## Weeks 3 & 4: SQL

The SQL Query Language Select Statements Joins, Aggregate and Nested Queries Insertions, Deletions and Updates Assertions, Views, Triggers and Access Control

CSC343 Introduction to Databases — University of Toronto

### SQL as a Query Language

- →SQL expresses queries in declarative way queries specify the properties of the result, not the way to obtain it.
- →Queries are translated by the query optimizer into the procedural language internal to the DBMS.
- →The programmer focuses on <u>readability</u>, not on efficiency.

SQL — 2

SQL - 1

### **SQL Queries**

 $\rightarrow$ SQL queries are expressed by the select statement.

 $\rightarrow$ Syntax:

select AttrExpr [[as] Alias ] {, AttrExpr [[as] Alias ] }
from Table [[as] Alias ] {, [[as] Alias ] }
[where Condition]

→The three parts of the query are usually called: target list, <u>from</u> clause, <u>where</u> clause.

The query first builds the Cartesian product of the tables in the from clause, then selects only the rows that satisfy the condition in the where clause and for each row evaluates the attribute expressions in the target list.

CSC343 Introduction to Databases — University of Toronto

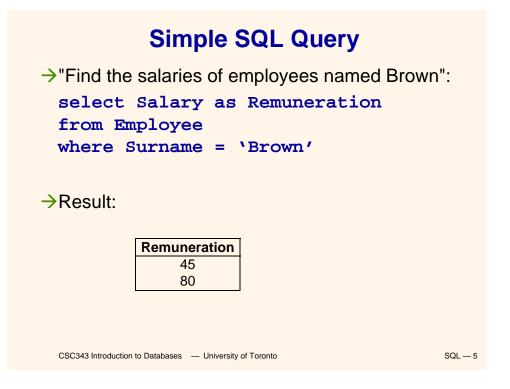
SQL - 3

### **Example Database**

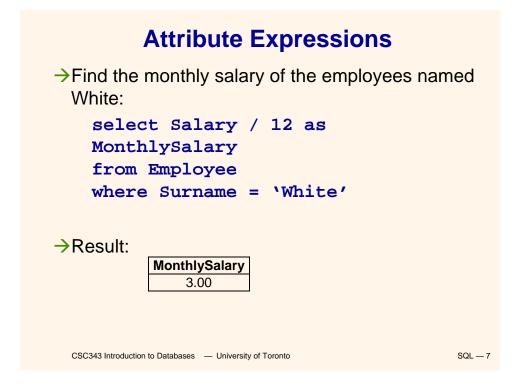
EMPLOYEE	FirstName	Surname	Dept	Office	Salary	City
	Mary	Brown	Administration	10	45	London
	Charles	White	Production	20	36	Toulouse
	Gus	Green	Administration	20	40	Oxford
	Jackson	Neri	Distribution	16	45	Dover
	Charles	Brown	Planning	14	80	London
	Laurence	Chen	Planning	7	73	Worthing
	Pauline	Bradshaw	Administration	75	40	Brighton
	Alice	Jackson	Production	20	46	Toulouse

DEPARTMENT	DeptName	Address	City
	Administration	Bond Street	London
	Production	Rue Victor Hugo	Toulouse
	Distribution	Pond Road	Brighton
	Planning	Bond Street	London
	Research	Sunset Street	San José

SQL-4

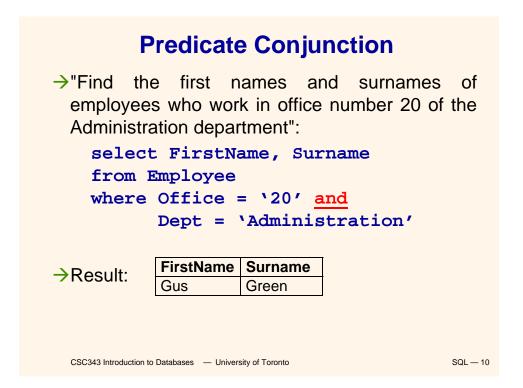


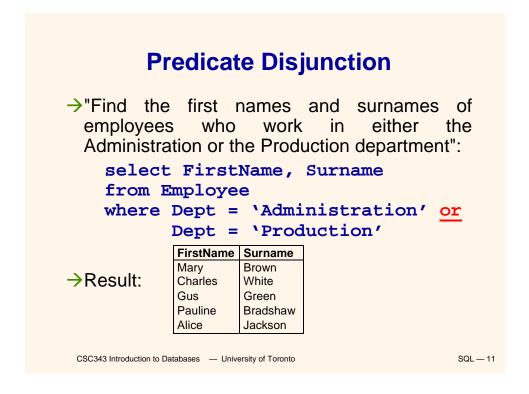
<ul> <li>* in the Target List</li> <li>"Find all the information relating to employees named Brown":</li> </ul>						
selec	t *					
from Employee						
	where Surname = 'Brown'					
			wn′			
			wn '			
where		me = 'Bro	Office	Salary	City	
where Result:	Surna	me = 'Bro		Salary 45	<b>City</b> Londor	

