SQL II

1 SQL Syntax

Basis syntax of SQL queries:

```
SELECT [ALL|DISTINCT] column1[,column2,column3,...]
FROM table1[,table2,table3,...]
[WHERE "conditions"];
```

- The ”conditions” in the WHERE clause can be:
  - A comparison predicate (e.g. salary > 10000)
  - A range predicate (e.g. salary BETWEEN 10000 AND 30000)
  - A set membership predicate (e.g. position IN(’Manager’,’Worker’))
  - A pattern matching predicate (e.g. address LIKE ’%Glaskow%’)
  - Combinations of the above with AND and OR
- But it can also be the result of another (independent) query (called subquery)
- Three types of subquery:
  1. A single-value (scalar) subquery (single column & single row)

     ```
     SELECT COUNT(*) AS myCount
     FROM PropertyForRent
     WHERE rent>350
     ```

  2. A multiple value subquery (one column & multiple rows)

     ```
     SELECT staffNo
     FROM Staff
     WHERE position='Manager'
     ```

  3. A table subquery (multiple columns/rows)

     ```
     SELECT clientNo, viewDate
     FROM Viewing
     WHERE propertyNo='PG4' AND comment IS NULL
     ```

2 Subquery

List staff who work in branch at ’163 Main St’

```
SELECT staffNo, fName, IName, position
FROM Staff
WHERE branchNo=
        (SELECT branchNo
         FROM Branch
         WHERE street='163 Main St')
```

- The inner SELECT:
  - Finds the branch number of the branch in 163 main street
  - Only one such branch (with branchNo=’B003’)⇒ scalar subquery
- The outer SELECT is equivalent with:

  ```
  SELECT staffNO, fName, IName, position
  FROM Staff
  WHERE branchNo='B003'
  ```

List all staff whose salary is greater than the average salary, and show by how much

- If we know that the average salary is 17000, then:

```
SELECT staffNo, fName, IName, position,
        salary-17000 AS SalDiff
FROM Staff
WHERE salary>17000
```

- We cannot write ”WHERE salary>AVG(salary)”
- Instead, we use a subquery

```
SELECT staffNo, fName, IName, position
        salary-(SELECT AVG(salary) FROM Staff) AS SalDiff
FROM Staff
WHERE salary>(SELECT AVG(salary) FROM Staff)
```

3 Nested Queries

Example - (scalar subquery and multi-value subquery) - use of the operator IN:
List the properties that are handled by staff who work in the branch with the address '163 Main St'

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN (SELECT staffNo
                FROM Staff
                WHERE branchNo=
                        (SELECT branchNo
                         FROM branch
                         WHERE street='163 Main St'))
```

- From the innermost query outwards:
  - The first query selects the branch number of the branch at 163 Main St
  - The second selects the staff working at this branch
  - Many staff ⇒ in the outermost query we use IN (”=” is not possible)
- In multi value subqueries:
  - Use of the operator ANY (or SOME) before the subquery means the WHERE condition is true if it is satisfied by at least one value returned by the subquery

Find all staff whose salary is larger than the salary of at least one member of staff at Branch B003

```
SELECT staffNo, fName, IName, position, salary
FROM Staff
WHERE salary> SOME(SELECT salary
                   FROM Staff
                   WHERE branchNo='B003')
```

An alternative would be:

```
WHERE salary>(SELECT MIN(salary)
              FROM Staff
              WHERE branchNo='B003')
```

4 Multi-Table Queries

- All examples so far have a major limitation: the whole information belongs to a single table
- We can extend queries to multiple tables either with subqueries that query different tables:
  List all Durham staff with salary greater than the average London-salary

  ```
  SELECT staffNo, fName, IName, position
  FROM DurhamStaff
  WHERE salary>(SELECT AVG(salary) FROM LondonStaff)
  ```

- Or by using a join operation:
  - Link data from two (or more) tables together (in a single query)
  - Include more than one table in the FROM clause
  - Separate these tables with a comma

5 Joins

- In joins, usually
  - Include a WHERE clause to specify the joined columns
  - We keep in the search only those rows which have the same values in the specified columns
  - For clarity, in the SELECT clause, we can putt the table name before the column name (e.g. Staff.staffNo)
  - Also possible to use an alias for a table in the FROM clause (useful for distinguishing column names in case of ambiguity)
  - Alias is separated from table name with a space
- Usually the syntax is

