Universiteit Antwerpen

Introduction to Databases
Lecture 4:
Structured Query Language

Floris Geerts

Universiteit Antwerpen

SQL

- How to retrieve data from the database?
  - Lingua franca = SQL
- SQL is the standard
  - Declarative language
  - Supported by all relational database systems

50 year of research on how to optimize

  - Provably hard problem

- We will cover a representative subset of SQL
  - See also http://www.w3schools.com/sql/

SQL Syntax

- Declarative: say WHAT you want, not how to compute it
- Basic query form:

SELECT [distinct] [list of attributes L]

FROM [tables T]

WHERE

[conditions C]

ORDER BY [list of attributes] (**ASCENDING/DESCENDING**);

(With distinct) equal to:
$\pi_L \sigma_C (T_1 \times T_2 \times \ldots \times T_k)$
Result is returned ordered.
If order by is absent, the order is undefined.

- Make any combination of tuples in T. For those that satisfy C, return L.
  - Caveat: keeps duplicates unless **distinct** is used!

Examples

- Works(<u>name</u>,salary)

Name	salary
Jan	1000
An	2000
George	2200

```
SELECT name
FROM works
WHERE salary > 1000;
```

name
An
George

```
SELECT distinct e1.name AS name
FROM works e1, works e2
WHERE e1.salary < e2.salary
ORDER BY name;
```

e1.name	e1.salary	e2.name	e2.salary
Jan	1000	Jan	1000
Jan	1000	An	2000
Jan	1000	George	2200
An	2000	Jan	1000
An	2000	An	2000
An	2000	George	2200
George	2200	Jan	1000
George	2200	An	2000
George	2200	George	2200

name
An
Jan
Jan

distinct →

name
An
Jan

Universiteit Antwerpen

Examples

- Works(name,salary)
- Manages(m_name,e_name) *// m_name manages e_name*
- Ask for employees earning more than their manager

```
SELECT distinct e.name
FROM works e, works m, manages man
WHERE
        e.name=man.e_name  AND
        m.name=man.m_name  AND
        e.salary > m.salary;
```

Syntax

- **SELECT** [list of attributes/expressions] or **SELECT ***
  - Attribute = *[Relation name].[Attribute name]*
    or *[Attribute name]* if there is no ambiguity
  - Expressions: A+B, 2C, A **AS** B
- **FROM** [list of tables]
  - Table = *[Table name]*
    or *[Table name]* ***AS*** *[alternative name]* if disambiguation is needed
  - E.g., FROM works e …. FROM works as e
  - Alternative name becomes the relation name of the copy
- **WHERE** [condition]
  - Condition = Boolean (AND, OR, NOT) combination of atoms
  - Atom = [Attribute or constant] OPER [Attribute or constant]
  - OPER can be =, <>, <, <=, >=, >

Exercises

Employee(ename, ecity)
Works(name,company)
Manages(mname,ename)
Located(company,city)

- Give all employees who live in their company's city.
- Give all employees who live in the same city as their managers.
- Give all pairs of companies that have employees living in the same city.
- Give all employees of a company in Brussels that are managed by a manager who lives in Antwerp.
- Give all employees who work for at least two companies.

Employee(ename, ecity)
Works(name,company)
Manages(mname,ename)
Located(company,city):

- Give all employees who live in their company's city.

```
SELECT E.ename
FROM Employee E, Works W, Located L
WHERE E.ename=W.name AND
        W.company=L.company AND
        E.ecity=L.city;
```

Employee(ename, ecity)

Works(name,company)

Manages(mname,ename)

Located(company,city)

- Give all employees who live in the same city as their managers.


```
SELECT E1.ename
FROM Employee E1, Employee E2, Manages M
WHERE E1.city=E2.city AND
        E1.ename=M.ename AND
        E2.ename=M.mname;
```

Employee(ename, ecity)
Works(name,company)
Manages(mname,ename)
Located(company,city)

- Give all pairs of companies that have employees living in the same city

```
SELECT  W1.company, W2.company
FROM Works W1, Works W2, Employee E1, Employee E2
WHERE   W1.name=E1.name AND
        W2.name=E2.name AND
        E1.ecity=E2.ecity;
```

- Give all employees of a company in Brussels that are managed by a manager who lives in Antwerp

