PREDICT 420

Atef Bader, PhD

Agenda

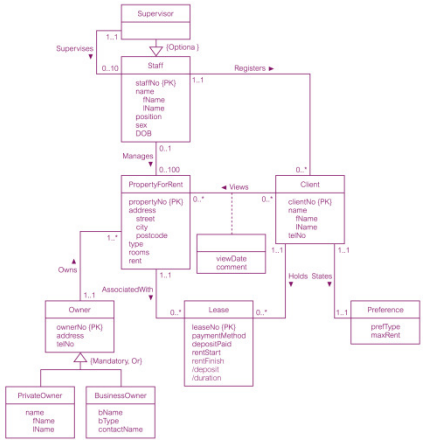
- SQL, RDBMS & Applications
- Joining Tables
- Assignment #2 Walkthrough & Deliverable
- Exercise #4 Walkthrough & Deliverable

RDBMS

How to build Database Application?

1

- ER Diagram
- UML



Conceptual Data Model

2

- Relations

Branch (branchNo, street, city, postcode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** postcode **Foreign Key** mgrStaffNo **references** Manager(staffNo)	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Staff (staffNo, fName, lName, position, sex, DOB, salary, supervisorStaffNo, branchNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Manager** (staffNo, mgrStartDate, bonus) **Primary Key** staffNo **Foreign Key** staffNo **references** Staff(staffNo)
PrivateOwner (ownerNo, fName, lName, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Viewing** (c ientNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent) **Primary Key** clientNo	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** clientNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, cl entNo, propertyNo) **Primary Key** easeNo **Alternate Key** propertyNo, rentStart **Alternate Key** c ientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** depos t (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	

Logical Data Model

3

- Tables
- SQL

Table 5.1 Result table for Example 5.1.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Implementation Model

