Lic. Informatique, Univ Lille 1, 2010-11

- **Introduction aux bases de données relationnelles**
  - 3ème séance: **Introduction à SQL**
- **Enseignante**: C. Kuttler
- **Biblio**: chapitre 3 de *Database Systems Concepts* de Silberschatz et al, McGraw-Hill (6ème edition, 2010)
- Ces transparents sont une adaptation de ceux disponibles sur le site du livre: **www.db-book.com**

Chapter 3: Introduction to SQL (first part)

Database System Concepts, 6th Ed.

Chapter 3: Introduction to SQL

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Aggregate Functions
- Additional Basic Operations

- **This slide set covers only part of Chap 3 of the textbook!**

History

- 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.

Domain Types in SQL

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

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

- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20) **not null,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))

- **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000) ☺
- **insert into** *instructor* **values** ('10211', null, 'Biology', 66000) ☹

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

Example: Declare *branch_name* as the primary key for *branch*

```
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 on an attribute automatically ensures **not null**

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) **primary key**,
  course_id **varchar**(8),
  sec_id **varchar**(8),
  semester **varchar**(6),
  year **numeric**(4,0),
  grade **varchar**(2),
  foreign key (*ID*) **references** *student,*
  foreign key (*course_id, sec_id, semester, year*) **references** *section*);

And more still

- **create table** *course* (
  course_id **varchar**(8) **primary key**,
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  foreign key *(dept_name*) **references** *department)*);

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.

Basic Query Structure

- A typical SQL query has the form:

$$\textbf{select } A_1, A_2, \ldots, A_n$$
$$\textbf{from } r_1, r_2, \ldots, r_m$$
$$\textbf{where } P$$

  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate, i.e. a test that evaluates to a boolean value for each tuple.

- The result of an SQL query is a relation.

The select Clause

- The **select** clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*

  from *instructor*

select name from instructor

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
- **select** NAME **from** INSTRUCTOR

select *dept_name* from *instructor*

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

- **Observation:** each department is listed once per instructor!

The select Clause: duplicates or not?

- SQL allows **duplicates** in relations as well as in query results.
- To force the **elimination of duplicates**, insert the keyword **distinct** after select.
- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword **all** specifies that duplicates not be removed.

```
select all dept_name
from instructor
```

The select Clause: asterisk & arithmetics

- An asterisk in the select clause denotes "all attributes"

  select *
  from *instructor*

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.

- **select** *ID, name, salary/12*
  from *instructor*

  returns a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

- Later, we'll see arithmetics for special types, e.g. dates

The where Clause

- The **where** clause selects **rows** for the result, that satisfy a specific **condition**
  - corresponds to the selection predicate of the relational algebra. Sorry this is an unhappy historic choice! ☹
  - evaluates to a boolean value (true or false) per row
- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Find all instructors in Comp. Sci. dept with salary > 70000

```
select name
        from instructor
        where dept_name = 'Comp. Sci.' and salary > 70000
```

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

select *name*

from *instructor*

where *dept_name* = 'Comp. Sci.'

and *salary* > 70000

name
Katz
Brandt

Exercise

- Comparisons can also be applied to results of arithmetic expressions.
- Which computer science instructors earn over 6000$ per month?
  - Write an SQL query!

The from Clause

- The **from** clause lists the relations needed for the query
  - Corresponds to the Cartesian product *operation* of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  select ∗
  from *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Cartesian Product

instructor

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

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Physics	95000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Physics	95000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Physics	95000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Physics	95000	10101	FIN-201	1	Spring	2010
10101	Srinivasan	Physics	95000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Physics	95000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…			
12121	Wu	Physics	95000	10101	CS-1			
12121	Wu	Physics	95000	10101				
12121	Wu	Physics	95000	10101				
12121	Wu	Physics	95000	10101				
12121	Wu	Physics	95000	15151				
12121	Wu	Physics	95000	22222	PHY-10			
…	…	…	…	…	…	…		
…	…			..	…	…		…

Sorry: bug in columns dept_name, salary, teaches.ID. I've dropped a mail to the book's Authors. It should be fixed later.

Queries on multiple relations: joins

- For all instructors who have taught courses, find their names and the course ID of the courses they taught.
-

select *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID*

name	*Course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Typical SQL query

- **select** A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P
- **select** clause
  - used to list the attributes desired in the result of the query
- **from** clause
  - list of the relations to be accessed in the evaluation of the query
- **where** clause
  - Predicate involving attributes of the relations in the from clause

What's the result of a query?

- Intuition: first look at **from**, then **where**, and then **select**!
- To understand the meaning:
  - Generate a cartesian product of relations liste in the from clause
  - Apply the predicates specified in the where clause on the result of step 1
  - For each tuple in the result of step 2, output the attributes (or results of expressions) specified in the select clause

1. This is **not** how the DBMS actually executes the query. It uses optimization techniques!

Queries on multiple relations: joins

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```
select section.course_id, semester, year, title
 from section, course
 where   section.course_id = course.course_id  and
              dept_name = 'Comp. Sci.'
```

Try Writing Some Queries in SQL

- Suggest queries to be written…..

Aggregate Functions

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

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

Aggregate Functions (Cont.)

- Find the number of tuples in the *course* relation
- **select count** (*)
  from *course*;
- 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

Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - **select** *dept_name*, **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

Aggregation (Cont.)

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list
  - /* erroneous query */
    select *dept_name*, *ID*, **avg** (*salary*)
    from *instructor*
    group by *dept_name*;

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

The Rename Operation

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

  old-name **as** *new-name*

- Example 1: **renaming attributes of the resulting relation**
  - **select** *ID, name, salary/12* **as** *monthly_salary*
    from *instructor*

- Example 2: **joining a relation with itself.**
  - 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, mamely in *Oracle*,it can't be used to rename a relation in the from clause. Simply use
  instructor T, instead of *instructor* **as** *T*

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)

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*

Friday 10 sept 2010: summary

- SQL as data-definition language and data-manipulation language
- SQL query structure: **select-from-where**
- **Joins** between 2 tables
- **Aggregate functions**:
  - avg,min,max,sum,count
  - **Group by** clause, **having** clause
- **Order by** clause, **as** clause (*renaming*)