Nested Queries

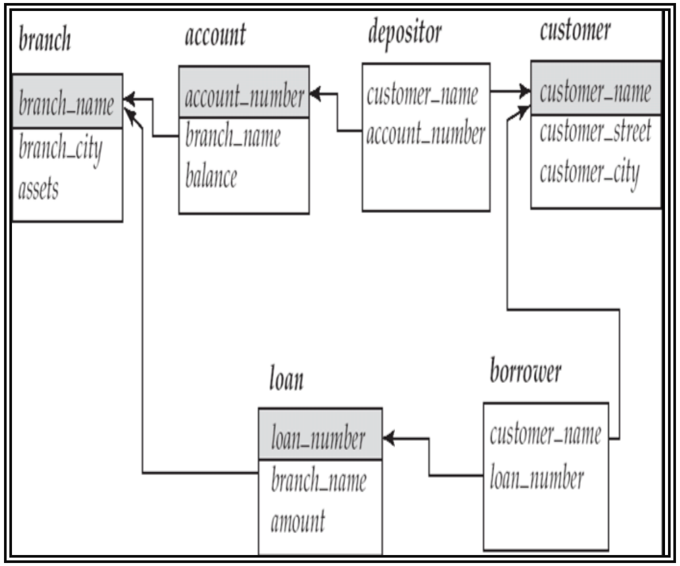
- **SQL provides a mechanism for the nesting of subqueries.**

- **A subquery is a select-from-where expression that is nested within another query.**

- **A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.**

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

```
select distinct customer_name
          from borrower
          where customer_name in (select customer_name
                                            from depositor )
```



In Construct- Nested Queries

- Find all customers who have a loan at the bank but do not have an account at the bank

```
select distinct customer_name
          from borrower
          where customer_name not in (select customer_name
                                      from depositor )
```

- Find all customers who have both an account and a loan at the Perryridge branch

```
select distinct customer_name
          from borrower, loan
          where borrower.loan_number = loan.loan_number and
           branch_name = 'Perryridge'  and
          (branch_name, customer_name ) in
                    (select branch_name, customer_name
                     from depositor, account
                     where depositor.account_number =
                       account.account_number )
```

“Some” Construct

- **Find 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'
```

- Same query using > **some** clause

```
select branch_name
        from branch
        where assets > some
                (select assets
                 from branch
                 where branch_city = 'Brooklyn')
```

“All” Construct

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

```
select branch_name
        from branch
        where assets > all
                (select assets
                from branch
                where branch_city = 'Brooklyn')
```

“Exists” Construct

- Find all customers who have an account at all branches located in Brooklyn.

```
select distinct S.customer_name
        from depositor as S
        where not exists (
                (select branch_name
                from branch
                where branch_city = 'Brooklyn')
                except
                (select R.branch_name
                from depositor as T, account as R
                where T.account_number = R.account_number and
                        S.customer_name = T.customer_name ))
```

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

Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- Find all customers who have at most one account at the Perryridge branch.

```
select T.customer_name
 from depositor as T
 where unique (
      select R.customer_name
      from account, depositor as R
      where T.customer_name = R.customer_name
              and
              R.account_number = account.account_number and
              account.branch_name = 'Perryridge')
```

- **Find all customers who have at least two accounts at the Perryridge branch.**

```
select distinct T.customer_name
from depositor  as T
where not unique (
     select R.customer_name
     from account, depositor as R
     where T.customer_name = R.customer_name and
               R.account_number = account.account_number  and
               account.branch_name = 'Perryridge')
```

- Variable from outer level is known as a **correlation variable**

Modification of the Database – Deletion

- **Delete all account tuples at the Perryridge branch**

```
delete from account
where branch_name = 'Perryridge'
```

- **Delete all accounts at every branch located in the city ‘Needham’.**

```
delete from account
where branch_name in (select branch_name
                      from branch
                      where branch_city = 'Needham')
```

- **Delete the record of all accounts with balances below the average at the bank.**

