# SQL

# SQL

- **A relational database consists of a collection of relations each of which is assigned a unique name**

- **SQL – Structured Query Language**

# SQL

- **The basic structure of an SQL expression consists of three clauses: select, from, and where**

- The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query
- The **from** clause corresponds to the Cartesian product operation of the relational algebra . It lists the relations to be scanned in the evaluation of the expression.
- The **where** clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause

# SELECT CLAUSE

A typical SQL query has the form

**Select $A_1, A_2, \ldots\ldots A_n$**

**from $r_1, r_2, \ldots\ldots r_m$**

**where P**

- Each $A_i$ represents an attribute
- Each $r_i$ is a relation
- P is a predicate
- The query is equivalent to the relational algebra expression

$$\Pi_{A1,A2,\ldots..An}(\sigma_{P}(r_1 \times r_2 \times \ldots\ldots \times r_m))$$

# SELECT CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all branches in the loan relation

**select *branch-name***

**from *loan***

# SELECT CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

To force the elimination of duplicates: the keyword **-distinct**

**select distinct *branch-name***

**from *loan***

# SELECT CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

The keyword **all** to specify explicitly that duplicates are not removed

**select all *branch-name***
**from *loan***

# SELECT CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

The asterisk symbol ‘*’ can be used to denote ‘all attributes’

**select *
from *loan***

# WHERE CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find all loan numbers 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***

# WHERE CLAUSE

**customer(customer-name, customer-street, customer-city)**
**branch(branch-name, branch-city, assets)**
**account(account-number, branch-name, balance)**
**loan(loan-number, branch-name, amount)**
**borrower(customer-name, loan-number)**
**depositor(customer-name, account-number)**

**select** *loan-number*
**from** *loan*
***where*** *amount* ***between*** *90000* ***and*** *100000*

Instead of

**select** *loan-number*
**from** *loan*
***where*** *amount >= 90000* ***and*** *amount <= 100000*

# FROM CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names, loan number and loan amount for all customers who have a loan from the bank

**select** ***customer-name, borrower.loan-number, amount***

**from** ***borrower, loan***

***where borrower.loan-number=loan.loan-number***

# FROM CLAUSE

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names, loan number and loan amount for all customers who have 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'***

# RENAME OPERATION

- Old name **as** new name
- The **as** clause can appear in both the select and from clauses
- < keyword *as* optional>

**select** ***customer-name, borrower.loan-number as load_id, amount***
**from** ***borrower, loan***
***where borrower.loan-number=loan.loan-number and***
***branch-name='Perryridge'***

# Tuple Variables

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names, loan number and loan amount for all customers who have a loan from the bank

**select T.*customer-name, T.loan-number, S.amount***

**from *borrower as T, loan as S***

***where T.loan-number=S.loan-number***

# Tuple Variables

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all branches that have assets greater than at least one branch located in Brooklyn

# Tuple Variables

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all branches that have assets greater than at least one 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'***

# String Operations

- **Pattern matching using the operator *like***
- **Percent ( %): The % character matches any substring**
- **Underscore ( _ ): The _ character matches any character**

**<u>Examples</u>**

- **‘Perry%’ matches any string beginning with ‘Perry’**
- **‘%idge%’ matches any string containing ‘idge’ as a substring**

  **Perryridge, Rock Ridge, Mianus Bridge, Ridgeway**
- **‘_ _ _’ matches any string of exactly three characters**
- **‘_ _ _ %’ matches any string of at least three characters**

# String Operations

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all customers whose street address includes the substring ‘Main’

**select** ***customer-name***

**from** ***customer***

***where customer-street like ‘%Main%’***

# String Operations

- Backslash ( \ ): the escape character
- like ‘ab\%cd%’ **escape** ‘\’ matches all strings beginning with ‘ab %cd’
- like ‘ab\\cd%’ **escape** ‘\’ matches all strings beginning with ‘ab\cd’
- Comparison operator : **not like**

# Order by Clause

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

The order by clause causes the tuples in the result of a query to appear in sorted order

**select distinct *customer-name***

**from *borrower, loan***

**where *borrower.loan-number=loan.loan-number and branch-name='Perryridge'***

**order by *customer-name***

# Order by Clause

- **The order by clause lists items in ascending order**
- **To specify the sort order: desc for descending order or asc for ascending order**

```
select *
from loan
order by amount desc, loan-number asc
```

