Intro to DB

CHAPTER 3
SQL

Chapter 3: SQL

- Overview
- Data Definition
- Basic 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 (Y2K!), SQL:2003, SQL:2008
- 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.

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:

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

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 *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.

Integrity Constraints in Create Table

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

Declare *dept_name* as to be a reference to *department.*

- **primary key** declaration on an attribute automatically ensures **not null**

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)*);
- **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

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)*);

  - Primary key declaration can be combined with attribute declaration as shown above

Drop **and** ***Alter Table*** **Constructs**

- **drop table:** deletes all information about the dropped relation from the database.

- **alter table**: used to add or drop attributes to an existing relation

  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.*

  - *null* is assigned to the new attribute for each tuple

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*

  - (dropping of attributes is not supported by many databases)

Basic Structure of SQL Queries

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  - A_is represent attributes
  - r_is represent relations
  - P is a predicate.

- The result of an SQL query is a relation.

The *select* Clause

- Find the names of all instructors:

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

select *
from *instructor*

- NOTE:
  - SQL does not permit the ‘-’ character in names (use ‘_’ in a real implementation).
  - SQL names are case insensitive.

The *select* Clause (cont.)

- 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

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

select all *dept_name*
from *instructor*

The *select* Clause (cont.)

- The **select** clause can contain arithmetic expressions
  - +, –, ∗, /
  - on constants or attributes of tuples.
- The query:

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

The *where* Clause

- Corresponds to the selection predicate of the relational algebra.
- Predicate involving attributes of the relations that appear in the **from** clause.
- Find all instructors in Comp. Sci. dept with salary > 80000

- Comparison conditions: >, <, =, <=, >=, !=
- Logical connectives: **and, or, not**
- Comparisons can be applied to results of arithmetic expressions
- SQL includes a **between** comparison operator

  where *salary* **between** 90000 **and** 100000

The *from* Clause

- Lists the relations to be scanned in the evaluation of the expression.
- Corresponds to the Cartesian product operation of the relational algebra.
- *Instructor* x *teaches*

  select *
  from *instructor, teaches*

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	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	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

