SQL (Structured Query Language)

[Astrahan, Gray, Lindsay, Selinger, …]

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

© Nick Roussopoulos

Chapter 3: Introduction to SQL

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

© Nick Roussopoulos

Domain Types in SQL

- **char(n)** Fixed length character string, with user-specified length *n.*
- **varchar(n)** Variable length character strings, with maximum length *n.*
- **int.** Integer (a finite subset of the integers).
- **smallint** Small integer (a subset of the integer domain type).
- **numeric(p,d)** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point.
- **real, double precision.** Floating point and double-precision floating point numbers.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- More domains are discussed in chapter 4 (date, currency, etc.)

© Nick Roussopoulos

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table *r* (*A*$_1$ *D*$_1$, *A*$_2$ *D*$_2$, ..., *A*$_n$ *D*$_n$,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each *A*$_i$ is an attribute name in the schema of relation *r*
  - *D*$_i$ is the data type of values in the domain of attribute *A*$_i$

  Integrity constraints
  - not null
  - primary key (*A*$_1$, ..., *A*$_n$)
  - foreign key (*A*$_m$, ..., *A*$_n$) references *r*

© Nick Roussopoulos

Integrity Constraints in Create Table

Example: Declare *ID* as the primary key for *instructor*

```
create table instructor (
    ID          char(5),
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2),
   primary key (ID),
   foreign key (dept_name) references department)
```

primary key declaration ensures not null

```
insert into instructor values ('10211', 'Smith', 'Biology', 66000);
insert into instructor values ('10211', null, 'Biology', 66000);
```

© Nick Roussopoulos

And a Few More Relation Definitions

- create table *student* (
  ID varchar(5) primary key,
  name varchar(20) not null,
  dept_name varchar(20),
  tot_cred numeric(3,0),
  foreign key *(dept_name)* references *department)* ;

- create table *takes* (
  ID varchar(5), Can we remove sec_id from the primary key?
  course_id varchar(8),
  sec_id varchar(8),
  semester varchar(6),
  year numeric(4,0),
  grade varchar(2),
  primary key (ID, course_id, sec_id, semester, year),
  foreign key (*ID*) references *student,*
  foreign key (*course_id, sec_id, semester, year*) references *section*);

© Nick Roussopoulos

Drop and Alter Table Constructs

- drop table

- alter table
  - alter table *r* add *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All tuples in the relation are assigned *null* as the value for the new attribute.
  - alter table *r* drop *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

© Nick Roussopoulos

Basic SQL Structure

- Basic commands
  - Retrieve: select
  - Update: insert, delete, update

Example

```
select   e.name
from     emp e          [from emp as e]
where    e.age >30;
```

Comments

- e is a tuple variable ranging over the emp relation
- a tuple variable followed by a dot and an attribute is an *indexed tuple variable* and specifies the corresponding attribute of the tuple
- what follows the select keyword is the *target list* (e.name)
- what follows the from is called the *tuple variable list* and consists of a list of relations and variable names
- what follows the where keyword is the *qualification clause.* It is an arbitrary Boolean expression.

© Nick Roussopoulos

SQL

- Basic format of select

  select [distinct] target_list
  from tuple_variable_list
  where qualification
  [group by group_list_subset]
  [order by target_list_subset];

- Semantics
- Form the Cartesian product of all relations in the from clause
- evaluate qualification: select the subset of the cartesian product that satisfies the qualification.
- apply grouping: partition the above subset into
- apply order and eliminate duplicates: if distinct, remove duplicate tuples and order the tuples according to the order by clause.
- evaluate target list: eliminate columns from above that are not in the target list ổ

© Nick Roussopoulos

SQL

- We'll write several queries on the following relational schema in SQL.

  sailors(sid,sname,rating)
  boats(bid, bname,color)
  reserve(sid,bid,date)

(1) Find the names of sailors who have reserved boat #2

```
select   s.sname
from     sailors s, reserve r
where    s.sid=r.sid and r.bid=2
```

(2) Find the names of sailors who have reserved red boat.

```
select s.sname
from   sailors s, boats b, reserve r
where s.sid=r.sid and r.bid=b.bid and b.color="red"
```

© Nick Roussopoulos

SQL

sailors(sid,sname,rating)
boats(bid, bname,color)
reserve(sid,bid,date)

