Updated 9/26/2022

Topic 1: SQL
L06: SQL intermediate

Wolfgang Gatterbauer

CS3200 Database design (fa22)

https://northeastern-datalab.github.io/cs3200/fa22s3/

9/26/2022

Class warm-up

- Last class summary
- Thanks for posting and answering on Piazza! Grades of anonymity
- Familiarity with recursion?
- Any particular SQL constructs that would interest you?

- SQL later today: "witnesses" (traditionally students find this topic the conceptually most difficult)
- SQL next: Nulls, outer joins, (recursion?)

Attributes for GROUP BY revisited

Product

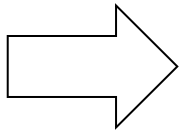
PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	GizmoWorks	65	Japan
3	Hitachi	15	Japan

Q: List company names and number of products they make [cname, num]

```
SELECT  C.cname, count(*) num
FROM    Product P, Company C
WHERE   P.cid=C.cid
GROUP BY C.cname
```



?

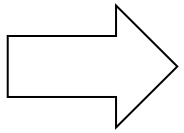
Attributes for GROUP BY revisited

P C

PName	Price	Category	cid	cid	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	1	1	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	1	1	GizmoWorks	25	USA
SingleTouch	$14.99	Photography	2	2	GizmoWorks	65	Japan
MultiTouch	$203.99	Household	3	3	Hitachi	15	Japan

Q: List company names and number of products they make [cname, num]

```
SELECT  C.cname, count(*) num
FROM    Product P, Company C
WHERE   P.cid=C.cid
GROUP BY C.cname
```



?

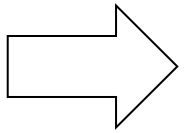
Attributes for GROUP BY revisited

P C

PName	Price	Category	cid	cid	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	1	1	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	1	1	GizmoWorks	25	USA
SingleTouch	$14.99	Photography	2	2	GizmoWorks	65	Japan
MultiTouch	$203.99	Household	3	3	Hitachi	15	Japan

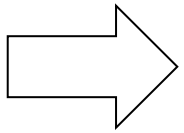
Q: List company names and number of products they make [cname, num]

```
SELECT  C.cname, count(*) num
FROM    Product P, Company C
WHERE   P.cid=C.cid
GROUP BY C.cname
```



CName	num
GizmoWorks	3
Hitachi	1

?



CName	num
GizmoWorks	2
GizmoWorks	1
Hitachi	1

Attributes for GROUP BY revisited

P C

PName	Price	Category	cid	cid	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	1	1	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	1	1	GizmoWorks	25	USA
SingleTouch	$14.99	Photography	2	2	GizmoWorks	65	Japan
MultiTouch	$203.99	Household	3	3	Hitachi	15	Japan

Q: List company names and number of products they make [cname, num]

```
SELECT  C.cname, count(*) num
FROM    Product P, Company C
WHERE   P.cid=C.cid
GROUP BY C.cname
```

CName	num
GizmoWorks	3
Hitachi	1

```
GROUP BY C.cname, C.cid
```

CName	num
GizmoWorks	2
GizmoWorks	1
Hitachi	1

```
GROUP BY C.cid
```

?

Attributes for GROUP BY revisited

P C

PName	Price	Category	cid	cid	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	1	1	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	1	1	GizmoWorks	25	USA
SingleTouch	$14.99	Photography	2	2	GizmoWorks	65	Japan
MultiTouch	$203.99	Household	3	3	Hitachi	15	Japan

Q: List company names and number of products they make [cname, num]

```
SELECT  C.cname, count(*) num
FROM    Product P, Company C
WHERE   P.cid=C.cid
GROUP BY C.cname
```

CName	num
GizmoWorks	3
Hitachi	1

```
GROUP BY C.cname, C.cid
```

```
GROUP BY C.cid
```

Group by can leverage keys!

CName	num
GizmoWorks	2
GizmoWorks	1
Hitachi	1

IN with multi-attributes = rows

Actor
id
fname
lname
gender

Director
id
fname
lname

300

```
SELECT *
FROM director
WHERE (fname, lname) IN
   (SELECT fname, lname
   FROM actor)
```

Does that work ?

IN with multi-attributes = rows

Actor
id
fname
lname
gender

Director
id
fname
lname

300

```
SELECT *
FROM director
WHERE (fname, lname) IN
   (SELECT fname, lname
   FROM actor)
```

9.23.2. IN

```
expression IN (subquery)
```

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-

```
row_constructor IN (subquery)
```

The left-hand side of this form of `IN` is a row constructor, as described in Section 4.2.13. The right-hand side is a parenthesized subquery, which must return exactly as many columns as there are expressions in the left-hand row. The left-hand expressions are evaluated and

4.2.13. Row Constructors

A row constructor is an expression that builds a row value (also called a composite value) using values for its member fields. A row constructor consists of the key word `ROW`, a left parenthesis, zero or more expressions (separated by commas) for the row field values, and finally a right parenthesis. For example:

```
SELECT ROW(1,2.5,'this is a test');
```

The key word `ROW` is optional when there is more than one expression in the list.