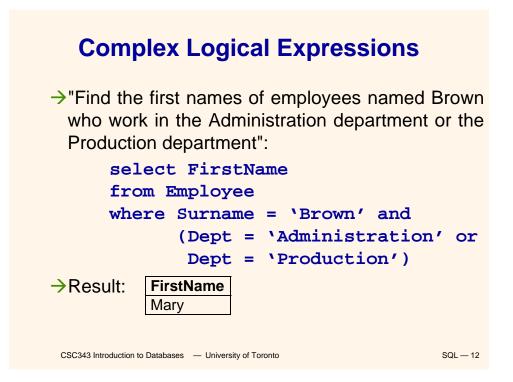


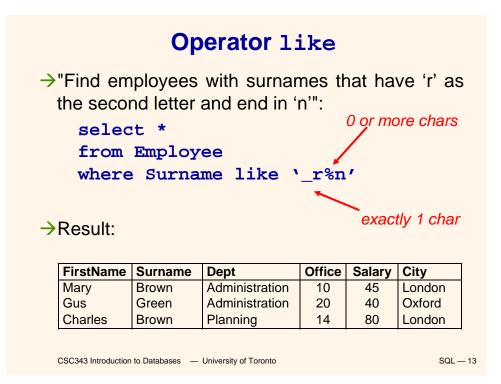
Simple Join Query						
→"Find the name	nes of en	nployees	s and the	eir cities of work":		
select Emp	loyee.	FirstNa	ame,			
Employe	e.Surna	ame, De	partm	ent.City		
from Emplo	yee, D	epartm	ent			
where Empl	oyee.D	ept = 1	Depart	ment.DeptName		
	FirstName	Surname	City			
→Result:	Mary	Brown	London			
	Charles	White	Toulouse			
	Gus	Green	London			
	Jackson	Neri	Brighton			
	Charles Brown London					
	Laurence Chen London					
	Pauline Bradshaw London					
	Alice	Jackson	Toulouse			
CSC343 Introduction to	o Databases — U	niversity of Toronto	)	SQL — 8		

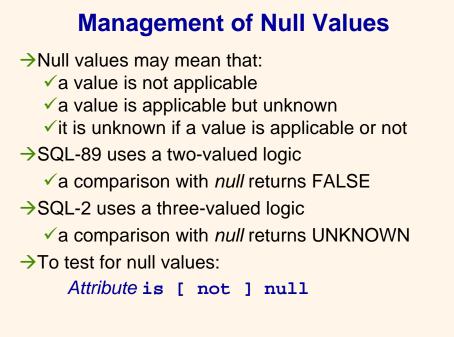
Table Aliases							
→"Find the names of employees and the cities where they work" (using an alias): select FirstName, Surname, D.City							
from En	nployee	e, Depa	rtment	E D			
where I	Dept =	DeptNa	me				
	FirstName	Surname	City				
	Mary	Brown	London				
→Result:	Charles	White	Toulouse				
	Gus	Green	London				
	Jackson	Neri	Brighton				
	Charles Brown London						
	Laurence Chen London						
	Pauline Bradshaw London						
	Alice	Jackson	Toulouse				
CSC343 Introduction to	Databases — Un	iversity of Toronto		SQL — 9			











CSC343 Introduction to Databases — University of Toronto

### Algebraic Interpretation of SQL Queries

→The generic query:

select  $T_1$ . Attr<sub>11</sub>, ...,  $T_h$ . Attr<sub>hm</sub> from Table<sub>1</sub>  $T_1$ , ..., Table<sub>n</sub>  $T_n$ where Condition

corresponds to the relational algebra query:

 $\pi_{T_1.Attr_{11},...,T_h.Attr_{hm}}(\sigma_{Condition}(Table_1 \times ... \times Table_n))$ 

SQL - 15

CSC343 Introduction to Databases — University of Toronto

**Duplicates**  $\rightarrow$ In the relational algebra and calculus the results of queries do not contain duplicates.  $\rightarrow$ In SQL, tables may have identical rows. →Duplicates can be removed using the keyword distinct: select City select distinct City from Department from Department City City London London Toulouse Toulouse Brighton **Brighton** London San José San José CSC343 Introduction to Databases — University of Toronto SQL — 16

### Joins in SQL-2

→SQL-2 introduced an alternative syntax for the representation of joins, representing them explicitly in the *from* clause:

select AttrExpr[[ as ] Alias ] {, AttrExpr[[as] Alias
from Table [[as] Alias ]
{[JoinType] join Table

[[as] Alias] on JoinConditions }
[ where OtherCondition ]

→JoinType can be any of inner, right [outer],
left [outer] or full [outer].