(3) find the colors of boats reserved by Pat

```
select   b.color
from     sailor s,  boats b, reserve r
where    s.sid=r.sid and r.bid=b.bid and s.sname="Pat"
```

(4) Find the names of sailors who have reserved at least one boat.

```
select   s.sname
from     sailor s, reserve  r
where    s.sid=r.sid
```

© Nick Roussopoulos

SQL

sailors(sid,sname,rating)
boats(bid, bname,color)
reserve(sid,bid,date)

(5) Find the names of sailor who have reserved a red or green boat

```
select   s.sname
from     sailors s, boats b, reserve r
where    s.sid=r.sid and r.bid=b.bid  and (b.color="red" or b.color="green")
```

(6) Find the names of sailors who have reserved both a red and a green boat

```
select     s.sname
from       sailors s, boats b, reserve r, boats b2, reserve r2
where      s.sid=r.sid and r.bid=b.bid and b.color="red"
and        s.sid=r2.sid and r2.bid=b2.bid and b2.color="green"
```

© Nick Roussopoulos

SQL

sailors(sid,sname,rating)
boats(bid, bname,color)
reserve(sid,bid,date)

More on target lists:

- * is an abbreviation for all attributes in the from clause list
- each item in a target list can be as general as attribute_name =expression, where expression is any arithmetic or string expression over indexed tuple variables and constants. It can also contain some built-in function like sqrt, sin, mod, etc., as well as aggregates (coming up)

Example: With rating an integer from 1 to 10, this query "gives bonus" to persons who sailed two different boats on the same day

```
select   s.sname, rrating=s.rating+2
from     sailors s,  reserve r, reserve r2
where    s.sid=r.sid and s.sid=r2.sid and r.date=r2.date and r.bid != r2.bid
```

45

© Nick Roussopoulos

SQL

sailors(sid,sname,rating)
boats(bid, bname,color)
reserve(sid,bid,date)

Qualifications: Each item in a qualification can be general as expression=expression.

Example:

```
select   name1=s1.sname, name2=s2.sname
from     sailors s1 s2
where    2*s1.rating=s2.rating-1
```

46

© Nick Roussopoulos

SQL

Further elaboration:

- **Tuple variables can be implicit if the system can figure out which relation each attribute belongs to.**
- **Table names can be used as tuple variables.**

Example: Find names, ages, and departments of employees who are over 40 and work on the first floor.

```
select  ename,age, emp.dname
from    emp, dept
where   age>40 and floor=1 and emp.dname=dept.dname
```


© Nick Roussopoulos

Natural Join

- **Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column**

```
select *
from instructor natural join teaches;
```

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

© Nick Roussopoulos

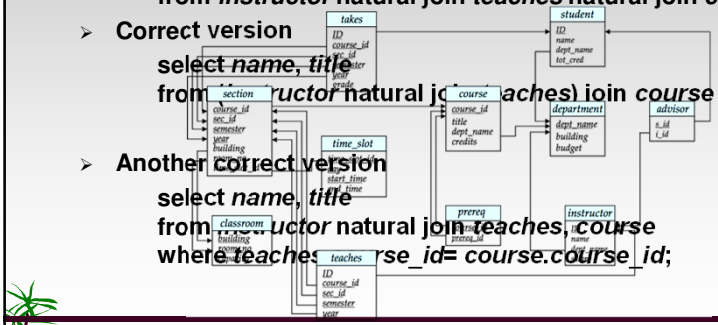
Natural Join (Cont.)

- Danger in natural join in comparing unrelated attributes with same name
  - List the names of instructors along with the titles of courses that they teach
  - Incorrect version (equates course.dept_name with instructor.dept_name)

    select *name*, *title*
    from *instructor* natural join *teaches* natural join *course*;
- Correct version

    select *name*, *title*
    from *instructor* natural join *teaches* join *course* using(*course_id*)
- Another correct version

    select *name*, *title*
    from *instructor* natural join *teaches*, *course*
    where *teaches.course_id= course.course_id*;