Screenshots from: https://www.postgresql.org/docs/current/functions-subquery.html , https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-ROW-CONSTRUCTORS
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

IN with multi-attributes = rows

Actor
id
fname
lname
gender

Director
id
fname
lname

300

```
SELECT *
FROM director
WHERE (fname, lname) IN
   (SELECT fname, lname
   FROM actor)
```

```
SELECT *
FROM director D
WHERE EXISTS
   (SELECT *
   FROM actor A
   WHERE A.fname=D.fnamee
   AND A.lname=D.lnamee)
```

9.23.2. IN

```
expression IN (subquery)
```

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-

```
row_constructor IN (subquery)
```

The left-hand side of this form of `IN` is a row constructor, as described in Section 4.2.13. The right-hand side is a parenthesized subquery, which must return exactly as many columns as there are expressions in the left-hand row. The left-hand expressions are evaluated and

4.2.13. Row Constructors

A row constructor is an expression that builds a row value (also called a composite value) using values for its member fields. A row constructor consists of the key word `ROW`, a left parenthesis, zero or more expressions (separated by commas) for the row field values, and finally a right parenthesis. For example:

```
SELECT ROW(1,2.5,'this is a test');
```

The key word `ROW` is optional when there is more than one expression in the list.

Screenshots from: https://www.postgresql.org/docs/current/functions-subquery.html , https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-ROW-CONSTRUCTORS
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

$\{q(A) \mid \exists r \in R, \exists s \in S[$
$q.A = r.A \wedge r.B = s.B]\}$

(a)

```
SELECT DISTINCT R.A
FROM R, S
WHERE R.B = S.B
```

(b)

```
SELECT DISTINCT R.A
FROM R
WHERE exists
 (SELECT *
 FROM S
 WHERE R.B = S.B)
```

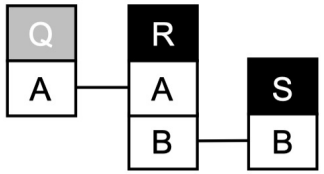
(c)

```
SELECT DISTINCT R.A
FROM R
WHERE R.B in
  (SELECT S.B
  FROM S)
```

(d)

```
SELECT DISTINCT R.A
FROM R
WHERE R.B = any
  (SELECT S.B
  FROM S )
```

(e)



(f)

$\{q(A) \mid \exists r \in R[q.A = r.A \wedge$
$\neg(\exists s \in S[r.B = s.B])]\}$

(g)

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
 (SELECT *
 FROM S
 WHERE R.B = S.B)
```

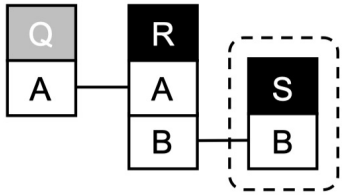
(h)

```
SELECT DISTINCT R.A
FROM R
WHERE R.B not in
 (SELECT S.B
 FROM S)
```

(i)

```
SELECT DISTINCT R.A
FROM R
WHERE R.B <> all
 (SELECT S.B
 FROM S)
```

(j)



(k)

$\{q(A) \mid \exists r \in R[q.A = r.A \wedge$
$\neg(\exists s \in S[r.B < s.B])]\}$

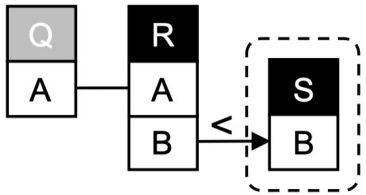
(l)

```
SELECT DISTINCT R.A
FROM R
WHERE not exists
 (SELECT *
 FROM S
 WHERE R.B < S.B)
```

(m)

