Database Modification Statements, Aggregate Functions, and Views

Database Modification

- SQL supports the following statements for modifying the database
  - Insert
  - Update
  - Delete

Database Modification

- SQL Insert statement
- Insert new rows into a table of the database

```
Insert Into table_name [(attribute_list)]
                    Values (value list);
```

- Examples:

```
insert into Account
values (9732, 'Smith', 'Main', 1200);
```

Since the attribute list is omitted, values have to be in the order of the columns defined. If they are not, insert will fail.

Database Modification

- Examples:

  insert into Account (cname, bal, bname, A#)

  values (‘Smith’, 1200, ‘Main’, 9732);

  With this approach you can also eliminate having the columns for which the value is unknown.

  insert into Customer (cname) values (‘Rahimi’);

  You can also achieve the same thing as:

  insert into Customer values (‘Rahimi’, Null, Null);

Database Modification

- Insert a new account for Rahimi with account number of 1111, at York branch, and the balance of 5000.

```
SQL> Insert into account values (1111,'Rahimi','York',5000.00);
```

- Verify the change by doing a select

```
SQL> Commit;
```

- Commit makes this insert permanent.

Database Modification

- Variations to the insert statement:
  - Some databases allow you to also insert many rows into a table at once

```
insert into Account (bname, A#, cname, bal)
    select bname, L#, cname, amt
    from Loan;
```

Database Modification

- Try it:
  - Create an account with the balance of 500 for all customers who have a loan with an amount $20000 or more in the bank.

Database Modification

- Try it:
  - Create an account with the balance of 500 for all customers who have a loan with an amount $20000 or more in the bank.

```
insert into account (cname, bname,A#, bal)
select cname , bname, L#, 500
from loan
where amt >= 20000;
```

- Verify the change by doing a select
- Throws away the change we made to the database by issuing an abort.

```
SQL> rollback;
```

Database Modification

- The CASE statement:
  - Repeat the same question as before but this time the following must be checked:
    - If the customer has a loan with an amount more than $20000, the account balance will be $500.
    - If the customer has a loan with an amount more than $30000, the account balance will be $750.

```
insert into account (cname, bname,A#, bal)
select cname , bname, L#,
case
       when amt >= 20000 and amt <30000 then 500
       when amt >= 30000 then 750
end as bal
from loan
where amt >= 20000;


SQL> rollback;
```

Database Modification

- Try the following as well:

```
select 3+2 from dual;

select user from dual;

select sysdate from dual;

Check SQL Functions for more examples on DUAL.

Set pagesize 64;
Set linesize 80;

select 'select * from ', tname, ' ;' from tab;

-- You can use this method to generate scripts automatically.
-- How do you generate a script that can drop all of your tables?
--
select 'drop table ', tname, ' cascade constraints;' from tab;
```

Database Modification

- Updating Rows
- Update command is used to modify one or more columns of one or more rows of a table
- Syntax:

```
Update table_name
Set <modified_column_list>
[where P];
```

Database Modification

- Examples:

```
update Loan
set amt = 0
where upper(cname) = ‘RAHIMI’;

update Account
set bal = bal * 1.01
where bal > 12000;
```

Database Modification

- Deleting Rows
- Delete operation is used to delete one or more rows from a table
- have to be very careful with the use of this statement
- The predicate in the delete statement indicates how many of the table’s row will be deleted (if any)

Database Modification

- Delete statement:

```
Delete [FROM] table_name
[where P];
```

- Examples:

```
delete Account
where bname IN (
            select bname
            from Branch
            where lower(bcity) = ‘east bloomington’);
```

Aggregate Functions

- SQL supports the following aggregate functions:
  - Avg
  - Min
  - Max
  - Sum
  - Count
  - Order by (Sort by)
- These functions work on a group (of rows)

Aggregate Functions

- Examples:

```
select bname, avg(bal)
from Account
group by bname;

select bname, avg(bal)
from Account
group by bname
having avg(bal) > 1200
order by bname ASC;
```

Aggregate Functions

- Examples with WHERE and HAVING

```
select bname, avg(bal)
from Account
Where lower(cname) LIKE '%o%'
group by bname
having avg(bal) > 120
order by bname ASC;

BNAME                AVG(BAL)
--------------- ----------
France                 3417
Main                 1835.5
Ridgedale               150
Southdale              1000
York               2894.875
```

Aggregate Functions

- Examples with WHERE and HAVING

```
column avg(bal) heading Average;
column avg(bal) format 99999.99;

select bname Branch_Name, avg(bal)
from Account
Where lower(cname) LIKE '%o%'
group by bname
having avg(bal) > 120
order by bname ASC;

BRANCH_NAME       Average
--------------- ---------
France            3417.00
Main              1835.50
Ridgedale          150.00
Southdale         1000.00
York              2894.88
```

Aggregate Functions

- The Count Function:
  - Returns the number of rows qualified a the select statement.

Table T

A	B	C	D
a1	b1	c1	
a2		c2	d1
a3	b2	c3	
a4	b3	c4	d2
a5	b2	c5	
a6		c6	

Select count(*) from T returns 6

Select count(a) from T returns 6

Select count(b) from T returns 4

Select count(distinct B) from T returns 3

Aggregate Functions

- Examples

```
select cname
from Account
where lower(bname) = ‘main’ AND
    (
     select count(*)
    from Loan
    where Account.cname = Loan.cname)  = 0
     ;
```

What does this statement do?