The keyword natural may precede JoinType (rarely implemented).
CSC343 Introduction to Databases — University of Toronto

SQL — 17

### **Inner Join in SQL-2** $\rightarrow$ "Find the names of the employees and the cities in which they work": select FirstName, Surname, D.City from Employee inner join Department as D on Dept = DeptName FirstName Surname City Mary Brown London $\rightarrow$ Result: Charles White Toulouse Gus Green London Jackson Neri **Brighton** Charles Brown London Chen Laurence London Pauline Bradshaw London Alice Jackson Toulouse CSC343 Introduction to Databases — University of Toronto SQL — 18

Another Example: Drivers and Cars									
DRIVER	Fire	stName	Su	Irname	Dr	iverID			
	Ma	ry	Bre	own	VF	R 2030020`	Y		
	Cha	arles	W	nite	ΡZ	Z 1012436E	3		
	Ma	rco	Ne	eri	AF	P 4544442F	२		
Аυтомов	BILE	CarReg	No	Make		Model	Dr	iverID	
		ABC 123	3	BMW		323	VF	R 2030020Y	
		DEF 456	6	BMW		Z3	VF	R 2030020Y	
		GHI 789		Lancia		Delta	PZ	2 1012436B	
		BBB 421		BMW		316	MI	2020030U	
CSC343 Introduction to Databases — University of Toronto SQL — 19									

Left Join
>"Find all drivers and their cars, if any":
 select FirstName,Surname,
 Driver.DriverID,CarRegNo,Make,Model
from Driver left join Automobile on
 (Driver.DriverID =
 Automobile.DriverID)

### →Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL

CSC343 Introduction to Databases — University of Toronto

### **Full Join**

→"Find all possible drivers and their cars":
 select
 FirstName,Surname,Driver.DriverID
 CarRegNo, Make, Model
 from Driver full join Automobile on
 (Driver.DriverID =
 Automobile.DriverID)

### →Result:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	NULL	BBB 421	BMW	316

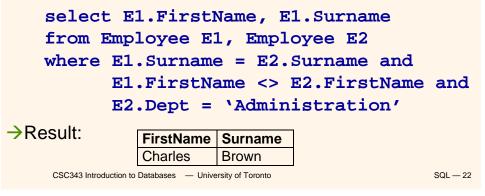
CSC343 Introduction to Databases — University of Toronto

SQL — 21

### **Table Variables**

→Table aliases may be interpreted as table variables. These correspond to the renaming operator  $\rho$ .

→"Find all first names and surnames of employees who have the same surname and different first names with someone in the Administration department":



### The order by Clause

 $\rightarrow$ order by — appearing at the end of a query orders the rows of the result; syntax: order by OrderingAttribute [asc | desc ] {, OrderingAttribute [ asc | desc ] }

>Extract the content of the Automobile table in descending order with respect to make and model:

select \*

from Automobile

BBB 421

CSC343 Introduction to Databases — University of Toronto

### order by Make desc, Model desc

BMW

→Result:

CarRegNo	Make	Model	DriverID
GHI 789	Lancia	Delta	PZ 1012436B
DEF 456	BMW	Z3	VR 2030020Y
ABC 123	BMW	323	VR 2030020Y

316

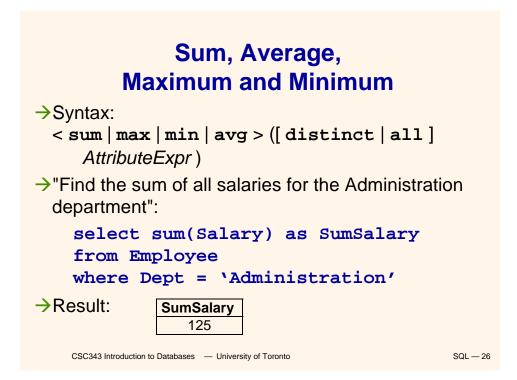
SQL - 23

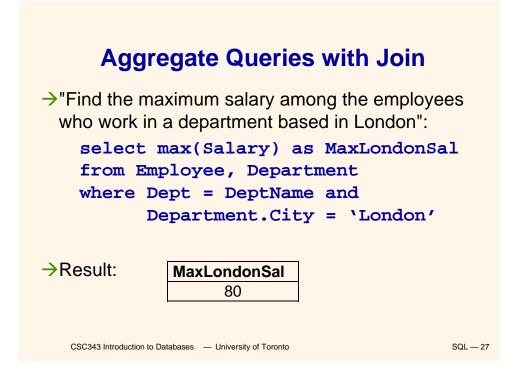
MI 2020030U

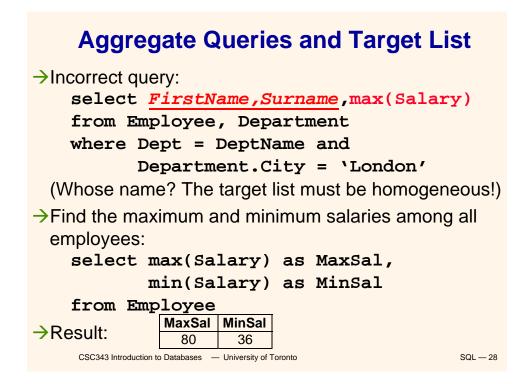
### **Aggregate Queries** →Aggregate queries cannot be represented in relational algebra. $\rightarrow$ The result of an aggregate query depends on functions that take as an argument a set of tuples. →SQL-2 offers five aggregate operators: ✓ count √ sum √max √min √avg CSC343 Introduction to Databases — University of Toronto SQL — 24

