# 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

# 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.

CSC343 Introduction to Databases — University of Toronto

SQL — 2

# **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
```

CSC343 Introduction to Databases - University of Toronto

SQL — 3

SQL - 1

# **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

PARTMENT	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

CSC343 Introduction to Databases — University of Toronto

DEP

# Simple SQL Query

 $\rightarrow$  "Find the salaries of employees named Brown":

select Salary as Remuneration
from Employee
where Surname = `Brown'

#### →Result:



#### CSC343 Introduction to Databases — University of Toronto

 $\rm SQL-5$ 

# \* in the Target List

→"Find all the information relating to employees named Brown":

select \*
from Employee
where Surname = `Brown'

#### $\rightarrow$ Result:

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	Brown	Planning	14	80	London

CSC343 Introduction to Databases — University of Toronto

SQL - 6

# **Attribute Expressions**

→Find the monthly salary of the employees named White:

select Salary / 12 as MonthlySalary from Employee where Surname = 'White'

#### →Result:



#### CSC343 Introduction to Databases — University of Toronto

SQL — 7

# **Simple Join Query**

 $\rightarrow$ "Find the names of employees and their cities of work":

select Employee.FirstName,

Employee.Surname, Department.City

from Employee, Department

where Employee.Dept = Department.DeptName

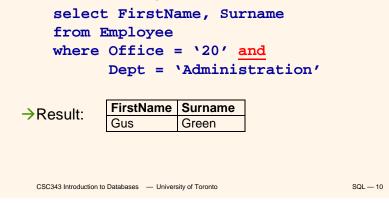
	FirstName	Surname	City
→Result:	Mary	Brown	London
	Charles	White	Toulouse
	Gus	Green	London
	Jackson	Neri	Brighton
	Charles	Brown	London
	Laurence	Chen	London
	Pauline	Bradshaw	London
	Alice	Jackson	Toulouse
			•

CSC343 Introduction to Databases — University of Toronto

	Table Aliases							
	→"Find the names of employees and the cities where they work" (using an alias):							
select	FirstN	ame, S	urname	e, D.City				
from En	nployee	, Depa	rtment	: D				
where I	Dept =	DeptNa	me					
	FirstName	Surname	City					
	Mary	Brown	London					
→Result:	Charles	White	Toulouse					
7 1 CO C C C	Gus	Green	London					
	Jackson	Neri	Brighton					
	Charles	Brown	London					
	Laurence	Chen	London					
	Pauline	Bradshaw	London					
	Alice	Jackson	Toulouse					
CSC343 Introduction to	CSC343 Introduction to Databases — University of Toronto SQL — 9							

# **Predicate Conjunction**

→"Find the first names and surnames of employees who work in office number 20 of the Administration department":



# **Predicate Disjunction**

→"Find the first names and surnames of employees who work in either the Administration or the Production department":

# select FirstName, Surname from Employee where Dept = `Administration' or Dept = `Production'

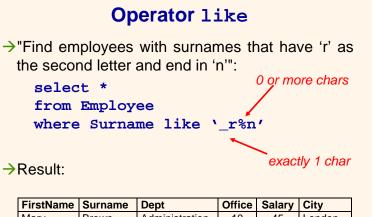
	FirstName	Surname
	Mary	Brown
→Result:	Charles	White
	Gus	Green
	Pauline	Bradshaw
	Alice	Jackson

 $\rm SQL-11$ 

# **Complex Logical Expressions**

→"Find the first names of employees named Brown who work in the Administration department or the Production department":

fro	ect FirstNa m Employee ere Surname	ame = `Brown' and	
	(Dept =	<pre>`Administration' ( `Production')</pre>	or
→Result:	FirstName Mary		
CSC343 Introduction t	o Databases — University of To	ronto SQL	— 12



FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Gus	Green	Administration	20	40	Oxford
Charles	Brown	Planning	14	80	London

CSC343 Introduction to Databases — University of Toronto

