L03: Structured Query Language (SQL) – Part 1

Dr. Kenneth LEUNG

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China
• SQL is the most common **Query Language**
  - Used in all commercial systems
• Discussion is based on the **SQL92 Standard**
  - Commercial products have different features of SQL, but the basic structure is the same
• **SQL features**
  - **Data Manipulation Language** (covered in this lecture)
  - **Data Definition Language**
  - **Constraint Specification**
  - **Embedded SQL**
  - **Transaction Management**
  - **Security Management**
  - …
SQL is based on set and relational operations with certain modifications and enhancements.

A typical SQL query has the form:

```
select A1, A2, ..., An
from R1, R2, ..., Rm
where P
```

- $A_i$ represent attributes
- $R_i$ represent relations
- $P$ is a predicate

This query is equivalent to the relational algebra expression:

```
π_{A_1, A_2, ..., A_n}(σ_P(R_1 × R_2 × ... × R_m))
```

The result of an SQL query is a relation (but may contain duplicates).

SQL statements can be nested.
Projection

- 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

- **Example:**
  Find the names of all branches in the *Loan* relation

```
select branch-name
from Loan
```

Equivalent to (but possibly with **duplicates**):

\[ \pi_{\text{branch-name}}(\text{Loan}) \]

- An asterisk in the select clause denotes "all attributes"

```
select *
from Loan
```

\[ \text{Loan (loan-number, amount, branch-name)} \]
• SQL allows duplicates in relations as well as in query results
• Use select distinct to force the elimination of duplicates

• Example:
  Find the names of all branches in the Loan relation, and remove duplicates

  select distinct branch-name
  from Loan

• The keyword all specifies that duplicates are not removed

  select all branch-name
  from Loan

  force the DBMS to remove duplicates

  force the DBMS not to remove duplicates
The select clause can contain arithmetic expressions involving the operators, $+$, $-$, $\div$ and $\ast$, and operating on constants or attributes of tuples.

Example:

```
select branch-name, loan-number, amount * 100
from Loan
```

Returns a relation which is the same as the Loan table, except that the attribute amount is multiplied by 100.
The ‘where’ Clause

- The **where** clause specifies **conditions** that tuples in the relations in the **from** clause must satisfy.

- **Example:**
  Find all loan numbers for loans made at the Perryridge branch with loan amounts greater than $1200.

```sql
select loan-number
from Loan
where branch-name=“Perryridge” and amount > 1200
```

- SQL allows logical connectives **and, or, and not**
- Arithmetic expressions can be used in the **comparison operators**
  - >, <, >=, <=, =, <> (not equal to)
- **Note:** attributes used in a query (both **select** and **where** parts) must be defined in the relations in the **from** clause

Loan (loan-number, amount, branch-name)
SQL includes the *between* operator for convenience

**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 corresponds to the Cartesian product operation of the relational algebra.
- Example: Find the Cartesian product (Borrower \( \times \) Loan)

  ```sql
  select * 
  from Borrower, Loan 
  ```

- Note: It is rarely used without a where clause (join).
- Example: Find the name and loan number of all customers having a loan at the Perryridge branch.

  ```sql
  select distinct customer-name, Borrower.loan-number 
  from Borrower, Loan 
  where Borrower.loan-number = Loan.loan-number and 
  branch-name = "Perryridge" 
  ```
The Rename Operation

- **Renaming** attributes using the `as` clause:
  
  \[ \text{old-name as new-name} \]

- **Example:**
  Find the name and loan number of all customers having a loan at the Perryridge branch; replace the column name `loan-number` with the name `loan-id`.

\[
\text{select distinct customer-name, Borrower.loan-number as loan-id from Borrower, Loan where Borrower.loan-number = Loan.loan-number and branch-name = “Perryridge”}
\]
• **Tuple variables** (or aliases) are defined in the `from` clause via the use of the `as` clause

• **Example:** Find the customer names and their loan numbers for all customers having a loan at some branch

```sql
select distinct customer-name, T.loan-number
from  Borrower as T, Loan as S
where T.loan-number = S.loan-number
```

• A tuple variable can be used as short hand, but it is more than just a short hand (see next slide)
Example: 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"
```
• Character attributes can be compared to a pattern using **like**:  
  - `%` matches any **substring**  
  - `_` matches any **single character**

• **Example:**  
  Find the name of all customers whose street includes the substring ‘Main’ (e.g., Mainroad, Smallmain Road, AMainroad, …)

```sql
select customer-name  
from Customer  
where customer-street like "%Main%"
```
Ordering the Display of Tuples

- **Example:** List in *alphabetical order* the names of all customers having a loan at Perryridge branch

