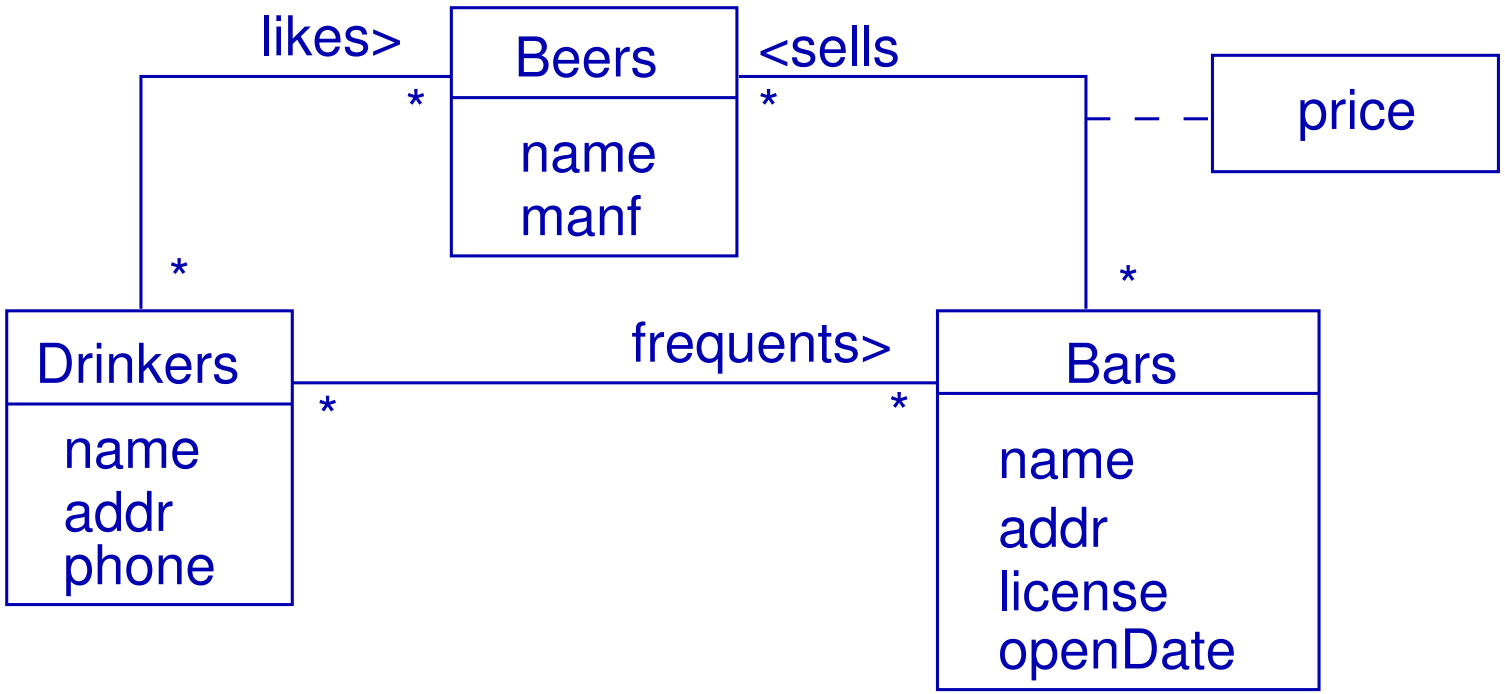
Database Management Systems

Lorraine Goeuriot
IUT 1, Université Grenoble Alpes
Auteur du cours : Marie-Christine Fauvet
Université Joseph Fourier, Grenoble – UFR IM^{2}AG

21 janvier 2016

Contenu

New Toy DB [1]



1. This DB is taken from : *Database Systems : The Complete Book (2e édition)*, écrit par Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom, 2008.