ER diagram of Staff and Branch entities and their attributes

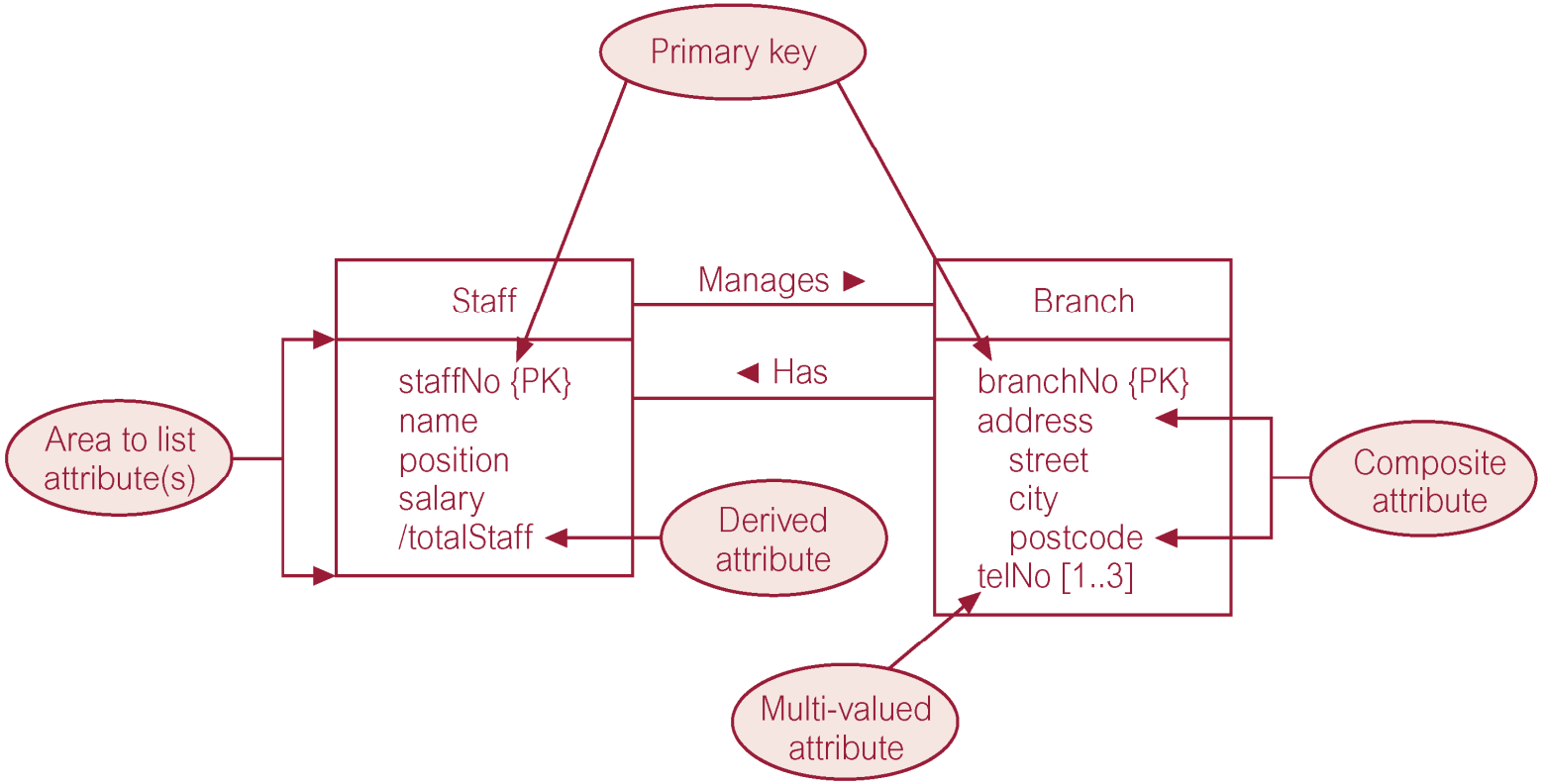
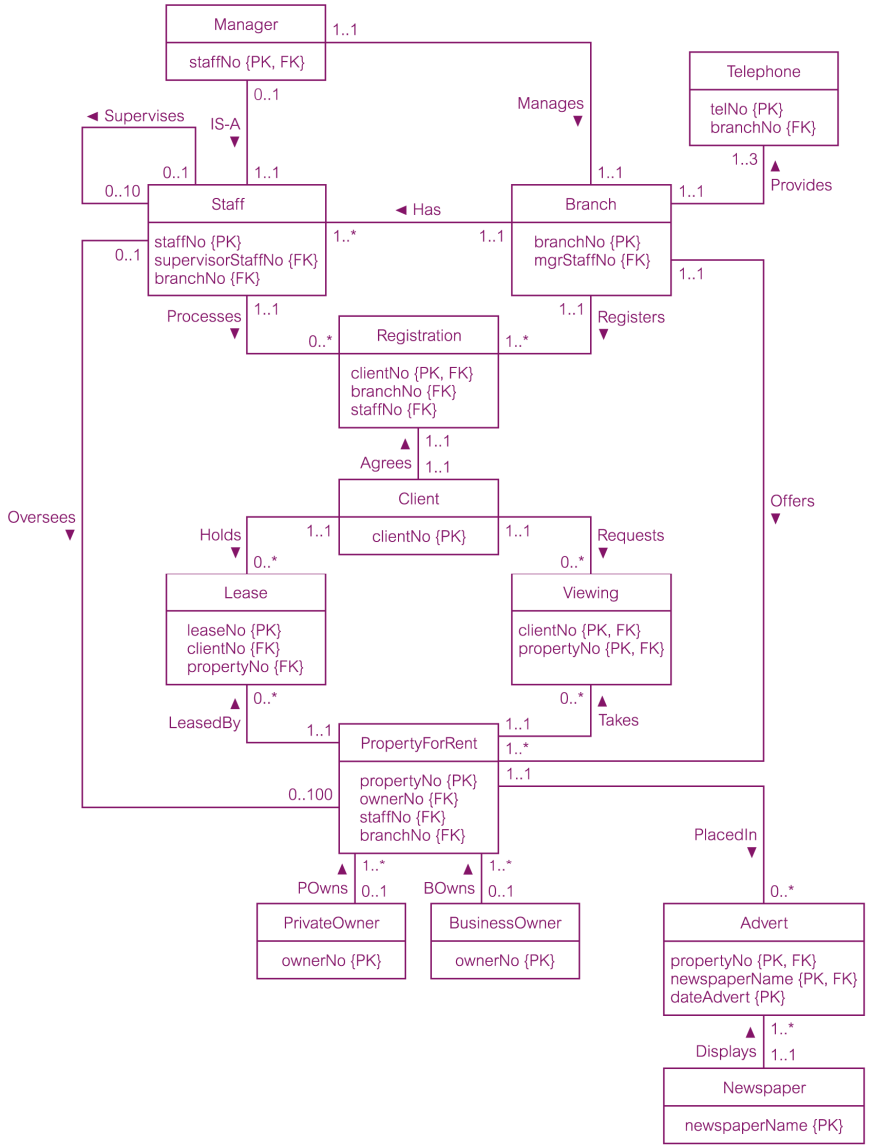