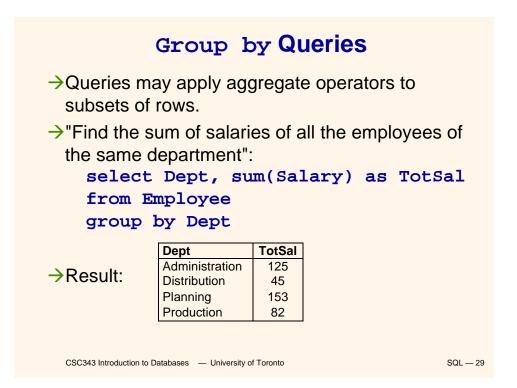
### **Operator** count

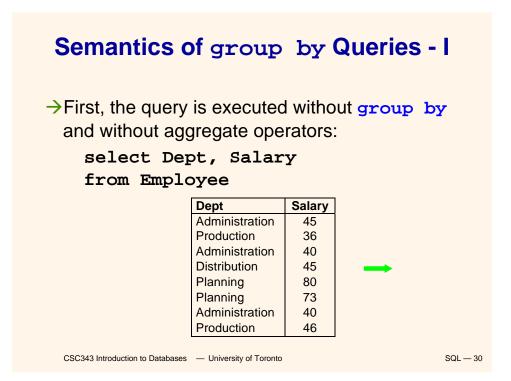
>count returns the number of elements (or, distinct elements) of its argument: count(< \* [[distinct|all]] AttributeList >) >"Find the number of employees": select count(\*)from Employee >"Find the number of different values on attribute Salary for all tuples in Employee": select count(distinct Salary) from Employee >"Find the number of tuples in Employee having nonnull values on the attribute Salary": select count(all Salary) from Employee











### Semantics of group by Queries - II

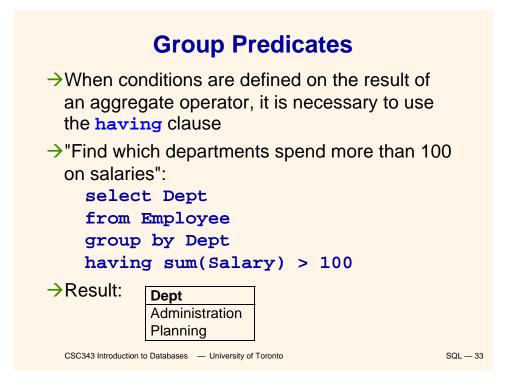
→... then the query result is divided in subsets characterized by the same values for the attributes appearing as argument of the group by clause (in this case attribute Dept):

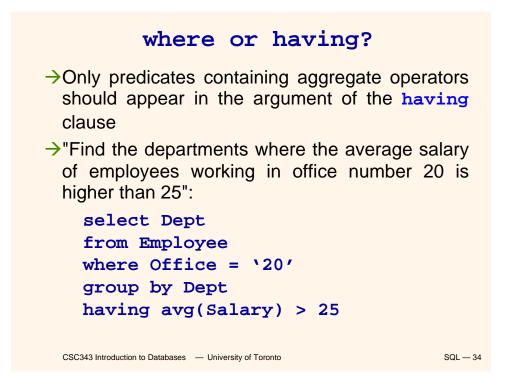
Finally, the aggregate operator is applied separately to each subset

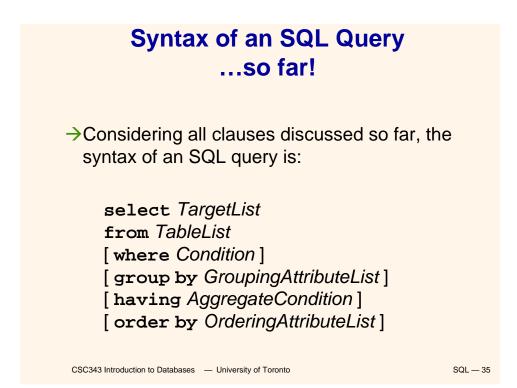
	Dept	Salary		
	Administration	45		Dept
	Administration	40		Administration
	Administration	40		
	Distribution	45	1	Distribution
	Planning	80		Planning
	Planning	73		Production
	Production	36		
	Production	46		
CSCS	343 Introduction to Databa	ases — Uni	versity of Toronto	