© Nick Roussopoulos

The Rename Operation

- The SQL allows renaming relations and attributes using the as clause:

  old-name as *new-name*
- E.g.,

  select *ID, name, salary/12* as *monthly_salary*
  from *instructor*
- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

  select distinct *T. name*
  from *instructor* as *T, instructor* as *S*
  where *T.salary > S.salary* and *S.dept_name = 'Comp. Sci.'*
- Keyword as is optional and may be omitted

  instructor as *T* ≡ *instructor T*

© Nick Roussopoulos

String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all instructors whose name includes the substring "dar".

```
select name
from instructor
where name like '%dar%'
```

- Match the string "100 %"

```
like '100 \%' escape '\'
```

- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

© Nick Roussopoulos

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

```
select distinct name
from    instructor
order by name
```

- We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default.
  - Example: order by *name* desc
- Can sort on multiple attributes
  - Example: order by *dept_name, name*

© Nick Roussopoulos

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)

```
select name
from instructor
where salary between 90000 and 100000;
```

- Tuple comparison

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

© Nick Roussopoulos

Duplicates

- In relations with duplicates, SQL treats the relations as multiset relations unless otherwise specified

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P
```

- **Multiset** versions of some of the relational algebra operators – given multiset relations r_1 and r_2:
  1. $\sigma_\theta(r_1)$: If there are c_1 copies of tuple t_1 in r_1, that satisfy θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$.
  2. $\Pi_A(r)$: For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$
  3. r_1 x r_2: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are c_1 x c_2 copies of the tuple $t_1.t_2$ in r_1 x r_2

© Nick Roussopoulos

Duplicates (Cont.)

- Example: Suppose multiset relations r_1 (*A, B*) and r_2 (*C*) are as follows:

  r_1 = {(1, *a*) (2, *a*)} r_2 = {(2) (3) (3)}

- $\Pi_B(r_1)$ ={(a), (a)}

- $\Pi_B(r_1)$ x r_2 ={(*a*,2), (*a*,2), (*a*,3), (*a*,3), (*a*,3), (*a*,3)}

© Nick Roussopoulos

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate is null can be used to check for null values.
  - Example: Find all instructors whose salary is null.

    select *name*
    from *instructor*
    where *salary* is null

© Nick Roussopoulos

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - Example*: 5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* or *true*) = *true*,
    (*unknown* or *false*) = *unknown*
    (*unknown* or *unknown*) = *unknown*
  - AND: *(true* and *unknown) = unknown,*
    (false and *unknown) = false,*
    (unknown and *unknown) = unknown*
  - NOT*: (*not *unknown) = unknown*
- Result of where clause predicate is treated as *false* if it evaluates to *unknown*

© Nick Roussopoulos

SQL Aggregates

- These functions operate on the multiset of values of a column of a relation, and return a value

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

© Nick Roussopoulos

SQL Aggregates

- Find the average salary of instructors in the Computer Science department
  - select avg (*salary*)
    from *instructor*
    where *dept_name*= 'Comp. Sci.';
- Find the total number of instructors who teach a course in the Spring 2010 semester
  - select count (distinct *ID*)
    from *teaches*
    where *semester* = 'Spring' and *year* = 2010
- Find the number of tuples in the *course* relation
  - select count (*)
    from *course*;

© Nick Roussopoulos

Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - select *dept_name*, avg (*salary*) as avg_salary
    from *instructor*
    group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

© Nick Roussopoulos

Aggregation (Cont.)

- cannot mix tuple and aggregate values (attributes in select clause outside of aggregate functions must appear in the "group by" list)

```
select dept_name, ID, avg (salary)
from instructor
group by dept_name;
```

- multiple attribute "group by"
  - count the number of rooms used by course sections for each building per year

```
select    building, year, count(room_no)
from      section
group by  building, year
```

61

© Nick Roussopoulos

Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups

62

© Nick Roussopoulos

Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor*
  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount (all values are null)
- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

© Nick Roussopoulos

Rownum

- select * from AUTHOR A
  where rownum = 1;

	ID	NAME
1	1	Anindya Datta

- select * from AUTHOR A
  where rownum = 3;

ID	NAME

- select * from (select A.*, rownum as rn from AUTHOR A)
  where rn = 3;

	ID	NAME	RN
1	3	Sandeepan Banerjee	3

- select ID,Name from (select A.*, rownum as rn from AUTHOR A)
  where rn = 3;

	ID	NAME
1	3	Sandeepan Banerjee

© Nick Roussopoulos

SQL Set Operators

- union, intersect, and minus (except).

Example: Find the names of employees who work in the toy department and make at most 60K.