Relations

Drinkers (<u>name</u>, addr, phone)
Beers (<u>name</u>, manf)
Bars (<u>name</u>, addr, license, openDate)
Likes (<u>drinker, beer</u>)
$Likes[drinker] \subseteq Drinkers[name]$
$Likes[beer] \subseteq Beers[name]$
Sells (<u>bar, beer</u>, price)
$price > 0$
$Sells[beer] \subseteq Beers[name]$
$Sells[bar] \subseteq Bars[name]$
Frequents (<u>drinker, bar</u>)
$Frequents[drinker] \subseteq Drinkers[name]$
$Frequents[bar] \subseteq Bars[name]$

Domains should be described too...

Relations Values

Bars

Name	**Addr**	**License**	**openDate**
Australia Hotel	The Rocks	123456	12/1/1940
Coogee Bay Hotel	Coogee	966500	31/8/1980
Lord Nelson	The Rocks	123888	11/11/1920
Marble Bar	Sydney	122123	1/4/2001
Regent Hotel	Kingsford	987654	29/2/2000
Rose Bay Hotel	Rose Bay	966410	31/8/2000
Royal Hotel	Randwick	938500	26/6/1986

Relations Values

Drinkers

Name	**Addr**	**Phone**
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321
Marie	Rose Bay	9371-2126
Adrian	Redfern	9371-1244

Frequents

Drinker	**Bar**
Adam	Coogee Bay Hotel
Gernot	Lord Nelson
John	Coogee Bay Hotel
John	Lord Nelson
John	Australia Hotel
Justin	Regent Hotel
Justin	Marble Bar
Marie	Rose Bay Hotel

Relations Values

Beers

Name	Manf
80/-	Caledonian
Bigfoot Barley - Wine	Sierra Nevada
Burragorang - Bock	George IV - Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old T	oohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada

Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Relations Values

Likes

Drinker	**Beer**
Adam	Crown Lager
Adam	Fosters
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Relations Values

Sells

Bar	**Beer**	**Price**
Australia Hotel	Burragorang Bock	3.50
Coogee Bay Hotel	New	2.30
Coogee Bay Hotel	Old	2.50
Coogee Bay Hotel	Sparkling Ale	2.80
Coogee Bay Hotel	Victoria Bitter	2.30

Query - Definition

A query is a *declarative program* aiming at retrieving data from a DB.
There are two ways to use them :

- Interpreted : directly query the database (e.g. via terminal or DBMS).
- Encapsulated : the query and results are treated by a programme (e.g. web application).

Example :

```
select bar, price /* a set of attributes */
from Sells /* a set of relations */
where beer = 'Victoria Bitter' /* a boolean expression */
```

In which bar and at which price the beer 'Victoria Bitter' is sold.

With MySQL...

```
SQL> select bar, price
2 from Sells
3 where beer = 'Victoria Bitter';
```

BAR	PRICE
Coogee Bay Hotel	2.3
Marble Bar	2.8
Regent Hotel	2.2
Royal Hotel	2.3

```
4 rows selected
SQL>
```

SQL Identifiers

- Objects : relations, attributes are identified by their names, with conventions similar to usual programming languages,
  BUT names are not case sensitive.
  Examples : Drinkers, Beer, select, From, wherE, And, OR, etc...
- Constants : string and number values have an implicit type.
  Examples : 'C\' is a string, "John's bike", 123.84, 4.23e-12, .01, 23., etc...

Types de données SQL

- Strings :

  char [n] : n characters (fixed length)
  varchar [n] : 0..n char (variable length)

- Numbers :

  smalling, integer, bigint
  integer (p)
  real, float, double
  float(p)

- Sets and tuples : list of values between (and)
  Examples : (12, 3, 9, 10) is a set (could be a tuple !)
  (123, 'John', '12-3-2001') is a tuple

`http://www.w3schools.com/sql/sql_datatypes_general.asp`

- Date, Time, Datetime... : various format for temporal types.

  Example :

  - *'2015-01-13' type Date*
  - *'15 :30 :01' type Time*
  - *'2015-01-13 15 :30 :01' type Datetime*

Operators and functions

- Comparison operators (work with every type) :
  $<$, $>$, $<>$, $!=$, $=$
- Boolean operators :
  NOT, AND, OR (from highest to lowest priority)
- Arithmetic operators :
  $+$ sum, - substraction, * multiplication, $/$ division
- Other numerical operators :
  abs, ceil, floor, truncate, round, exp, pow, mod, sqrt, etc...
  sin, cos, tan, atan, etc...