```
delete from account
        where balance < (select avg (balance )
                               from account )
```

- Problem: as we delete tuples from deposit, the average balance changes
- Solution used in SQL:
  1. First, compute **avg** balance 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 *account***

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

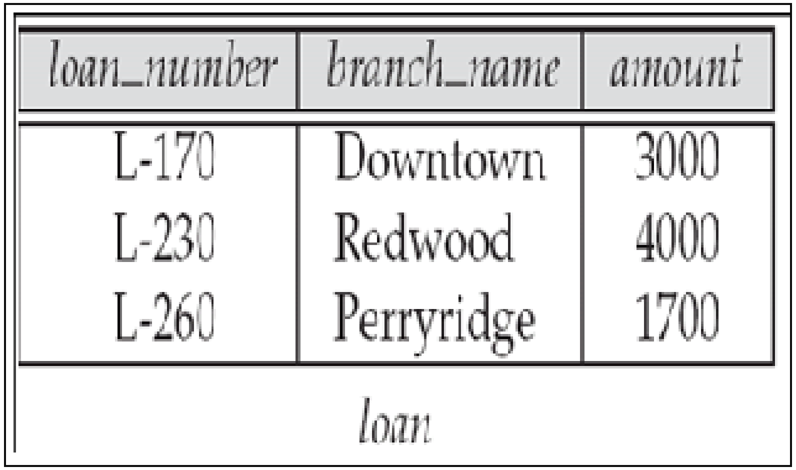
or equivalently

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

- **Add a new tuple to *account* with *balance* set to null**

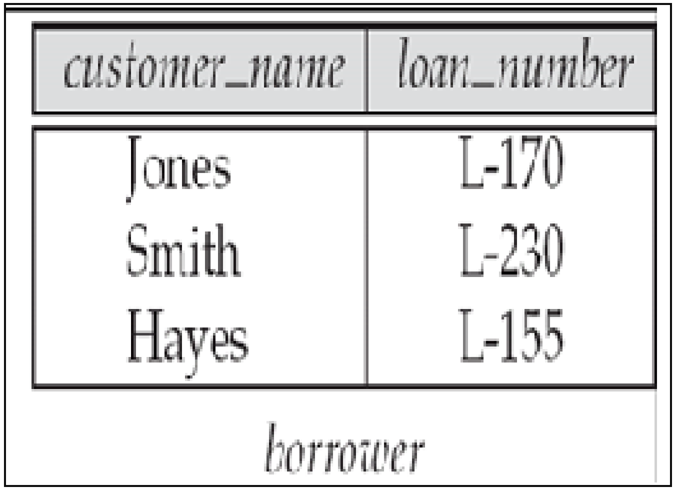
```
            insert into account
                    values ('A-777','Perryridge',  null )
```

Joined Relations – Datasets



loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan



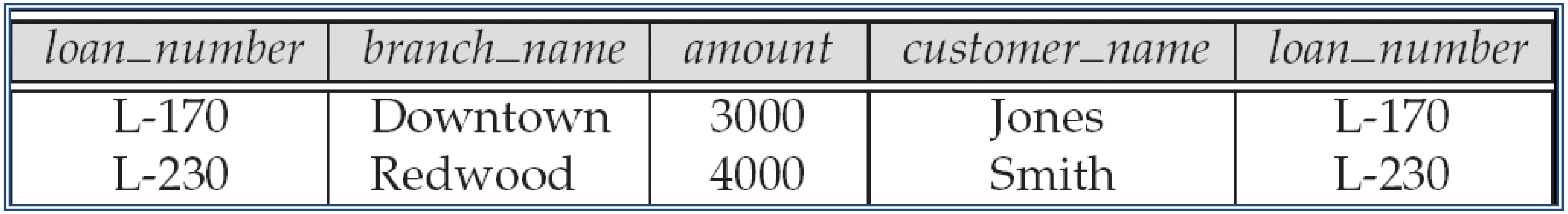
customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Note: borrower information missing for L-260 and loan information missing for L-155

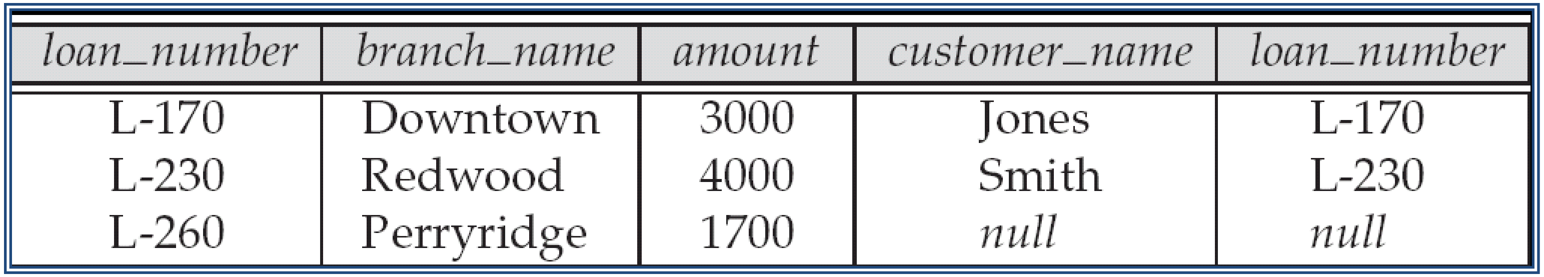
Joined Relations

- loan **inner join** borrower **on**
  loan.loan_number = borrower.loan_number



loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

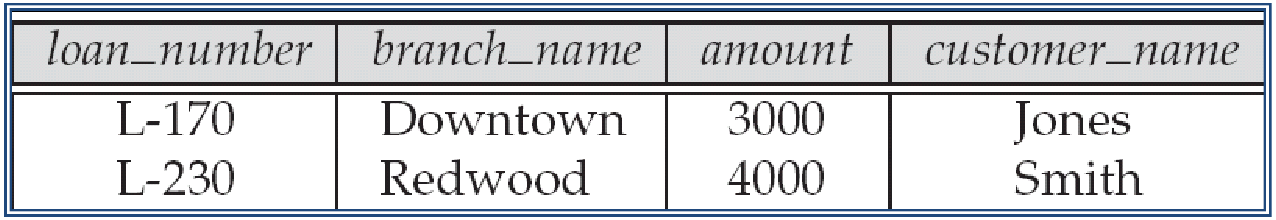
- loan **left outer join** borrower **on**
  loan.loan_number = borrower.loan_number



loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

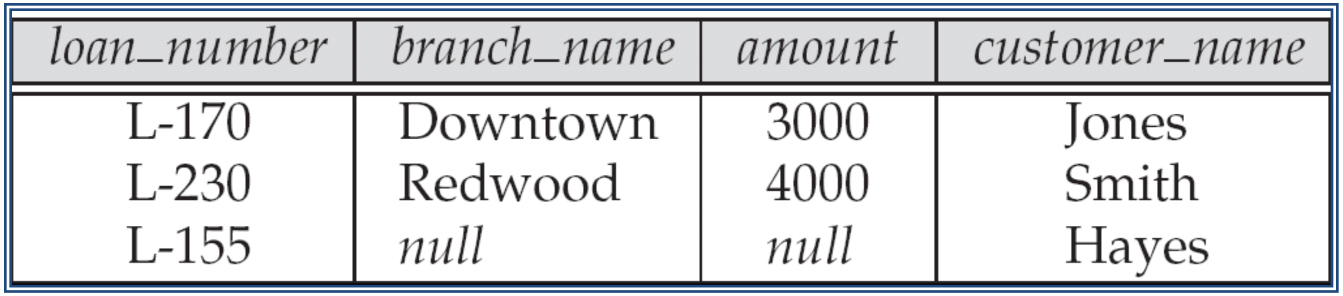
Joined Relations

- *loan* **natural inner join** *borrower*



loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

loan **natural right outer join** *borrower*



loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Find all customers who have either an account or a loan (but not both) at the bank.

