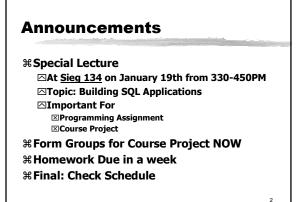
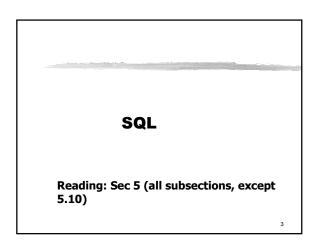
Introduction to Database Systems

CSE 444

Lecture #3 Jan 10 2001





Selection and Projection SELECT name, stockPrice FROM Company WHERE country="USA" AND stockPrice > 50 Input schema: Company(sticker, name, country, stockPrice) Output schema: R(name, stock price)

Removing Duplicates

Product(pid, name, maker, category, price)

SELECTDISTINCT categoryFROMProductWHEREprice > 100

Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: find total sales for the entire database

SELECT Sum(price * quantity) FROM Purchase

Example 1': find total sales of bagels

SELECTSum(price * quantity)FROMPurchaseWHEREproduct = 'bagel'

5

Grouping,	Aggregation
-----------	-------------

Purchase(product, date, price, quantity)

Example 2: find total sales after 9/1 per product.

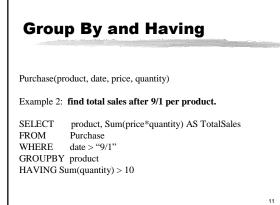
SELECTproduct, Sum(price*quantity) AS TotalSalesFROMPurchaseWHEREdate > "9/1"GROUPBYproduct

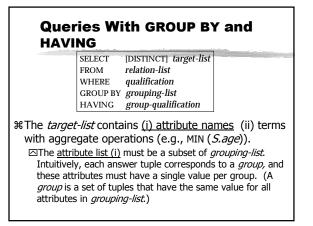
Product	Date	Price	Quantity
Banana	10/19	0.52	17
Banana	10/22	0.52	7
Bagel	10/20	0.85	20
Bagel	10/21	0.85	15

Γ

Bagel \$29.75 Banana \$12.48 SELECT product, Sum(price*quantity) AS TotalSales FROM Purchase			
SELECT product, Sum(price*quantity) AS TotalSales	gel	\$29.75	
	nana	\$12.48	
WHERE date > "9/1" GROUPBY product	OM Purc	hase	juantity) AS TotalSales

Exam	ple		
Product	SumSales	MaxQuantity	
Banana	\$12.48	17	
Bagel	\$29.75	20	
or every pro	duct, what is the total	sales and max quan	ity sold?
SELECT		* quantity) AS Sum ity) AS MaxQuantit	
FROM	Purchase		
GROUP BY	product		





Conceptual Evaluation

- #The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary'* fields are deleted, as before.
- **%** The remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- **#**The *group-qualification* is then applied to eliminate some groups.

#One answer tuple is generated per qualifying group.

Find the age of the youngest sailor with age 18, for each rating with at least 2 such sailors

					-
CELECT S noting MIN (S age)	<u>sid</u>	sname	rating	age	
SELECT S.rating, MIN (S.age) FROM Sailors S	22	dustin	7	45.0	
WHERE S.age $>= 18$	31	lubber	8	55.5	
GROUP BY S.rating	71	zorba	10	16.0	
HAVING COUNT (*) > 1	64	horatio	7	35.0	
	29	brutus	1	33.0	
# Only S.rating and S.age are	58	rusty	10	35.0	
mentioned in the SELECT, GROU	JP rating	g age			
BY or HAVING clauses; other	1	33.0			
attributes ` <i>unnecessary</i> '.	7	45.0	rating		
₩ 2nd column of result is	7	35.0	7	35.0	
unnamed. (Use AS to name it	, 0	55.5			
	10	35.0	Answer	relation	1

Joins

Product (pname, price, category, maker) Purchase (buyer, seller, store, product) Company (cname, stockPrice, country) Person(per-name, phoneNumber, city)

Find names of people living in Seattle that bought gizmo products, and the names of the stores they bought from

 SELECT
 per-name, store

 FROM
 Person, Purchase

 WHERE
 per-name=buyer AND city="Seattle" AND product="gizmo"