select ∗ **from** *instructor, teaches*

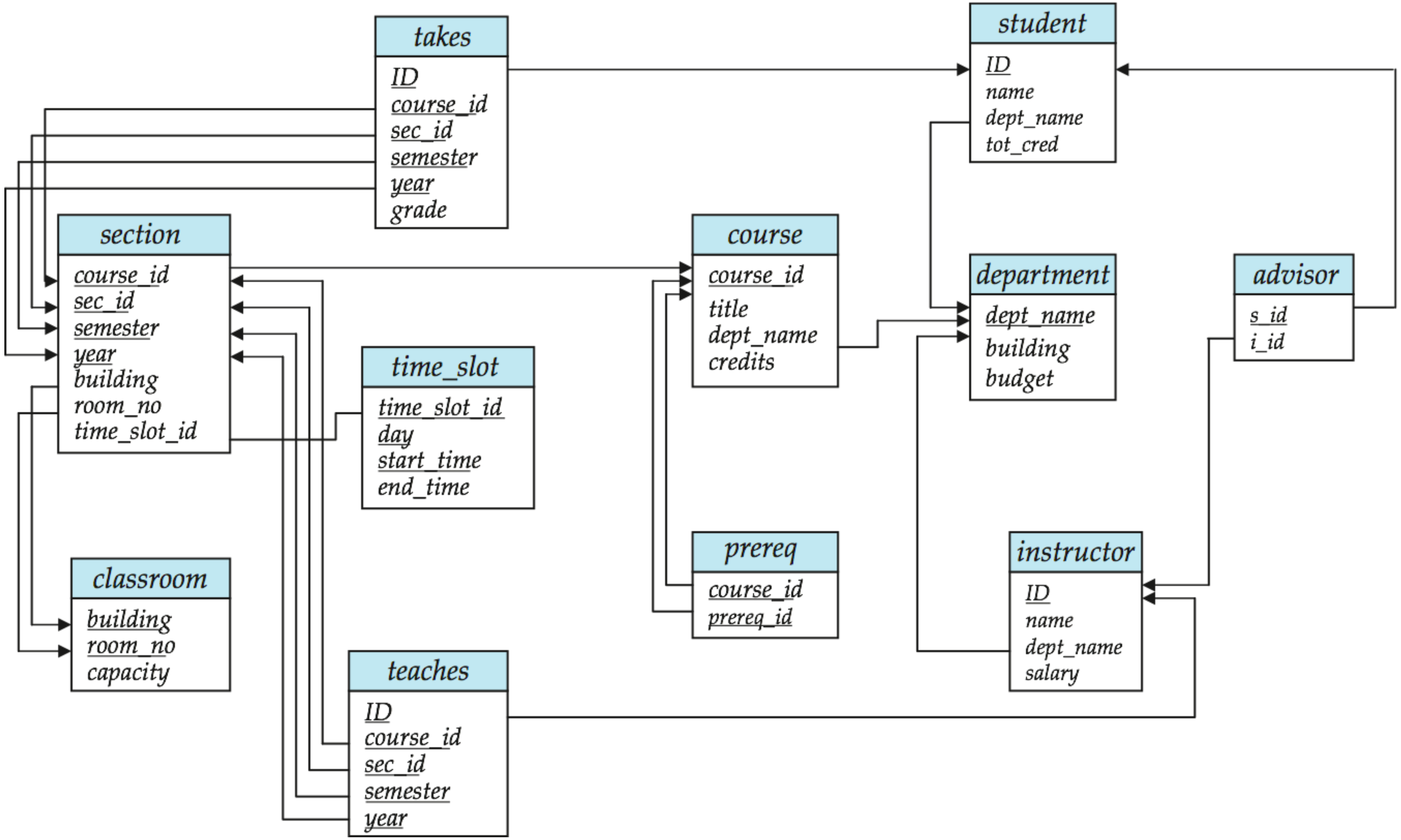
Joins

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

- 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.'

Schema Diagram for the University



SQL Examples

- Find the IDs of students advised by an instructor named Einstein.

- Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

The *Rename* Operation

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

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

  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’.

- *S*, *T* are called ***tuple variables***
- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

String Operations

- For comparisons on character strings
- Patterns are described using special characters:
  - percent (%): matches any substring.
  - underscore (_). matches any character.
- Find all courses whose title includes the substring “data”.

- Escape character to specify % and \ within string

  like ‘100\%’
- A variety of string operations such as
  - concatenation (using “||”)
  - case conversion, string length, substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order all instructors

- **desc** for descending order or **asc** for ascending order (default)
  - **order by** *name* **desc**

- Can sort on multiple attributes
  - **order by** *dept_name, name*
  - **order by** *dept-name* **desc**, *name* **asc**

Set Operations

- The set operations: **union, intersect, except** correspond to the relational algebra operations $\cup, \cap, -$.
-
- To retain all duplicates use multiset versions:

  union all, intersect all and **except all.**

  Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:
  - in *r* **union all** *s*
  - in *r* **intersect all** *s*
  - in *r* **except all** *s*

Set Operations (cont.)

- *Find all customers who have a loan, an account, or both:*

  (select *customer-name* **from** *depositor*)
  union
  (select *customer-name* **from** *borrower)*

- *Find all customers who have both a loan and an account.*

  (**select** *customer-name* **from** *depositor*)
  intersect
  (select *customer-name* **from** *borrower)*

- *Find all customers who have an account but no loan.*

  (**select** *customer-name* **from** *depositor*)
  except
  (select *customer-name* **from** *borrower)*

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
-
- 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.

  Find all instructors whose salary is null.

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

  - Why not the following?

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

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*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
-

Aggregate 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

Aggregate Functions (Cont.)

- 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

- Find the number of tuples in the *course* relation

  select count (*)
  from *course*;

Aggregate Functions – Group By

- Find the average salary of instructors in each department

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

  - Note: departments with no instructor will not appear in result

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*;

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

-

Nested Subqueries

- A ***subquery*** is

- Common use of subqueries:

  perform tests for set membership, set comparisons, and set cardinality.

Nested Query – Examples

- Find courses offered in Fall 2009 and in Spring 2010

  select distinct *course_id*
  from *section*

- Find courses offered in Fall 2009 but not in Spring 2010

  select distinct *course_id*
  from *section*

Nested Query – Examples

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

select count (**distinct** *ID*)
from *takes*

- Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features

Set Comparison

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

select distinct *T.name*
from *instructor* **as** *T*, *instructor* **as** *S*

- Same query using > **some** clause

select *name*
from *instructor*

Set Comparison (cont.)

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

select *name*
from *instructor*

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Correlation Variables

- Yet another way of specifying the query “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
- *S*: Correlation name or correlation variable

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')
```

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

Modification of the Database – Deletion

- Delete all courses of Appl. Math department

  delete from *course*
  where *dept_name* = 'Appl. Math'

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

Modification of the Database – Deletion

- Delete all instructors whose salary is less than the average salary of instructors.

  delete from *instructor*

  where *salary*< (**select avg** (*salary*) **from** *instructor*);

  - Problem:

  - Solution used in SQL:

    1. First, compute **avg** salary and find all tuples to delete
    2. Next, delete all tuples found above
       (without recomputing **avg** or retesting the tuples)

Modification of the Database – Insertion

- Add a new tuple to *course*

  insert into *course*
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- or equivalently

- Add a new tuple to *student* with *tot_creds* set to null

  insert into *student*
  values ('3003', 'Green', 'Finance', *null*);

Insertion (Cont.)

- Add all instructors to the *student* relation with tot_creds set to 0

  insert into *student*
  select *ID, name, dept_name, 0*
  from *instructor*

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation
  - otherwise queries like the following would cause problems

    (if *table1* did not have any primary key defined)

    insert into *table*1 **select** * **from** *table*1

Modification of the Database – Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - Write two **update** statements:
  - The order is important

END OF CHAPTER 3