```
SELECT DISTINCT R.A
FROM R
WHERE R.B >= all
 (SELECT S.B
 FROM S)
```

(n)



(o)

Figure 15: Example 14: SQL has a redundant syntax, especially if interpreted under set semantics ("SELECT DISTINCT"), binary logic (tables contain no null values) and compared with TRC. Here, queries (a)–(e), queries (g)–(j), and queries (l)–(n) are equivalent. On the right, (f), (k), and (o) show the three corresponding Relational Diagrams (Section 4) that abstract away the syntactic variants and focus on the logical patterns of the queries.

The SQL standard

← ICS ← 35 ← 35.060

ISO/IEC 9075-2:2016

Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)

Abstract

Preview

ISO/IEC 9075-2:2016 defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data.

ISO/IEC 9075-2:2016 specifies the syntax and semantics of a database language:

- For specifying and modifying the structure and the integrity constraints of SQL-data.

- For declaring and invoking operations on SQL-data and cursors.

- For declaring database language procedures.

- For embedding SQL-statements in a compilation unit that is otherwise written in a particular programming language (host language).

- For deriving an equivalent compilation unit in the host language. In that equivalent compilation unit, each which invoke an SQL externally-invoked procedure that, when executed, has an effect equivalent to exe-cuting the SQL-statement.

- For direct invocation of SQL-statements.

- To support dynamic preparation and execution of SQL-statements.

ISO/IEC 9075-2:2016 provides a vehicle for portability of data definitions and compilation units between SQL-implementations.

ISO/IEC 9075-2:2016 provides a vehicle for interconnection of SQL-implementations.

Implementations of this ISO/IEC 9075-2:2016 can exist in environments that also support application program-ming languages, end-user query languages, report generator systems, data dictionary systems, program library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

Buy this standard

Format	Language
✓ PDF	English

CHF 198 Buy

General information

Status : ✓ Published

Publication date : 2016-12

Edition : 5

Number of pages : 1707

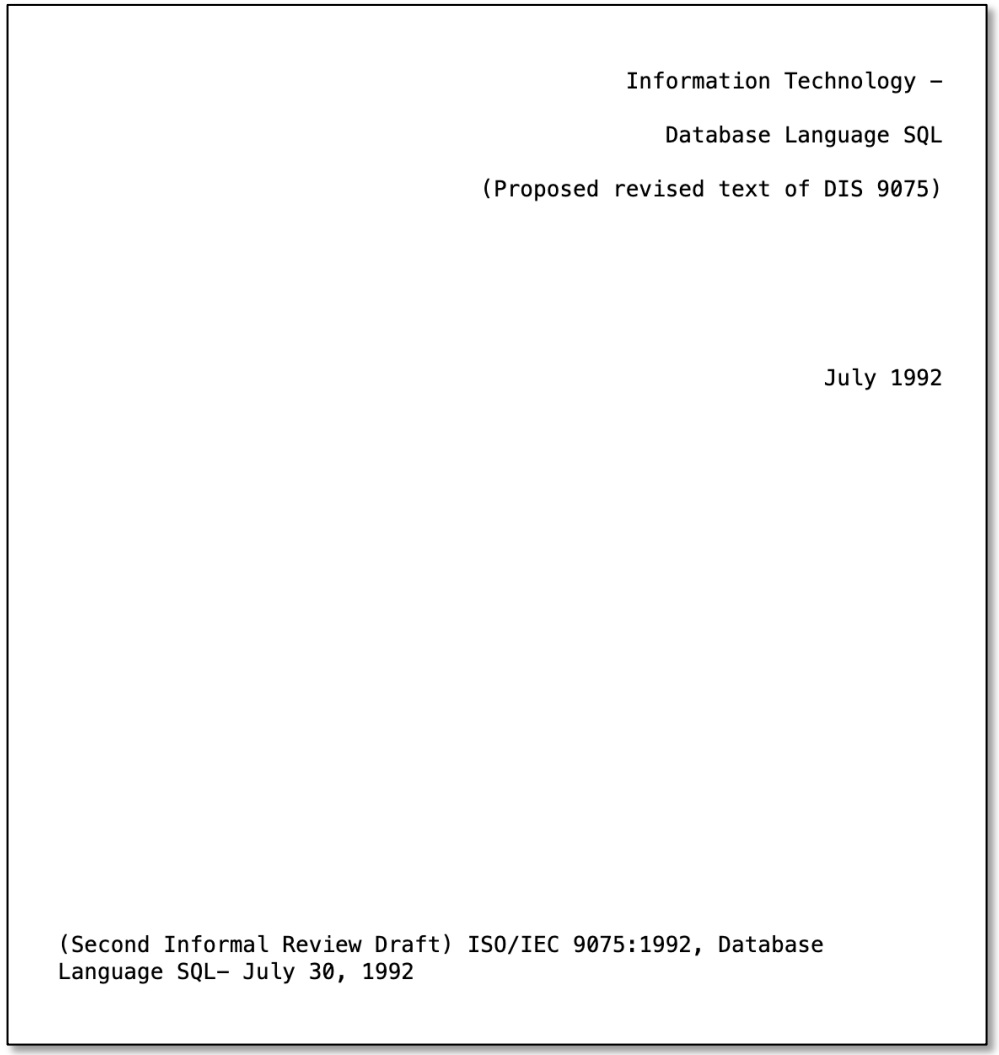
Technical Committee : ISO/IEC JTC 1/SC 32 Data management and interchange

ICS : 35.060 Languages used in information technology

✓ ISO/IEC 9075-2:2016/Cor 2:2022 Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation) — Technical Corrigendum 2

Source: https://www.iso.org/standard/63556.html

The SQL standard



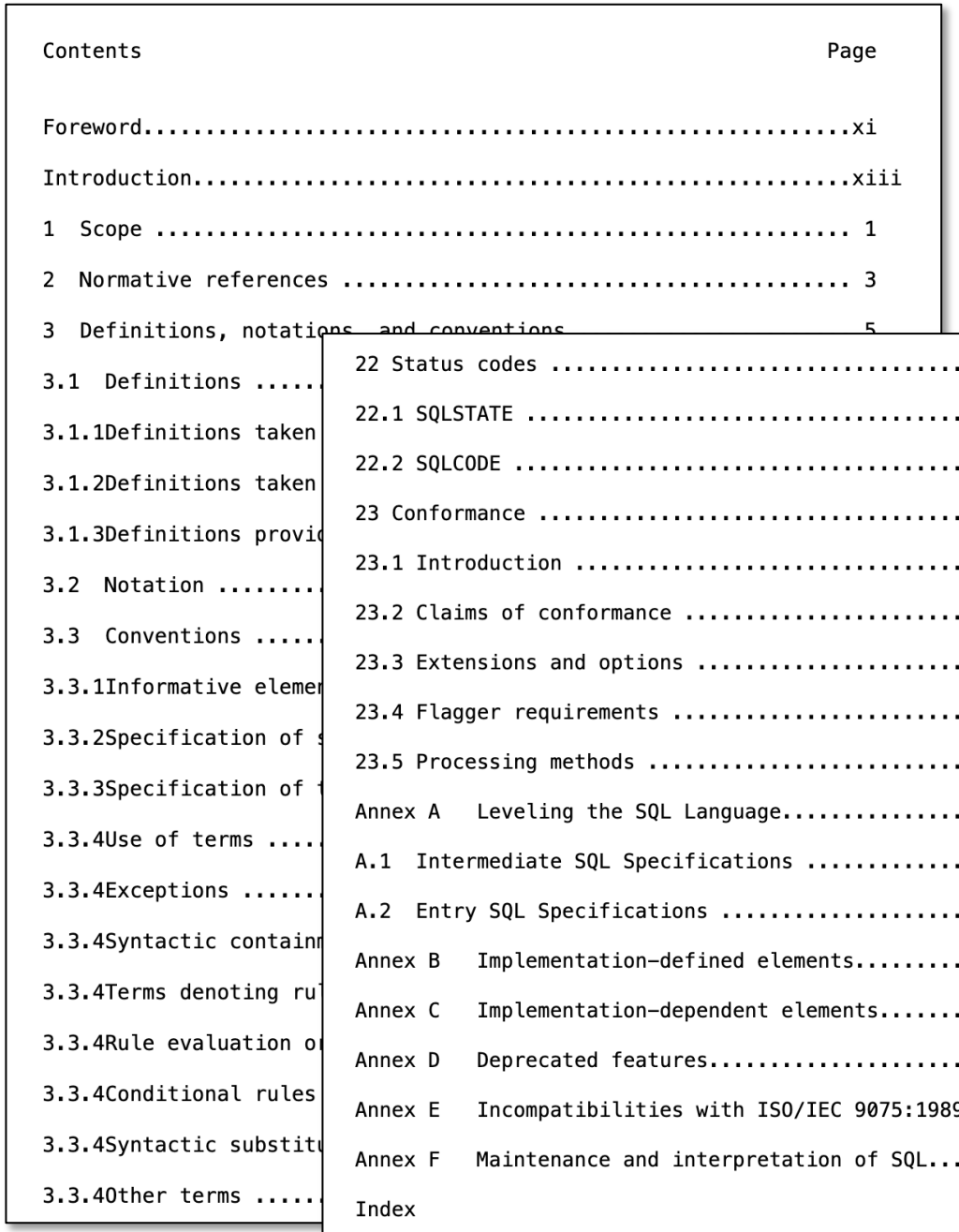
Information Technology -

Database Language SQL

(Proposed revised text of DIS 9075)

July 1992

(Second Informal Review Draft) ISO/IEC 9075:1992, Database Language SQL- July 30, 1992



Contents Page

Table of Contents ix

Source: https://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Thoughts on indendation and capitalization

