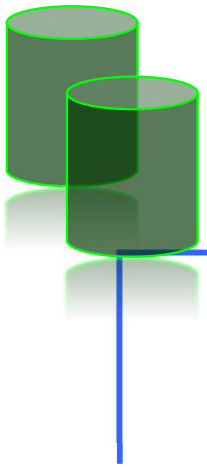


CZ2007
Introduction to Databases

NANYANG TECHNOLOGICAL UNIVERSITY

Querying Relational Databases using SQL
Part-2

Dr. Quah T.S., Jon

School of Computer Science and Engineering
Nanyang Technological University, Singapore

Summary and roadmap

NANYANG TECHNOLOGICAL UNIVERSITY

- Introduction to SQL
- SELECT
  FROM
  WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins

- Next
  - Subquery
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - …

Subqueries

- A subquery is an SQL query nested inside a larger query
- Queries with subqueries are referred to as nested queries
- A subquery may occur in
  - SELECT
  - FROM **SQL subquery**
  - WHERE **SQL subquery**

A special subquery: Scalar Subquery

Scalar Subquery

- return a single value which is then used in a comparison.
- If query is written so that it expects a subquery to return a single value, and it returns multiple values or no values, a run-time error occurs.

Example Query

From Sells(bar, beer, price), find the bars that serve Heineken for the same price Junior bar charges for Tiger.

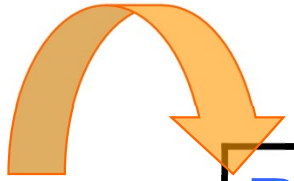
Example Scalar Subquery

Sells

Bar	Beer	Price
Clinic	Heineken	8.00
Clinic	Tiger	6.60
Junior	Tiger	7.90
MOS	Heineken	7.90
Junior	Heineken	8.00

Find the price Junior charges for Tiger.

```
SELECT    price
FROM      Sells
WHERE     bar = `Junior'
AND beer = `Tiger';
```



Price
7.90

Find the bars that serve Heineken at that price.

```
SELECT    bar
FROM      Sells
WHERE     beer = `Heineken'
          AND price = 7.90;
```

Bar
MOS

Example Scalar Subquery

```
SELECT   bar
FROM     Sells
WHERE    beer = 'Heineken' AND
         price = (SELECT price
                  FROM   Sells
                  WHERE bar = 'Junior'
                    AND beer = 'Tiger');
```

Without using Scalar Subquery, how ?

```
SELECT   S1.bar
FROM     Sells S1, Sells S2
WHERE    S1.beer = 'Heineken'
AND        S2.bar = 'Junior'
AND        S2.beer = 'Tiger'
AND        S1.price = S2.price;
```

Use two copies of the table

Subqueries in FROM

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all products in the 'phone' category with prices under 1000
- SELECT X.PName
  FROM (SELECT *
  FROM Product
  WHERE category = 'Phone') AS X
  WHERE X.Price < 1000

Subqueries in FROM (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all products in the 'phone' category with prices under 1000
- SELECT PName
  FROM Product
  WHERE Category = 'Phone'
  AND Price < 1000
- This is a much more efficient solution

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that make some products with price < 100
- SELECT DISTINCT CName
  FROM Company AS X
  WHERE X.CName IN
  (SELECT Y.CName
  FROM Product AS Y
  WHERE Y.Price < 100)

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that make some products with price < 100
- SELECT DISTINCT CName
  FROM Company AS X
  WHERE X.CName IN
  (SELECT *
  FROM Product AS Y
  WHERE Y.Price < 100)

Error!

•The number of attributes in the SELECT clause in the subquery must match the number of attributes compared to with the comparison operator.

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that make some products with price < 100
- SELECT DISTINCT CName
  FROM Company AS X
  WHERE EXISTS
  (SELECT * FROM Product AS Y
  WHERE X.CName = Y.Cname
  AND Y.Price < 100)

•A nested query is **correlated** with the outer query if it contains a reference to an attribute in the outer query.

•A nested query is ***correlated*** with the outside query if it must be re-computed for every tuple produced by the outside query.

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that make some products with price < 100
- SELECT DISTINCT CName
  FROM Company AS X
  WHERE 100 > ANY
  (SELECT Price FROM Product AS Y
  WHERE X.CName = Y.Cname)

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that make some products with price < 100
- SELECT DISTINCT CName
  FROM Product
  WHERE Price < 100
- This is more efficient than the previous solutions

Operators in Subqueries

IN

<tuple> IN <relation> is true if and only if the tuple is a member of the relation.

ANY/SOME

x = ANY(<relation>) is a boolean cond. meaning that *x* equals at least one tuple in the relation.

EXISTS

- EXISTS(<relation>) is true if and only if the <relation> is not empty.
- Returns true if the nested query has 1 or more tuples.

