Basic Query Structure

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

  \[
  \text{select } A_1, A_2, \ldots, A_n \\
  \text{from } r_1, r_2, \ldots, r_m \\
  \text{where } P
  \]

  - \( A_i \) represents an attribute, \( R_i \) represents a relation.
  - \( P \) is a predicate.

- This query is equivalent to the relational algebra expression:

  \[
  \Pi_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))
  \]

- 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
  - projection operation of relational algebra
- Example: find the names of all branches in the loan relation:
  ```sql
  select branch_name
  from loan
  ```
- In the relational algebra, the query would be:
  $$\pi_{branch\_name}(loan)$$
The select Clause (Cont.)

- SQL allows duplicates in query results.
  - To force the elimination of duplicates, insert the keyword `distinct` after `select`.

- Ex. Find the names of all branches in the `loan` relations, and remove duplicates

  ```sql
  select distinct branch_name
  from loan
  ```
The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”
  
  ```sql
  select *
  from loan
  ```

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

- The query:
  
  ```sql
  select loan_number, branch_name, amount * 100
  from loan
  ```

  would return a relation similar to the loan relation, except that values for amounts are multiplied by 100.
The where Clause

- The *where* clause specifies conditions that the result must satisfy
  - Relational algebra’s selection predicate.
- 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**.

Comparisons can be applied to results of arithmetic expressions.

SQL includes a **between** comparison operator.

**Example**: Find the loan number of those loans with loan amounts between $90,000 and $100,000 (that is, $\geq 90,000$ and $\leq 100,000$)

```sql
select loan_number
from loan
where amount between 90000 and 100000
```
The from Clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.

- **Ex. borrower X loan**
  
  ```sql
  select *
  from borrower, loan
  
  No where clause!
  ```
The from Clause (Cont.)

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

```sql
SELECT customer_name, borrower.loan_number, amount 
FROM borrower, loan 
WHERE borrower.loan_number = loan.loan_number 
AND branch_name = 'Perryridge'
```
The Rename Operation

- The SQL allows renaming relations and attributes using the as clause:
  
  \[ \text{old-name as new-name} \]

- Find the name, loan number and loan amount of all customers; rename the column name \text{loan_number} as \text{loan_id}.

```
select customer_name,
       borrower.loan_number as loan_id, amount
from borrower, loan
where borrower.loan_number = loan.loan_number
```
Tuple Variables / Aliases

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

```sql
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.

\[
\begin{align*}
\text{select distinct } & \quad T.\text{branch\_name} \\
\text{from } & \quad \text{branch as } T, \text{ branch as } S \\
\text{where } & \quad T.\text{assets} > S.\text{assets} \\
\text{and } & \quad S.\text{branch\_city} = \text{‘Brooklyn’}
\end{align*}
\]
String Operations

- SQL includes a pattern matching operator for comparisons on character strings.
- The operator “like” uses patterns that are described using two special characters:
  - percent % or *: matches any substring.
  - underscore _ or ?: matches any character.
- *E.g.* Find the names of all customers whose street includes the substring “Main”.

```sql
select customer_name
from customer
where customer_street like '%Main%'
```
String Operations

- Streets that match the name “Main%”
  - % (or *) are part of the substring
    … like ‘Main\%’ escape ‘\’
- E.g. Any street name with exactly 5 characters
  … like ‘_ _ _ _ _’
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.
Set Operations

- The set operations **union**, **intersect**, and **except** operate on relations and correspond to the relational algebra operations $\cup$, $\cap$, $\neg$.
- Each of the above operations automatically eliminates duplicates;
- to retain all duplicates use the corresponding multiset versions
  - **union all**, **intersect all** and **except all**.
Set Operations

• Find all customers with a loan, an account, or both:
  \[(\text{select } \text{customer}._\text{name} \text{ from } \text{depositor})\]
  \[\text{union}\]
  \[(\text{select } \text{customer}._\text{name} \text{ from } \text{borrower})\]

• Find all customers with both a loan and an account:
  \[(\text{select } \text{customer}._\text{name} \text{ from } \text{depositor})\]
  \[\text{intersect}\]
  \[(\text{select } \text{customer}._\text{name} \text{ from } \text{borrower})\]

• Find all customers with an account but no loan:
  \[(\text{select } \text{customer}._\text{name} \text{ from } \text{depositor})\]
  \[\text{except}\]
  \[(\text{select } \text{customer}._\text{name} \text{ from } \text{borrower})\]
Null Values

The predicate **is null** is used to check for null values.

- Example: Find all loan number which appear in the *loan* relation with null values for *amount*.

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

- There is also a **is not null** option.
Nested Subqueries

- 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.
Nested Subqueries Examples

• 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 )
```