```
DROP TABLE IF EXISTS Company;
create table Company (
       CName varchar(20) PRIMARY KEY,
       StockPrice int,
       Country varchar(20) );
```

```
CREATE TABLE Company (CName varchar(20) PRIMARY KEY,
                      StockPrice int,
                      Country varchar(20) );
```

```
CREATE TABLE IF NOT EXISTS Company (
  CName varchar(20) PRIMARY KEY,
  StockPrice int,
  Country char(20) );
```

```
create table COMPANY (
  cname VARCHAR(20) PRIMARY KEY,
  stockprice INT,
  country VARCHAR(20));
```

```
CREATE TABLE Company(
cname varchar(20) PRIMARY KEY,
stockprice int,
country varchar(20));
```

```
CREATE TABLE Company (
  CName varchar(20) PRIMARY KEY,
  StockPrice int,
  Country varchar(20)
  );
```

```
create table COMPANY (
  cname varchar(20) PRIMARY KEY,
  stockprice int,
  country varchar(20));
```

```
CREATE TABLE Company (CName varchar(20) PRIMARY KEY, StockPrice int, Country varchar(20));
```

No final word on capitalization

Product (pname, price, category, manufacturer)
Company (cname, stockprice, country)

Q: Find all US companies that manufacture products in the 'Gadgets' category!