- String functions :

  concat, lower, upper, substring, length, etc...

- Temporal functions :

  to_date (string,string) → date
  / to_date ('05 Dec 2001', 'DD Mon YYYY') */*
  to_char (date, string) → string
  / to_char (d,f) gives a string representing d following format f. */*
  / to_char (to_date ('05 Dec 2001', 'DD Mon YYYY'), 'DD/MM/YY') = '05/12/01' */*

Other temporal functions :

add_months (date, integer) → date
*/*add_months (d1, n) gives date d1 plus one month.*/*

Numerical operators can be used on temporal values :

d1, d2 : date, n : int ≥ 0
d1 - d2 amount of days between d1 and d2
d1 + n is the date n days after d1
d1 - n is the date n days before d1

Other functions :

sysdate → date, etc...

Projection, selection

Give the bars (name and address) located in Sydney
In SQL :

```
select name, addr /*projection*/
from Bars /*relation(s)*/
where addr = 'Sydney' /*selection*/
```

In the query *select ... from ... where P*, P is a predicate, built with :

- a simple condition, such as :
  <att. name> < comp. op.> <att. name> ou
  <att. name> <comp. op.> <cst. val.>
- or a complex boolean expression such as :
  <cond.> <bool. op.> <cond.>
  where <cond.> is a simple condition or a complex boolean expression.

Comparison operators : $=$, $\neq$, $<$, $>$, etc.
Boolean operators : not, and, or
P can contain brackets

Exercises

1) Evaluate the following expressions :

- A and B or C
- A and (B or C)
- not A and A

when A=false, B=true, C=false

2) Evaluate the following expressions : :

- (3>5) and (6=6)
- (3>5) or (6=6)
- (X≥Y) and (B or (X=3)), when X=4 et Y=3

Expressions in a Select clause

For each bar, give its name, address and for how many years it has been open

```
select name, addr,
       to_number (to_char (sysdate, 'YYYY'))
       - to_number (to_char (openDate, 'YYYY'))
from Bars;
```

Many operators and functions exist :

`https://dev.mysql.com/doc/refman/5.7/en/functions.html`

Multiple Values

Give the beers that cost less than $2.5

The expected result is :

Beer
New
Old
Victoria Bitter

In SQL :

select Beer from Sells
where price <= 2.5

And the result is :

Beer
New
Old
Victoria Bitter
New
Victoria Bitter
New
Old
Victoria Bitter

Multiple Values - get rid of them

To remove repetitions from the results :

select distinct Beer from Sells where price <= 2.5

Use it with caution :

- Think about the cost !
- Often useless : *select name from Drinkers*

Use of the star (*)

The character * means “no projection”
Example : FGive all the information about the drinkers In SQL : *select * from Drinkers*
Result :

name	addr	phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321
Marie	Rose Bay	9371-2126
Adrian	Redfern	9371-1244

- Very useful to visualize the schema and content of a table

Rename attributes in a *select*

Attributes can be renamed in a query using *AS* in the *select* clause :

select name as Drinkers, addr, phone from Drinkers

Result :

drinkers	addr	phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321
Marie	Rose Bay	9371-2126
Adrian	Redfern	9371-1244

Order tuples

The *order by* clause can only be applied to attributes in the *select* clause :

select bar, beer, price from Sells
order by price desc, bar asc

Result :

bar	beer	price
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Australia Hotel	Burragorang Bock	3.5
Coogee Bay Hotel	Sparkling Ale	2.8
Marble Bar	New	2.8
Marble Bar	Victoria Bitter	2.8
Marble Bar	Old	2.8
Coogee Bay Hotel	Old	2.5
...		

Relational junction (intern)

```
select A1, A2, ..., An                          /* → projection*/
from R1 join R2 on .. join Rp on ...            /* → relation grouping*/
where C                                         /* → selection*/
```

For each beer sold by Australia Hotel, give its price and manufacturer

In SQL :

```
select Beer, Price, Manf                        /* projection */
from Beers join Sells on (Name = Beer)          /* join condition */
where Bar = 'Australia Hotel'                   /* selection condition */
/* The clause "on" can contain any predicate. */
```

To join two (or more) relations

Name ambiguity in a query

How can we handle several occurrences of the same name in a query ?

