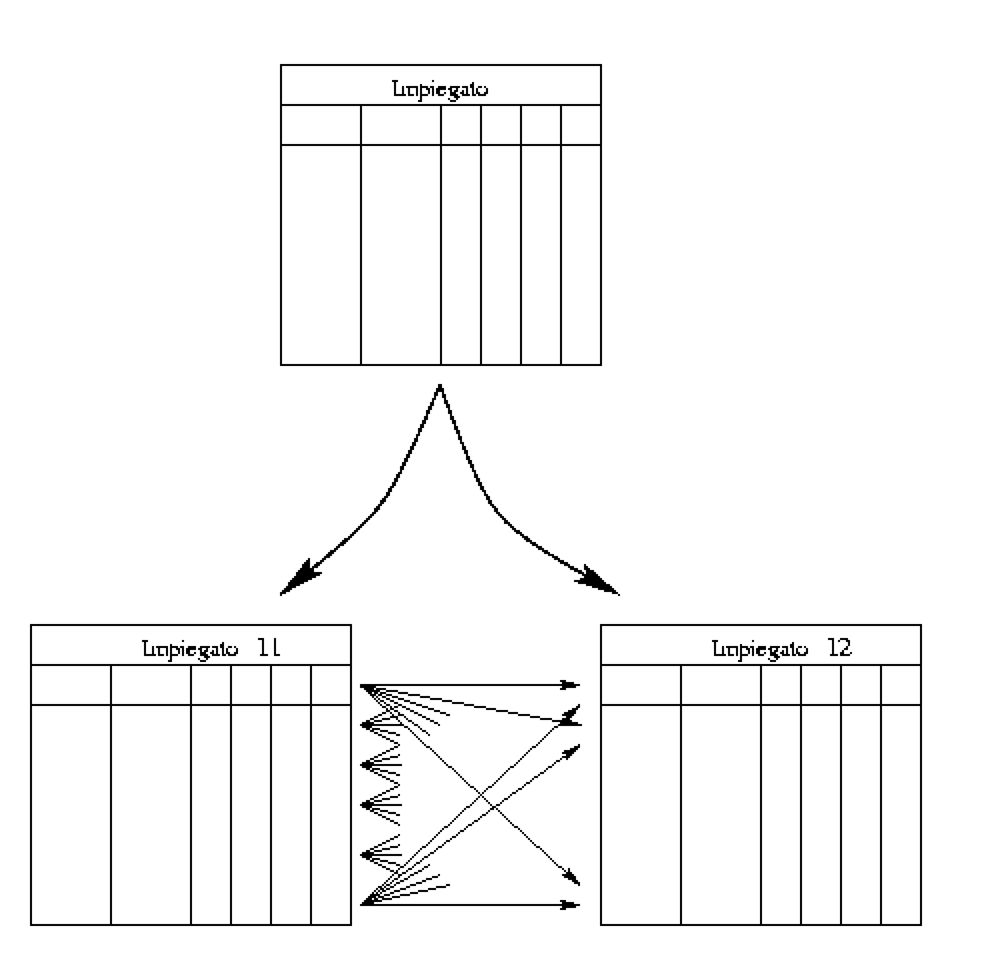
```
select Name
from Student, Exam, Course
where  Student.Id = Exam.Id
  and Course.Code = Exam.Code
  and Title like ’Mat%' and Mark = 10
```

$$\Pi_{\text{Name}}\ \sigma_{(\text{Title like 'Mat\%'}) \wedge (\text{Mark}=10)}\ (\text{Student} \rhd\!\lhd \text{Exam} \rhd\!\lhd \text{Course})$$

Variables in SQL

- The alias name of a table can be used as a variable, referring to the entire table.
- Alias must be used whenever you need to refer to a table **more** than once.
- Using variables is similar to the ρ (rename) operator of RA.