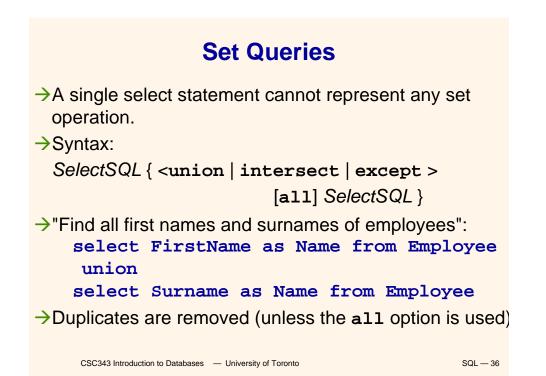
SQL — 31

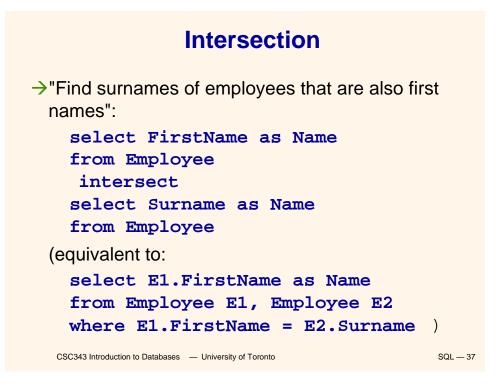
group by Queries and Target List  $\rightarrow$ Incorrect query: select Office from Employee group by Dept  $\rightarrow$ Incorrect query: select DeptName, count(\*), D.City from Employee E join Department D on (E.Dept = D.DeptName) group by DeptName  $\rightarrow$ Correct query: select DeptName,count(\*),D.City from Employee E join Department D on (E.Dept = D.DeptName) group by DeptName, D.City CSC343 Introduction to Databases — University of Toronto SQL — 32

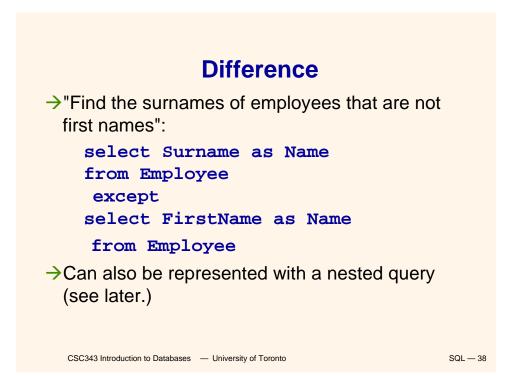












### **Nested Queries**

 $\rightarrow$ A where clause may include predicates that:

✓Compare an attribute (or attribute expression) with the result of an SQL query;

syntax: ScalarValue Op <any | all> SelectSQL any – the predicate is true if at least one row returned by SelectSQL satisfies the comparison all – predicate is true if all rows satisfy comparison;

Use the existential quantifier on an SQL query; syntax: exists SelectSQL

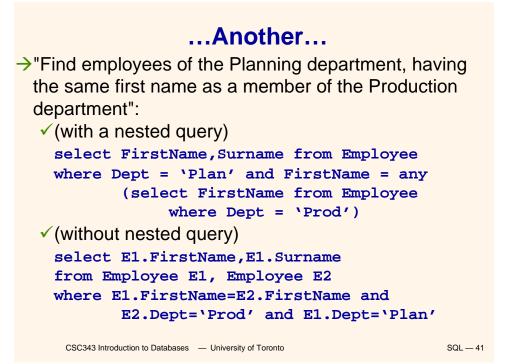
the predicate is true if *SelectSQL* is non-empty.

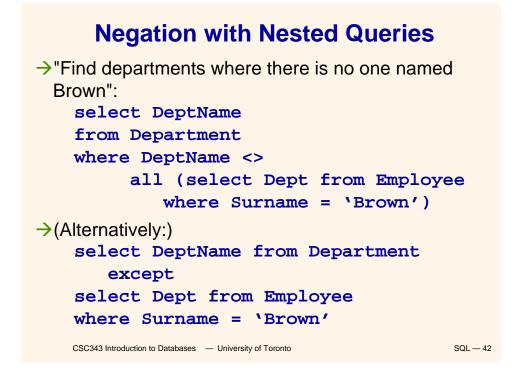
The query appearing in the where clause is called a nested query.

SQL - 39

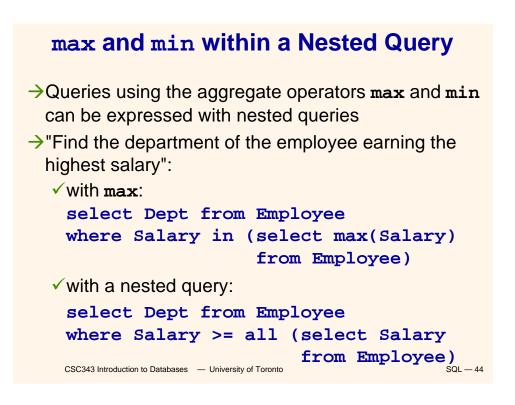
CSC343 Introduction to Databases — University of Toronto