ALL

x <> ALL(<relation>) is true if and only if for every tuple *t* in the relation, *x* is not equal to *t*.

Note: The keyword NOT can proceed any of the operators (*s* NOT IN *R)*

Avoiding Nested Queries

- In general, nested queries tend to be more inefficient than un-nested queries
  - query optimizers of DBMS do not generally do a good job at optimizing queries containing subqueries
- Therefore, they should be avoided whenever possible
- But there are cases where avoiding nested queries is hard…

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that do not make any product with price < 100
- SELECT DISTINCT CName
  FROM Company AS X
  WHERE NOT EXISTS
  (SELECT * FROM Product AS Y
  WHERE X.CName = Y.Cname
  AND Y.Price < 100)

Subqueries in WHERE (cont.)

Company

CName	StockPrice	Country
...	...	...

Product

PName	Price	Category	CName
...	...	...	...

- Find all companies that do not make any product with price < 100
- SELECT DISTINCT CName
  FROM Company AS X
  WHERE 100 <= ALL
  (SELECT Price FROM Product AS Y
  WHERE X.CName = Y.Cname)

Subquery – Rules to Remember

- The ORDER BY clause may not be used in a subquery.

- Column names in a subquery refer to the table name in the FROM clause of the subquery by default.

Summary and roadmap

- Introduction to SQL
- SELECT
  FROM
  WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery

- Next
  - Aggregations
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - …

Reference: Chapter 6.3 of our TextBook

Aggregation

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
…	…	…

- What is the average price of the models from Toyota?
- How many models are there from BMW?

Aggregation: Count

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
…	…	…

- Count the number of car models from Toyota:
- SELECT COUNT(*)
  FROM Cars
  WHERE Maker = 'Toyota'

Aggregation: Count (cont.)

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
i8	BMW	50

- Count the number of car makers:
- SELECT COUNT(Maker)
  FROM Cars

Error!

Aggregation: Count (cont.)

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
i8	BMW	50

- Count the number of car makers:
- SELECT COUNT(DISTINCT Maker)
  FROM Cars

Aggregation: Average

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
…	…	…

- Compute the average price of car models from Toyota:
- SELECT AVG(Price)
  FROM Cars
  WHERE Maker = 'Toyota'

Aggregation: Min

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
…	…	…

- Compute the minimum price of car models from Toyota:
- SELECT MIN(Price)
  FROM Cars
  WHERE Maker = 'Toyota'

Aggregation: Max

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
…	…	…

- Compute the maximum price of car models from Toyota:
- SELECT MAX(Price)
  FROM Cars
  WHERE Maker = 'Toyota'

Aggregation: Sum

Cars

Model	Maker	Price
Corolla	Toyota	1000
E89	BMW	2000
…	…	…

- Compute the sum of prices of car models from Toyota:
- SELECT SUM(Price)
  FROM Cars
  WHERE Maker = 'Toyota'

Aggregation: Sum (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- Compute the gross sales of oranges:
- SELECT SUM(Price * Quantity)
  FROM Purchase
  WHERE Product = 'Orange'

Aggregation: Sum (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- Is it possible to obtain this?
- Yes
- Use GROUP BY

Product	GrossSales
Orange	80
Banana	90

Aggregation: Group By

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, SUM(Price * Quantity) AS GrossSales
  FROM Purchase
  GROUP BY Product

Product	GrossSales
Orange	80
Banana	90

Aggregation: Group By (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, Date, SUM(Price * Quantity) AS GrossSales,
  FROM Purchase
  GROUP BY Product, Date

Product	Date	GrossSales
Banana	2011.1.1	10
Orange	2011.1.1	30
...	...	...

Aggregation: Group By (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
~~Banana~~	~~2011.1.2~~	~~1~~	~~20~~
Banana	2011.1.3	4	15

- SELECT Product, SUM(Price * Quantity) AS GrossSales
  FROM Purchase
  WHERE Price >= 2
  GROUP BY Product

Product	GrossSales
Orange	80
Banana	70

Aggregation: Group By (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, SUM(Quantity) AS TotalQuan, Max(Price) as MaxPrice
  FROM Purchase
  GROUP BY Product

Product	TotalQuan	MaxPrice
Orange	20	5
Banana	40	4

Aggregation: Having (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, SUM(Quantity) AS TotalQuan
  FROM Purchase
  GROUP BY Product
  HAVING Max(Price) > 4

Product	TotalQuan	MaxPrice
Orange	20	5

Aggregation: Having (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, SUM(Quantity) AS TotalQuan
  FROM Purchase
  GROUP BY Product
  HAVING Max(Price) > 4
  OR TotalQuan > 20

Product	TotalQuan	MaxPrice
Orange	20	5
Banana	40	4

Common Mistakes

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, Date,
  SUM(Quantity) AS TotalQuan
  FROM Purchase
  GROUP BY Product

Error!