Employee(ename, ecity)
Works(name,company)
Manages(mname,ename)
Located(company,city)

SELECT E1.ename
FROM Employee E1, Employee E2, Manages M, Works W, Located L
WHERE E1.ename=W.name **AND** W.company=L.company **AND** L.city="Brussels"
AND E1.ename=M.ename **AND** M.mname=E2.ename **AND** E2.ecity="Antwerp"

Employee(ename, ecity)

Works(name,company)

Manages(mname,ename)

Located(company,city)

- Give all employees who work for at least two companies

```
SELECT W1.name
FROM Works W1, Works W2
WHERE W1.name=W2.name  AND W1.company <> W2.company
```

Other Basic Constructions: Set Operations

- Result of a query is a relation (with duplicates)
- If two queries are compatible (same schema, same order of attributes), then we can use set operations to combine them
  - **UNION**
  - **INTERSECT**
  - **EXCEPT** (MINUS in many systems)
- These operations remove duplicates. If you want duplicates:
  - **UNION ALL**
  - **INTERSECT ALL**
  - **EXCEPT ALL**

Examples: Set Operations

name	city	salary	age
John	Antwerp	30	43
Mike	Antwerp	50	44
Ella	Brussels	40	23
Mia	Gent	20	42
Mike	Gent	60	60

Select city **from** members **where** age>40;

city
Antwerp
Antwerp
Gent
Gent

INTERSECT

=

city
Antwerp
Gent

Select city **from** members **where** salary<60;

city
Antwerp
Antwerp
Brussels
Gent

Examples: Set Operations

name	city	salary	age
John	Antwerp	30	43
Mike	Antwerp	50	44
Ella	Brussels	40	23
Mia	Gent	20	42
Mike	Gent	60	60

Select city **from** members
where age>40;

city
Antwerp
Antwerp
Gent
Gent

INTERSECT ALL **=**

city
Antwerp
Antwerp
Gent

Select city **from** members
where salary<60;

city
Antwerp
Antwerp
Brussels
Gent

Examples: Set Operations

name	city	salary	age
John	Antwerp	30	43
Mike	Antwerp	50	44
Ella	Brussels	40	23
Mia	Gent	20	42
Mike	Gent	60	60

Select city **from** members **where** age>40;

city
Antwerp
Antwerp
Gent
Gent

UNION

=

city
Antwerp
Brussels
Gent

Select city **from** members **where** salary<60;

city
Antwerp
Antwerp
Brussels
Gent

Examples: Set Operations

name	city	salary	age
John	Antwerp	30	43
Mike	Antwerp	50	44
Ella	Brussels	40	23
Mia	Gent	20	42
Mike	Gent	60	60

Select city **from** members **where** age>40;

city
Antwerp
Antwerp
Gent
Gent

UNION ALL **=**

city
Antwerp
Antwerp
Antwerp
Antwerp
Brussels
Gent
Gent
Gent

Select city **from** members **where** salary<60;

city
Antwerp
Antwerp
Brussels
Gent

Examples: Set Operations

name	city	salary	age
John	Antwerp	30	43
Mike	Antwerp	50	44
Ella	Brussels	40	23
Mia	Gent	20	42
Mike	Gent	60	60

Select city **from** members **where** age>40;

city
Antwerp
Antwerp
Gent
Gent

EXCEPT

=

city

Select city **from** members **where** salary<60;

city
Antwerp
Antwerp
Brussels
Gent

Universiteit Antwerpen

Examples: Set Operations

name	city	salary	age
John	Antwerp	30	43
Mike	Antwerp	50	44
Ella	Brussels	40	23
Mia	Gent	20	42
Mike	Gent	60	60

Select city **from** members **where** age>40;

city
Antwerp
Antwerp
Gent
Gent

EXCEPT ALL **=**

city
Gent

Select city **from** members **where** salary<60;

city
Antwerp
Antwerp
Brussels
Gent

Universiteit Antwerpen

Other Basic Constructions: Joins

- In FROM:
  - R **JOIN** S:
    - table over the attributes of R U S, without duplicates, made out of all “compatible” pairs of tuples
  - If attribute names are different, can use:
    R **JOIN** S **on [condition]**

```
SELECT distinct e1.name AS name
FROM works e1 JOIN works e2 ON e1.name=e2.name
WHERE e1.company <> e2.company;
```

