



Database Systems

Lecture 5:

Introduction to SQL (part 1)

Dr. Momtazi
momtazi@aut.ac.ir

based on the slides of the course book



Outline

- **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



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.



Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- And as we will see later, also other information such as
 - The set of indices to be maintained for each relations.
 - Security and authorization information for each relation.
 - The physical storage structure of each relation on disk.



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,d*)**. Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **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-constraint1),  
                ...,  
                (integrity-constraintk))
```

- 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),  
    dept_name varchar(20),  
    salary     numeric(8,2))
```



Integrity Constraints in Create Table

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

Example:

```
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 department(  
    dep_name      varchar(20),  
    building      varchar(15),  
    budget        numeric(12,2),  
    primary key (dept_name));
```




And a Few More Relation Definitions

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



And a Few More Relation Definitions

```
■ create table section (  
    course_id          varchar(8),  
    sec_id             varchar(8),  
    semester           varchar(6),  
    year                numeric(4,0),  
    building            varchar(15),  
    room_number        varchar(7),  
    time_slot_id       varchar(4),  
    primary key (course_id, sec_id, semester, year),  
    foreign key (course_id) references course);
```



And a Few More Relation Definitions

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



And a Few More Relation Definitions

- **create table** *takes* (
 ID **varchar**(5),
 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*);

- Note: *sec_id* can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester



Updates to tables

■ Insert

- **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);

■ Delete

- Remove all tuples from the *student* relation
 - **delete from** *student*

■ Drop Table

- **drop table** *r*

■ Alter

- **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 exiting 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.



Outline

- 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



Basic Query Structure

- A typical SQL query has the form:

select A_1, A_2, \dots, A_n
from r_1, r_2, \dots, r_m
where P

- A_i represents an attribute
 - R_i represents a relation
 - P is a predicate.
- The result of an SQL query is a relation.



The select Clause

- The **select** clause lists 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*

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
 - E.g., *Name* \equiv *NAME* \equiv *name*
 - Some people use upper case wherever we use bold font.



Example: select clause

<i>name</i>
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Figure 3.2 Result of “*select name from instructor*”.



The select Clause

- SQL allows duplicates in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name  
from instructor
```

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

```
select all dept_name  
from instructor
```



Example: select clause

<i>dept_name</i>
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

Figure 3.3 Result of “select *dept_name* from *instructor*”.



The select Clause

- An asterisk in the select clause denotes “all attributes”

select *
from *instructor*

- An attribute can be a literal with no **from** clause

select '437'

- Results is a table with one column and a single row with value “437”
- Can give the column a name using:

select '437' as FOO

- An attribute can be a literal with **from** clause

select 'A'
from *instructor*

- Result is a table with one column and N rows (number of tuples in the *instructors* table), each row with value “A”



The select Clause

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

- The query:

```
select ID, name, salary/12  
from instructor
```

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

- Can rename “*salary/12*” using the **as** clause:

```
select ID, name, salary/12 as monthly_salary
```



The where Clause

- The **where** clause specifies conditions that the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.

- To find all instructors in Comp. Sci. dept

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```



The where Clause

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**

- To find all instructors in Comp. Sci. dept with salary > 70000

select *name*

from *instructor*

where *dept_name* = 'Comp. Sci.' **and** *salary* > 70000

- Comparisons can be applied to results of arithmetic expressions.



Example: where clause

<i>name</i>
Katz
Brandt

Figure 3.4 Result of “Find the names of all instructors in the Computer Science department who have salary greater than \$70,000.”



The from Clause

- The **from** clause lists the relations involved in 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.
 - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).



Cartesian Product

instructor

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

teaches

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
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

<i>Inst.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...
...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...
...



Example: from clause

- Find the names of all instructors who have taught some course and the course_id
 - **select** *
from *instructor* , *teaches*
where *instructor.ID* = *teaches.ID*



Example: from Clause

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
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 3.8 The natural join of the *instructor* relation with the *teaches* relation.



Example: from clause

- Find all instructors who have taught some course
 - **select** *
from *instructor* , *teaches*
where *instructor.ID* = *teaches.ID*

- This query can be written more concisely using the natural-join operation in SQL as:
 - **select** *
from *instructor* **natural join** *teaches*



Example: from clause

- Find the names of all instructors who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor , teaches*
where *instructor.ID = teaches.ID*
 - **select** *name, course_id*
from *instructor natural join teaches*



Example: from Clause

<i>name</i>	<i>course_id</i>
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	HN-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

Figure 3.7 Result of “For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught.”



Examples: from clause

- Find the names of all instructors who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor , teaches*
where *instructor.ID = teaches.ID*

- Find the names of all instructors in the Art department who have taught some course and the course_id
 - **select** *name, course_id*
from *instructor , teaches*
where *instructor.ID = teaches.ID and instructor. dept_name = 'Art'*



The Rename Operation

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

old-name as new-name

- 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 \equiv *instructor T*



Cartesian Product Example

- Relation *emp-super*

<i>person</i>	<i>supervisor</i>
Bob	Alice
Mary	Susan
Alice	David
David	Mary

- Find the supervisor of “Bob”

Select *supervisor*
from *emp-super*
where *person*= 'Bob'



Cartesian Product Example

- Relation *emp-super*

<i>person</i>	<i>supervisor</i>
Bob	Alice
Mary	Susan
Alice	David
David	Mary

- Find the supervisor of the supervisor of “Bob”

Select *B.supervisor*
from *emp-super as A, emp-super as B*
where *A.supervisor = B.person*



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 '\'
```

in that above we use backslash (\) as the escape character.



String Operations (Cont.)

- Patterns are case sensitive.
- Pattern matching examples:
 - 'Intro%' matches any string beginning with "Intro".
 - '%Comp%' matches any string containing "Comp" as a substring.
 - '___' matches any string of exactly three characters.
 - '___ %' matches any string of at least three characters.
- 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.



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*



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, $\geq \$90,000$ and $\leq \$100,000$)
 - **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'*);



Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- **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 , and t_1 satisfies selections σ_{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 $\Pi_A(t_1)$ denotes the projection of the single tuple t_1 .
 3. $r_1 \times r_2$: 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 $c_1 \times c_2$ copies of the tuple $t_1 \cdot t_2$ in $r_1 \times r_2$



Duplicates (Cont.)

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

$$r_1 = \{(1, a) (2, a)\} \quad r_2 = \{(2), (3), (3)\}$$

- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$, while $\Pi_B(r_1) \times r_2$ would be $\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$
- SQL duplicate semantics:

select A_1, A_2, \dots, A_n
from r_1, r_2, \dots, r_m
where P

is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$



Questions?