**Simple Nested Query**  $\rightarrow$ "Find the employees who work in departments in London": select FirstName, Surname from Employee where Dept = any (select DeptName from Department where City = 'London') (Equivalent to: select FirstName, Surname from Employee, Department D where Dept = DeptName and D.City = `London' ) CSC343 Introduction to Databases — University of Toronto SQL — 40



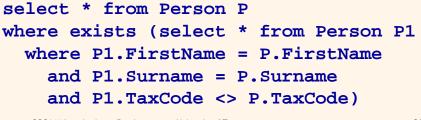


### Operators in and not in >Operator in is a shorthand for = any select FirstName, Surname from Employee where Dept in (select DeptName from Department where City = `London') >Operator not in is a shorthand for <> all select DeptName from Department where DeptName not in (select Dept from Employee where Surname = `Brown')



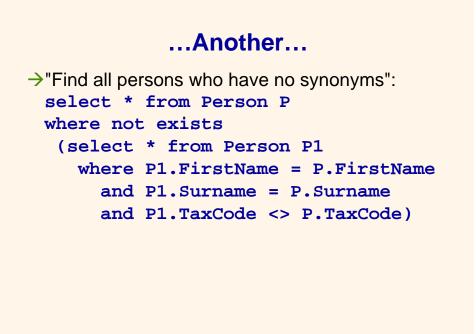
### **A Complex Nested Query**

- A nested query may use variables of the outer query ('transfer of bindings').
- Semantics: the nested query is evaluated for each row of the outer query.
- Find all persons who have the same first name and surname with someone else ("synonyms"), but different tax codes":

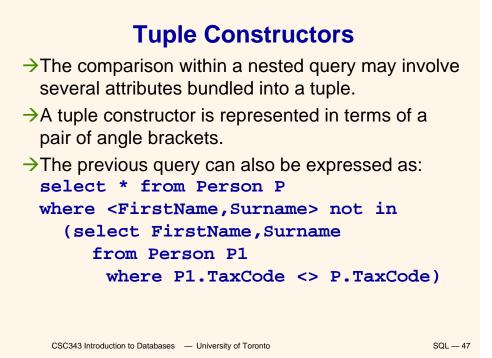


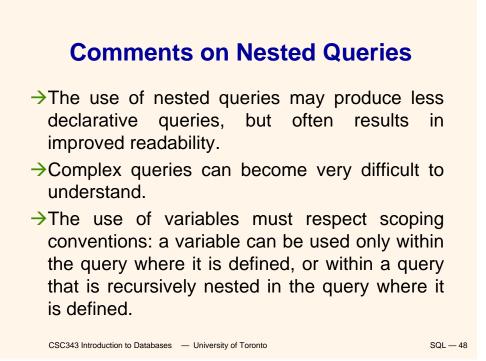
CSC343 Introduction to Databases — University of Toronto

SQL — 45



CSC343 Introduction to Databases — University of Toronto





# Scope of Variables

>Incorrect query: select \* from Employee where Dept in (select DeptName from Department D1 where DeptName = `Production') or Dept in (select DeptName from Department D2 where D2.City = D1.City) >What's wrong?

CSC343 Introduction to Databases — University of Toronto

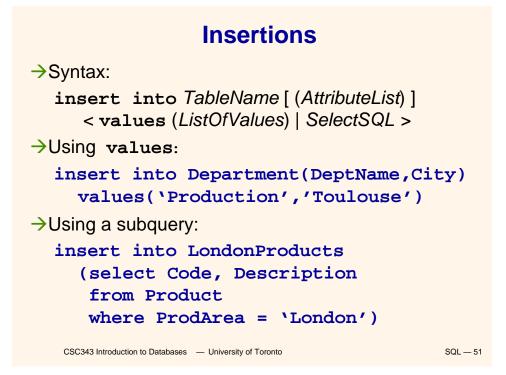
Data Modification in SQL
Modification statements include:

Insertions (insert);
Deletions (delete);
Updates of attribute values (update).

All modification statements operate on a set of tuples (no duplicates.)
In the *condition* part of an update statement it is possible to access other relations.