Subqueries

- A subquery is a query inside another query
  - Everywhere a relation is expected, we can use a subquery instead
  - Everywhere a value is expected, we can use a subquery producing one result
- We can also use subqueries in combination with operators EXISTS, ANY, and ALL
- Subqueries can use attributes from the outer query
  - Called *correlated subquery*

Other Basic Constructions: Comparing with sets

- In Where:
  - EXISTS (query): true if query produces an answer
  - [Attribute or constant] OPER ANY (query Q): true if there is an answer A in Q such that [Attribute or constant] OPER A holds
    E.g.: SELECT e.name FROM works e WHERE e.salary > **ANY** (**SELECT** salary **FROM** works)
  - [Attribute or constant] OPER ALL (query Q): true if for all answers A in Q [Attribute or constant] OPER A holds
    E.g.: SELECT e.name FROM works e WHERE e.salary > **ALL** (**SELECT** salary **FROM** works)
  - IN: same as "= ANY"

NOT EXISTS

- Until now: all queries of the type:
  - If A and B and C and D, then: report E

- What if we want to output E if something is missing?
  - Employee NOT working for “UAntwerpen”
  - Employee NOT earning more than any of his/her managers?

NOT EXISTS

Employee(name, city)
Works(name,company)
Manages(m_name,e_name)
Located(company,city)

- Employee NOT working for “UAntwerpen”

- What’s wrong with:

```
SELECT e.name
FROM works e
WHERE e.company <> “UAntwerpen”;
```

NOT EXISTS

Employee(name, city)
Works(name,company)
Manages(m_name,e_name)
Located(company,city)

- Manager NOT earning less than any of his/her employees?
- What's wrong with:

```
SELECT m.name
FROM works e, works m, manages man
WHERE
        e.name=man.e_name  AND
        m.name=man.m_name  AND
        e.salary <= m.salary;
```

NOT EXISTS

- We need “NOT EXISTS” in **WHERE** clause

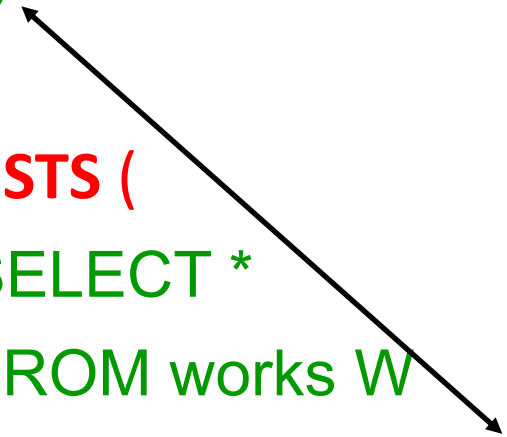
- Syntax:
  - (NOT) EXISTS (SQL Query Q)
  - Q can reuse attribute names of the outer query
  - Is “TRUE” if Q does (not) return any answer

Correct Answers: NOT EXIST

Employee(name, city)
Works(name,company)
Manages(m_name,e_name)
Located(company,city)

- Employee NOT working for “UAntwerpen”

```
SELECT e.name
FROM works e
WHERE
        NOT EXISTS (
                SELECT *
                FROM works W
                WHERE W.name=e.name AND W.company =“UAntwerpen”
        );
```



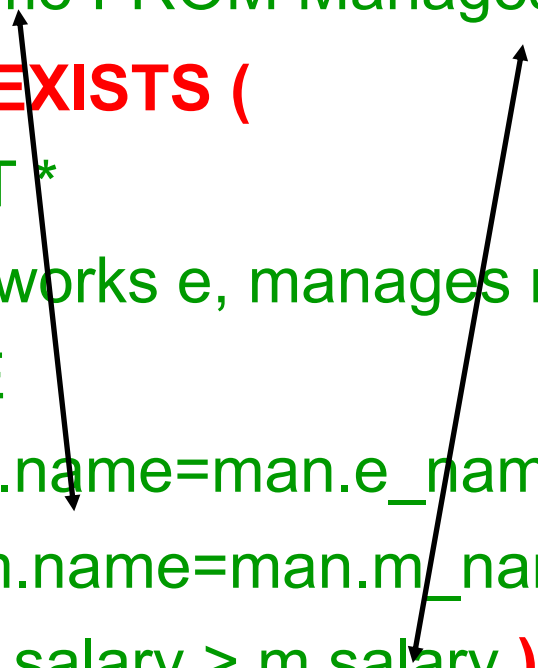
Universiteit Antwerpen

Correct Answers: NOT EXIST

Employee(name, city)
Works(name,company)
Manages(m_name,e_name)
Located(company,city)