```
(select   ename
 from     emp
 where    dname="toy")
minus
 (select  ename
  from    emp
  where   sal>60K)
```

© Nick Roussopoulos

SQL Nested Queries

Emp	ename	sal	dept
	Gary	30K	toy
	Shirley	35K	candy
	Christos	37K	shoe
	Robin	22K	toy
	Uma	30K	shoe
	Tim	12K	

Dept	dept	floor	mgr
	candy	1	Irene
	toy	2	Jim
	men	2	John
	shoe	1	George

- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.
  - "in" tests for set membership
  - "not in" tests for non membership

Example: Find names of employees who work on the 1st floor.

```
select   ename
from     emp
where    dname in
                 (select   dname
                  from     dept
                  where    floor=1)
```

```
select   ename
from     emp
where    dname  not in
                 (select   dname
                  from     dept
                  where    floor=1)
```

The same query in flat form is:

```
select   ename
from     emp, dept
where    emp.dname=dept.dname and floor=1
```

© Nick Roussopoulos

SQL

- Scoping of variables works exactly as in C

Example: Find the names who take a course from their advisor

```
select   sname
from     student
where    s# in
                (select   s#
                 from     enroll
                 where  c# in
                              (select    c#
                               from      class
                               where     prof=student.advisor))
```

67

© Nick Roussopoulos

SQL Connectives

- <op> any
- <op> all where <op> is one of { =, !=, <,>, >=, <=}

Example: Find names of employees who make more than everybody on the first floor.

```
select   ename
from     emp
where    sal>all
                (select    sal
                 from      emp, dept
                 where     emp.dname=dept.dname and floor=1)
```

68

© Nick Roussopoulos

Set Comparison

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept name = 'Biology');
```

- Same query without "> some" clause

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept name = 'Biology';
```


© Nick Roussopoulos

Definition of Some Clause

- F <op> some $r \Leftrightarrow \exists\, t \in r$ such that (F <op> t)
  Where <op> can be: $<, \leq, >, =, \neq$

(5 < some {0, 5, 6}) = true (read: 5 < some tuple in the relation)

(5 < some {0, 5}) = false

(5 = some {0, 5}) = true

(5 ≠ some {0, 5}) = true (since 0 ≠ 5)

Note: (= some) ≡ in But (≠ some) ≢ not in

© Nick Roussopoulos

Example Query

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = 'Biology');
```

© Nick Roussopoulos

Definition of all Clause

- F <op> all $r \Leftrightarrow \forall t \in r$ (F <op> t)

(5 < all $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}$) = false

(5 < all $\begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}$) = true

(5 = all $\begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}$) = false

(5 ≠ all $\begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}$) = true (since 5 ≠ 4 and 5 ≠ 6)

Note (≠ all) is equivalent to not in see example 4 above
But (= all) is not equivalent to in see example 3 above

© Nick Roussopoulos

Test for Empty Relations

- The exists construct returns the value true if the argument subquery is nonempty.

```
select  *
from instructor
exists  (select *
            from section
            where year ='2020';
```

- not exists is true if subquery returns a nonempty result

© Nick Roussopoulos

Correlation Variables

- "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

```
select course_id
from section as S
where semester = 'Fall' and year= 2009 and
        exists (select *
                from section as T
                where semester = 'Spring' and year= 2010
                        and S.course_id= T.course_id);
```

- Correlated subquery
- Correlation name or correlation variable

© Nick Roussopoulos

Not Exists

- Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

- Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- *Note:* Cannot write this query using = all and its variants

75

© Nick Roussopoulos

Test for Absence of Duplicate Tuples

- The unique construct tests whether a subquery has any duplicate tuples in its result.
- Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
                    and R.year = 2009);
```

76

© Nick Roussopoulos

Derived Relations in the "from clause"

- SQL allows a subquery expression to be used in the "from clause"
- Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

```
select dept_name, avg_salary
 from (select dept_name, avg (salary) as avg_salary
         from instructor
         group by dept_name)
 where avg_salary > 42000;
```

- Note that we do not use the having clause
- Another way to write above query

```
select dept_name, avg_salary
 from (select dept_name, avg (salary)
         from instructor
         group by dept_name) as dept_avg (dept_name, avg_salary)
 where avg_salary > 42000;
```

77

© Nick Roussopoulos

Views

- In some cases, it is not desirable for all users to see the entire table
- Students need to know an instructors name and department, but not the salary.