```
SQL — 13
```

# **Management of Null Values**

 $\rightarrow$ Null values may mean that:

- $\checkmark$  a value is not applicable
- ✓ a value is applicable but unknown
- ✓it is unknown if a value is applicable or not
- →SQL-89 uses a two-valued logic
  - ✓a comparison with *null* returns FALSE
- →SQL-2 uses a three-valued logic
  - ✓ a comparison with *null* returns UNKNOWN
- $\rightarrow$ To test for null values:

Attribute is [ not ] null

CSC343 Introduction to Databases — University of Toronto

SQL - 14

# Algebraic Interpretation of SQL Queries

 $\rightarrow$ 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

# **Duplicates**

- →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 from Department select <u>distinct</u> City from Department

City
London
Toulouse
Brighton
London
San José

City	1
London	
Toulouse	
Brighton	
San José	

CSC343 Introduction to Databases — University of Toronto

# 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**

→"Find the names of the employees and the cities in which they work":

	FirstName	Surname	City	
Deevilte	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 — University of Toronto

SQL - 18

# Another Example: Drivers and Cars

DRIVER	FirstName	Surname	DriverID
	Mary	Brown	VR 2030020Y
	Charles	White	PZ 1012436B
	Marco	Neri	AP 4544442R

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

 $\rightarrow$ 

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

 $\rm SQL-20$ 

# **Full Join**

 $\rightarrow$ "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)

#### $\rightarrow$ 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					S

```
Table Variables
```

- $\rightarrow$ Table aliases may be interpreted as table variables. These correspond to the renaming operator  $\rho$ .
- $\rightarrow$ "Find all first names and surnames of employees who have the same surname and different first names with someone in the Administration department":

select E	1.Firstl	Name, E	1.Surname	
from Emp	loyee E	1, Emplo	oyee E2	
where E1	.Surname	e = E2.8	Surname and	
E1.FirstName <> E2.FirstName and				
E2.Dept = `Administration'				
→Result:		-	1	
Zitesuit.	FirstName	Surname		
	Charles	Brown		

CSC343 Introduction to Databases	<ul> <li>University of Toronto</li> </ul>

SQL - 22

# The order by Clause

 $\rightarrow$ order by — appearing at the end of a query orders the rows of the result: svntax:

> 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

#### order by Make desc, Model desc

	CarRegNo	Make	Model	DriverID	
→Result:	GHI 789	Lancia	Delta	PZ 1012436B	
	DEF 456	BMW	Z3	VR 2030020Y	
	ABC 123	BMW	323	VR 2030020Y	
	BBB 421	BMW	316	MI 2020030U	
CSC343 Introduction to Databases — University of Toronto					SQL — 23

# **Aggregate Queries**

- →Aggregate queries cannot be represented in relational algebra.
- $\rightarrow$ The result of an aggregate guery depends on functions that take as an argument a set of tuples.
- $\rightarrow$ SQL-2 offers five aggregate operators:
  - √ count
  - √ sum
  - √max
  - √min
  - √avq

CSC343 Introduction to Databases — University of Toronto

# **Operator** count

 $\rightarrow$ count returns the number of elements (or, distinct elements) of its argument:

count(< \* | [ distinct | all ] AttributeList >)

 $\rightarrow$ "Find the number of employees":

#### select count(\*)from Employee

 $\rightarrow$ "Find the number of different values on attribute Salary for all tuples in Employee":

#### select count(distinct Salary) from Employee

 $\rightarrow$ "Find the number of tuples in Employee having nonnull values on the attribute Salary":

select count(all Salary) from Employee CSC343 Introduction to Databases - University of Toronto SQL — 25

# Sum, Average, **Maximum and Minimum**

 $\rightarrow$ Syntax:

< sum | max | min | avg > ([ distinct | all ] AttributeExpr)

 $\rightarrow$ "Find the sum of all salaries for the Administration department":

select sum(Salary) as SumSalary from Employee

where Dept = 'Administration'

 $\rightarrow$ Result:



CSC343 Introduction to Databases - University of Toronto

SOI - 26

SOI - 28



# **Aggregate Queries and Target List**

select FirstName,Surname,max(Salary)

from Employee, Department

where Dept = DeptName and

Department.City = 'London'

(Whose name? The target list must be homogeneous!)

 $\rightarrow$ Find the maximum and minimum salaries among all



# Group by Queries

- →Queries may apply aggregate operators to subsets of rows.
- →"Find the sum of salaries of all the employees of the same department":

select Dept, sum(Salary) as TotSal
from Employee
group by Dept

Decult	Administration	125
→Result:	Distribution	45
	Planning	153
	Production	82