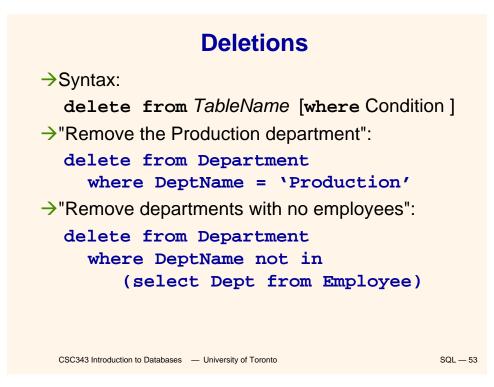
 $\rm SQL-50$ 

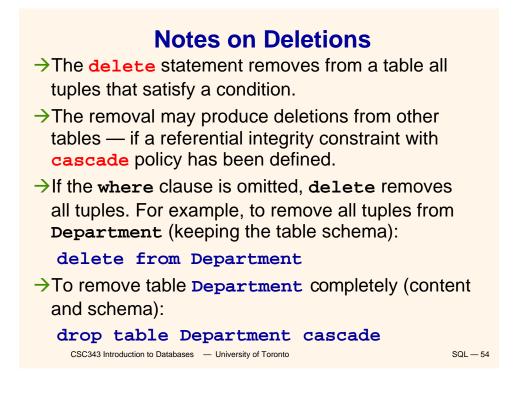
SQL - 49

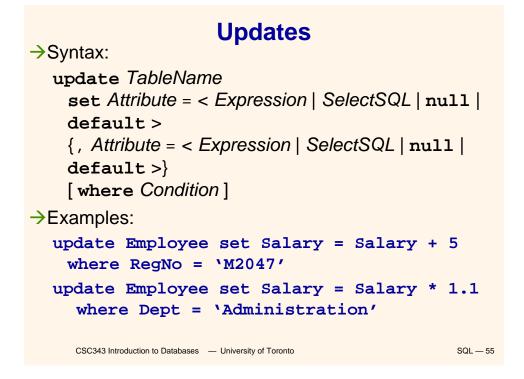


### **Notes on Insertions**

- The ordering of attributes (if present) and of values is meaningful -- first value for the first attribute, etc.
- →If *AttributeList* is omitted, all the relation attributes are considered, in the order they appear in the table definition.
- →If AttributeList does not contain all the relation attributes, left-out attributes are assigned default values (if defined) or the null value.









As with any side effect statement, the order of updates is important:
 update Employee
 set Salary = Salary \* 1.1
 where Salary <= 30</li>
 update Employee
 set Salary = Salary \* 1.15
 where Salary > 30
 In this example, some employees may get a double raise! How can we fix this?

# Generic Integrity Constraints -> The check clause can be used to express arbitrary constraints during schema definition. -> Syntax: check (Condition) -> Condition is what can appear in a where clause — including nested queries. -> For example, the definition of an attribute Superior in the schema of table Employee: Superior character(6) check (RegNo like "1%" or Dept = (select Dept from Employee E where E.RegNo = Superior)

SQL - 57

CSC343 Introduction to Databases — University of Toronto

**Assertions permit the definition of constraints** independently of table definitions.
Assertions are useful in many situations -- e.g., to express generic inter-relational constraints.
An assertion associates a name to a check clause; syntax:
create assertion AssertName check (Condition)
There must always be at least one tuple in table Employee":
create assertion AlwaysOneEmployee
check (1 <= (select count(\*) from Employee))

### Views

>Views are "virtual tables" whose rows are computed from other tables (base relations). >Syntax: create view ViewName [(AttributeList)] as SelectSQL [with [local]cascaded] check option] >Examples: create view AdminEmployee (RegNo,FirstName,Surname,Salary) as select RegNo,FirstName,Surname,Salary from Employee where Dept = 'Admin' and Salary > 10 create view JuniorAdminEmployee as select \* from AdminEmployee where Salary < 50 with check option</pre>

CSC343 Introduction to Databases — University of Toronto

SQL — 59

# <section-header><list-item><list-item><list-item><list-item><list-item><list-item><list-item><table-row><table-row><table-row>



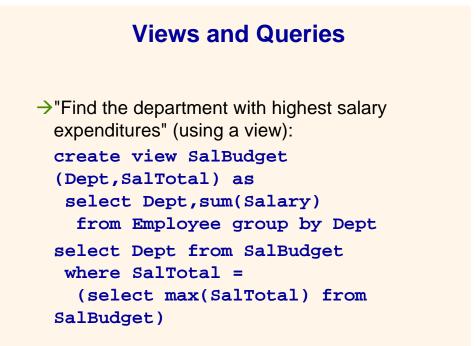
→"Find the department with highest salary expenditures" (without using a view):

select Dept from Employee
group by Dept
having sum(Salary) >= all
(select sum(Salary) from
Employee

group by Dept)

This solution may not work with all SQL systems.

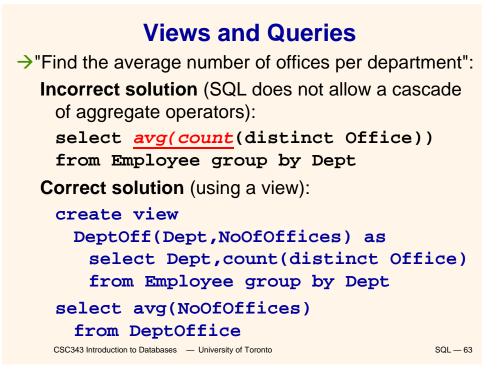
CSC343 Introduction to Databases — University of Toronto



CSC343 Introduction to Databases — University of Toronto

SQL — 62

SQL - 61





- →Every element of a schema can be protected (tables, attributes, views, domains, etc.)
- →The owner of a resource (the creator) assigns privileges to the other users.
- →A predefined user <u>system</u> represents the database administrator and has access to all resources.
- $\rightarrow$ A privilege is characterized by:
  - ✓a resource;
  - the user who grants the privilege;
  - the user who receives the privilege;
  - the action that is allowed on the resource;
  - ✓ whether or not the privilege can be passed on to other users. CSC343 Introduction to Databases — University of Toronto
    SQL-64