- Manager NOT earning less than any of his/her employees?

```
SELECT m.name FROM Manages m
WHERE NOT EXISTS (
        SELECT *
        FROM  works e, manages man
        WHERE
                e.name=man.e_name  AND
                m.name=man.m_name  AND
                e.salary > m.salary );
```



Universiteit Antwerpen

Examples: Subqueries

Serves(bar, beer)
Visits(drinker, bar)

Assumption: All the bars in Serves are also bars in Visits and vice versa.

Select all drinkers that visit some bar where only Duvel is served:
(same as saying: select all drinkers that visit some bar where Duvel is served and *no other* beer is served):

```
SELECT bar
FROM Serves
WHERE beer="Duvel" AND
        bar NOT IN
       (SELECT bar FROM Serves
        WHERE beer <> "Duvel");
```

```
SELECT bar
FROM Serves S1
WHERE beer="Duvel" AND
        NOT EXISTS
       (SELECT bar FROM Serves S2
        WHERE S1.bar=S2.bar AND
         S2.beer<> "Duvel");
```

Examples: Subqueries

Serves(bar, beer)
Visits(drinker, bar)

Assumption: All the bars in Serves are also bars in Visits and vice versa.

Select all drinkers that visit some bar where only Duvel is served:
(same as saying: select all drinkers that visit some bar where Duvel is served and *no other* beer is served):

Examples: Subqueries

Serves(bar, beer)
Visits(drinker, bar)

Assumption: all the bars in Serves are also bars in Visits and vice versa.

Select all drinkers that visit some bar where only Duvel is served:
(same as saying: select all drinkers that visit some bar where Duvel is served and *no other* beer is served):

```
SELECT DISTINCT Visits.drinker
FROM Visits, Serves
WHERE Visits.bar = Serves.bar AND Serves.beer = 'Duvel'
      AND Serves.bar NOT IN
          (SELECT Bar1.bar
          FROM Serves Bar1
          WHERE Bar1.beer <> 'Duvel');
```

Can also be done using **EXCEPT**

```
SELECT DISTINCT drinker
FROM Visits
WHERE bar IN
          (SELECT bar
           FROM Serves
           WHERE beer = 'Duvel')
    AND
      bar NOT IN
         (SELECT bar
          FROM Serves
          WHERE beer <> 'Duvel');
```

Universiteit Antwerpen

Views

- CREATE VIEW: Make a virtual table

```
CREATE VIEW temp AS
        SELECT bar
         FROM Serves
         WHERE beer <> 'Duvel';

SELECT DISTINCT Visits.drinker
FROM Visits, Serves
WHERE Visits.bar = Serves.bar
        AND Serves.beer = 'Duvel'
        AND Serves.bar NOT IN temp;
```

```
SELECT DISTINCT Visits.drinker
FROM Visits, Serves
WHERE Visits.bar = Serves.bar
        AND Serves.beer = 'Duvel'
        AND Serves.bar NOT IN
                (SELECT Bar1.bar
                FROM Serves Bar1
                WHERE Bar1.beer <> 'Duvel');
```

Exercises

Band_member(bname,mname)

Concert(bname,date,location)

- Give all bands that have never concerted in “Wembley Stadium”
- Give all bands that have concerted in any of the places in which “Queen” concerted.
- Give all bands that have never concerted on the same day in the same location as “Ramstein”
- Give all bands that have at least one member in common with the band “Abba”
- Give all bands of which “Mick Jagger” is a member and have no members in common with a band of which “John Lennon” is a member

Exercises

Band_member(bname,mname)

Concert(bname,date,location)

- Give all bands that have never concerted in “Wembley Stadium”

```
SELECT bname
FROM Band_member B
WHERE NOT EXISTS (
        SELECT * FROM Concert C
        WHERE C.bname=B.bname AND C.location=“WS”);
```

Exercises

Band_member(bname,mname)