```
CSC343 Introduction to Databases — University of Toronto
```

```
SQL — 29
```

# Semantics of group by Queries - I

→First, the query is executed without group by and without aggregate operators:

select Dept, Salary
from Employee

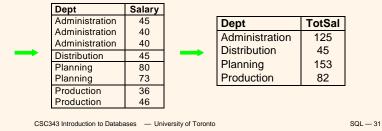
Dept	Salary	
Administration	45	
Production	36	
Administration	40	
Distribution	45	
Planning	80	
Planning	73	
Administration	40	
Production	46	

```
CSC343 Introduction to Databases — University of Toronto
```

SQL — 30

# 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

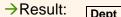


group by Queries and Target List  $\rightarrow$ Incorrect query: select Office from Employee group by Dept  $\rightarrow$ Incorrect query: select DeptName, count(\*), D.City from Employee E join Department D on (E.Dept = D.DeptName) group by DeptName  $\rightarrow$ Correct query: select DeptName,count(\*),D.City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

# **Group Predicates**

- When conditions are defined on the result of an aggregate operator, it is necessary to use the having clause
- →"Find which departments spend more than 100 on salaries":

select Dept
from Employee
group by Dept
having sum(Salary) > 100



Administration
Planning

CSC343 Introduction to Databases — University of Toronto

SQL — 33

# where or having?

- Only predicates containing aggregate operators should appear in the argument of the having clause
- →"Find the departments where the average salary of employees working in office number 20 is higher than 25":

select Dept
from Employee
where Office = `20'
group by Dept
having avg(Salary) > 25

CSC343 Introduction to Databases - University of Toronto

SQL — 34

# Syntax of an SQL Query ...so far!

→Considering all clauses discussed so far, the syntax of an SQL query is:

select TargetList
from TableList
[where Condition]
[group by GroupingAttributeList]
[having AggregateCondition]
[order by OrderingAttributeList]

SQL — 35

# **Set Queries**

→A single select statement cannot represent any set operation.

→Syntax:

SelectSQL { <union | intersect | except >
 [all] SelectSQL }

→"Find all first names and surnames of employees": select FirstName as Name from Employee union

select Surname as Name from Employee

→Duplicates are removed (unless the **all** option is used)

CSC343 Introduction to Databases — University of Toronto

# Intersection

→"Find surnames of employees that are also first names":

select FirstName as Name from Employee intersect select Surname as Name from Employee

(equivalent to:

select E1.FirstName as Name
from Employee E1, Employee E2
where E1.FirstName = E2.Surname )

CSC343 Introduction to Databases — University of Toronto

SQL — 37

# Difference

→"Find the surnames of employees that are not first names":

select Surname as Name from Employee except select FirstName as Name from Employee

→Can also be represented with a nested query (see later.)

CSC343 Introduction to Databases — University of Toronto

SQL — 38

# **Nested Queries**

→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.

CSC343 Introduction to Databases — University of Toronto

SQL — 39

# **Simple Nested Query**

→"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 Introducton to Databases - University of Toronto SQL-40

# ...Another...

"Find employees of the Planning department, having the same first name as a member of the Production department":

```
(with a nested query)
select FirstName,Surname from Employee
where Dept = 'Plan' and FirstName = any
    (select FirstName from Employee
        where Dept = 'Prod')
```

(without nested query)

CSC343 Introduction to Databases — University of Toronto

