SQL (SECTION 3.2, 3.3, 3.4, 3.5, 3.7, and 3.8 up to 3.8.5, 3.9)

Note: Parts of this lecture were developed by Professor Zartoshty
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

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- The set of indices to be maintained for each relation.
- 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 $n$ 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.
SQL INTRODUCTION

• For the purposes of the following, assume the existence of these two relations:
  Emp (SS#, name, age, salary, dno)
  Dept (dno, dname, floor, mgrSS#)

• **Structured Query Language (SQL)** consists of four basic commands: Select, Insert, Update, and Delete.

• SQL provides commands to change the state of database: table creation/deletion, **insert**, **delete**, and **update**
Create Table Construct

- An SQL relation is defined using the **create table** command:
  
  \[
  \text{create table } r \ (A_1 D_1, A_2 D_2, \ldots, A_n D_n, \\
  \text{(integrity-constraint}_1), \\
  \ldots, \\
  \text{(integrity-constraint}_k) )
  \]

  - \( r \) 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:
  
  ```sql
  \text{create table branch}
  \ (branch\_name \ char(15) \textbf{not null},
  \ branch\_city \ char(30),
  \ assets \ integer)
  ```
Integrity Constraints in Create Table

- not null
- primary key \((A_1, \ldots, A_n)\)

Example: Declare \textit{branch\_name} as the primary key for \textit{branch} and ensure that the values of \textit{assets} are non-negative.

```sql
create table branch
  (branch\_name char(15),
   branch\_city char(30),
   assets integer,
   primary key (branch\_name))
```

\textbf{primary key} declaration on an attribute automatically ensures \textbf{not null} in SQL-92 onwards, needs to be explicitly stated in SQL-89
CREATE TABLE COMMAND

• **Emp** (SS#, name, age, salary, dno)
  **Dept** (dno, dname, floor, mgrSS#)

• The allowed integrity constraints include:
  – Primary key (Aj1, Aj2, …, Ajm): states that attributes Aj1, Aj2, …, Ajm constitute
    the primary key for relation r: non-null and unique.
  – Check(P): states that every record in the relation must satisfy the specified
    predicate P, e.g., P = “age > 10”

• Create table **Emp** (SS# int, name char(15), age int, salary int, dno int, **primary key** (SS#), **check** (salary > 24000))
• Create table **Dept** (dno int, dname char(15), floor int, mgrss# int, **primary key** dno)
Basic Query Structure

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

```sql
select A_1, A_2, ..., A_n
from r_1, r_2, ..., 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.
SELECT COMMAND

- The select command has the following syntax:

```
select [distinct] target-list
from tuple variable list
where qualification
[order by target list subset]
[having set-qualification]
```

the items in square brackets ([ ]) are electives.

- Qualification list can be a boolean combination (and, or, not) of either selection and join clauses.

- A selection clause is a comparison between an indexed tuple variable and a constant. A comparison operator might be `{=, ≠, ≤, ≥, >, <}`.
SELECT COMMAND (Cont…)

- A **join clause** is a comparison between two indexed tuple variable. Once again, a comparison operator might be \{=, \neq, \leq, \geq, >, <\}. In the following example, e.dno=d.dno is a join clause while d.dname='Toy' is a selection clause:

```sql
select e.name
from Emp e, Dept d
where e.dno = d.dno and d.dname = 'Toy'
```

- The results of a select query is a relation.

- To eliminate duplicates from the result, use the key word distinct in the target list. For example, the following query shows all distinct salaries above 50,000 with no duplicates:

```sql
select distinct e.salary
from Emp e
where e.salary > 50000
```
SELECT COMMAND (Cont…)

Conceptualize the execution of a select query as one of:
1. evaluate qualification to locate qualifying tuples,
2. project out columns of these tuples using the target-list,
3. make the resulting tuples into a relation and eliminate duplicates if necessary.
SELECT COMMAND (Cont…)

• SQL provides set operations: **union, intersect, minus**. Example: Retrieve the social security of those employees who work in both the shoe and toy departments.

```sql
(select distinct e.SS#
from Emp e, Dept d
where e.dno = d.dno and d.dname = 'Toy')
intersect
(select distinct e.SS#
from Emp e, Dept d
where e.dno = d.dno and d.dname = 'Shoe')
```

• Note that the following query is **WRONG**!

```sql
select distinct e.SS#
from Emp e, Dept d
where e.dno = d.dno and d.dname = 'Toy' and d.dname=Shoe'
```
This is because the department name is single value (either Toy or Shoe). The result of this query is an empty relation.
SELECT COMMAND (Cont…)

- SQL has set membership operator: **in**. Example: select those employees in the shoe department who earn more than 50,000 and who work for the toy department.