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

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

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

  Note: This query can be written in a much simpler manner. The formulation below is simply to illustrate SQL features.

```sql
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 )
```
Set comparison: the Some clause

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

```sql
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

```sql
select branch_name
from branch
where assets > some
  (select assets
   from branch
   where branch_city = 'Brooklyn')
```
Definition of Some Clause

- $F \ <\text{comp}\ > some \ r \iff \exists\ t \in\ r\ such\ that\ (F\ <\text{comp}\ > t)$

Where $\text{<comp>}$ can be: $<, \le, >, =, \neq$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$F &lt; \text{some } r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

(read: $5 < \text{some tuple in the relation}$)

$(5 < \text{some } 5) = true$

$(5 < \text{some } 5) = false$

$(5 = \text{some } 5) = true$

$(5 \neq \text{some } 5) = true$ (since $0 \neq 5$)

$(= \text{some}) \equiv \text{in}$

However,

$(\neq \text{some}) \not\equiv \text{not in}$
Set comparison: the All clause

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

```sql
select branch_name
from branch
where assets > all
  (select assets
   from branch
   where branch_city = 'Brooklyn')
```
Definition of all Clause

\[ \text{F <comp> all } r \iff \forall t \in r \ (F <\text{comp}> t) \]

\[
\begin{align*}
(5 < \text{all } & 5 \quad ) = \text{false} \\
5 6 & \\
(5 < \text{all } & 6 10 \quad ) = \text{true} \\
6 & \\
(5 = \text{all } & 4 5 \quad ) = \text{false} \\
4 & \\
(5 \neq \text{all } & 4 6 \quad ) = \text{true} \\
4 & \\
\end{align*}
\]

(\neq \text{all}) \equiv \text{not in}

However,

(= \text{all}) \neq \text{in}

(since 5 \neq 4 \text{ and } 5 \neq 6)
Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \iff r \neq \emptyset$
- **not exists** $r \iff r = \emptyset$
- *E.g.* Find all customers that have both an account and a loan

```sql
select customer_name
from borrower
where exists
  (select *
   from depositor
   where depositor.customer_name = borrower.customer_name)
```
Another Exists Query

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

```sql
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 ))
```

- For each customer, we need to check whether the set of all branches he has an account contains the set of all branches in Brooklyn.
- Note that $X - Y = \emptyset \iff X \subseteq Y$
Joined Relations

- **Join operations** take two relations and return as a result another relation.

- These additional operations are typically used as subquery expressions in the **from** clause
  - SQL92 style: only relations in from clause. *E.g.*

```
select customer_name, T.loan_number, S.amount
from  borrower as T, loan as S
where  T.loan_number = S.loan_number
```
Joined Relations (cont.)

- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.

- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

<table>
<thead>
<tr>
<th>Join types</th>
<th>Join Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>inner join</td>
<td>natural</td>
</tr>
<tr>
<td>left outer join</td>
<td>on &lt;predicate&gt;</td>
</tr>
<tr>
<td>right outer join</td>
<td>using ((A_1, A_1, \ldots, A_n))</td>
</tr>
<tr>
<td>full outer join</td>
<td></td>
</tr>
</tbody>
</table>
### Joined Relations – Datasets for Examples

- **Relation borrower and loan**

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
<th>customer_name</th>
<th>loan_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>L-260</td>
<td>Perryridge</td>
<td>1700</td>
<td>Hayes</td>
<td>L-155</td>
</tr>
</tbody>
</table>

- Note: borrower information missing for L-260 and loan information missing for L-155
Joined Relations – Examples (cont.)

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

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
<th>customer_name</th>
<th>loan_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
<th>customer_name</th>
<th>loan_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>L-260</td>
<td>Perryridge</td>
<td>1700</td>
<td><code>null</code></td>
<td><code>null</code></td>
</tr>
</tbody>
</table>
### Joined Relations – Examples (cont.)

- **loan natural inner join** borrower

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
<th>customer_name</th>
<th>loan_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
</tbody>
</table>

- **loan natural right outer join** borrower

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>L-155</td>
<td>null</td>
<td>null</td>
<td>Hayes</td>
</tr>
</tbody>
</table>
Joined Relations – Examples (cont.)

- `loan full outer join borrower using (loan_number)`

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
<th>customer_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>L-260</td>
<td>Perryridge</td>
<td>1700</td>
<td>null</td>
</tr>
<tr>
<td>L-155</td>
<td>null</td>
<td>null</td>
<td>Hayes</td>
</tr>
</tbody>
</table>

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

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