- Anything in the SELECT list should either be (i) an aggregate function or (ii) in the GROUP BY list

Common Mistakes (cont.)

Purchase

Product	Date	Price	Quantity
Orange	2011.1.1	3	10
Banana	2011.1.1	2	5
Orange	2011.1.2	5	10
Banana	2011.1.2	1	20
Banana	2011.1.3	4	15

- SELECT Product, Date,
  SUM(Quantity) AS TotalQuan
  FROM Purchase
  GROUP BY Product
  HAVING Price > 2

Error!

- The HAVING clause specifies conditions on each group, but not conditions on each tuple. Anything in Having should either be (i) an aggregate function or (ii) in the GROUP BY list

HAVING vs. Where

Purchase(product, date, price, quantity)

- Find total sales after 10/1/2005 per product, for those products that have more than 100 buyers.

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING   SUM(quantity) > 100
```

HAVING clauses contains conditions on **groups**

*Whereas WHERE clauses condition on **individual tuples**…*

A summary of Grouping and Aggregation

NANYANG TECHNOLOGICAL UNIVERSITY

```
SELECT     S
FROM       R1,…,Rn
WHERE      C1
GROUP BY   a1,…,ak
HAVING     C2
```

Evaluation steps:

1. Evaluate FROM-WHERE: apply condition C_1 on the attributes in $R_1,\ldots,R_n$
2. GROUP BY the attributes $a_1,\ldots,a_k$
3. Apply condition C_2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Exercise

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the number of usernames under each real name
- SELECT COUNT(*)
  FROM Author
  GROUP BY RealName

Exercise

Author

UserName	RealName
...	...

Wrote

UserName	Article
...	...

- Find the number of articles written by each user, group by real names
- SELECT RealName, COUNT(*)
  FROM Author AS A, Wrote AS W
  WHERE A.UserName = W.UserName
  GROUP BY RealName

Exercise

Author

UserName	RealName
...	...

Wrote

UserName	Article
...	...

- Find the real names of the persons who wrote more than 10 articles
- SELECT RealName
  FROM Author AS A, Wrote AS W
  WHERE A.UserName = W.UserName
  GROUP BY RealName
  HAVING COUNT(*) > 10

Exercise

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the real names of the persons who wrote more than 10 articles
- SELECT DISTINCT RealName
  FROM Author AS A
  WHERE 10 <
  (SELCT COUNT(Wrote.Article)
  FROM Wrote AS W
  WHERE A.UserName = W.UserName)

Error: it only counts the number of articles of each username. However, a realname may have multiple usernames

Summary and roadmap

- Introduction to SQL
- SELECT
  FROM
  WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations

- Next
  - UNION, INTERSECT, EXCEPT
  - NULL
  - Outerjoin
  - …

Union

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the usernames (i) with real names starting with 'Chris' OR (ii) have written more than 10 articles
- (SELECT A.UserName
  FROM Author AS A
  WHERE RealName LIKE 'Chris%')
  UNION
  (SELECT W.UserName
  FROM Wrote AS W
  GROUP BY W.UserName
  HAVING COUNT(*) > 10)

(subquery)

UNION

(subquery)

Union

Author

UserName	RealName
...	...

Wrote

UserName	Article
...	...

- Find the usernames (i) with real names starting with 'Chris' OR (ii) have written more than 10 articles
- (SELECT A.UserName
  FROM Author AS A
  WHERE RealName LIKE 'Chris%')
  UNION
  (SELECT W.UserName
  FROM Wrote AS W
  GROUP BY W.UserName
  HAVING COUNT(*) > 10)

Note: UNION automatically removes duplicates

Union

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the usernames (i) with real names starting with 'Chris' OR (ii) have written more than 10 articles
-
```
(SELECT *
 FROM Author AS A
 WHERE RealName LIKE 'Chris%')
UNION
(SELECT *
 FROM Wrote AS W
 GROUP BY W.UserName
 HAVING  COUNT(*) > 10)
```

Error!

Intersect (Not in some DBMS)

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the usernames (i) with real names starting with 'Chris' AND (ii) have written more than 10 articles
-
```
(SELECT A.UserName
 FROM Author AS A
 WHERE RealName LIKE 'Chris%')
INTERSECT
(SELECT W.UserName
 FROM Wrote AS W
 GROUP BY W.UserName
 HAVING  COUNT(*) > 10)
```

Except (Not in some DBMS)

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the usernames who wrote more than 10 articles but do not have a real name starting with 'Chris'
-
```
(SELECT W.UserName
 FROM Wrote AS W
 GROUP BY W.UserName
 HAVING  COUNT(*) > 10)
EXCEPT
(SELECT A.UserName
 FROM Author AS A
 WHERE A.RealName LIKE 'Chris%')
```

Alternative solution?

Author

UserName	RealName
…	…

Wrote

UserName	Article
…	…

- Find the usernames who wrote more than 10 articles but do not have a real name starting with 'Chris'
- SELECT W.UserName
  FROM Wrote AS W, Author AS A
  WHERE RealName NOT LIKE 'Chris%'
  AND W.UserName = A.UserName
  GROUP BY W.UserName
  HAVING COUNT(*) > 10)

Exercise

- Player (PID, PName, Ranking, Age)
  Court (CID, Type, Location)
  Reserves (PID, CID, Date)
- Find the names of the players who have reserved courts of 'Clay' type
- SELECT PName
  FROM Player, Court, Reserves
  WHERE Player.PID = Reserves.PID AND
  Court.CID = Reserves.CID AND
  Type='Clay';

Exercise

- Player (PID, PName, Ranking, Age)
  Court (CID, Type, Location)
  Reserves (PID, CID, Date)
- Find the PID of the players who have reserved 'Clay' courts but not 'Grass' courts
- (SELECT R1.PID
  FROM Court AS C1, Reserves AS R1
  WHERE R1.CID = C1.CID AND Type = 'Clay')
  EXCEPT
  (SELECT R2.PID
  FROM Court AS C2, Reserves AS R2
  WHERE R2.CID = C2.CID AND Type = 'Grass')

Exercise

- Player (PID, PName, Ranking, Age)
  Court (CID, Type, Location)
  Reserves (PID, CID, Date)
- Find the PIDs of players who have a ranking of 3 or who have reserved court with CID 100
- (SELECT Player.PID
  FROM Player
  WHERE Ranking = 3)
  UNION
  (SELECT Reserves.PID
  FROM Reserves
  WHERE CID = 100)

Exercise

- Player (PID, PName, Ranking, Age)
  Court (CID, Type, Location)
  Reserves (PID, CID, Date)
- Find the PIDs of players who have NOT reserved a 'Clay' court before
- ```
  (SELECT  Player.PID
   FROM    Player)
  EXCEPT
  (SELECT  Reserves.PID
   FROM    Reserves, Court
   WHERE   Reserves.CID = Court.CID
           AND Type = 'Clay')
  ```
  ```