```
SELECT   cname
FROM     Product P, Company
WHERE    country = 'USA'
AND      P.category = 'Gadgets'
AND      P.manufacturer = cname
```

~~*My recommendation for capitalization*~~

~~*1. SQL keywords in ALL CAPS,*~~
~~*2. Table names and aliases with Initial Caps*~~
~~*3. Column names all in lowercase.*~~

PostgreSQL treats all in lowercase.
Except if you write:

CREATE TABLE "Product" (…)

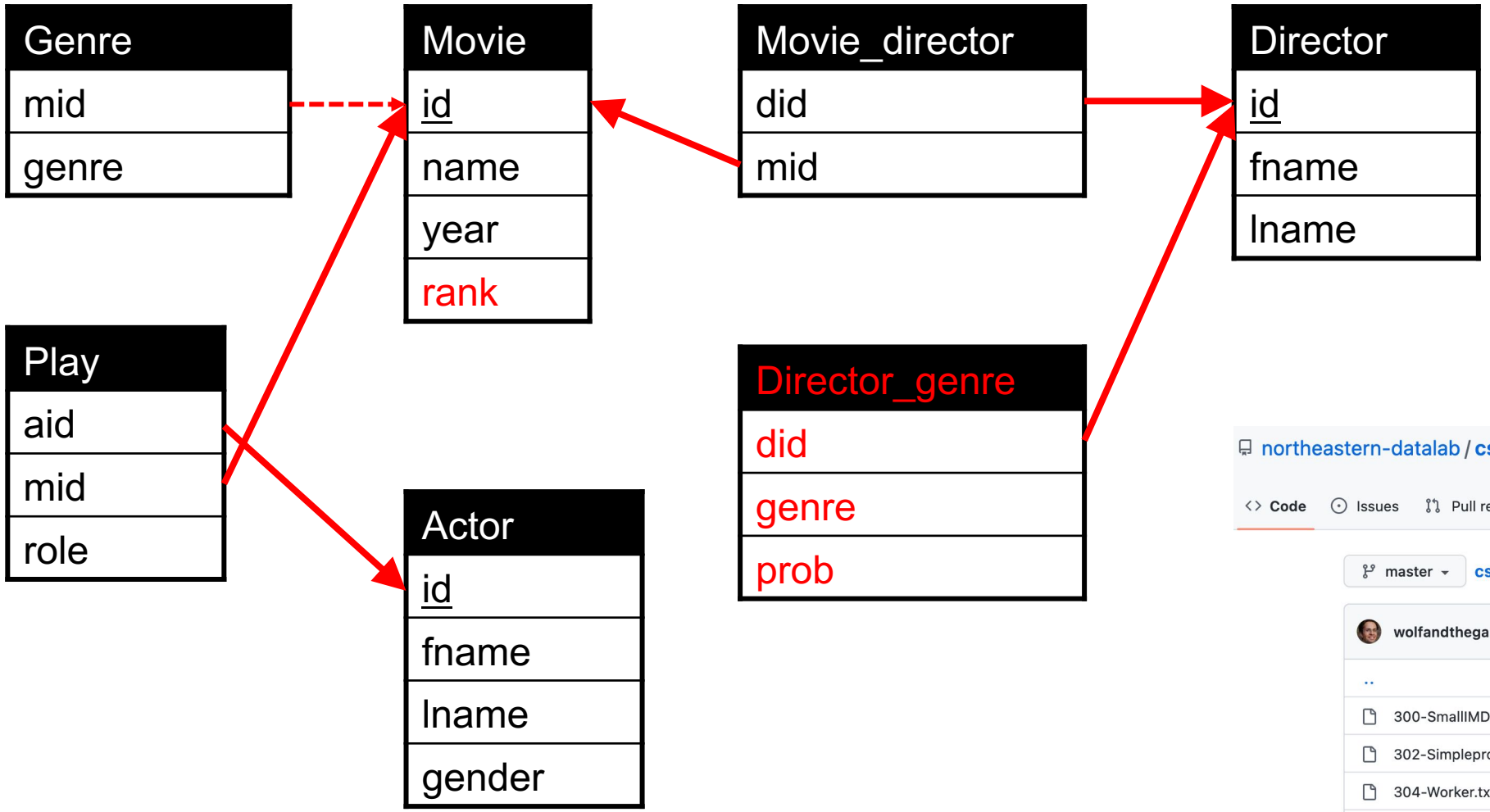
This will preserve capitalization of table name
But … you need to always use quotations

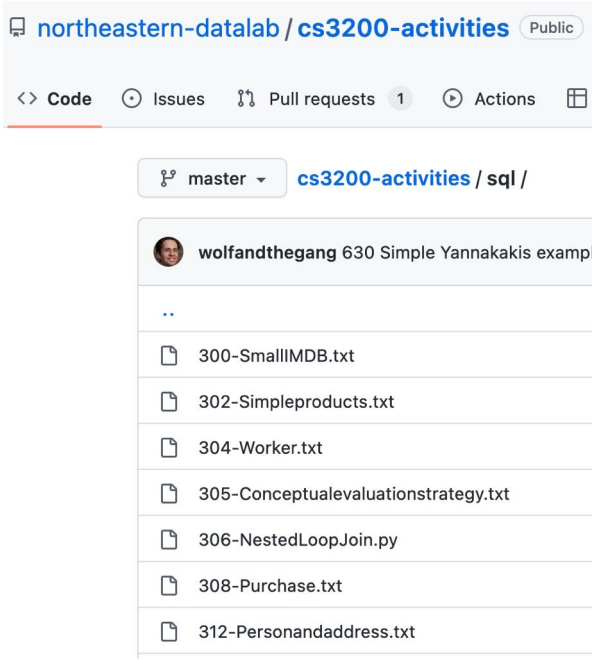
More information: http://blog.lerner.co.il/quoting-postgresql/ , https://stackoverflow.com/questions/6331504/omitting-the-double-quote-to-do-query-on-postgresql

Small IMDB schema

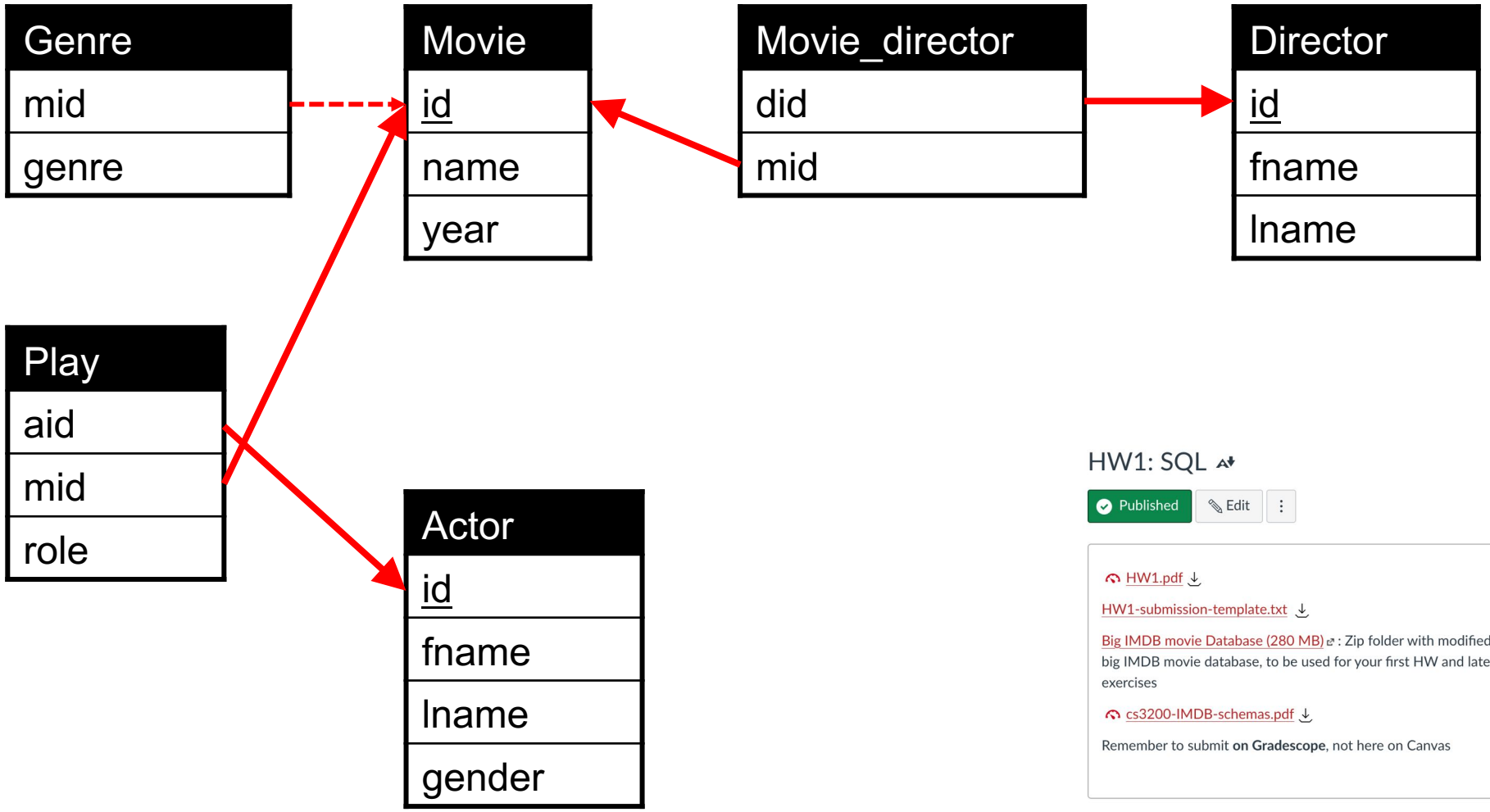
This is a far smaller movie database in our SQL folder

→ 300





Big IMDB schema (Postgres)



HW1: SQL

Published | Edit

HW1.pdf

HW1-submission-template.txt

Big IMDB movie Database (280 MB) : Zip folder with modified big IMDB movie database, to be used for your first HW and later exercises

cs3200-IMDB-schemas.pdf

Remember to submit **on Gradescope**, not here on Canvas

https://drive.google.com/file/d/1k4FwFQEEqygaQvNB38

imdb-cs3200.zip

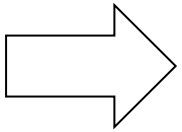
WITH clause = CTEs (Common Table Expressions)

Not discussed in our books. For more info see e.g.:
https://modern-sql.com/feature/with

Subqueries in FROM clause = Derived tables

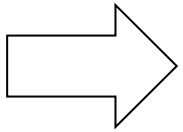
Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



X

Product	TQ
Bagel	40
Banana	70



MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

Q2: Find the maximal total quantities purchased across all products.