```
select ID, name, dept_name
from instructor
```

- Any relation other than the "base relational schema" but is made visible to a user via a query ("virtual relation") is called a **view**.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

78

© Nick Roussopoulos

View Definition

- A view is defined using the create view statement which has the form

  create view *v* as < query expression >

  where <query expression> is any legal SQL expression. The view name is represented by *v.*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving the definition of a query expression;

- Query Modification: when a query uses a view, the defining expression is substituted in its place.

© Nick Roussopoulos

Example Views

- A view of instructors without their salary

```
create view faculty as
    select ID, name, dept_name
    from instructor;
```

- Create a view of department salary totals

```
create view departments_total_salary(dept_name, total_salary) as
    select dept_name, sum (salary)
    from instructor
    group by dept_name;
```

© Nick Roussopoulos

Views Defined Using Other Views

- create view *physics_fall_2009* as
  select *course.course_id*, *sec_id*, *building*, *room_number*
  from *course*, *section*
  where *course.course_id* = *section.course_id*
  and *course.dept_name* = 'Physics'
  and *section.semester* = 'Fall'
  and *section.year* = '2009';

- create view *physics_fall_2009_watson* as
  select *course_id*, *room_number*
  from *physics_fall_2009*
  where *building*= 'Watson';

© Nick Roussopoulos

With Clause

- The with clause provides a way of defining a temporary view whose definition is available only to the query in which the with clause occurs.

- Find all departments with the maximum budget

```
with max_budget (value) as
   (select max(budget)
    from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

© Nick Roussopoulos

Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

© Nick Roussopoulos

SQL Updates

- Insert command format:

  insert into relation_name values (value_list)

  or

  insert into relation_name select_statement

- Semantics of insert:
  - Format 1: Add the tuple corresponding to value_list into relation_name.
  - Format 2: Execute the select statement, and then add the resulting tuples into relation_name.

Example:

```
insert into student
    values(1, "Carey", "CS", "Stonebraker")
```

© Nick Roussopoulos

SQL Updates

Example: Suppose we have a relation
register(s#, name, paid),

in which registered students are recorded. After the end of registration week, we execute

```
insert into  student
             select   r.s#, r.name
             from     register r
             where    r.paid="yes"
```

© Nick Roussopoulos

SQL Delete

- Delete command format:
  delete relation_name **where** qualification

- Semantics of delete: Execute the corresponding select command i.e.,

```
select    full_target_list
from      relation_name
where     qualification
```

and then remove the resulting tuples from relation_name.

© Nick Roussopoulos

SQL Delete

The following command expels CS majors who received less than 1.5 in a CS course:

```
delete   student
where    major="CS" and s# in
                    (select   s# from enroll, course
                     where    enroll.s#student.s# and grade < 1.5
                     and      enroll.c#course.c# and dept="CS")
```

© Nick Roussopoulos

SQL Update

- Update format

```
update   relation_name
set      target_list
where    qualification
```

1. Semantics of update: it is equivalent to executing the following:

```
1. insert into   del_temp
                 select   full_target_list
                 from     relation_name
                 where    qualification
2. insert into   app_temp
                   select  ext_target_list
                   from    relation_name
                   where   qualification
```

The difference of ext_target_list from full_ target_list is that the former is augmented with tuple_variable.attr_name for all attributes of the from relation(s) that don't appear in full_target_list

3. remove the tuples in del_temp from relation_name
4. add the tuples in app_temp into relation_name.

© Nick Roussopoulos

SQL Update

Example: Give a 10% grade raise in every CS major in CS564.

```
update   enroll
set      grade=1.1 * grade
where    c#="CS564" and s# in
         (select s#
          from student
          where major="CS")
```

The equivalent

```
insert into del_temp
    select s#,c#,grade
    from enroll
    where c#="CS564" and s# in
       (select s#
        from student
        where major="CS")
```

```
insert into app_temp
    select s#,c#,grade=1.1 * grade
    from enroll
    where c#="CS564" and s# in
       (select s#
        from student
        where major="CS")
```

then do (enroll minus del_temp) union app_temp

© Nick Roussopoulos

Outer Joins

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.