  ```
  SELECT "list-of-columns"
  WHERE table1,table2,...
  WHERE "search conditions"
  ```

List the details of all clients who have viewed a property, along with any comment supplied

```
SELECT Client.ClientNo, Client.fname, Client.IName, Viewing.propertyNo, Viewing.comment
FROM Client, Viewing
WHERE Client.clientNo=Vieweing.clientNo
```

Using an alias:

```
SELECT c.clientNo, c.fName, c.IName, v.propertyNo, v.comment
FROM Client c, Vieweing v
WHERE c.clientNo=v.clientNo
```

This type of join is also known as a natural inner join:

- Keeps the rows that coincide in the specified columns (in the WHERE clause)
- Ignores all rows that do not meet the join conditions
- The most common type of Join

5.1 Three table Join

For each branch, list staff who manage properties, including the city in which branch is located and the properties they manage

```
SELECT b.branchNo, b.city, s.staffNo, s.fName, s.IName, p.propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo=s.branchNo AND s.staffNo=p.staffNo
ORDER BY b.branchNo, s.staffNo, p.propertyNo
```

An alternative formulation of this is

```
FROM (Branch b JOIN Staff s USING branchNo)
        JOIN PropertyForRent p USING staffNo
```

5.2 Inner Joins

- Instead of demanding the same column values in the matching columns we can demand different relations between the column values
  List all Durham-Staff who have salary 10% more than some staff member in London

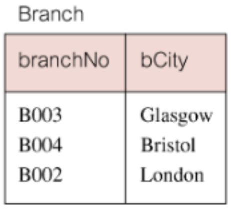
  ```
  SELECT dur.staffNo,dur.fName,dur.IName, dur.position, dur.salary
  FROM DurhamSaff dur, LondonStaff lon
  WHERE dur.salary>1.1*lon.salary
  ```

- This type of join is an inner join:
  - We add the term ”natural”, if we demand equality for the columns with the same name in the two tables (e.g. dur.salary=lon.salary)
  - Inner joins still ignore all rows that do not meet the join conditions

5.3 Outer Joins

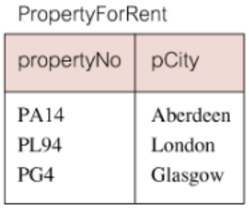
- Inner join: If one row of a table is unmatched, the row is omitted from the output table
- Outer join: It retains (some of) the rows that do not satisfy the join conditions
- Left outer join: It retains the rows of the left table that are unmatched with rows from the right table
- Right outer join: Retain the unmatched rows of the right table
- Full outer join: Retain the unmatched rows of both tables

5.3.1 Example



Branch

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

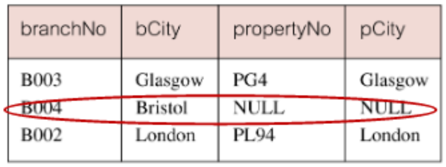


PropertyForRent

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Left outer Join
List the branch offices which have any properties that are in the same city

```
SELECT b.*,p.*
FROM Branch b LEFT JOIN PropertyForRent p
        ON b.bCity=p.pCity
```

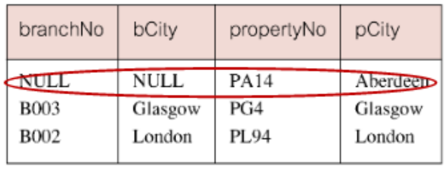


branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

- Includes the Bristol row of the left table unmatched with rows from the right table
- No rows corresponding to the properties in Aberdeen

Right outer join
List all properties and any branch offices that are in the same city