Concert(bname,date,location)

- Give all bands that have concerted in any of the places in which "Queen" concerted.

```
SELECT B.bname
FROM Band_member B
WHERE EXISTS (
   SELECT * FROM Concert C1, Concert C2
   WHERE C1.bname=B.bname AND C2.bname="Queen" AND C1.location=C2.location
);
```

Exercises

Band_member(bname,mname)

Concert(bname,date,location)

- Give all bands that have never concerted on the same day in the same location as “Ramstein”

```
SELECT B.bname FROM Band_member B
WHERE NOT EXISTS (
   SELECT * FROM Concert C1, Concert C2
   WHERE C1.bname=B.bname AND C2.bname=“Ramstein” AND
   C2.date=C1.date AND C2.location=C1.location);
```

Band_member(bname,mname)

Concert(bname,date,location)

Exercises

- Give all bands that have at least one member in common with the band “Abba”

```
SELECT B1.bname FROM Band_member B1
WHERE EXISTS (
   SELECT * FROM  Band_member B2
   WHERE  B2.bname=“Abba” AND B2.mname=B1.mname);

SELECT B1.bname FROM Band_member B1, Band_member B2
WHERE B2.bname=“Abba” AND B2.mname=B1.mname);
```

Exercises

Band_member(bname,mname)

Concert(bname,date,location)

- Give all bands of which “Mick Jagger” is a member and have no members in common with a band of which “John Lennon” is a member

```
SELECT B.bname FROM Band_member B
WHERE B.mname=“Mick Jagger”
   AND NOT EXISTS (
       SELECT * FROM Band_member B1, Band_member B2, Band_member B3
       WHERE B1.bname=B.bname AND B2.mname=B1.mname AND
         B3.bname=B2.bname AND B3.mname=“John Lennon”);
```

Advanced Functionalities: Aggregation

- Aggregation (avg(A), min(A), max(A), sum(A), count(A), count(distinct A), …)
  - group by / having (optional)
  - **All non-aggregated attributes in SELECT must appear in GROUP BY**

SELECT w.company **as** company, **avg**(salary) **as** salary

FROM works w

GROUP BY Company

HAVING min(salary) > 1000

ORDER BY salary;

Advanced Functionalities: Aggregation

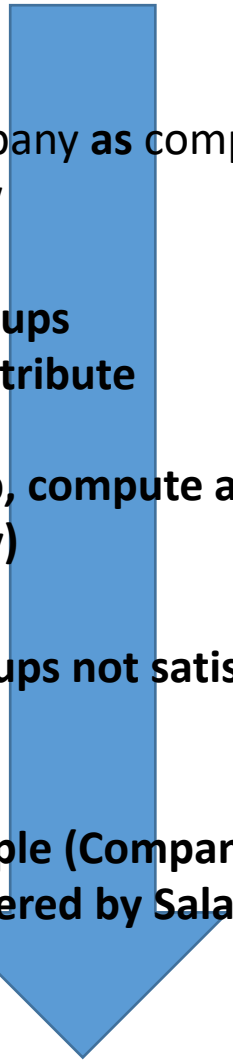
SELECT w.company **as** company, **avg**(salary) **as** salary

FROM works w

GROUP BY Company

HAVING min(salary) > 1000

ORDER BY salary;



SELECT w.company **as** company, salary
FROM works w

Divide into groups
by company attribute

For each group, compute avg(salary)
and min(salary)

Throw out groups not satisfying
HAVING

Output one tuple (Company,avg(salary))
per group, ordered by Salary

Manages(m_name,e_name)

Employee(name, salary)

Aggregation Examples

- Give per manager the number of people they manage:

```
SELECT m_name, count(e_name)
FROM Manages
GROUP BY m_name ;
```

Manages(m_name,e_name)

Employee(name, salary)

Aggregation Examples

- Give all managers that manage more than 3 employees with a salary exceeding 3000 EUR.

```
SELECT m_name
FROM Manages JOIN Employee ON e_name=name
WHERE salary>3000
GROUP BY m_name
HAVING count(e_name) > 3;
```

Manages(m_name,e_name)

Employee(name, salary)

Aggregation Examples

- Give per manager the number of people they manage with a salary exceeding 3000 EUR.

```
SELECT m_name, count(e_name)
FROM Manages JOIN (select * from Employee
                   WHERE salary>3000)
                   ON e_name=name
GROUP BY m_name ;
```

NOT CORRECT for managers with zero such employees!