```sql
select e.name
from Emp e, Dept d
where e.dno = d.dno and d.dname = 'Shoe' and e.salary > 50,000
and e.SS# in (  
    select e.SS#
    from Emp e, Dept d
    where e.dno = d.dno and d.dname = 'Toy')
```

- SQL has set comparison operators: **contains, some, all**. It is important to note that these are boolean operators that produce either True or False as their final output. These operators do NOT produce a set of tuples as output. Set comparison operators are used in the qualification list of a query.
SELECT COMMAND (Cont…)

• A contains B means set A is a superset of set B. Example: Social security number of employees who manage all the departments on the first floor.

```sql
select d1.mgrSS#
from Dept d1
where
   (select d2.dname
    from Dept d2
    where d1.mgrSS#=d2.mgrSS#)
contains
   (select dname
    from Dept
    where floor = 1)
```

Note that a manager may manage department on the second and third floor.
### set A: shoe

<table>
<thead>
<tr>
<th>Dno</th>
<th>Dname</th>
<th>Floor</th>
<th>mgrSS#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>Toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>Men</td>
<td>2</td>
<td>323</td>
</tr>
<tr>
<td>4</td>
<td>Women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>
set A: shoe, toy

<table>
<thead>
<tr>
<th>Dno</th>
<th>Dname</th>
<th>Floor</th>
<th>mgrSS#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>Toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>Men</td>
<td>2</td>
<td>323</td>
</tr>
<tr>
<td>4</td>
<td>Women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>
set A: shoe, toy, men

<table>
<thead>
<tr>
<th>Dno</th>
<th>Dname</th>
<th>Floor</th>
<th>mgrSS#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>Toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>Men</td>
<td>2</td>
<td>323</td>
</tr>
<tr>
<td>4</td>
<td>Women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>
set A: shoe, toy, men

set B:

<table>
<thead>
<tr>
<th>Dno</th>
<th>Dname</th>
<th>Floor</th>
<th>mgrSS#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>Toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>Men</td>
<td>2</td>
<td>323</td>
</tr>
<tr>
<td>4</td>
<td>Women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>
set A: shoe, toy, men  
set B: shoe, toy

<table>
<thead>
<tr>
<th>Dno</th>
<th>Dname</th>
<th>Floor</th>
<th>mgrSS#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>Toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>Men</td>
<td>2</td>
<td>323</td>
</tr>
<tr>
<td>4</td>
<td>Women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>
SELECT COMMAND (Cont…)

- Find the name of employees whose salary is higher than the salary of everyone on the first floor.

```
select name
from Emp
where salary > all
    (select e.salary
     from Emp e, Dept d
     where e.dno = d.dno and d.floor=1)
```
SELECT COMMAND (Cont…)  

• Find the name of employees whose salary is higher than the salary of some of the employees on the second floor.