Table : All Columns, All Rows

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

ERD for *DreamHome*



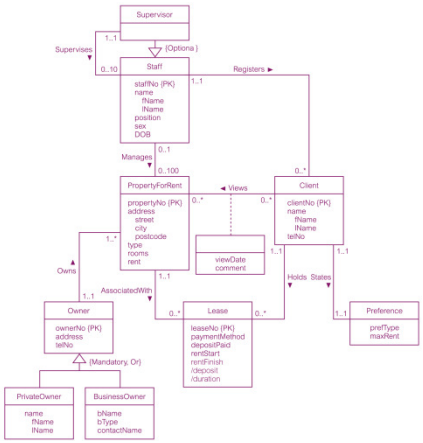
Global relation diagram for *DreamHome*

Branch (branchNo, street, city, postcode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** postcode **Foreign Key** mgrStaffNo **references** Manager(staffNo)	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Staff (staffNo, fName, lName, position, sex, DOB, salary, supervisorStaffNo, branchNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Manager** (staffNo, mgrStartDate, bonus) **Primary Key** staffNo **Foreign Key** staffNo **references** Staff(staffNo)
PrivateOwner (ownerNo, fName, lName, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Viewing** (c ientNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent) **Primary Key** clientNo	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** clientNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, cl entNo, propertyNo) **Primary Key** easeNo **Alternate Key** propertyNo, rentStart **Alternate Key** c ientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** depos t (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	

How to build Database Application?

1

- ER Diagram
- UML



Conceptual Data Model

2

- Relations

Branch (branchNo, street, city, postcode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** postcode **Foreign Key** mgrStaffNo **references** Manager(staffNo)	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Staff (staffNo, fName, lName, position, sex, DOB, salary, supervisorStaffNo, branchNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Manager** (staffNo, mgrStartDate, bonus) **Primary Key** staffNo **Foreign Key** staffNo **references** Staff(staffNo)
PrivateOwner (ownerNo, fName, lName, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Viewing** (c ieriNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent) **Primary Key** clientNo	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** clientNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, cl entNo, propertyNo) **Primary Key** easeNo **Alternate Key** propertyNo, rentStart **Alternate Key** c ientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** depos t (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(pnsperty No) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	

Logical Data Model

3

- Tables
- SQL

YOU ARE HERE

Table 5.1 Result table for Example 5.1.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Implementation Model

3

Implementation Model

SQL:
Data Manipulation and Data Definition

SQL & ISO

File Edit View History Bookmarks Tools Help

ISO/IEC 9075-11:2011 - In...

www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53685

Français | Русский Members area

Standards About us Standards Development News Store

Search ISO

Standards catalogue Handbooks and packages Checklists

ISO Store › Store › Standards catalogue › By TC › JTC 1 Information technology › SC 32

ISO/IEC 9075-11:2011

Information technology -- Database languages -- SQL -- Part 11: Information and Definition Schemas (SQL/Schemata)

(Only available in English)

Abstract

Preview ISO/IEC 9075-11:2011

ISO/IEC 9075 defines Structured Query Language (SQL). The scope of SQL is the definition of data structure and the operations on data stored in that structure. ISO/IEC 9075-1, ISO/IEC 9075-2 and ISO/IEC 9075-11 en-

FORMAT LANGUAGE

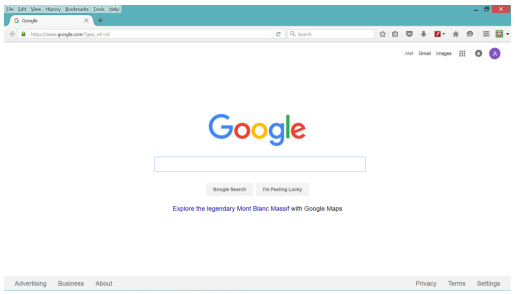
PDF English

CHF 198 Add to basket

What is common between SQL and Google?