Variables in SQL



Sample Query

Find the employees managed by Giorgio

Employee

Id	Name	HireDate	Salary	Manager
1	Piero	1-1-95	3 M	2
2	Giorgio	1-1-97	2,5 M	null
3	Giovanni	1-7-96	2 M	2

Employees Managed by Giorgio

```
select  X.Name, X.Manager, Y.Id,
  Y.Name
from Employee as X, Employee as Y
where X.Manager = Y.Id
  and Y.Name = 'Giorgio'
```

X.Name	X.Manager	Y.Id	Y.Name
Piero	2	2	Giorgio
Giovanni	2	2	Giorgio

Modify Commands

Modify Commands in SQL

- Aim at:
  - inserting (**`insert`**);
  - removing (**`delete`**);
  - modifying values of attributes (**`update`**).
- All the instructions work over sets (set-oriented).
- The command may include a condition, where one (or more) external table(s) can be referenced.

Insert

- Syntax:

  insert into *Table* [(*AttributeList*)]

  < **values** (*ValueList*) | *SelectSQL*>

- Examples:

```
insert into Student
values (’456878’, ’Giorgio Rossi’,
         ’Bologna’, ’Logistics’)
insert into Bolognesi
values (select *
    from Student
    where City = ’Bologna’)
```

Insert

- The sequence according to which attributes and values are cited is **relevant** (positional notation: the first value refers to the first attributes and so on).
- If *AttributeList* is omitted, SQL refers to all the attributes of the relation, in the sequence they appeared in the `create table` statement.
- If *AttributeList* does not include all the attributes of the relation, the remainder attributes will assume the default value (or, if not defined, the null value).

Insert

- By an *AttributeList*:

```
insert into Student(Id,Name,City,Dept)
   values ('456878', 'Giorgio Rossi',
          'Bologna', 'Logistics')
```

- By a query with *AttributeList*:

```
insert into Bolognesi(Id,Name,City,Dept)
   values (select Id, Name, City, Dept
          from Student
          where City = 'Bologna')
```

Remove

- Syntax:

  `delete from` *Table* [**`where`** *Condition*]

- Remove the student whose Id is 678678:

```
    delete from Student
           where Id = '678678'
```

- Remove the students who never passed an examination:

```
    delete from Student
           where Id not in
            (select Id from Exam)
```

Remove

- The **`delete`** command removes from the table all the tuples that fulfill the condition.
- The command may generate removal of tuples in other tables, if a referential integrity constraint is defined with a **`cascade`** policy.
- If the **`where`** clause is omitted, the **`delete`** command removes all the tuples.
- To remove all the tuples of Student (keeping the empty schema of the table):

```
delete from Student
```

- To completely remove the table Student (including the schema of the table):

```
drop table Student cascade
```

Modify

- Syntax:

update *Table*
set *Attribute* **=** < *Expression* | *SelectSQL* | **null** |
default>
{, *Attribute* **=** < *Expression* | *SelectSQL* | **null** |
default >}
[**where** *Condition*]

- Example:

```
update Exam
       set Mark = 10
             where Date = 1-4-03


update Exam
       set Mark = Mark + 1
             where Id = '787989'
```

Modify

- As the language is **set-oriented,** the order according to which we issue the commands is very relevant:

```
Update Employee
  set Salary = Salary * 1.1
   where Salary <= 30
update Employee
  set Salary = Salary * 1.15
   where Salary > 30
```

- If commands are issued according to the order above, some lucky employees may receive a double increase.

Use of the `in` Token

Increase of 5 euro the amount of all the orders which include the product ‘456’.

```
update Order
  set Amount = Amount + 5
  where OrderId in
   select OrderId
   from Detail
   where ProductId = ’456’
```

Use of Nested Queries

Assign to the attribute TotPieces the sum of the quantities of the lines inside an order.

```
update Order O
 set TotPieces =
  (select sum(Qty)
  from Detail D
  where D.OrderId = O.OrderId)
```