For each bar that John frequents, give its name, the beers it sells and their price.

select Bar, Beer, Price
from Sells join Frequents (Bar = Bar)
where Drinker = 'John'
ERROR at line 2 :
ORA-00918 : column ambiguously defined

Attribute name ambiguity - solution

The complete name of an attribute is characterized by the relation name :

select Sells.Bar, Sells.Beer, Sells.Price
from Sells join Frequents (Sells.Bar = Frequents.Bar)
where Frequents.Drinker = 'John'

- Each attribute is characterized by the relation it belongs to (easier to read)
- Necessary for the attributes defined in more than one relation in the from clause.

Natural join : first form

The join condition relates to all the common attributes of the 2 relations.

```
select Bar, Sells.Beer, Sells.Price
/* Bar appears only once in the resulting schema */
from Sells natural join Frequents
/* The attribute bar is common to both relations */
where Frequents.Drinker = 'John'
```

The `Bar` attribute (appearing in both `Sells` and `Frequents`) cannot be prefixed by the relation name.

Natural join : 2nd form

The condition does not relate to all the common attributes

```
select Bars.name, Addr, Drinkers.name
from Bars join Drinkers using (addr)
      /* the join condition uses only the common attribute addr*/
```

Another version :

```
select Bars.name, Bars.Addr, Drinkers.name
from Bars join Drinkers on (Bars.addr=Drinkers.addr)
      /* The join condition is explicit.*/
```

Natural join and relational join in the same clause `from`

For each beer, give its name, manufacturer, the name of drinkers who like it, and for each of these drinkers the bars they frequent.

```
select name, manf, drinker, bar
from Beers join Likes on (name=beer) natural join Frequents
```

/* The following expression is not correct : the drinker attribute is ambiguous : it belongs to both relations Frequents natural join Beers and Likes.

```
select name, manf, drinker, bar
from Frequents natural join Beers join Likes on (name=beer)
```

/* The following expression is correct : */

```
select name, manf, drinker, bar
from Frequents natural join (Beers join Likes on (name=beer))
```

Join operations all have the same priority, they are evaluated from left to right. Parenthesis allow to force the order of evaluations.

Cross join

select ....
from Bars cross join Drinkers
where Drinkers.name = 'John'

Join : summary

R and S are relations defined as : R (X, Y, Z) et S (Y, Z, T)

- Cross join : from R cross join S
  schema : R.X, R.Y, R.Z, S.Y, S.Z, S.T
- Relational join : from R join S on (P)
  schema : R.X, R.Y, R.Z, S.Y, S.Z, S.T
  P is a predicate, that must be True on R.X, R.Y, R.Z, S.Y, S.Z, S.T
  The join condition is P
- Natural join (1st form) : from R natural join S
  schema : R.X, Y, Z, S.T
  The join condition is R.Y = S.Y and R.Z = S.Z (All the common attributes are concerned)
- Natural join (2nd form) : from R join S using (Y)
  schema : R.X, Y, R.Z, S.Z, S.T
  The join condition is R.Y = S.Y

Multiple use of a relation

Give all the pairs drinkers who like the same beer

```
select L1.drinker, L2.Drinker
from Likes L1 join Likes L2
      on (L1.beer = L2.beer and L1.drinker <> L2.drinker)
```

Questions :

Is the clause `select distinct` necessary to remove duplicates ?

How can we remove symetrical pairs ?
R is antisymetrical if :
$<X, Y> \in R \Longrightarrow (<Y, X> \notin R \text{ or } X = Y)$
We know that for all X, $<X, X> \notin R$

```
select distinct L1.drinker, L2.Drinker
from Likes L1 join Likes L2
        on (L1.Beer = L2.Beer and L1.Drinker < L2.Drinker)
```

Union, Intersection, Difference

Q1 and Q2 are queries with a similar structure `select... from....` with compatible schema :

- select A, B from... is not compatible with `select C from...`
- `select A, B from...` is compatible with `select C, D from...` if and only if A and C are comparables, as well as B and D.