# Set Operations

- **Union operation**
- **Intersect operation**
- **Except operation**

# Union Operation

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find all the bank customers having a loan, an account or both at the bank

**(select *customer-name***

**from depositor*)***

***union***

**(select *customer-name***

**from borrower*)***

# Union Operation

- Union  operation automatically eliminates duplicates
- If we want to retain all duplicates
- Query: Find all the bank customers having a loan, an account or both at the bank

**(select *customer-name***

**from depositor*)***

***union all***

**(select *customer-name***

**from borrower*)***

# Intersect Operation

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find all customers who have both a loan and an account at the bank

  **(select *customer-name***

  **from depositor*)***

  ***intersect***

  **(select *customer-name***

  **from borrower*)***

# Intersect Operation

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find all customers who have both a loan and an account at the bank

  **(select *customer-name***

  **from depositor*)***

  ***intersect all***

  **(select *customer-name***

  **from borrower*)***

# Except Operation

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find all customers who have an account but no loan at the bank

  **(select *customer-name***

  **from depositor*)***

  ***except***

  **(select *customer-name***

  **from borrower*)***

# Except Operation

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find all customers who have an account but no loan at the bank

  **(select *customer-name***

  **from depositor*)***

  ***except all***

  **(select *customer-name***

  **from borrower*)***

# Aggregate Functions

- **Aggregate functions are functions that take a collection of values as input and return a single value**

- **Average : avg**
- **Minimum: min**
- **Maximum: max**
- **Total: sum**
- **Count : count**

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the average account balance at the Perryridge branch

```
select avg(balance)
from account
where branch-name= ‘Perryridge’
```

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the average account balance at each branch

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the average account balance at each branch

```
select branch-name, avg(balance)
from account
group by branch-name
```

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the number of depositors for each branch

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the number of depositors for each branch

```
select branch-name, count (distinct customer-name)
from depositor, account
where depositor.account-number=account.account-number
group by branch-name
```

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find those branches where the average account balance is more than $1200.

```
select branch-name, avg(balance)
from account
group by branch-name
having avg(balance)>1200
```

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the number of tuples in the customer relation

  **select count (*)**

  **from customer**

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the average balance for each customer who lives in Harrison and has at least three accounts

# Aggregate Functions

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the average balance for each customer who lives in Harrison and has at least three accounts

  **select depositor.customer-name, avg(balance)**

  **from depositor, account, customer**

  **where depositor.account-number=account.account-number and depositor.customer-name=customer.customer-name and**

  **customer-city='Harrison'**

  **group by depositor.customer-name**

  **having count(distinct depositor.account-number)>=3**

# Set Membership

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find all the customers who have both a loan and an account at the bank

```
select distinct customer-name
from borrower
where customer-name in (select customer-name from
                        depositor)
```

# Set Membership

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- **not in operator**

# Set Membership

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

- Query: Find the names of customers who have a loan at the bank and whose names are neither Smith nor Jones

```
select distinct customer-name
from borrower
where customer-name not in ('Smith', 'Jones')
```

# Set comparison

- **The phrase ‘greater than at least one’ corresponds to *>some***
- **The phrase ‘greater than all’ corresponds to *>all***

# Set comparison

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all branches that have assets greater than at least one 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'***

# Set comparison

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all branches that have assets greater than at least one branch located in Brooklyn

**select** ***branch-name***

**from** ***branch***

***where assets > some (select assets***

***from branch***

***where branch-city='Brooklyn')***

# Set comparison

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the names of all branches that have assets greater than all branches located in Brooklyn

**select** ***branch-name***

**from** ***branch***

***where assets > all (select assets***

***from branch***

***where branch-city='Brooklyn')***

# Set comparison

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the branch that has the highest average balance

- **Cannot use max(avg(…))**

# Set comparison

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find the branch that has the highest average balance

```
select branch-name
from account
group by branch-name
having avg(balance) >=all (select avg (balance)
                           from account
                           group by branch-name)
```

# Test for Empty Relations

- ***exists*** construct returns the value true if the argument subquery is nonempty
- Non existence of tuples by ***not exists*** construct
- ***unique*** construct returns the value true if the argument subquery contains no duplicate tuples
- Existence of duplicate tuples by ***not unique*** construct