### **Types of Privileges**

 $\rightarrow$ SQL offers six types of privilege:

- vinsert: to insert a new object into the
  resource;
- ✓ update: to modify the resource content;
- delete: to remove an object from the
   resource;
- select: to access the resource content;
- references: to build a referential integrity constraint with the resource;
- ✓ usage: to use the resource in a schema definition (e.g., a domain)

SQL - 65

CSC343 Introduction to Databases — University of Toronto

spant a privilege to a user: grant < Privileges | all privileges > on Resource to Users [with grant option] \*grant option specifies whether the privilege can be propagated to other users. \*For example, grant select on Department to stefano \*To take away privileges: revoke Privileges on Resource from Users [restrict | cascade ]

### **Database Triggers**

Triggers (also known as ECA rules) are element of the database schema.

→General form:

- on <event> when <condition> then <action>
- Event- request to execute database operation
- Condition predicate evaluated on database state
- Action execution of procedure that might involve database updates

→Example:

on "updating maximum enrollment limit"

if "# registered > new max enrollment limit "
 then "deregister students using LIFO policy"
 CSC343 Introduction to Databases - University of Toronto
SQL - 67

### **Trigger Details**

→Activation — occurrence of the event that activates the trigger.

Consideration — the point, after activation, when condition is evaluated; this can be immediate or deferred.

 Deferred means that condition is evaluated when the database operation (*transaction*) currently executing requests to commit.

→Condition might refer to both the state before and the state after event occurs.

### **Trigger Execution**

This is the point when the *action* part of the trigger is carried out.

- With deferred consideration, execution is also deferred.
- With immediate consideration, execution can occur immediately after consideration or it can be deferred
  - ✓ If execution is immediate, execution can occur before, after, or instead of triggering event.
  - Before triggers adapt naturally to maintaining integrity constraints: violation results in rejection of event.

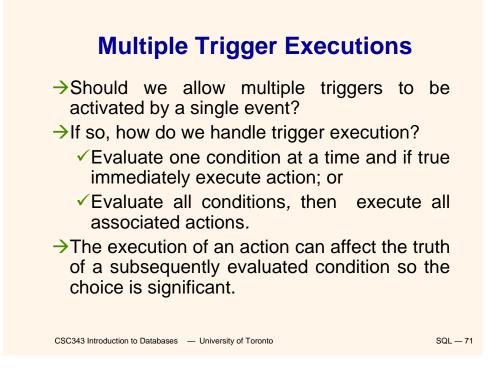
CSC343 Introduction to Databases — University of Toronto

SQL — 69

### **Event Granularity**

Event granularity can be:

- →Row-level: the event involves change of a single row,
  - This means that a single update statement might result in multiple events;
- Statement-level: here events result from the execution of a whole statement; for example, a single update statement that changes multiple rows constitutes a single event.



### **Triggers in SQL-3**

- →Events: insert, delete, or update statements or changes to individual rows caused by these statements.
- $\rightarrow$ Condition: Anything allowed in a where clause.
- Action: An individual SQL statement or a program written in the language of Procedural Stored Modules (PSM) -- which can contain embedded SQL statements.

### **Triggers in SQL-3**

- Consideration = immediate condition can refer to both the state of the affected row or table before and after the event occurs.
- Execution = immediate can be before or after the execution of the triggering event
- →Note that the action of a before-trigger cannot modify the database.
- →Granularity: Both row-level and statement-level.

SQL - 73

CSC343 Introduction to Databases — University of Toronto

**Before-Trigger with Row Granularity** Check that CREATE TRIGGER Max\_EnrollCheck enrollment **BEFORE INSERT ON Transcript** < limit **REFERENCING NEW AS N** --row to be added FOR EACH ROW WHEN ((SELECT COUNT (T.StudId) FROM Transcript T WHERE T.CrsCode = N.CrsCode **AND T.**Semester = N.Semester) >= (SELECT C.MaxEnroll FROM Course C WHERE C.CrsCode = N.CrsCode )) THEN ABORT TRANSACTION Action CSC343 Introduction to Databases — University of Toronto SQL — 74

### After-Trigger with Row Granularity

CREATE TRIGGER LimitSalaryRaise AFTER UPDATE OF Salary ON Employee REFERENCING OLD AS O NEW AS N FOR EACH ROW

WHEN (N.Salary - O.Salary > 0.05 \* O.Salary) THEN UPDATE **Employee** -- action SET Salary = 1.05 \* O.Salary WHERE Id = O.Id

[Note: The action itself is a triggering event; however, in this case a chain reaction is not possible.]

CSC343 Introduction to Databases — University of Toronto

SQL — 75



CSC343 Introduction to Databases — University of Toronto