```
select name
from Emp
where salary > some
  (select e.salary
   from Emp e, Dept d
   where e.dno = d.dno and d.floor=2)
```
• Assume a yacht club that maintains a reservation system to enable its sailors to reserve boats. For each sailor, this reservation system maintains their sailor id, name, and rank. For each boat, the system maintains a boat id, its color, and name. Sailors may reserve different boats for different dates.

Sailors (sid, sname, rank)
Boats(bid, bname, color, sid)
Assume a yacht club that maintains a reservation system to enable its sailors to reserve boats. For each sailor, this reservation system maintains their sailor id, name, and rank. For each boat, the system maintains a boat id, its color, and name. Sailors may reserve different boats for different dates.

Sailors \((\text{sid}, \text{sname}, \text{rank})\)

Boats \((\text{bid}, \text{bname}, \text{bcolor})\)

Reserve \((\text{sid}, \text{bid}, \text{date})\)
• Query: Retrieve the sailors with a rank higher than 5.
REVIEW

• Query: Retrieve the sid of those sailors who have reserved a red boat.

• Query: Retrieve the name of those sailors who have reserved a red boat.
REVIEW

- Query: Retrieve the sid of those sailors with at least one reservation.

Sailors (sid, sname, rank)
Boats(bid, bname, color)
Reserve (sid, bid, date)
• Query: Retrieve the name of those sailors who have reserved both a red and a green boat.
• Query: Retrieve those sailors who have reserved both a red and a green boat.

Sailors (sid, sname, rank)
Boats(bid, bname, color)
Reserve (sid, bid, date)
REVIEW

• Query: Retrieve those sailors who have reserved either a red or a green boat.

Sailors (sid, sname, rank)
Boats(bid, bname, color)
Reserve (sid, bid, date)
NULL Values

• SQL allows NULL as a special value for attributes; also called null value. What does a null value mean?
NULL VALUES

- SQL allows NULL as a special value for attributes; also called null value. What does a null value mean?
  - Value is unknown, e.g., an unknown birthday.
  - Value is inapplicable: there is no value that makes sense here. A spouse attribute value for an employee who is not married.
  - Value withheld: For security/privacy/other reasons, the system is not providing the value, e.g., an unlisted phone number for a famous person.
- A comparison involving NULL is UNKNOWN:
  - e.g., Dept.dname = “Toy” computes to UNKNOWN when the dname attribute value of a record is NULL.
- The correct way to find those rows whose dname attribute value is NULL:
  ```sql
  SELECT *
  FROM Dept
  WHERE dname IS NULL
  ```
**NULL VALUES (Cont…)**

- Given that the “Where” computes to either TRUE or FALSE, what are the rules with comparisons involving UNKNOWN? E.g., consider the following predicate $\text{d.dname} = \text{“toy” and d.floor=5}$ for a row with NULL as its d.dname and 5 as its floor value; does this predicate return TRUE or FALSE?
NULL VALUES (Cont…)

• Given that the “Where” computes to either TRUE or FALSE, what are the rules with comparisons involving UNKNOWN? E.g., consider the following predicate \texttt{d.dname = “toy” and d.floor=5} for a row with NULL as its \texttt{d.dname} and 5 as its floor value; does this predicate return TRUE or FALSE?

• Think of TRUE as 1, UNKNOWN at 0.5, and FALSE as 0:
  – AND of two truth values is the minimum of those values. X AND Y is FALSE if either X or Y is FALSE; it is UNKNOWN if neither is FALSE but at least one is UNKNOWN, and it is TRUE only when both X and Y are TRUE.
  – OR of two truth values is the maximum of those values. X OR Y is TRUE if either X or Y is TRUE; it is UNKNOWN if neither is TRUE but at least one is UNKNOWN, and it is FALSE when both are FALSE.
  – Negation of truth value \( v \) is 1-\( v \). NOT X is TRUE when X is FALSE, the value FALSE when x is TRUE, and the value UNKNOWN when X has value UNKNOWN.
JOIN PREDICATES: Inner Join

- “Emp inner join Dept on Emp.dno = Dept.dno” maintains the joining column twice.

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
<th>dno</th>
<th>dname</th>
<th>Floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
<td>3</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
</tbody>
</table>
JOIN PREDICATES: Natural Join

- “Emp natural join Dept on Emp.dno = Dept.dno” maintains the joining column once.

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
<th>dname</th>
<th>Floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
</tbody>
</table>
JOIN PREDICATES: Left Outer Join

- “Dept left outer join Emp on Dept.dno = Emp.dno” maintains the rows of the left hand side (Dept) with no joining records of the right hand side (Emp) as NULLs.

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
<th>ss#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
</tbody>
</table>
JOIN PREDICATES: Right Outer Join

• “Emp right outer join Dept on Emp.dno = Dept.dno” maintains the rows of the right hand side (Dept) with no joining records of the left hand side (Emp) as NULLs.

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ss#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>
JOIN PREDICATES: Full Outer Join

• “Emp full outer join Dept on Emp.dno = Dept.dno” maintains the rows of the right hand side (Emp) with no joining records of the left hand side (Dept) as NULLs AND rows of the left hand side (Dept) with no joining records of the right hand side (Emp) as NULLS.

<table>
<thead>
<tr>
<th>SS#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
<tr>
<td>565</td>
<td>Kate</td>
<td>25</td>
<td>30000</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ss#</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
<td>3</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
<tr>
<td>565</td>
<td>Kate</td>
<td>25</td>
<td>30000</td>
<td>4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
SELECT COMMAND

• Why have tuple variables?
SELECT COMMAND (Cont…)

• SQL supports five aggregate functions that can be applied to any attribute:
  1. `count([distinct] X)`: number of [distinct] values for attribute X
  2. `sum([distinct] X)`: sum of [distinct] values for attribute X
  3. `avg([distinct] X)`: average of [distinct] values for attribute X
  4. `max(X)`: maximum value for attribute X
  5. `min(X)`: minimum value for attribute X
     – The result of each of these functions is a single value.

• Find the number of employees whose salary is higher than 100,000:
SELECT COMMAND (Cont…)  

• Aggregate functions can be used only in the target list of a query.

• Qualified aggregates: the set on which the aggregate is applied can be restricted by the where clause. Example: Find the average salary of the employees in the toy department.

