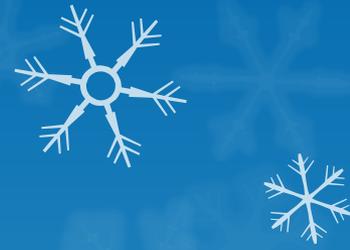


**Course Name:**  
**Database Management**  
**Systems**



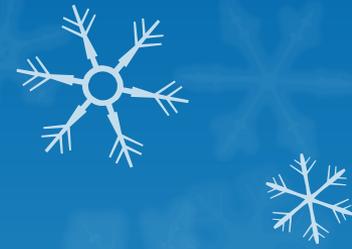
# Lecture 10

## Topics to be covered

- DDL
- Tuple Calculus



# Data Definition Language



Allows the specification of:

- The schema for each relation, including attribute types.
- Integrity constraints
- Authorization information for each relation.
- Non-standard SQL extensions also allow specification of
  - The set of indices to be maintained for each relations.
  - The physical storage structure of each relation on disk.



# Create Table Construct

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

```
create table r (A1 D1, A2 D2, ..., An Dn,  
                (integrity-constraint1),  
                ...,  
                (integrity-constraintk))
```

- *r* is the name of the relation
- each *A*<sub>*i*</sub> is an attribute name in the schema of relation *r*
- *D*<sub>*i*</sub> is the data type of attribute *A*<sub>*i*</sub>

Example:

```
create table branch  
  (branch_name char(15),  
   branch_city  char(30),  
   assets       integer)
```

# 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.
- More are covered in Chapter 4.

# Integrity Constraints on Tables

- **not null**
- **primary key** ( $A_1, \dots, A_n$ )

Example: Declare *branch\_name* as the primary key for *branch*

```
create table branch
(branch_name char(15),
 branch_city char(30) not null,
 assets integer,
 primary key (branch_name))
```

**primary key** declaration on an attribute automatically ensures **not null** in SQL-92 onwards, needs to be explicitly stated in SQL-89

# Basic Insertion and Deletion of Tuples

- Newly created table is empty
- Add a new tuple to *account*

```
insert into account  
values ('A-9732', 'Perryridge', 1200)
```

- Insertion fails if any integrity constraint is violated
- Delete *all* tuples from *account*

```
delete from account
```

Note: Will see later how to delete selected tuples

# Drop and Alter Table Constructs

- The **drop table** command deletes all information about the dropped relation from the database.
- The **alter table** command is used to add 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  $A$ .

- All tuples in the relation are assigned *null* as the value for the new attribute.
- The **alter table** command can also be used to drop attributes of a relation:

**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:

```
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.
- This query is equivalent to the relational algebra expression.

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

- 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 branches in the *loan* relation:  
**select** *branch\_name*  
**from** *loan*

- In the relational algebra, the query would be:

$$\square_{branch\_name} (loan)$$

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g. *Branch\_Name*  $\equiv$  *BRANCH\_NAME*  $\equiv$  *branch\_name*
  - Some people use upper case wherever we use bold font

# 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 branches in the *loan* relations, and remove duplicates

```
select distinct branch_name  
from loan
```

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

```
select all branch_name  
from loan
```

# The select Clause (Cont.)

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

```
select *  
from loan
```

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

- E.g.:

```
select loan_number, branch_name, amount * 100  
from loan
```

# 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 loan number for loans made at the Perryridge branch with loan amounts greater than \$1200.

```
select loan_number
from loan
where branch_name = 'Perryridge' and amount >
1200
```

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

# 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 *borrower X loan*

```
select *  
from borrower, loan
```

- Find the name, loan number and loan amount of all customers having a loan at the Perryridge branch.

```
select customer_name, borrower.loan_number, amount  
from borrower, loan  
where borrower.loan_number = loan.loan_number and  
branch_name = 'Perryridge'
```

# The Rename Operation

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

*old-name as new-name*

- E.g. Find the name, loan number and loan amount of all customers; rename the column name *loan\_number* as *loan\_id*.

```
select customer_name, borrower.loan_number as  
loan_id, amount  
from borrower, loan  
where borrower.loan_number = loan.loan_number
```

# Tuple Variables

- Tuple variables are defined in the **from** clause via the use of the **as** clause.
- Find the customer names and their loan numbers and amount for all customers having a loan at some branch.

```
select customer_name, T.loan_number, S.amount  
from borrower as T, loan as S  
where T.loan_number = S.loan_number
```

- Find the names of all branches that have greater assets than some branch located in Brooklyn.

```
select distinct T.branch_name  
from branch as T, branch as S  
where T.assets > S.assets and S.branch_city = 'Brooklyn'
```

- Keyword **as** is optional and may be omitted  
*borrower as T*  $\equiv$  *borrower T*

- Some database such as Oracle *require* **as** to be omitted

# Example Instances

- We will use these instances of the Sailors and Reserves relations in our examples.
- If the key for the Reserves relation contained only the attributes *sid* and *bid*, how would the semantics differ?

*R1*

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

*S1*

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

*S2*

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0