```
SQL — 41
```

# **Negation with Nested Queries**

```
>"Find departments where there is no one named
Brown":
    select DeptName
    from Department
    where DeptName <>
        all (select Dept from Employee
        where Surname = `Brown')
>(Alternatively:)
    select DeptName from Department
        except
    select Dept from Employee
    where Surname = `Brown'
```

CSC343 Introduction to Databases — University of Toronto

SQL - 42

11

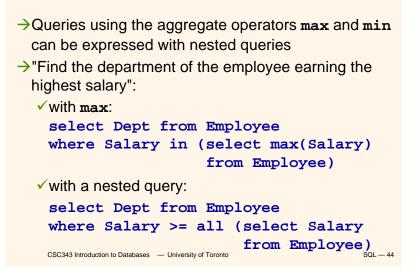
# Operators in and not in

>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')

CSC343 Introduction to Databases — University of Toronto

```
SQL — 43
```

# max and min within a Nested Query



# **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":
  - select \* from Person P
  - where exists (select \* from Person P1
    where P1.FirstName = P.FirstName
    and P1.Surname = P.Surname

```
and P1.TaxCode <> P.TaxCode)
```

```
CSC343 Introduction to Databases — University of Toronto
```

```
SQL — 45
```

# ...Another...

```
>"Find all persons who have no synonyms":
   select * from Person P
   where not exists
   (select * from Person P1
      where P1.FirstName = P.FirstName
      and P1.Surname = P.Surname
      and P1.TaxCode <> P.TaxCode)
```

CSC343 Introduction to Databases — University of Toronto

SQL — 46

# **Tuple Constructors**

- →The comparison within a nested query may involve several attributes bundled into a tuple.
- →A tuple constructor is represented in terms of a pair of angle brackets.
- The previous query can also be expressed as: select \* from Person P where <FirstName,Surname> not in (select FirstName,Surname from Person P1

```
where P1.TaxCode <> P.TaxCode)
```

SQL — 47

# **Comments on Nested Queries**

- The use of nested queries may produce less declarative queries, but often results in improved readability.
- Complex queries can become very difficult to understand.
- →The use of variables must respect scoping conventions: a variable can be used only within the query where it is defined, or within a query that is recursively nested in the query where it is defined.

# **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 = <u>D1.City</u>)
```

→What's wrong?

```
CSC343 Introduction to Databases — University of Toronto
```

SQL — 49

# **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.

CSC343 Introduction to Databases — University of Toronto

SQL - 50

# Insertions

```
→Syntax:
```

→Using values:

```
insert into Department(DeptName,City)
  values('Production','Toulouse')
```

 $\rightarrow$ Using a subquery:

```
insert into LondonProducts
 (select Code, Description
  from Product
  where ProdArea = `London')
```

```
SQL — 51
```

# **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.

```
CSC343 Introduction to Databases — University of Toronto
```

# **Deletions**

→Syntax:
 delete from TableName [where Condition ]
→"Remove the Production department":
 delete from Department
 where DeptName = `Production'
→"Remove departments with no employees":
 delete from Department
 where DeptName not in
 (select Dept from Employee)

```
CSC343 Introduction to Databases - University of Toronto
```

 $\rm SQL-53$ 

# **Notes on Deletions**

- The delete statement removes from a table all tuples that satisfy a condition.
- →The removal may produce deletions from other tables — if a referential integrity constraint with cascade policy has been defined.
- If the where clause is omitted, delete removes all tuples. For example, to remove all tuples from Department (keeping the table schema):

#### delete from Department

→To remove table Department completely (content and schema):

# drop table Department cascade

CSC343 Introduction to Databases — University of Toronto

SQL — 54

# **Updates**

→Syntax:

```
update TableName
set Attribute = < Expression | SelectSQL | null |
default >
{, Attribute = < Expression | SelectSQL | null |
default >}
[where Condition]
>Examples:
update Employee set Salary = Salary + 5
```

```
where RegNo = `M2047'
```

```
update Employee set Salary = Salary * 1.1
  where Dept = `Administration'
```

```
\rm SQL-55
```

# **Notes on Updates**

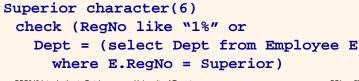
As with any side effect statement, the order of updates is important:
 update Employee
 set Salary = Salary \* 1.1
 where Salary <= 30</li>
 update Employee
 set Salary = Salary \* 1.15
 where Salary > 30
 In this example, some employees may get a double raise! How can we fix this?