Query
Database of Things

Query
Internet of Things



Objectives of SQL

- **Ideally, database language should allow user to:**
  1. **create the database and relation structures;**
  2. **perform insertion, modification, deletion of data from relations;**
  3. **perform simple and complex queries.**

Objectives of SQL

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.

- Until SQL:1999, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.

Objectives of SQL

- **Consists of standard English words:**

```
1) CREATE TABLE Staff(staffNo VARCHAR(5),
                      lName VARCHAR(15),
                      salary DECIMAL(7,2));

2) INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);

3) SELECT staffNo, lName, salary
   FROM Staff
   WHERE salary > 10000;
```

History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.

History of SQL

- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- In 1987, ANSI and ISO published an initial standard for SQL.
- In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released.

Writing SQL Commands

- **SQL statement consists of *reserved words* and *user-defined words*.**
  - **Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.**
  - **User-defined words are made up by user and represent names of various database objects such as relations, columns, views.**

SELECT Statement

```
SELECT [DISTINCT | ALL]
    {* | [columnExpression [AS newName]] [,...] }
FROM                TableName [alias] [, ...]
[WHERE    condition]
[GROUP BY           columnList]  [HAVING condition]
[ORDER BY           columnList]
```

SELECT Statement

FROM	**Specifies table(s) to be used.**
WHERE	**Filters rows.**
GROUP BY	**Forms groups of rows with same column value.**
HAVING	**Filters groups subject to some condition.**
SELECT	**Specifies which columns are to appear in output.**
ORDER BY	**Specifies the order of the output.**

Example : All Columns, All Rows

List full details of all staff.

```
SELECT staffNo, fName, lName, address,
       position, sex, DOB, salary, branchNo
FROM Staff;
```

- **Can use * as an abbreviation for 'all columns':**

```
SELECT *
FROM Staff;
```

Example : All Columns, All Rows

Table

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Example : Specific Columns, All Rows

Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff;
```

Example : Specific Columns, All Rows

Table

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Example : Use of DISTINCT

- **Use DISTINCT to eliminate duplicates:**

```
SELECT DISTINCT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG36

Example : Comparison Search Condition

List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
 FROM Staff
 WHERE salary > 10000;
```

Table

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Example : Set Membership

List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

Table

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Example : Pattern Matching

Find all owners with the string 'Glasgow' in their address.

```
SELECT ownerNo, fName, lName, address, telNo
 FROM PrivateOwner
 WHERE address LIKE '%Glasgow%';
```

Table

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

SELECT Statement - Aggregates

- ISO standard defines five aggregate functions:

  1. COUNT returns number of values in specified column.
  2. SUM returns sum of values in specified column.
  3. AVG returns average of values in specified column.
  4. MIN returns smallest value in specified column.
  5. MAX returns largest value in specified column.

Example : Use of COUNT(*)

How many properties cost more than $350 per month to rent?

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

myCount
5

Example : Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin,
    MAX(salary) AS myMax,
     AVG(salary) AS myAvg
   FROM Staff;
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT Statement - Grouping

- **Use GROUP BY clause to get sub-totals.**
- **SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:**
  - **column names**
  - **aggregate functions**
  - **constants**
  - **expression involving combinations of the above.**

Example : Use of GROUP BY

Find number of staff in each branch and their total salaries.

```
SELECT      branchNo,
            COUNT(staffNo) AS myCount,
            SUM(salary) AS mySum
FROM        Staff
GROUP BY    branchNo
ORDER BY    branchNo;
```

Example : Use of GROUP BY

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Restricted Groupings – HAVING clause

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

Example : Use of HAVING

For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT      branchNo,
            COUNT(staffNo) AS myCount,
            SUM(salary) AS mySum
FROM        Staff
GROUP BY    branchNo
HAVING COUNT        (staffNo) > 1
ORDER BY    branchNo;
```

Example : Use of HAVING

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

- **Some SQL statements can have a SELECT embedded within them.**
- **A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a *subquery* or *nested query*.**
- **Subselects may also appear in INSERT, UPDATE, and DELETE statements.**

Example : Subquery with Equality

List staff who work in branch at '163 Main St'.