```
select customer_name
        from (depositor natural full outer join borrower )
        where account_number is null or loan_number is null
```

Derived Relations

- SQL allows a subquery expression to be used in the **from** clause
- **Find the average account balance of those branches where the average account balance is greater than Rs 1200.**

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
           from account
           group by branch_name )
          as branch_avg ( branch_name, avg_balance )
where avg_balance > 1200
```

Note : We do not need to use the **having** clause, since we compute the temporary (view) relation branch_avg in the **from** clause, and the attributes of branch_avg can be used directly in the **where** clause.

View Definition

- **A relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a view.**

- **A view is defined using the create view statement which has the form**

  create view v as < query expression >

  where <query expression> is any legal SQL expression. The view name is represented by v.

- **Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.**

- **A view consisting of branches and their customers**

```
create view all_customer as
          (select branch_name, customer_name
           from depositor, account
          where depositor.account_number =
                    account.account_number )
          union
          (select branch_name, customer_name
          from borrower, loan
          where borrower.loan_number = loan.loan_number )
```

- **Find all customers of the Perryridge branch**

```
select customer_name
          from all_customer
          where branch_name = 'Perryridge'
```

Uses of Views

- Hiding some information from some users
  - Consider a user who needs to know a customer's name, loan number and branch name, but has no need to see the loan amount.

  - Define a view

```
(create view cust_loan_data as
 select customer_name, borrower.loan_number, branch_name
  from borrower, loan
  where borrower.loan_number = loan.loan_number )
```

  - Grant the user permission to **read cust_loan_data, but not borrower or loan**

  - Predefined queries to make writing of other queries easier
  - Common example: Aggregate queries used for statistical analysis of data

Processing of Views

- **When a view is created**
  - **Query expression is stored in the database along with the view name**
  - **Expression is substituted into any query using the view**
- **Views definitions containing views**
  - **One view may be used in the expression defining another view**
  - **A view relation v_1 is said to depend directly on a view relation v_2 if v_2 is used in the expression defining v_1**
  - **A view relation v_1 is said to depend on view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2**
  - **A view relation v is said to be recursive if it depends on itself.**

View Expansion

- A way to define the meaning of views defined in terms of other views.

- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.

- View expansion of an expression repeats the following replacement step:

```
repeat
   Find any view relation $v_i$ in $e_1$
   Replace the view relation $v_i$ by the expression defining $v_i$
until no more view relations are present in $e_1$
```

As long as the view definitions are not recursive, this loop will terminate

With Clause

- The **with** clause provides a way of defining a temporary view whose definition is available only to the query in which the **with** clause occurs.

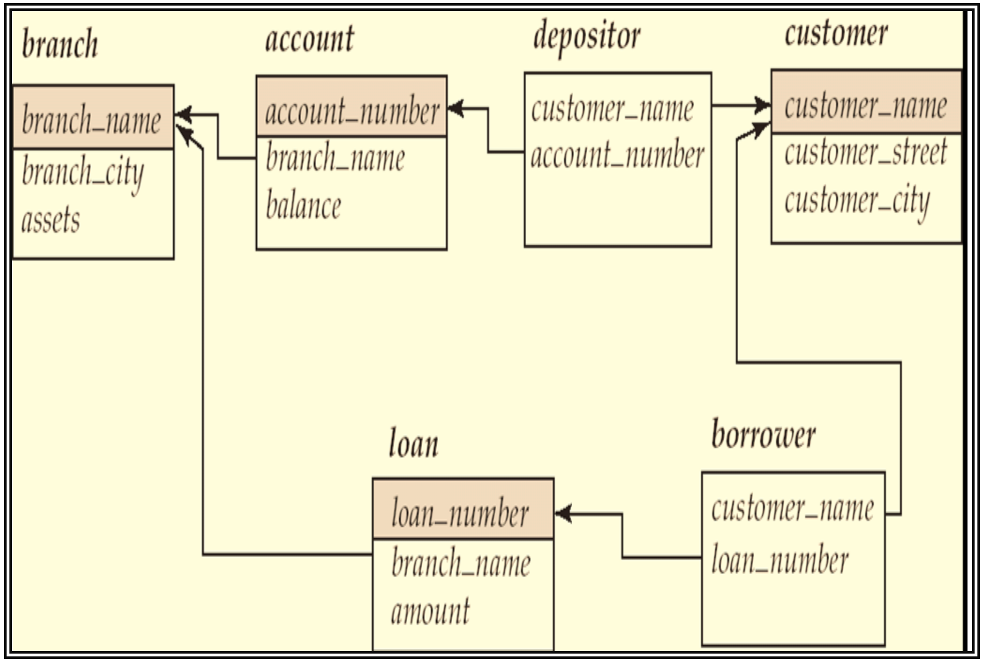
- **Find all accounts with the maximum balance**

```
with max_balance (value) as
    select max (balance)
    from account
select account_number
from account, max_balance
where account.balance = max_balance.value
```