# **Generic Integrity Constraints**

- →The check clause can be used to express arbitrary constraints during schema definition.
- $\rightarrow$ 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:



```
CSC343 Introduction to Databases — University of Toronto
```

```
SQL — 57
```

# Assertions

- →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":

CSC343 Introduction to Databases — University of Toronto

SQL — 58

# Views

- →Views are "virtual tables" whose rows are computed from other tables (*base relations*).
- $\rightarrow$ 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
```

# **Notes on Views**

- →SQL views cannot be mutually dependent (no recursion).
- →check option executes when a view is updated.
- →Views can be used to formulate complex queries -views decompose a problem and produce more readable solutions.
- →Views are sometimes necessary to express certain queries:
  - Queries that combine and nest several aggregate operators;
  - ✓ Queries that make fancy use of the union operator.

CSC343 Introduction to Databases — University of Toronto

# **Views and Queries**

```
→ "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
```

SQL — 61

# **Views and Queries**

→ "Find the department with highest salary
expenditures" (using a view):
create view SalBudget
(Dept,SalTotal) as
select Dept,sum(Salary)
from Employee group by Dept
select Dept from SalBudget
where SalTotal =
 (select max(SalTotal) from
SalBudget)

CSC343 Introduction to Databases — University of Toronto

SQL — 62

# **Views and Queries**

→ "Find the average number of offices per department":
Incorrect solution (SQL does not allow a cascade
 of aggregate operators):
 select <u>avg(count</u>(distinct Office))
 from Employee group by Dept
Correct solution (using a view):
 create view
 DeptOff(Dept,NoOfOffices) as
 select Dept,count(distinct Office)
 from Employee group by Dept
select avg(NoOfOffices)
 from DeptOffice

# **Access Control**

- →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 \_system 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;
  - $\checkmark$  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
```

# **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)

```
CSC343 Introduction to Databases — University of Toronto
```

SQL — 65

# grant and revoke

 $\rightarrow$ To grant 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

 $\rightarrow$ To take away privileges:

revoke Privileges on Resource from Users

[restrict | cascade ] CSC343 Introduction to Databases — University of Toronto

SQL — 66

# **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.

CSC343 Introduction to Databases — University of Toronto

# **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.

CSC343 Introduction to Databases	<ul> <li>University of Toronto</li> </ul>
----------------------------------	---

SQL — 70

# **Multiple Trigger Executions**

- →Should we allow multiple triggers to be activated by a single event?
- $\rightarrow$ If so, how do we handle trigger execution?
  - Evaluate one condition at a time and if true immediately execute action; or
  - ✓ Evaluate all conditions, then execute all associated actions.
- The execution of an action can affect the truth of a subsequently evaluated condition so the choice is significant.

# **Triggers in SQL-3**

- →Events: insert, delete, or update statements or changes to individual rows caused by these statements.
- →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.

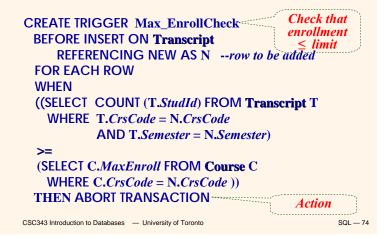
# **Triggers in SQL-3**

- →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*.

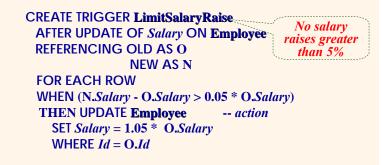
```
CSC343 Introduction to Databases — University of Toronto
```

```
SQL — 73
```

# **Before-Trigger with Row Granularity**



# After-Trigger with Row Granularity



[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