SQL: Structured Query Language

SELECT Statement

SELECT [DISTINCT] select-list
FROM from-list
[WHERE qualification ]

- from-list contains the list of database relations from which the data is to be retrieved.
- select-list contains the list of relation attributes (possibly modified) to be returned in the answer to the query.
- qualification contains the conditions which must be satisfied by a database record to be put into the answer set.

Evaluation of SELECT statement:

1. from-list defines a cartesian product of all relations in it.
2. qualification defines selection and join conditions on the data.
3. select-list defines the final look of the answer set (i.e., the projection attributes).
4. DISTINCT specifies duplicate elimination in the final answer set (default for SELECT QUERY: no duplicate elimination).

SQL select Statement and Relational Algebra

As one can guess from its evaluation, SELECT statement implements Relational Algebra operations selection, projection, cartesian product and join.
select and selection

Relational algebra operation: $\sigma_C(R)$

SELECT statement

```sql
SELECT *
FROM R
WHERE C
```

Note: “*” is a special notation for the selection list that includes all available attributes.

Example: RA: $\sigma_{\text{salary}>20000}(\text{Employee})$

SQL:

```sql
SELECT *
FROM Employee
WHERE salary > 20000
```

select and projection

Relational algebra operation: $\pi_F(R)$

SELECT statement

```sql
SELECT DISTINCT F
FROM R
```

Note: Relational algebra operations return sets (with no duplicates), hence to represent true relational algebra projection, we need to use DISTINCT here. While we may ignore this sometimes, please, keep this fact in mind.

Example: RA: $\pi_{\text{name, salary, position}}(\text{Employee})$

SQL:

```sql
SELECT DISTINCT name, salary, position
FROM Employee
```

select and cartesian product

Relational algebra operation: $R_1 \times R_2$

SELECT statement

```sql
SELECT *
FROM R1, R2
```

Note: As mentioned above, more than one relation in the from list represents cartesian product.
Example: RA: $Employee \times Client$

SQL:

```
SELECT *
FROM Employee, Client
```

select and cartesian product

Relational algebra operation: $R_1 \bowtie_{R_1.A=R_2.B} R_2$

**SELECT** statement

```
SELECT *
FROM $R_1, R_2$
WHERE $R_1.A = R_2.B$
```

Note: For clarity, a simple equijoin is used here. This conversion can be extended to other types of join.

Example: RA: $Employee \bowtie_{Employee.manages=Client.Id} Client$

SQL:

```
SELECT *
FROM Employee, Client
WHERE Employee.manages = Client.Id
```

From-list

**From-list** of the **SELECT** statement has the following format:

```
TableName [TableAlias] [. . .]
```

- **TableName** is a name of the existing database relation.
- **TableAlias** also known as range variable is an identifier that can be used instead of **TableAlias**.

While there can be two **TableNames** that have the same value in a from-list, all **TableAliases** are unique!

Table aliases allow representation of self-joins in SQL.

Example

*Find all employees in John Smith’s department who have bigger salary and output their records side by side.*

```
SELECT *
FROM Employee E1, Employee E2
WHERE E1.department = E2.department AND E1.salary < E2.salary
AND E1.name = "John Smith"
```
Select-list

A simple select-list is a comma-separated sequence of attribute names. However, more complex expressions can appear in a select list.

- \(*\): As mentioned above, this is a shortcut for "all attributes of all relations involved in the query in their natural order".
- \(AttributeName\): a simple attribute name can be part of a select-list if the attribute name alone is sufficient to uniquely identify the attribute among all attributes involved in the query;
- \(TableName.AttributeName\): if two different relations involved in the query have attributes with the same name, table name must preface the attribute name to guarantee unambiguous identification.
- \(TableAlias.AttributeName\): whenever table alias is defined for a relation it is always safe to reference the attribute in this manner. This is also the only way to indicate the exact attribute to be included into select-list in self-joins.
- \(Expression\): An expression over different attributes involved in a query is can also be returned in select-list.
- \(AttributeSpec as NewAttributeName\): this form is used to rename the attribute in the result of the query. It can be used to give names to the attributes that are results of expressions as well as to replace TableName.AttributeName names with simpler attribute names.
- \(Aggregate Expressions\): select queries also allow for return of aggregate expressions (sums, averages, minimums and maximums, etc.) of attributes. More on that later.

Examples

1. Different attribute identifiers.
   
   Find all books that were borrowed at least twice in 2002 and output the names of the two borrowers in each record in chronological order.
   
   ```sql
   SELECT Title, Book.Id, P1.Name, P2.Name 
   FROM Books, Loans L1, Loans L2, Patrons P1, Patrons P2 
     P1.Id = L1.Pid AND P2.Id = L2.Pid 
     AND L1.Date < L2.Date AND L1.Date > 12/31/2001
   ```

   Note: This query is a 5-way join that includes two self-joins: on Patrons and Loans. Title attribute is unambiguous as it exists only in Books. Book qualifier is needed for Book.Id because Patrons relation also has Id field. Finally, P1 and P2 qualifiers for the two Name attributes are needed to indicate which attribute comes from which copy of the Patrons relation.

2. Expressions as columns.
   
   Output for each employee the sum of his/her salary and bonus.
   
   ```sql
   SELECT Name, Salary + Bonus 
   FROM Employee
   ```
3. Renaming output columns

Output for each employee the sum of his/her salary and bonus.

```
SELECT Name AS Employee_Name, Salary + Bonus AS Compensation
FROM Employee
```

Note: same query as above, only the first field is renamed "Employee_Name" and the second field is given a new name "Compensation".