15

17

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
 Compute the cross-product of *relation-list*.
 Discard resulting tuples if they fail *qualifications*.
 Delete attributes that are not in *target-list*.
 If DISTINCT is specified, eliminate duplicate rows.
- * This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Meaning (Semantics) of SQL Queries

SELECT a1, a2, ..., ak FROM R1 AS x1, R2 AS x2, ..., Rn AS xn WHERE Conditions

4. Translation to Relational algebra:

 $\Pi_{a1,...,ak}$ ($\sigma_{Conditions}$ (R1 x R2 x ... x Rn))

Select-From-Where queries are precisely Select-Project-Join

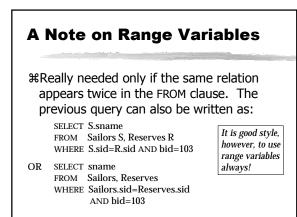
Meaning (Semantics) of SQL Queries

1. Nested loops: Answer = {} for x1 in R1 do for x2 in R2 do

> for xn in Rn do if Conditions then Answer = Answer U {(a1,...,ak)} return Answer

Example In	st	^{R1} and	$\frac{\text{sid}}{22}$ 10		<u>ay</u> 10/96
			22 10 58 10		2/96
#We will use these	S1	<u>sid</u>	sname	rating	age
instances of the		22	dustin	7	45.0
Sailors and		31	lubber	8	55.5
Reserves		58	rusty	10	35.0
relations in our examples.	S 2	sid	sname	rating	age
examplest		28	yuppy	9	35.0
		31	lubber	8	55.5
		44	guppy	5	35.0
		58	rusty	10	35.0

	ple atio					
	FROM	S.snar Sailor S1.sid	s S1, F			bid=103
(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96



Find sailors who've reserved at least one boat

SELECT S.sid FROM Sailors S, Reserves R WHERE S.sid=R.sid

#Would adding DISTINCT to this query make a difference?

SQL is Tricky!

SELECT R.A FROM R, S, T WHERE R.A=S.A OR R.A=T.A

Looking for $R \cap (S \cup T)$

But what happens if T is empty?

23

 Select S.sname

 FROM Sailors S

 WHERE S.sid IN (SELECT R.sid

 FROM Reserves R

 WHERE clause can itself contain an SQL query!

 % To find sailors who've not reserved #103, use NOT IN.

 % To understand semantics of nested queries, think of a

 <u>nested loops</u> evaluation: For each Sailors tuple, check the qualification by computing the subquery.

Nested Queries with Correlation

Find names of sailors who've reserved boat #103: SELECT S sname FROM Sailors S WHERE EXISTS (SELECT * FROM Reserves R WHERE R.bid=103 AND S.sid=R.sid) # EXISTS is another set comparison operator, like IN.

If UNIQUE is used, and * is replaced by *R.bid*, finds sailors with at most one reservation for boat #103. (UNIQUE checks for duplicate tuples; * denotes all attributes. Why do we have to replace * by *R.bid*?) # Illustrates why, in general, subquery must be re-computed for

each Sailors tuple.

More on Set-Comparison Operators

₩We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.

%Also available: *op* SOME, *op* ALL

Example: Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

SELECT Company.name FROM Company, Product WHERE Company.name=maker AND Product.name IN (SELECT product FROM Purchase WHERE buyer = "Joe Blow");

Here the subquery returns a set of values

27

Example: Subqueries Returning Relations

Equivalent to:

SELECT Company.name FROM Company, Product, Purchase WHERE Company.name=maker AND Product.name = product AND buyer = "Joe Blow"

Is this query equivalent to the previous one ?