Opérateurs

- Union [all]
  Duplicates are not removed with option `all`
- Intersect (intersection)
- Minus (différence)

Duplicates are removed, except with `union all`.

Difference : example

Give the beers sold for less than \$3, that John does not like.

select Beer from Sells where Price $<$ *3*
minus
select Beer from Likes where Drinker $=$ *'John'*

Both subquery must create relations with compatibles schema.

Intersection : example

Give the drinkers and the beers such as drinkers like a given beer and frequent a bar that sells it.

select Drinker, Beer from Likes
intersect
select Drinker, Beer from Sells natural join Frequents

Another version :

select distinct Drinker, Beer
from Likes natural join Sells natural join Frequents
/ Joins are evaluated from left to right. */*

Union : example

Give the drinkers who like Sparkling Ale or frequent Lord Nelson.

```
select Drinker from Likes
where Beer = 'Sparkling Ale'
union
select Drinker from Frequents
where Bar = 'Lord Nelson'
```

→

Drinker
Gernot
Justin
John

```
select Drinker from Likes
where Beer = 'Sparkling Ale'
union all
select Drinker from Frequents
where Bar = 'Lord Nelson'
```

→

Drinker
Gernot
Gernot
Justin
John

Queries in the clause from

The clause `from` can contain SQL expressions.
Give the name and manufacturer of the beer sold less than $3 that John does not like
We already saw the query giving the beers sold less than $3 that John does not like :

select Beer from Sells where Price $<$ *3*
minus
select Beer from Likes where Drinker $=$ *'John'*

We call `R` the relation resulting from this query.

The final query is (using R) :

select Beer, Manf
from Beers join R on (Beer = Name)

If we substitute R to its expression :

```
select Beer, Manf
from Beers join (select Beer from Sells where Price < 3
                 minus
                 select Beer from Likes where Drinker = 'John') R
                 /* < b > ∈ R ⟺ b is sold less than $3
                 et John n'aime pas b. */
     on (Beer = Name)
```

A name has to be given to the subquery, also called nested query (R in this case).

Why are embedded queries useful ?

- Decompose the query in simpler subqueries.
- The expression and test of the subqueries are independent.

Rule : the relation embedded in the clause `from` MUST be specified.
Clumsy use :

```
select beer, price
from  (select distinct beer from Sells where price ≥ 10) X
      /* < b > ∈ X ⟺ there is a bar selling beer b for more than $10. */
      natural join Sells
where bar = 'Coogee Bay Hotel'
```

A shorter and simpler version :

```
select beer, price from Sells
where price ≥ 10 and bar = 'Coogee Bay Hotel'
```

Aggregate Functions

Used to reduce a list of values to a single value.

- Count (*) → number of tuples
- Count (A) → number of values for `A`
- Count (distinct A) → number of distinct values for `A`
- Avg (A) → average value for `A`
- Min (A) (resp. Max) → minimum (resp. maximum) value of `A`
- Sum (A) → sum of values for `A`

Aggregate functions : Example

What is the average price for beers sold in Australia Hotel ?

select avg (price) from Sells where Bar = 'Australia Hotel'

How many bars are located in The Rocks ?

select count (*) from Bars where Addr = 'The Rocks'

Aggregate functions : more examples

How many bars are selling beers for less than $2.5 ?

select count (Bar) from Sells where price < 2.5

Careful : This query is not correct, why ?

A bar is counted as many times as it is selling beers for less than $2.5

Correct query :

select count (distinct Bar) from Sells where price < 2.5

Which bars sell beer `New` for the lowest price ?

```
select bar
from Sells join
     (select min(price) as minP from Sells
     where beer='New') Min
     /* <m> ∈ Min ⟺ m is the lowest price for beer New
     among all the bars.*/
     on (price = minP)
where beer='New'
```

Partition

Partition a query to apply an aggregate function on each class separately

How many beers do drinkers like ?