Complex Queries using With Clause

- **Find all branches where the total account deposit is greater than the average of the total account deposits at all branches.**

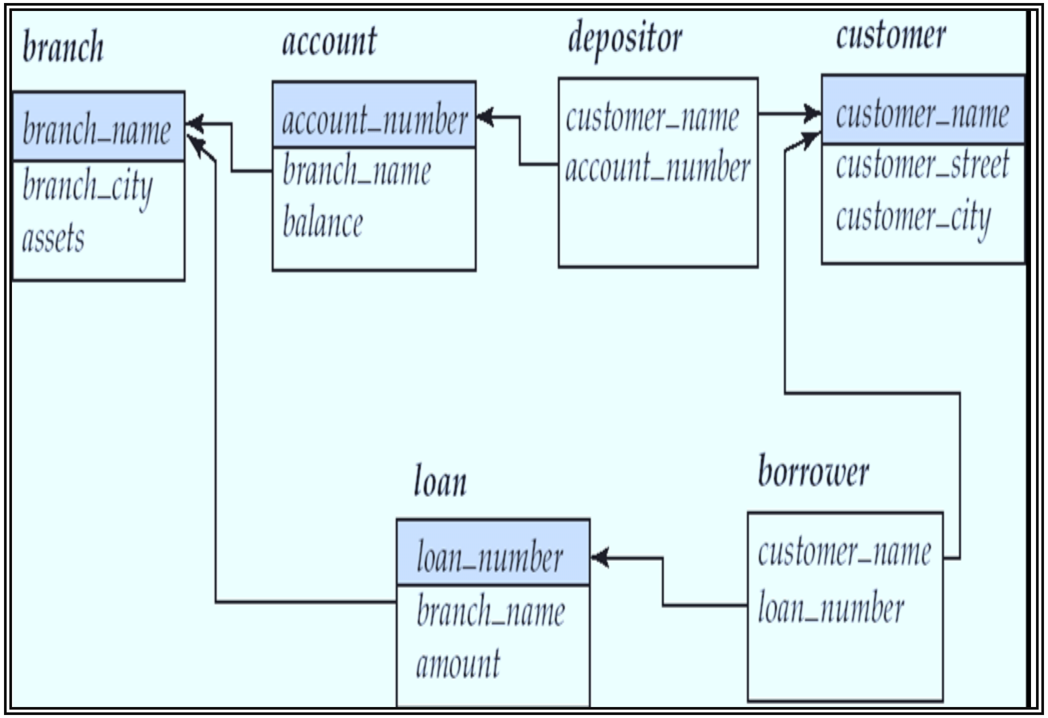
```
with branch_total (branch_name, value) as
        select branch_name, sum (balance)
        from account
        group by branch_name


with branch_total_avg (value) as
        select avg (value)
        from branch_total


select branch_name
 from branch_total, branch_total_avg
 where branch_total.value >= branch_total_avg.value
```



Update of a View

- **Create a view of all loan data in the loan relation, hiding the amount attribute**

create view loan_branch **as**
select loan_number, branch_name
from loan



Add a new tuple to loan_branch

insert into loan_branch
values ('L-37‘, 'Perryridge‘)

This insertion must be represented by the insertion of the tuple

('L-37', 'Perryridge', null)

into the loan relation

Updates Through Views ..contd.