```
SELECT MAX(TQ) as MTQ
FROM X
```

Subqueries in FROM clause = Derived tables

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
      FROM Purchase
      GROUP BY product) X
```

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

Q2: Find the maximal total quantities purchased across all products.

```
SELECT MAX(TQ) as MTQ
FROM X
```

Common Table Expressions (CTE): WITH clause

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
      FROM Purchase
      GROUP BY product) X
```

MTQ
70

CTE (Common Table Expression)

Query using CTE

```
WITH  X as
      (SELECT product, SUM(quantity) as TQ
      FROM Purchase
      GROUP BY product)
SELECT MAX(TQ) as MTQ
FROM X
```

The WITH clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

WITH Keyword

Slightly adapted
Student-contributed slide

If you need to use a query and want it to run only once, the WITH keyword lets you define one without it running multiple times (which may happen with a subquery).

WITH {*subqueryName*} AS {subquery}
SELECT …
FROM … *subqueryName*
WHERE …

$y = f(x)$

$z = g(f(x))$ $\quad$ $z = g(y)$

$v = h(f(x))$ $\quad$ $v = h(y)$

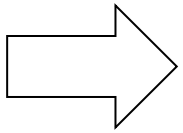
A tricky example to comapre WHERE and HAVING

HAVING vs. WHERE clauses (see links below)

Slightly adapted
Student-contributed slide

Sales	
Product	**SaleAmount**
iPhone	500
Laptop	800
iPhone	1000
Speakers	400
Laptop	600

Query: Find total sales for iPhone and for speakers



Results | Messages

	Product	TotalSales
1	iPhone	1500
2	Speakers	400

1: using WHERE, but not HAVING:

?

2: using HAVING, but not WHERE:

?

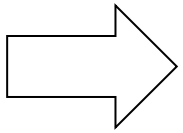
Source: SQL scripts and YouTube video: http://csharp-video-tutorials.blogspot.com/2015/09/difference-between-where-and-having-in.html , https://www.youtube.com/watch?v=15ndzPfa6-E
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

HAVING vs. WHERE clauses (see links below)

Slightly adapted
Student-contributed slide

Sales	
Product	**SaleAmount**
iPhone	500
Laptop	800
iPhone	1000
Speakers	400
Laptop	600

Query: Find total sales for iPhone and for speakers



Results | Messages

	Product	TotalSales
1	iPhone	1500
2	Speakers	400

1: using WHERE, but not HAVING:

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product IN ('iPhone','Speakers')
GROUP BY Product
```

2: using HAVING, but not WHERE:

?