Aggregate Function

- What does the following code do?

```
select distinct a.cname
from  account a
where  a.bname in (        select bname
                           from branch
                           where lower(bcity) = 'edina'
                           )
group by a.cname
having count (a.bname) = (           select count (bname)
                                     from branch
                                      where lower(bcity) = 'edina'
                                     );
```

Aggregate Function

- The statement prints the name of customers who have an account in every branch that is in Edina.
  - When running the command, Woods and Rahimi are returned as answers. But Rahimi is not the right answer since Rahimi does not have an account in France which is a branch in Edina.
  - To solve the problem we need to change the statement as shown on the next page:

Aggregate Function

```
select distinct a.cname
from  account a
where  a.bname in (select bname
                   from branch
                   where lower(bcity) = 'edina'
                 )
group by a.cname
having count (distinct a.bname) = (select count (bname)
                                  from branch
                                  where lower(bcity) = 'edina'
                                  );
```

Creating Views

- View is a relation derived from one or more **base** relations.
- Defined by a select statement.
- Selection, projection, join and union commonly used.
- Most DBMS's don't materialize views

Creating Views

- View are used for security and/or ease of use purposes
- A view can be defined on one or more tables
- CREATE VIEW *view_name* [**(***column-list***)**] AS

  select_statement;

Creating Views

- Examples:

```
Create view Names AS
    Select Fname, Lname, Minit
    From Employee;

Create view Cust_Account AS
    Select Loan.cname, Loan.L#, Account.A#
    From Loan, Account
    Where Loan.cname = Account.cname;
```

Creating Views

- For the view created on the previous page:

```
Select * from cust_account
Where cname LIKE ‘%o%’;

Gets mapped to:

   Select Loan.cname, Loan.L#, Account.A#
   From Loan, Account
   Where Loan.cname = Account.cname AND
             Loan.cname LIKE ‘%o%’;
```

Using Views

- Once created, views are used just like any other base tables.
- Access to the views need to be controlled just like any other base tables in the system
- Queries against views are mapped to queries against the actual base table(s) by the DBMS
- If a view is not defined as read-only then users can update the base tables through the view

Updating Views

- Updating views is troublesome since not all the columns from the base table(s) are seen in the view
- If these columns are defined as “Not Null” the view can not be updated
- Most DBMSs do not allow updating a view

Updating Views

- Example: Suppose we want to have a view of the customers' names and the branch city where they have a loan

```
SQL>    select cname, bcity, branch.bname
  2     from Branch, Loan
  3     where Branch.bname = Loan.bname
  4     order by cname, bcity, branch.bname;
Cook            Edina           France
Cook            Edina           York
Cook            Minnetonka      Main
James           Edina           France
Jones           Edina           France
Jones           Edina           Southdale
Jones           Edina           York
Melcher         Edina           France
Rahimi          Edina           France
Rahimi          Edina           Southdale
Rahimi          Edina           York
Tomczak         Edina           France
Woods           Edina           France

13 rows selected.

SQL> Create view Cname_Bcity AS
  2     select cname, bcity, branch.bname
  3     from Branch, Loan
  4     where Branch.bname = Loan.bname
  5     order by cname, bcity, branch.bname;

View created.
```

Updating Views

- Now the view is created, we can select from it just like any other table

```
SQL> select * from cname_bcity;
Cook                Edina               France
Cook                Edina               York
Cook                Minnetonka          Main
James               Edina               France
Jones               Edina               France
Jones               Edina               Southdale
Jones               Edina               York
Melcher             Edina               France
Rahimi              Edina               France
Rahimi              Edina               Southdale
Rahimi              Edina               York
Tomczak             Edina               France
Woods               Edina               France

13 rows selected.
```

- But, we cannot insert into it.
- Why?

Business Rules

- Business rules in the database reflect the constraints that organizations have on data items of a database.
- Examples of business rules:
  - Every employee must work for a department
  - Every department must have a department manager
  - Age of employees must be a positive number
  - In Oracle, a rule is assigned to a column by a “Check Constraint”
  - SQL Server can use check constraint and also has rule as a separate construct

Oracle Rules

- Create a table called Min_cities.
- This table has two columns
  - Column city_name is varchar(20) and can only accept city names (MPLS, STP, STC, MNKTO)
  - Column population Number (8)
- Show how this rule works by trying to insert an invalid city into the database
- Remove the rule from the column it is attached to

Oracle Rules

- Answer:

```
SQL> create table Mn_cities (
  2     city_name varchar(20),
  3     population number(8),
  4  constraint nameCT check (city_name IN ('MPLS', 'STP', 'STC', 'MNKTO')));

Table created.

SQL> insert into Mn_cities values ('MPLS', 2000000);

1 row created.

SQL> insert into Mn_cities values ('Minneapolis', 2000000);
insert into Mn_cities values ('Minneapolis', 2000000)
*
ERROR at line 1:
ORA-02290: check constraint (SKR.NAMECT) violated


SQL> alter table Mn_cities drop constraint nameCT;

Table altered.
```

Database Control

- Access to the database is controlled by the use of Grant/Revoke statements
- Syntax:

```
Grant DBA|select|update|insert|delete
On {table_name[.column][,]} 1..*
To user_name|group_name
[with Grant Option];
```

- Examples:

```
Grant select on employee to public;
Revoke update on employee from Saeed;
Grant DBA to Saeed with grant option;
```

Database Control

- Example:
  - Supposed Sam creates table Emp
    - Therefore Sam owns table Emp.
  - If Sam wants Joe to be able to select, insert, delete, update the table, Sam issues the following:

    Grant select, insert, update, delete on Emp to Joe;
  - Joe then can use the table:

    Select * from sam.Emp;

    Insert into sam.Emp values (…..);