Example: Subqueries Returning Relations

You can also use: s > ALL R s > ANY R EXISTS R

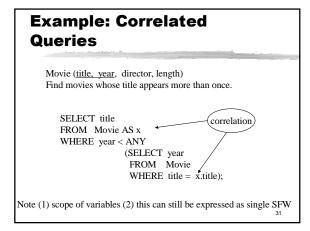
Product (pname, price, category, maker) Find products that are more expensive than all those produced By "Gizmo-Works"

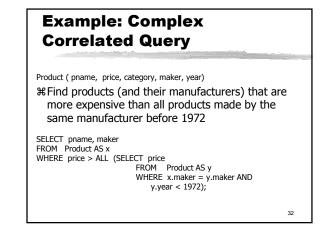
SELECT name FROM Product WHERE price > ALL (SELECT price FROM Purchase WHERE maker="Gizmo-Works")

29

Example: Conditions on Tuples

SELECT Company.name FROM Company, Product WHERE Company.name=maker AND (Product.name,price) IN (SELECT product, price) FROM Purchase WHERE buyer = "Joe Blow"); 28





<section-header><section-header><text><text>

Union, Intersection, Difference

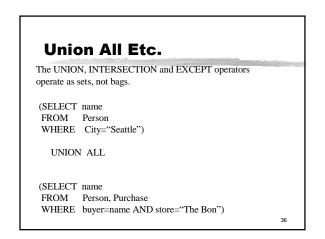
(SELECT name FROM Person WHERE City="Seattle")

UNION

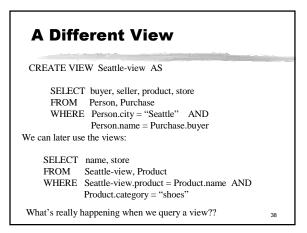
(SELECT name FROM Person, Purchase WHERE buyer=name AND store="The Bon")

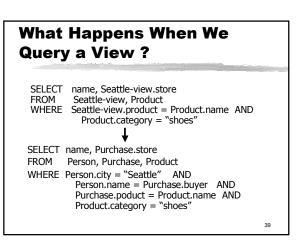
Similarly, you can use INTERSECT and EXCEPT. You must have the same attribute names (otherwise: rename). 34

Find sid's of sailors <u>or</u> a green boat	who've reserved a red
 UNION: Can be used to compute the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries). Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?) 	SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid AND R.bid=B.bid AND (B.color='red' OR B.color='green') SELECT S.sid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid AND R.bid=B.bid FROM Sailors S, Boats B, Reserves R WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'



Defining Views Views are relations, except that they are not physically stored. They are used mostly in order to simplify complex queries and to define conceptually different views of the database to different classes of users. View: purchases of telephony products: CREATE VIEW telephony-purchases AS SELECT product, buyer, seller, store FROM Purchase, Product WHERE Purchase.product = Product.name AND Product.category = "telephony"





Null Values and Outerjoins #If x=Null then 4*(3-x)/7 is still NULL #If x=Null then x="Joe" is UNKNOWN #Three boolean values: Image: Delta SE = 0

 $\Box UNKNOWN = 0.5$ $\Box TRUE = 1$

= 1

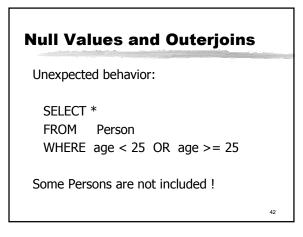
40

Null Values and Outerjoins

 \Re C1 AND C2 = min(C1, C2) \Re C1 OR C2 = max(C1, C2) \Re NOT C1 = 1 - C1

> SELECT * FROM Person WHERE (age < 25) AND (height > 6 OR weight > 190)

Rule in SQL: include only tuples that yield $\text{TRUE}_{_{41}}$

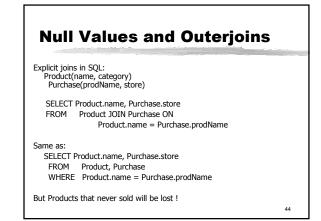


Null Values and Outerjoins

SELECT * FROM Person WHERE age < 25 OR age >= 25 OR age IS NULL

43

Now it includes all Persons



Build Database and Database Set outer joins in SQL: Product(name, category) Purchase(prodName, store) SELECT Product.name, Purchase.store FROM Product LEFT OUTER JOIN Purchase ON Product.name = Purchase.prodName

oduct		_	Purcha	se	and the second se	
Name	Catego	ry	ProdNa	me	Store	
Gizmo	gadge	t	Gizmo)	Wiz	
Camera	Photo		Came	a	Ritz	
OneClick	Photo		Camer	a	Wiz	
		Name	Store			
		Gizmo	Wiz			
		Camera	Ritz			
		Camera	Wiz			
		OneClick	-			