© Nick Roussopoulos

Left Outer Join

Course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- ***course* natural left outer join *prereq***

course_id	title	dept_name	credits	prere_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

© Nick Roussopoulos

Right Outer Join

Course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- ***course* natural right outer join *prereq***

course_id	title	dept_name	credits	prere_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

© Nick Roussopoulos

Full Outer Join

Course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- ***course* natural full outer join *prereq***

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

© Nick Roussopoulos

Update of a View

- **Add a new tuple to *faculty* view which we defined earlier**

  insert into *faculty* values ('30765', 'Green', 'Music');

  This insertion causes an insertion into the *instructor* relation of the tuple:

  ('30765', 'Green', 'Music', null)

- **Most SQL implementations allow updates only on simple views**
  - **The key is in the view definition and the value is specified; other attributes can be omitted and are set to null.**
  - **No expressions, aggregates, or distinct specification in the select clause**
  - **No join, group by or having clause views**
  - **The query does not have a group by or having clause.**

© Nick Roussopoulos

Integrity Constraints

- Numerical constraints
  - keys
  - quantified constraints of relationships (1-1, 1-N)
- Domain constraints
  - type declarations
    - fixed length char strings
    - [small] integer
    - floating point, double precision
  - extensions
    - date
    - currency
  - NULL values
    - for values to be filled in later on
    - primary keys cannot have NULL value
    - some attributes can be specified as NOT NULL
- Key constraints
  - keys have unique values
  - primary key- a candidate key declared primary
  - unique key- a candidate key
  - foreign key- a set of attributes that are primary keys for other relations)

© Nick Roussopoulos

Referential Integrity

- a value in a tuple of a relation must appear in at least one tuple of another relation, e.g. every value of eno in WORKS-IN is in at least one tuple of EMP

  EMP(eno,ename,sal)
  DEPT(dno,dname,floor)
  WORKS-IN(eno,dno,hours)
  eno: foreign key dno: foreign key
- referential integrity

  π_{eno} (WORKS-IN) $\subseteq$ π_{eno} (EMP)
- Updates may violate ref. integrity constraints
  - insertion: inserts in the referencing relation a value that is not in the referenced one
  - update: referencing relation (as insertions)
  - deletion: delete a tuple referenced from the referencing relation
  - update: referenced relation (as deletions)

© Nick Roussopoulos

Assertions

- declare arbitrary expressions that have to be satisfied
  assertion = predicate expression
  e.g. GPA > 2.8
  sum(all_charges) < credit_line
- SQL:
  create assertion balance on deposit: balance >= 0
  assert address on insertion deposit::
  exists (select * from customer
  where customer.customer.name = deposit.customer.name)
- When assertion is introduced, the condition is validated and if not rejected. After that it stays a a watchdog for violations- VERY expensive

© Nick Roussopoulos

Triggers

- trigger is a statement that is executed when an update is applied to the database
- the trigger has 2 parts
  - condition under which the trigger is executed
  - the actions to be taken when executed

e.g.

```
define trigger overdraft on update of account T
        (if new T.balance < 0 then ( insert into borrow values
          (T.branch.name,T.account-number,T.customer-name, T.balance)
         update deposit S
          set S.balance = 0
           where S.account-number = T.account-number))
```

- triggers make the system reactive
- triggers are also called active rules and they are also VERY expensive

© Nick Roussopoulos

Large-Object Types

- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*:
  - blob: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - clob: character large object -- object is a large collection of character data
  - When a query returns a large object, a pointer is returned rather than the large object itself.

© Nick Roussopoulos

Authorization Specification in SQL

- The grant/revoke statement is used to authorize/de-authorize

```
grant select on (id,ename) on emp to Jeff,Jim,Joe;
grant update (adr) on emp to public;
grant delete  on emp to Joe;
revoke select on emp from Jeff;
revoke update (adr) on emp from public;
```

- The format is

  grant <privilege list> on <relation name or view name> to <user list>
  - <user list> is: a user-id, public
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the dba).

© Nick Roussopoulos