# Test for Empty Relations

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Find all customers who have both an account and a loan at the bank

```
select customer-name
from  borrower
where exists (select *
                from depositor
                where depositor.customer-name=borrower.customer-name)
```

# Schema Definition in SQL

**create table** customer

(customer-name char(20),

customer-street char(30),

customer-city char(30),

**primary key** (customer-name))

**create table** branch

(branch-name char(15),

branch-city char(30),

assets numeric(16,2),

**primary key** (branch-name))

- To remove a relation from an SQL database we use the drop table command

**Drop table r**

- Deletes not only all tuples of r, but also the schema for r

## Delete from r

- Retains relation r, but deletes all tuples in r

- Alter table command to add attributes to an existing relation
- All tuples in the relation are assigned ***null*** as the value for the new attribute

**alter table r add A D**

- r is the name of an existing relation, A is the name of the attribute to be added and D is the domain of the added attribute

- We can drop attributes from a relation

**alter table r drop A**

- r is the name of an existing relation, A is the name of the attribute of the relation

# Modification of Database

- Deletion
- Insertion
- Updates

# Deletion

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Delete all account tuples in the perryridge branch

**delete from** account

**where** branch-name=‘perryridge’

# Deletion

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Delete all loan with loan amounts between $1300 and $1500

```
delete from loan
where amount between 1300 and 1500
```

# Deletion

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Delete all account tuples at every branch loacted in Brooklyn

```
delete from account
where branch-name in (select branch-name
                      from branch
                      where branch-city = 'Brooklyn')
```

# Insertion

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Insert the fact that there is an account A-9732 at the perryridge branch and it has a balance of $1200

**insert into** account

**values (‘A-9732’, ‘perryridge’,1200)**

- Values are specified in the order in which the corresponding attributes are listed in the relation schema

# Insertion

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Insert the fact that there is an account A-9732 at the perryridge branch and it has a balance of $1200

**insert into** account

**values (‘A-9732’, ‘perryridge’,1200)**

- Values are specified in the order in which the corresponding attributes are listed in the relation schema

# Insertion

- For the benefit of users who may not remember the order of the attributes

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Insert the fact that there is an account A-9732 at the perryridge branch and it has a balance of $1200

**insert into** account **(branch-name, account-number, balance)**

**values (‘perryridge’, ‘A-9732’,1200)**

# Insertion

- Insert tuples on the basis of the result of a query

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

```
insert into account
        select loan-number, branch-name, 200
        from loan
        where branch-name ='perryridge'
```

# Updates

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: All balances are to be increased by 5 percent

```
update account
        set balance=balance *1.05
```

# Updates

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: if balance is $1000 or more then, increase by 5 percent

```
update account
        set balance=balance *1.05
        where balance>=1000
```

# Updates

**customer(customer-name, customer-street, customer-city)**

**branch(branch-name, branch-city, assets)**

**account(account-number, branch-name, balance)**

**loan(loan-number, branch-name, amount)**

**borrower(customer-name, loan-number)**

**depositor(customer-name, account-number)**

Query: Pay 5 percent interest on accounts whose balance is greater than average

```
update account
        set balance=balance *1.05
        where balance> (select avg(balance) from account)
```

# Updates

Query: All accounts with balances over $10,000 receive 6 percent interest, whereas all others receive 5 percent

```
update account
        set balance=balance *1.06
        where balance >10000

update account
        set balance=balance *1.05
        where balance <=10000
```

# Updates

Query: All accounts with balances over $10,000 receive 6 percent interest, whereas all others receive 5 percent

```
update account
        set balance=balance *1.06
        where balance >10000
update account
        set balance=balance *1.05
        where balance <=10000
```

- **Order of the two statements is important.**
- **Otherwise an account with a balance just under $10,000 would receive 11.3 percent interest**

# Case construct

Query: Pay 5 percent interest on accounts whose balance is less than 10,000 and all others pay 6 percent

```
update account
        set balance= case
                        when balance <=10000 then balance *1.05
                        else balance*1.06
                        end
```

# Case construct

The general form of the case statement is

case

when $predicates_1$ then $result_1$

when $predicates_2$ then $result_2$

when $predicates_n$ then $result_n$

else $result_0$

end

# THANK YOU