- **Some updates through views are impossible to translate into updates on the database relations**

  - **create view** v **as**
    select loan_number, branch_name, amount
    from loan
    where branch_name = ‘Perryridge’

    insert into v **values** **(** 'L-99','Downtown', '23'**)**

- Most SQL implementations allow updates only on simple views (without aggregates) defined on a single relation

Null Values

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
- Null signifies an unknown value or that a value does not exist.
- The predicate **is null** can be used to check for null values.

Find all loan number which appear in the loan relation with null values for amount.

```
select loan_number
from loan
where amount is null
```

The result of any arithmetic expression involving null is null

Example: 5 + null returns null

However, aggregate functions simply ignore nulls

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

- **Result of where clause predicate is treated as false if it evaluates to unknown**

Null Values and Aggregates

- Total all loan amounts

```
select sum (amount )
from loan
```

  - Above statement ignores null amounts
  - Result is null if there is no non-null amount

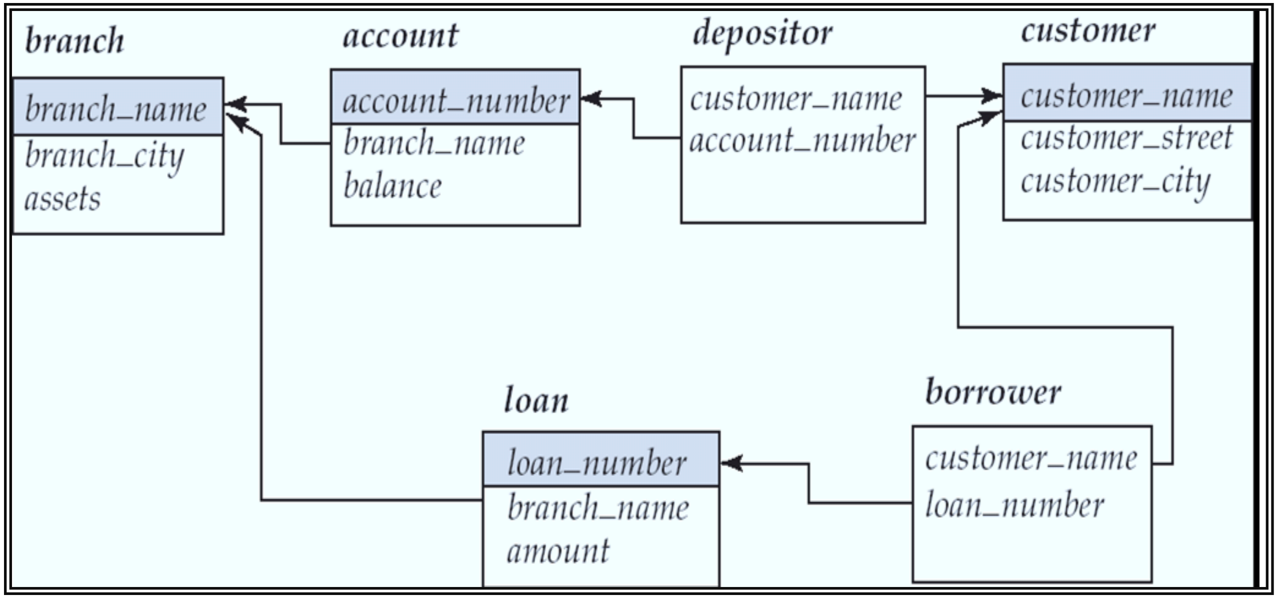
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

The where Clause ● ● ● Contd.

- SQL includes a **between** comparison operator

- Find the loan number of those loans with loan amounts between Rs 90,000 and Rs 100,000 (that is, >= Rs 90,000 and <= Rs100,000)

```
select loan_number
        from loan
        where amount between 90000 and 100000
```


branch
branch_name
branch_city
assets
account
account_number
branch_name
balance
depositor
customer_name
account_number
customer
customer_name
customer_street
customer_city
loan
loan_number
branch_name
amount
borrower
customer_name
loan_number