  ```sql
  select avg(salary)
  from Emp e, Dept d
  where e.dno = d.dno and d.dname = 'Toy'
  ```

• Find the employees who earn more than the average salary:
SELECT COMMAND (Cont…)

• Aggregate functions can be used only in the target list of a query.

• Qualified aggregates: the set on which the aggregate is applied can be restricted by the where clause. Example: Find the average salary of the employees in the toy department.

   select  avg(salary)
   from    Emp e, Dept d
   where   e.dno = d.dno and d.dname = ‘Toy’

• Find the employees who earn more than the average salary:
SELECT COMMAND (Cont…)

• At times, one might want the system to apply a function and do the grouping at the same time. Example: compute the average salary for each department.
  
  ```sql
  select d.dno, d.dname, avg(salary) as AvgSalary
  from Emp e, Dept d
  where e.dno = d.dno
  group by d.dno, dname
  ```

• **having** can be used to restrict the relevant groups. Example: Find the average salary of those departments with more than 10 employees.
INSERT COMMAND

- Insert is of two types:
  1. `insert into` rel-name `values` value list
  2. `insert into` rel-name `select`

- To illustrate, assume the existence of two relations:
  register(sid, sname, paid, course#) and CSCI485(sid,sname).

- If Joe and Bob register for csci485 without having paid:
  `insert into` register `values`
    (666-66-6666, `Joe', No, 485)
    (777-77-7777, `Bob', No, 485)
INSERT COMMAND (Cont…)

• To insert all CSCI485 student into CSCI485 relation who have paid:

```sql
insert into CSCI485
    select sid, name
    from register r
    where r.paid = 'yes' and r.course#=485
```

• Note that the target list of the select command must confirm to the schema of CSCI485
DELETE COMMAND

• Delete has the following syntax:
  `delete from` rel-name `where` qualification

• Example: Fire all those sailors whose rating is less than 2.
  `delete from` sailors
  `where` rating < 2

• The semantic of `delete` is as follows:
  1. execute query:
     `select *`
     `from` rel-name
     `where` qualification
  2. remove tuples found in Step 1 from rel-name.

• Why have this semantic for delete?
DELETE COMMAND

• Delete has the following syntax:
  delete rel-name where qualification
• Example: Fire all those sailors whose rating is less than 2.
  delete sailors
  where rating < 2
• The semantic of delete is as follows:
  1. execute query:
     select *
     from rel-name
     where qualification
  2. remove tuples found in Step 1 from rel-name.

• Why have this semantic for delete?
  Delete sailors
  Where rating < all (select avg(rating) from sailors)
UPDATE COMMAND

• Update command has the following syntax:
  update rel-name
  set target-list
  where qualification

• Example: Give a 10% raise to all employees in the toy department.
  update Emp
  set salary = 1.1 × salary
  where SS# in (  
    select e.SS#
    from Emp e, Dept d
    where e.dno = d.dno and d.dname = ‘Toy’)

UPDATE COMMAND (Cont…)

The semantics of update are as follows:

1. Execute the following two queries:
   a) `insert into` del-temp
      `select` full-target-list
      `from` rel-name
      `where` qualification
   b) `insert into` app-temp
      `select` extended target list
      `from` rel-name
      `where` qualification

   Extended target list in our example would be: (SS#,name, age, sal × 1.1, dno). Full target list in our example would be: (SS#,name, age, sal, dno).

2. Remove tuples in del-temp from rel-name

3. Insert tuples in app-temp into rel-name
<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
<td>3</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
<td>1</td>
</tr>
<tr>
<td>324</td>
<td>John</td>
<td>40</td>
<td>55000</td>
<td>2</td>
</tr>
<tr>
<td>333</td>
<td>Mike</td>
<td>25</td>
<td>60000</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoe</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>men</td>
<td>2</td>
<td>323</td>
</tr>
<tr>
<td>4</td>
<td>women</td>
<td>2</td>
<td>324</td>
</tr>
<tr>
<td>ssnnum</td>
<td>name</td>
<td>age</td>
<td>salary</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>123</td>
<td>Mary</td>
<td>22</td>
<td>55000</td>
</tr>
<tr>
<td>323</td>
<td>Joe</td>
<td>33</td>
<td>34000</td>
</tr>
<tr>
<td>324</td>
<td>John</td>
<td>40</td>
<td>55000</td>
</tr>
<tr>
<td>333</td>
<td>Mike</td>
<td>25</td>
<td>60000</td>
</tr>
</tbody>
</table>

dname = “Toy”

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
</tbody>
</table>
dname = “Toy”

<table>
<thead>
<tr>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>toy</td>
<td>1</td>
<td>323</td>
</tr>
</tbody>
</table>

Emp.dno = Dept.dno and d.dname = “Toy”

<table>
<thead>
<tr>
<th>sssnum</th>
<th>name</th>
<th>age</th>
<th>salary</th>
<th>dno</th>
<th>dname</th>
<th>floor</th>
<th>mgrssnum</th>
</tr>
</thead>
<tbody>
<tr>
<td>324</td>
<td>John</td>
<td>40</td>
<td>55000</td>
<td>2</td>
<td>Toy</td>
<td>1</td>
<td>323</td>
</tr>
</tbody>
</table>