Expected result :

Drinker	Beer
Adam	3
Gernot	2
John	4
Justin	2

Partition - creation

1. Create a partition on `Likes`

Adam	Crown Lager
Adam	Fosters
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

In SQL :

```
select ...
from
Likes
group by
Drinker
```

2. Reduce each class of the partition to a single tuple

Drinker list (partition criteria)
→ Drinker (one of the values)
Beer list
→ int (# values)

In SQL :

select Drinker, count (Beer)
from Likes
group by Drinker

Proper use of the clause *select*

In a query containing the clause *group by*, *select* only contains :

- One or several attributes among the partition criteria
- One or several aggregations applied to the attributes

Wrong :

```
select Drinker, Addr, count (Beer)
from Likes join Drinkers
on (Drinker=Name)
group by Drinker
```

Right :

```
select Drinker, Addr, count (Beer)
from Likes join Drinkers
on (Drinker=Name)
group by Drinker, Addr
```

Partition filter

For each bar selling more than 2 beers, give the number of beers it is selling and the number of drinkers frequenting it.

```
select Bar,
       count (distinct S.Beer) as NbBeers,
       count (distinct F.Drinker) as NbDrinkers
from Sells S natural join Frequents F
group by Bar
having count (distinct S.Beer) > 2
```

More examples...

Which bars are selling all the beers ?
We know that : $| A | = | B | \wedge A \subseteq B \implies A = B$
For each bar, how many beers ?

select bar, count(beer) as nbBeers from Sells group by bar

How many beers sold in total ?

select count(distinct beer) as nbTot from Sells

The final query is :

```
select bar
from (select bar, count(beer) as nbBeers from Sells group by bar) X1
     join (select count(distinct beer) as nbTot from Sells) X2 on
     (nbBeers = nbTot)
```

Which bars are selling the maximum number of beers ?

- For each bar, how many beers ? see X1 above.
- In X1, which bar is associated to the maximum ?

```
select bar from X1 join (select max(nbBeers) as nbMax from X1) Y
                        on (nbBeers = nbMax)
```

The final query :

```
select bar
from (select bar, count(beer) as nbBeers from Sells group by bar) X1
      join (select max(nbBeers) as nbMax from X1) Y
      on (nbBeers = nbMax)
```

For each drinker who likes all the beers sold in Australia Hotel, give the name and the bars she/he likes.

```
/* How many beers are sold in Australia hotel ? */
select count(Beer) from Sells where Bar = 'Australia Hotel'
/* How many beers each drinker likes ? */
select drinker, count(beer) as nbBeers from Likes group by Drinker

/* Final query : */
select drinker, bar
from (select count(Beer) as nbBeers
      from Sells where Bar = 'Australia Hotel') X1
      natural join
      (select drinker, count(beer) as nbBeers
      from Likes) X2
      natural join Frequents
```

This query is not correct, why ?

Correct query :

```
select drinker, bar
from (select count(Beer) as nbBeers
      from Sells where bar = 'Australia Hotel') X1
      natural join
      (select drinker, count(beer) as nbBeers
      from Likes natural join Sells
      where bar = 'Australia Hotel') X2
      natural join Frequents
```

Evaluation order in queries

select *5. projections or aggregate functions*
select *5. projection on a partition criteria and aggregate functions*
from *1. relation junction*
where *2. selection on the joined relations*
group by *3. partition*
having *4. filter on the partition*

Partition : common error

For each bar, give its name, address, the amount of beers it sells, and the drinkers frequenting it.

```
select bar, count (distinct beer) as nbBeers, addr, drinker
                                                  *
from Bars join Sells on (name = bar)
            natural join Frequents
group by bar
```

ERROR at line 1 :
ORA-00979 : not a GROUP BY expression

Why ?

First problem : attribute Addr

```
select bar, count (beer) as nbBeers, addr
        /* distinct is not ncecessary. */
from Bars join Sells on (name = bar)
group by bar, addr
```

Expressions *group by name* and *group by name, addr* relate to the same partition, as a bar has only one address.

Result

Bar	nbBeers	Addr	Drinker
Australia Hotel	1	The Rocks	{John}
Coogee Bay Hotel	4	Coogee	{Adam, John}
Lord Nelson	2	The Rocks	{Gernot, John}
Marble Bar	3	Sydney	{Justin}
Regent Hotel	2	Kingsford	{Justin}

Column `Drinker`'s type is a set !