```
SELECT b.*,p.*
FROM Branch b RIGHT JOIN PropertyForRent p
        ON b.bCity=p.pCity
```

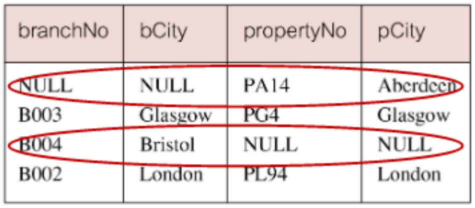


branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

- Includes the Aberdeen-row of the right table unmatched with rows from the left table
- No rows corresponding to branches in Bristol

Full outer join *List the branch offices and properties that are in the same city, along with any unmatched branches or properties*

```
SELECT b.*,p.*
FROM Branch b FULL JOIN PropertyForRent p
        ON b.bCity=p.pCity
```



branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

6 Database Updates

Three SQL statements for modifying the contents of the (existing) tables in the database

- INSERT: adds new rows of data into the table

```
INSERT INTO TableName [columnList]
VALUES (data ValueList)
```

- UPDATE: Modifies existing data in a table

```
UPDATE TableName
SET columnName1=dataValue1[,columnName2=dataValue2]
[WHERE searchCondition]
```

- DELETE: Removes rows of data from a table

```
DELETE FROM TableName
[WHERE searchCondition]
```

7 Data Definition Language Overview

Basic commands

- CREATE: create a new table
  - Assign a name to the table and define the names and domains of each of the columns in the table
- ALTER: Amend the relation schema (i.e. table structure)
  - If it is necessary to change the structure of a table because of design error, or just because the design has changed
- Specify integrity and referential constraints
  - PRIMARY KEY, FOREIGN KEY
- DROP: Delete a table.
- CREATE VIEW: define a virtual table
  - Virtual relation (table) that appears to the user
  - It is derived from a query on a ”real table”

8 Main domain types in SQL

CHAR(n): character string of fixed length n
VARCHAR(n):character string of variable length at most n
BIT(n): bit string of fixed length n
INTEGER: large positive/negative integer values
SMALLINT: small positive/negative integer values (up to 32767)
NUMERIC(p,d): a (positive/negative) decimal number with at most:

- Precision p (total number of all digits)
- Scale d: total number of decimal digits

9 Other domain types in SQL

We can define also our custom domain types specifically for out needs

Change name of a known type

```
CREATE DOMAIN Postcode AS VARCHAR(8);
```

With additional constraints

```
CREATE DOMAIN SexType AS CHAR(1)
        CHECK(VALUE IN ('M','F'));
```

More complex (nested) definitions

```
CREATE DOMAIN StaffNumber AS VARCHAR(5)
        CHECK(VALUE IN(SELECT staffNo
                        FROM Staff))
```

10 Constraints

- Referential actions when defining a table (i.e. for FOREIGN KEY)
  - ON UPDATE (what to do when the corresponding primary key is updated)
  - ON DELETE (what to do when the corresponding primary key is deleted)
- Available options for these actions
  - CASCADE (when update/delete: update the foreign key/delete the tuple)
  - SET NULL (when update/delete: set the foreign key to NULL)
  - SET DEFAULT (when update/delete: set the foreign key to the default value)
  - NO ACTION (when update/delete: do nothing - this is dangerous)

11 Create Table Construct

- An SQL relation is defined using the create table command:

  ```
  CREATE TABLE R(
          Attribute1 Domain1 [NOT NULL|UNIQUE],
          Attribute2 Domain2 [NOT NULL|UNIQUE],
          ...,
          Integrity & Referential constraints)
  ```

- A good strategy:
  - Before the `CREATE TABLE R(...)` it is always safe to write `DROP TABLE R`;
- Two options for `DROP TABLE R`
  - `RESTRICT` (the `DROP` is rejected if there are other objects that depend for their existence upon the continued existence of R)
  - `CASCADE` (default option: the `DROP` is proceeds anyway and SQL drops all dependant objects, and upon all dependent objects)

12 Defining Views

- View: a relation (table) that:
    - Depends on other relations and
    - Is not physically stored as a table
- Main use of views
    - For presenting different information to different users
    - Simplify complex queries

```
CREATE VIEW Developers AS
        SELECT name, project
        FROM Employee
        WHERE department='Development'
```