Chapter 3: SQL

- Data Definition
- Basic Query Structure
- Set Operations
- Aggregate Functions
- Null Values
- Nested Subqueries
- Complex Queries
- Views
- Modification of the Database
- Joined Relations
IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory

Renamed Structured Query Language (SQL)

ANSI and ISO standard SQL:
- SQL-86
- SQL-89
- SQL-92
- SQL:1999 (language name became Y2K compliant!)
- SQL:2003

Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- Not all examples here may work on your particular system.

### Data Definition Language (DDL)

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.
Data Manipulation Language (DML)

- Used to search and query the database, and
- To update the database: Three basic constructs
  1. Insert,
  2. Delete
  3. Update
- Sometimes the term “Query Language” is used as a synonym of DML

---

Figure 3.1: Database Schema

```sql
branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (customer_name, loan_number)
account (account_number, branch_name, balance)
depositor (customer_name, account_number)
```
Modification of the Database – Deletion

- Delete all account tuples at the Perryridge branch
  
  ```sql
  delete from account
  where branch_name = 'Perryridge'
  ```

- Delete all accounts at every branch located in the city ‘Needham’.
  
  ```sql
  delete from account
  where branch_name in (select branch_name
                         from branch
                         where branch_city = 'Needham')
  ```

---

Example Query

- Delete the record of all accounts with balances below the average at the bank.
  
  ```sql
  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`
  
  ```sql
  insert into account
  values ('A-9732', 'Perryridge', 1200)
  ```

  or equivalently

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

- Add a new tuple to `account` with `balance` set to null
  
  ```sql
  insert into account
  values ('A-777', 'Perryridge', null)
  ```

Modification of the Database – Insertion

- Provide as a gift for all loan customers of the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account
  
  ```sql
  insert into account
  select loan_number, branch_name, 200
  from loan
  where branch_name = 'Perryridge'
  ```

- The `select from where` statement is evaluated fully before any of its results are inserted into the relation (otherwise queries like
  
  ```sql
  insert into table1 select * from table1
  ```
  
  would cause problems)
Modification of the Database – Updates

- Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.
  - Write two update statements:
    
    ```sql
    update account
    set balance = balance * 1.06
    where balance > 10000
    
    update account
    set balance = balance * 1.05
    where balance <= 10000
    ```

  - The order of stored tuples is not important
  - The order between statements is important
  - Can be done better using the case statement (next slide)

Case Statement for Conditional Updates

- Same query as before: Increase all accounts with balances over $10,000 by 6%, all other accounts receive 5%.
  
  ```sql
  update account
  set balance = case
    when balance <= 10000 then balance * 1.05
    else balance * 1.06
  end
  ```
Updating Though Views

- Bad Idea—Avoid it!

Joined Relations – Datasets for Examples

- Relation loan
- Relation borrower

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

<table>
<thead>
<tr>
<th>customer_name</th>
<th>loan_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<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

- **loan inner join borrower on**
  \[
  \text{loan.loan\_number} = \text{borrower.loan\_number}
  \]

<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**
  \[
  \text{loan.loan\_number} = \text{borrower.loan\_number}
  \]

<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>null</td>
<td>null</td>
</tr>
</tbody>
</table>

- **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

- 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
```