Exercise

- Player (PID, PName, Ranking, Age)
  Court (CID, Type, Location)
  Reserves (PID, CID, Date)
- Find the PIDs of players who have reserved each court at least once
- ```
  SELECT  P.PID
  FROM    Player AS P
  WHERE NOT EXIST
     ( (SELECT C.CID
        FROM Court AS C
        EXCEPT
       (SELECT R.CID
        FROM Reserves AS R
        WHERE R.PID = P.PID) )
  ```
  ```

A bit theory: Bag Semantics vs. Set Semantics

- Set semantics → No duplicates, each item appears only once
  - Default for UNION, INTERSECT, and EXCEPT is **set**

- Bag semantics → Duplicates allowed, i.e., a multiset
  - Default for SELECT-FROM-WHERE is **bag**

How to change the default?

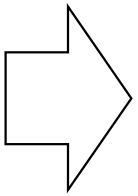
- Force set semantics with **DISTINCT** after SELECT
- Force bag semantics with **ALL** after UNION, etc.

DISTINCT: Change Bag Semantics to Set Semantics

Product

PName	Price	Category
iPhone x	888	Phone
iPad	668	Tablet
Mate 10	798	Phone
EOS 550D	1199	Camera

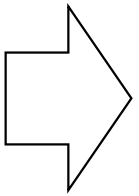
```
SELECT DISTINCT Category
FROM   Product
```



Category
Phone
Tablet
Camera

Versus

```
SELECT Category
FROM   Product
```



Category
Phone
Tablet
Phone
Camera

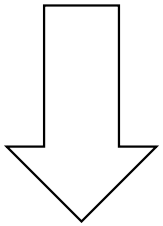
ALL: Change Set Semantics to BAG Semantics

Product_A

PName	Price
iPhone x	888
iPad	668
Mate 10	798
EOS 550D	1199

Product_B

PName	Price
iPhone x	888
Mate 20	798



```
(SELECT *
 FROM Product_A)
UNION ALL
(SELECT *
 FROM Product_)
```

PName	Price
iPhone x	888
iPad	668
Mate 10	798
EOS 550D	1199
iPhone x	888
Mate 20	798

Summary and roadmap

- Introduction to SQL
- SELECT
  FROM
  WHERE
- Eliminating duplicates
- Renaming attributes
- Expressions in SELECT Clause
- Patterns for Strings
- Ordering
- Joins
- Subquery
- Aggregations
- UNION, INTERSECT, EXCEPT

- Next
  - NULL
  - Outerjoin
  - Insert/Delete tuples
  - Create/Alter/Delete tables
  - …

Reference: Chapter 6.2 & 6.4 of our TextBook