More joins

(R NATURAL FULL OUTER JOIN S)

A	B	C
a	b	c
d	e	null
null	f	g

(R NATURAL LEFT OUTER JOIN S)

A	B	C
a	b	c
d	e	null

(R NATURAL RIGHT OUTER JOIN S)

A	B	C
a	b	c
null	f	g

(R FULL OUTER JOIN S ON R.A > S.C)

A	B_1	B_2	C
d	e	b	c
a	b	null	null
null	null	f	g

(R LEFT OUTER JOIN S ON R.A > S.C)

A	B_1	B_2	C
d	e	b	c
a	b	null	null

(R RIGHT OUTER JOIN S ON R.A > S.C)

A	B_1	B_2	C
d	e	b	c
null	null	f	g

OUTER JOIN examples

R

A	B
a	b
d	e

S

B	C
b	c
f	g

Universiteit Antwerpen

Manages(m_name,e_name)

Employee(name, salary)

Aggregation Examples

- Give per manager the number of people they manage with a salary exceeding 3000 EUR.

```
SELECT m_name, count(e_name)
FROM Manages LEFT OUTER JOIN (select * from Employee
                              WHERE salary>3000)
                    ON e_name=name
GROUP BY m_name ;
```

What About Conditions With NULL ?

- Recall NULL being a special value indicating something is missing or unknown
  - Arithmetic operations (x, +, -, ...) with NULL return NULL.
  - Comparisons (>, <, =, ...) with NULL return UNKNOWN.
    - NULL=NULL ➔ UNKNOWN ; NULL <> NULL ➔ UNKNOWN
  - Use IS to test on NULL

```
SELECT distinct e_name
FROM manages
WHERE m_name=NULL;
```

Empty. Always. Because X=NULL is always UNKOWN even if X=NULL

```
SELECT distinct e_name
FROM manages
WHERE m_name IS NULL;
```

All employees for which manager is missing/unknown

Universiteit Antwerpen

Three-Valued Logic

- Being able to evaluate complex Boolean expressions with NULL values requires an extension of the usual binary logic to a three-valued logic

X	Y	X AND Y	X OR Y	NOT X
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNK.	UNK.	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNK.	TRUE	UNK.	TRUE	UNK.
UNK.	UNK.	UNK.	UNK.	UNK.
UNK.	FALSE	FALSE	UNK.	UNK.
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNK.	FALSE	UNK.	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

- Only tuples for which the condition in the WHERE clause evaluates to TRUE are returned.

Exercises

- Band_member(bname,mname)
- Concert(bname,date,location)

- Give the band(s) with fewest members.
- Give all bands that played more often in “Wembley stadium” than “Queen” did.

Exercises

Band_member(<u>bname</u>,<u>mname</u>)
Concert(<u>bname</u>,<u>date</u>,<u>location</u>)

Give the band(s) with fewest members.

```
CREATE VIEW nMembers AS (SELECT bname, count(mname) AS  number
                                FROM Band_member GROUP BY bname);

CREATE VIEW minimum AS (SELECT min(number) FROM nMembers);


SELECT B.bname from nMembers B
WHERE EXISTS (SELECT * FROM minimum M
                       WHERE M.number=B.number);
```

Exercises

Band_member(bname,mname)
Concert(bname,date,location)

Give all bands that played more often in “Wembley stadium” than “Queen” did.

```
CREATE VIEW numWS AS (SELECT bname, count(date)  AS  num
                      FROM Concert
                      WHERE location=“WS” GROUP BY bname);

SELECT B.bname
FROM numWS B, numWS Q
WHERE Q.bname=“Queen” AND Q.num< B.num);
```

Summary

- The standard of all relational database query languages:
  The Structured Query Language (SQL)
  - Declarative language: say what you want, not how to compute it
  - Allows for aggregation (sum, min, max, avg, ...)
  - Fully “composable”: output can be input of a SQL query
  - Has “bag” semantics
- Optimizer translates SQL to (an extended version of) the more procedural RA, and creates a physical query plan for RA query
- Expressive power: SQL without aggregation and without duplicates has same expressive power as the relational algebra