  ```sql
  select distinct customer-name
  from Borrower, Loan
  where Borrower.loan-number = Loan.loan-number and
  branch-name = "Perryridge"
  order by customer-name
  ```

- **Alternatively:** `order by customer-name desc, amount asc`
  - `desc` for descending order; `asc` for ascending order (default)

- **Note:** SQL must perform a sort to fulfill an `order by` request. Since sorting a large number of tuples may be costly, it is desirable to sort only when necessary.

  - Loan (loan-number, amount, branch-name)
  - Borrower (customer-name, loan-number)
Set Operations

• The set operation union, intersect, and except operate on relations and correspond to the relational algebra operations ∪, ∩ and −

• Each of the above operations automatically eliminates duplicates; to retain all duplicates use union all, intersect all and except all

• Suppose a tuple occurs m times in relation R and n times in relation S. Then, it occurs:
  - m + n times in (R union all S)
  - min(m, n) times in (R intersect all S)
  - max(0, m-n) times in (R except all S)
• **Example:** Find all customers who have a loan, an account, or both:

\[
\text{(select customer-name from Depositor)} \cup \text{(select customer-name from Borrower)}
\]

• **Example:** Find all customers who have both a loan and an account:

\[
\text{(select customer-name from Depositor)} \cap \text{(select customer-name from Borrower)}
\]

• **Example:** Find all customers who have an account but no loan:

\[
\text{(select customer-name from Depositor)} \setminus \text{(select customer-name from Borrower)}
\]
Every SQL statement returns a relation/set in the result; remember a relation could be null or merely contain a single atomic value.

You can replace a value or set of values with a SQL statement (i.e., a subquery):

```
select * from Loan
where amount > 1200
```

```
select * from Loan
where amount > select avg(amount) from Loan
```

**Note:** It is illegal if the subquery returns the wrong type for the comparison.
• **Example:** Find all customers who have *both* an account and a loan in the bank.

```sql
select distinct customer-name 
from   Borrower 
where customer-name in ( select customer-name 
                          from   Depositor)
```

Check for each borrower if he/she is *also* a depositor

Return the set of depositors

Depositor (*customer-name, account-number*)
Borrower (*customer-name, loan-number*)
The ‘not in’ Clause

- **Example:** 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)
```

Make sure each borrower is *not also* a depositor

Return the set of depositors

Depositor (*customer-name, account-number*)
Borrower (*customer-name, loan-number*)
• **Example:** Find all branches that have greater assets than some branch located in Brooklyn
  
  - Equivalent to “find all branches that have greater assets than the **minimum** assets of any branch located in Brooklyn”

```sql
select branch-name
from Branch
where assets > some
    ( select assets
        from Branch
        where branch-city = "Brooklyn")
```

Assets of all branches in Brooklyn
The ‘some’ Clause – Semantics

Note:
- \((= \text{some})\) is equivalent to \(\text{in}\)
- However, \((\neq \text{some})\) is not equivalent to \(\text{not in}\)

\[
\begin{array}{c|c|c}
5 < \text{some} & 0 & 5 \\
\hline
& 5 & 6 \\
\end{array}
\]

\(= \text{true} \ (5 < 6)\)

\[
\begin{array}{c|c|c}
5 < \text{some} & 0 & 5 \\
\hline
& 5 & \\
\end{array}
\]

\(= \text{false}\)

\[
\begin{array}{c|c|c}
5 = \text{some} & 0 & 5 \\
\hline
& 5 & \\
\end{array}
\]

\(= \text{true}\)

\[
\begin{array}{c|c|c}
5 \neq \text{some} & 0 & 5 \\
\hline
& 5 & \\
\end{array}
\]

\(= \text{true} \ (\text{since } 0 \neq 5)\)
The ‘all’ Clause

• Example: Find the names of all branches that have greater assets than all branches located in Brooklyn.
  – Equivalent to “find all branches that have greater assets than the maximum assets of any branch located in Brooklyn”

\[
\text{select branch-name}
\text{from Branch}
\text{where assets > all}
\text{( select assets}
\text{from Branch}
\text{where branch-city=“Brooklyn”)}
\]

Assets of all branches in Brooklyn

Branch (branch-name, branch-city, assets)
The ‘all’ Clause – Semantics

\[
\begin{array}{c}
(5 < \text{all} \begin{array}{c} 0 \\ 5 \\ 6 \end{array} ) = \text{false} \\
(5 < \text{all} \begin{array}{c} 6 \\ 10 \end{array} ) = \text{true} \\
(5 = \text{all} \begin{array}{c} 4 \\ 5 \end{array} ) = \text{false} \\
(5 \neq \text{all} \begin{array}{c} 6 \\ 10 \end{array} ) = \text{true}
\end{array}
\]

Note:
- \((\neq \text{all})\) is equivalent to not in
- However, \((= \text{all})\) is not equivalent to in
• **exists** returns true if the argument subquery is nonempty

• **Example:** Find all customer names who have both a loan and an account:

```sql
select customer-name from Depositor as D where exists
    ( select * from Borrower as B
    where D.customer-name = B.customer-name)
```

• **Example:** Find all customer names who have an account but no loan:

```sql
select customer-name from Depositor as D where not exists
    ( select * from Borrower as B
    where D.customer-name = B.customer-name)
```
Test for Absence of Duplicate Tuples

- **unique** tests whether a subquery has any duplicate tuples in its result.

- **Example:** Find all customers who have *only one* account at the Perryridge branch:

  ```sql
  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")
  ```

  Depositors at Perryridge with same name as T

Depositors with same name as T

Depositor (customer-name, account-number)
Account (account-number, balance, branch-name)
Example: Find all customers with at least 2 accounts at the Perryridge branch.

```
select 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")
```
Example: Find all customers with an account at all branches located in Brooklyn.

```
select distinct S.customer-name
from Depositor as S
where not exist
    ( ( 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 S...

Branches in Brooklyn where customer S does not have an account

Branches where customer S does have an account

\[ X - Y = \emptyset \iff X \subseteq Y \]