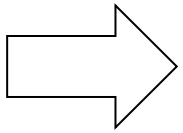
Source: SQL scripts and YouTube video: http://csharp-video-tutorials.blogspot.com/2015/09/difference-between-where-and-having-in.html , https://www.youtube.com/watch?v=15ndzPfa6-E
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

HAVING vs. WHERE clauses (see links below)

Slightly adapted
Student-contributed slide

Sales	
Product	**SaleAmount**
iPhone	500
Laptop	800
iPhone	1000
Speakers	400
Laptop	600

Query: Find total sales for iPhone and for speakers



Results | Messages

	Product	TotalSales
1	iPhone	1500
2	Speakers	400

1: using WHERE, but not HAVING:

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product IN ('iPhone','Speakers')
GROUP BY Product
```

2: using HAVING, but not WHERE:

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING Product IN ('iPhone','Speakers')
```

Source: SQL scripts and YouTube video: http://csharp-video-tutorials.blogspot.com/2015/09/difference-between-where-and-having-in.html , https://www.youtube.com/watch?v=15ndzPfa6-E
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Witnesses

Motivation: What should these "queries" return?

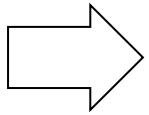
Product

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products [cid, cname, mp]



?

Motivation: What should these "queries" return?

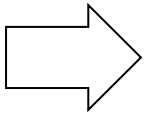
Product

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

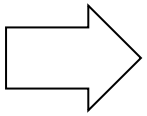
Find for each company id, the maximum price amongst its products [cid, cname, mp]



cid	cname	mp
1	GizmoWorks	20
2	Apple	300

cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products [cid, cname, mp, pname]



?

Motivation: What should these "queries" return?

Product

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products [cid, cname, mp]

cid	cname	mp
1	GizmoWorks	20
2	Apple	300

cid	mp
1	20
2	300

Find for each company id, the product(s) with maximum price amongst its products (Recall that "group by cid" can just give us one single tuple per cid)

cid	cname	mp	pname
1	GizmoWorks	20	SuperGizmo
2	Apple	300	iTouch1
2	Apple	300	iTouch2

Witnesses: simple (1/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Witnesses: simple (2/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery

1.
```
SELECT   max(P1.price)
FROM     Product P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (3/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...

2.
```
SELECT  P2.pname, P2.price
FROM    Product P2
```

1.
```
SELECT  max(P1.price)
FROM    Product P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (4/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...
  and compare the price with the max price

```
SELECT  P2.pname, P2.price
FROM    Product P2
WHERE   P2.price =
        (SELECT  max(P1.price)
        FROM     Product P1)
```

Witnesses: with joins (1/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Witnesses: with joins (2/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company

Witnesses: with joins (2/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company

2.
```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company C2, Product P2
WHERE   C2.cid = P2.cid
```

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
- 3. Compare the price with the max price

2.
```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company C2, Product P2
WHERE   C2.cid = P2.cid
```

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (4/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
- 3. Compare the price with the max price

```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company C2, Product P2
WHERE   C2.cid = P2.cid
    and P2.price =
        (SELECT max(P1.price)
        FROM    Product P1
        WHERE   P1.cid = C2.cid)
```

How many aliases do we actually need?

Witnesses: with joins (5/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
  and compare the price with the max price

```
SELECT  cname, pname, price
FROM    Company, Product
WHERE   Company.cid = Product.cid
    and price =
        (SELECT max(price)
        FROM    Product
        WHERE   cid = Company.cid)
```

We need no single alias here.

Next: can we eliminate the max operator in the inner query?

Witnesses: with joins (6/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute all prices in a subquery, for a given company
- 2. Compute each product and its price, per company
  and compare the price with the all prices

```
SELECT  cname, pname, price
FROM    Company, Product
WHERE   Company.cid = Product.cid
    and price >= ALL
        (SELECT price
        FROM    Product
        WHERE   cid = Company.cid)
```

But: "ALL" does not work in SQLite ☹

Witnesses: with joins (6/6)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute all prices in a subquery, for a given company
- 2. Compute each product and its price, per company
  and compare the price with the all prices

```
SELECT  cname, pname, price
FROM    Company, Product
WHERE   Company.cid = Product.cid
    and price >= ALL
        (SELECT price
        FROM    Product
        WHERE   cid = Company.cid)
```

Next: can you write this query as uncorrelated nested query (e.g. with WITH clause)

?

Witnesses: with FROM (1/4)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Another Plan:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

1.
```
SELECT  cid, max(price) as MP
FROM    Product
GROUP   BY cid
```

Finding the maximum price for each company was easy.
But we want the "witnesses", i.e. the products with max price.

Witnesses: with FROM (2/4)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Another Plan:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

```
2. SELECT  C2.cname, P2.pname, X.MP
   FROM    Company C2, Product P2,
         1. ( SELECT   cid, max(price) as MP
              FROM     Product
              GROUP    BY cid                  ) as X
   WHERE   C2.cid = P2.cid
     and   C2.cid = X.cid
     and   P2.price = X.MP
```

Let's write the same query with a "WITH" clause

Witnesses: with FROM (3/4)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Another Plan with WITH:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

```
WITH  X as
      (SELECT   cid, max(price) as MP
      FROM      Product
      GROUP     BY cid)
SELECT C2.cname, P2.pname, X.MP
FROM   Company C2, Product P2, X
WHERE  C2.cid = P2.cid
   and C2.cid = X.cid
   and P2.price = X.MP
```

Witnesses: with FROM (4/4)

Product (pname, price, cid)
Company (cid, cname, city)

Q: For each company, find the most expensive product + its price

Another Plan with WITH:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

```
WITH X (cid, cname, MP) as
        (SELECT   C.cid, cname, max(price)
        FROM      Product P, Company C
        WHERE     C.cid = P.cid
        GROUP     BY C.cid)
SELECT  X.cname, P.pname, X.MP
FROM    Product P, X
WHERE   P.cid = X.cid
    and P.price = X.MP
```

~~SELECT P.cid, cname, max(price)~~
~~FROM Product P, Company C~~
~~WHERE C.cid = P.cid~~
~~GROUP BY P.cid~~

Notice that above query does not work. Do you see why?
If not, observe the error message!