Result

Bar	nbBeers	Addr	Drinker
Australia Hotel	1	The Rocks	John
Coogee Bay Hotel	4	Coogee	Adam
Coogee Bay Hotel	4	Coogee	John
Lord Nelson	2	The Rocks	Gernot
Lord Nelson	2	The Rocks	John
Marble Bar	3	Sydney	Justin
Regent Hotel	2	Kingsford	Justin

Pour résoudre le problème sur Drinker

```
select bar, nbBeers, addr, drinker
from (select bar, count (beer) as nbBeers, addr
      from Bars join Sells on (name = bar)
      group by bar, addr) X
      /* <b, n, a> ∈ X ⟺ bar b sells n beers and its
      address is a.*/
      natural join Frequents
```

Operator IN

- *The result of a query is a relation.*
- tuple IN relation $\Longleftrightarrow$ tuple $\in$ relation
- Inverse : tuple NOT IN relation

Give the name and manufacturer for beers that John likes.

```
select Name, Manf from Beers
where Name in (select Beer from Likes
                    where Drinker = 'John')
```

Another expression :

```
select B.name, B.manf
      from Beers B join Likes L on (B.name = L.beer)
      where L.drinker = 'John'
```

Operator IN - syntax

The operator IN can be generalized to deal with tuples of any length

- $< X_1, X_2, ..., X_n >$ IN relation (with attributes $A_1, A_2, ..., A_n$) is true $\Longleftrightarrow$ <X1, X2, ...Xn > $\in$ relation.

select ... from ...
where $(X_1, X_2, ..., X_n)$ in
(select $A_1, A_2, ..., A_n$ from ....)

Obviously, for each i=1..n,
types for X_i et A_i must be comparable

Use of nested queries

Give the bars that serve `New` at the same price as `Victoria Bitter` in `Coogee Bay Hotel`.

```
select Bar from Sells
where Beer = 'New' and Price =
      (select Price from Sells
      where Bar = 'Coogee Bay Hotel' and Beer = 'Victoria Bitter')
```

If the nested query gives a single tuple, the system gives it back, else it returns an error

Mieux :

```
select Bar from Sells
where Beer = 'New' and Price IN
      (select Price from Sells
      where Bar = 'Coogee Bay Hotel' and Beer = 'Victoria Bitter')
```

Utilisation de sous-requêtes

```
select Bar from Sells
where Beer = 'New' and Price =
      (select Price from Sells
      where Bar = 'Coogee Bay Hotel' and Beer = 'Victoria Bitter')
```

Better :

```
select Bar from Sells
where Beer = 'New' and Price IN
      (select Price from Sells
      where Bar = 'Coogee Bay Hotel' and Beer = 'Victoria Bitter')
```

Even better :

```
select s1.bar from Sells s1 join Sells s2 on (s1.price=s2.price)
where s1.beer = 'New' and
      s2.bar = 'Coogee Bay Hotel' and s2.beer = 'Victoria Bitter'
```

Parameters range

Range rule : Parameters can be used in any nested query
For each bar, give the cheapest beer and its price (in this bar)

```
select Bar, Beer, Price
from Sells S1 /* S1 is a parameter */
where Price in (select min (Price)
                from Sells S2 where S1.Bar = S2.Bar)
```

Another expression :

```
select Bar, Beer, Price
from Sells S join
     (select bar, min(price) as minPrice
     from Sells
     group by bar) as X
     on (S.bar=X.bar and price=minPrice)
```

Motivation

There can be a need to use empty values :

- Temporarily unknown values :
  I don't know this address yet, but will know it soon
- Temporarily forbidden values :
  She's not married, she can't have a marital name yet
- Strictly forbidden values :
  He didn't pass the exam, so can't have a mark.
- etc...

From DBs perspective, there is only one "empty value" concept.