Subqueries in

SELECT clause

FROM clause

WHERE clause

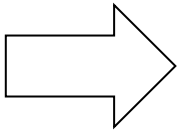
HAVING clause (similar to WHERE)

Witnesses: with aggregates per group (1/10)

First: How to get the product that is sold with maximum price?

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	mp
Banana	4

```
SELECT   product, max(price) as mp
FROM
WHERE
GROUP BY
HAVING
```

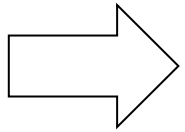
???

Witnesses: with aggregates per group (2/10)

First: How to get the product that is sold with maximum price?

Purchase *1) Find the maximum price*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



(no name)
4

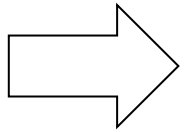
```
SELECT  max(price)
FROM    Purchase
```

Witnesses: with aggregates per group (3/10)

First: How to get the product that is sold with maximum price?

Purchase *2) Now you need to find product with price = maximum price*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	mp
Banana	4

```
SELECT  P2.product, P2.price as mp
FROM    Purchase P2
WHERE   P2.price = (
            SELECT  max(price)
            FROM    Purchase
)
```

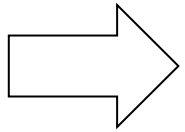
Witnesses: with aggregates per group (4/10)

First: How to get the product that is sold with maximum price?

Another way to formulate this query

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	mp
Banana	4

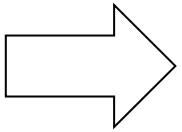
```
SELECT   P2.product, P2.price as mp
FROM     Purchase P2
WHERE    P2.price >= ALL (
             SELECT   price
             FROM     Purchase
)
```

Witnesses: with aggregates per group (5/10)

Second: How to get the product that is sold with max sales (=quanities sold).

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	sales
Banana	70

```
SELECT
FROM
WHERE
GROUP BY
HAVING
```

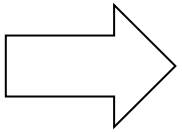
?

Witnesses: with aggregates per group (6/10)

Second: How to get the product that is sold with max sales (=quanities sold).

Purchase *1: find the total sales (sum of quantity) for each product*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	sales
Bagel	40
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY product
```

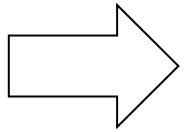
Witnesses: with aggregates per group (7/10)

Second: How to get the product that is sold with max sales?

2: we can use the same query as nested query

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



(no name)
40
70

```
SELECT max(Q)
FROM (
    SELECT    sum(quantity)
    FROM      Purchase
    GROUP BY product
) as Q
) X
```

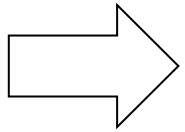
Witnesses: with aggregates per group (8/10)

Second: How to get the product that is sold with max sales?

3: putting things together

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	sales
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY  product
HAVING    sum(quantity) =
          (SELECT max (Q)
          FROM   (SELECT    sum(quantity) Q
                  FROM      Purchase
                  GROUP BY  product) X )
```

Next: Can you write the query with only 2 nestings?

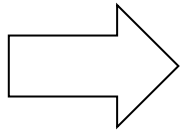
Witnesses: with aggregates per group (9/10)

Second: How to get the product that is sold with max sales?

Another way to formulate this query with "ALL"

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	sales
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY product
HAVING    sum(quantity) >= ALL (
            SELECT    sum(quantity)
            FROM      Purchase
            GROUP BY product
          )
```

Next: Can you write the query without having to repeat the SUM aggregate twice?

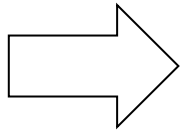
Witnesses: with aggregates per group (10/10)

Second: How to get the product that is sold with max sales?

Another way to formulate this query with a CTE (= WITH clause)

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

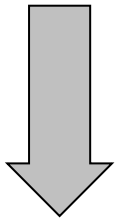


Product	sales
Banana	70

```
WITH X as
      (SELECT product, sum(quantity) as sales
      FROM Purchase
      GROUP BY product)
SELECT product, sales
FROM X
WHERE sales =
      (SELECT max (sales)
      FROM X)
```

WITH clause: temporary relations


```
SELECT  pname, price
FROM    Product
WHERE   price =
        (SELECT  max(price)
        FROM     Product)
```



```
WITH    Max_price(value) as
        (SELECT  max(price)
        FROM     Product)
SELECT  pname, price
FROM    Product, Max_price
WHERE   price = value
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

CTE (Common Table Expression)

(In this example, the result is a **scalar**)

Query using CTE

WITH clause: temporary relations


```
SELECT  pname, price
FROM    Product
WHERE   price =
        (SELECT  max(price)
        FROM     Product)
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

```
WITH    Max_price as
        (SELECT  max(price) as value
        FROM     Product)
SELECT  pname, price
FROM    Product, Max_price
WHERE   price = value
```