Empty value characteristics

- The empty value is called `NULL`.
- `NULL` values don't appear in the queries output (nothing appears).
- `NULL` belongs to all types.
- All the operators have a specific behaviour with empty values.
- Two new operators : `IS NULL` - `IS NOT NULL`

Operators and empty values

- Comparison operators : A is an attribute
  $\forall$ op $\in \{<, >, <>, =\}$, A op Null, Null op A equals Null
- Arithmetic operators :
  $\forall$ op $\in \{+, -, /, *, ...\}$, A op Null, Null op A equals Null
- Boolean operators :

AND	True	False	Null
True	True	False	Null
False	False	False	Null
Null	Null	False	Null

Similarly, NOT null equals null

OR	True	False	Null
True	True	True	True
False	True	False	Null
Null	True	Null	Null

- min, max, sum, avg ignore NULL values.
  If all the values are NULL, the result is NULL.
- count(*) doesn't take into account NULL values : it only counts tuples.
- count (distinct A) et count (A) only count non NULL values.

NULL isn't 0, { }, '', nor ()
NULL is nothing !

Queries and missing values

R	**StuId**	**Mark**	**Course**
	12	45	maths
	11	90	french
	11	75	maths
	12	65	physics
	12		french

Give the information on the students who got a mark different from 90

select * from R where note <> 90

R	**StuId**	**Mark**	**Course**
	12	45	maths
	11	75	maths
	12	65	physics

A tuple is selected if it satisfies the `where` condition
(if the evaluated condition is `True`)

Give the information on the students who got a mark different from 90 (including those who didn't pass the exam)

select * from R where note <> 90 or note is null

R	**Stuld**	**Mark**	**Course**
	12	45	maths
	11	75	maths
	12	65	physics
	12		french

For each student, calculate its average mark. Non passed exams count as 0.

```
select NumEt, avg (nvl (note, 0)) as Moy
from R group by NumEt

/* nvl (x, y) : if x is not null then x else y*/
```

R	**StuId**	**Moy**
	12	36.67
	11	82.5

Générer des valeurs absentes

For each bar, give the drinkers frequenting it (including bars where nobody goes)

```
select Bar, Drinker from Frequents
        union
select Name, Null from Bars where Name not in (select Bar from Frequents)
```

The expected result is :

Bar	Drinker
Coogee Bay Hotel	Adam
Lord nelson	Gernot
Coogee Bay Hotel	John
Lord Nelson	John
Australia Hotel	John
Regent Hotel	Justin
Marble Bar	Justin
Royal Hotel	

This query is not correct : the schema `select Bar, Drinker from Frequents` is not compatible with `select Name, Null from Bars...` (Drinker belongs to Frequents keys, it cannot contain `NULL` values).

For each bar, give the drinkers frequenting it (including bars where nobody goes)

```
select B.Name, F.Drinker from Bars B left outer join
                                Frequents F on (B.name = F.bar)
```

For each tuple in Bars which do not satisfy the join relation with anothe tuple from Frequents, a tuple with NULL values is created.
What's the result of :

```
select F.Bar, F.Drinker
from Bars B left outer join
     Frequents F on (B.name = F.bar)
```

Attributes in the junction must be carefully chosen !

Another example :

Students (<u>stuId</u>, name, firstname)
Course (<u>courseID</u>, courseName)
Registrations (<u>*stuID, courseID*</u>)

For each student (ID, name and firstname), give the identifiers of courses she/he is registered (also give students who didn't register any course)

```
select S.stuID, S.name, S.lastname, R.courseID
from Students S natural left outer join Registrations R
```

For each tuple in *Students* that do not satisfy the join condition (no tuple from registration), a tuple with NULL values for `Registration` is returned.

Another example :
For each student (stuID, name, firstname), give the identifiers and name of the courses she/he registered to (also give students not registered to any course)

```
select S.stuID, S.name, S.firstname, R.courseID, C.courseName
from Student S natural left outer join Registrations R
        natural left outer join Course C
```

The query below give the same result :

```
select S.stuID, S.name, S.firstname, R.courseID, C.courseName
from Course C natural right outer join Registrations R
        natural right outer join Students S
```

- Often, there are several ways of expressing the same query
  - Which one should be chosen ? Not an easy question
  - The queries are optimized by the DBMS, so developers do not need to worry about this...
  - However, with some DBMS, execution time can be critical