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

EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries.
- Produce a simple true/false result.
- True if and only if there exists at least one row in result table returned by subquery.
- False if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.

Example : Query using EXISTS

Find all staff who work in a London branch.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
  (SELECT *
   FROM Branch b
   WHERE s.branchNo = b.branchNo AND
                        city = 'London');
```

Example : Query using EXISTS

Table

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Joining Tables

Tables Join

- A SQL join clause combines records from two or more tables in a relational database.
- It creates a set that can be saved as a table or used as it is.
- A JOIN is a means for combining fields from two tables (or more) by using values common to each.

Tables Join

- ANSI-standard SQL specifies different types of JOIN, for example: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS.

- As a special case, a table (base table, view, or joined table) can JOIN to itself in a self-join.

Tables Join

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Cross Join

- CROSS JOIN returns the Cartesian product of rows from tables in the join.
- It will produce rows which combine each row from the first table with each row from the second table

Cross Join

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Rafferty	31	Sales	31
Jones	33	Sales	31
Heisenberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
Williams	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Heisenberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
Williams	NULL	Engineering	33
Rafferty	31	Clerical	34
Jones	33	Clerical	34
Heisenberg	33	Clerical	34
Smith	34	Clerical	34
Robinson	34	Clerical	34
Williams	NULL	Clerical	34
Rafferty	31	Marketing	35
Jones	33	Marketing	35
Heisenberg	33	Marketing	35
Smith	34	Marketing	35
Robinson	34	Marketing	35
Williams	NULL	Marketing	35

Natural Join

- Natural join is a binary operation between two tables/relations, R and S.

- The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names

Natural Join

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Inner Join

- **An inner join requires each record in the two joined tables to have matching records**
- **Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate.**
  - SELECT *
  - FROM employee
  - INNER JOIN department ON employee.DepartmentID = department.DepartmentID;

Tables Join

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31

Equi Join

- An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

- An example of an equi-join:

  - SELECT *
  - FROM employee JOIN department
  - ON employee.DepartmentID = department.DepartmentID;

Natural Join

- A natural join is a type of equi-join where the join predicate arises implicitly by comparing all columns in both tables that have the same column-names in the joined tables.

- The resulting joined table contains only one column for each pair of equally named columns.

LEFT OUTER JOIN

- The result of a left outer join (or simply left join) for tables A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B).

- This means that if the ON clause matches 0 (zero) records in B (for a given record in A), the join will still return a row in the result (for that record)—but with NULL in each column from B. A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with NULL (empty) values in the link field.

LEFT OUTER JOIN

- For example, this allows us to find an employee's department, but still shows the employee(s) even when they have not been assigned to a department

```
SELECT *
FROM employee
LEFT OUTER JOIN department ON employee.DepartmentID =
department.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33

RIGHT OUTER JOIN

- A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B.

- A right outer join returns all the values from the right table and matched values from the left table (NULL in the case of no matching join predicate).

- For example, this allows us to find each employee and his or her department, but still show departments that have no employees.

RIGHT OUTER JOIN

```
SELECT *
FROM employee RIGHT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

SQL Tutorial

File Edit View History Bookmarks Tools Help

SQL Tutorial

www.w3schools.com/sql/default.asp

w3schools.com THE WORLD'S LARGEST WEB DEVELOPER SITE

HTML CSS JAVASCRIPT SQL PHP BOOTSTRAP TUTORIALS REFERENCES

SQL Tutorial

- SQL HOME
- SQL Intro
- SQL Syntax
- SQL Select
- SQL Distinct
- SQL Where
- SQL And & Or
- SQL Order By
- SQL Insert Into
- SQL Update
- SQL Delete
- SQL Injection
- SQL Select Top
- SQL Like
- SQL Wildcards
- SQL In
- SQL Between

SQL Tutorial

« W3Schools Home Next Chapter »

SQL is a standard language for accessing databases.

Our SQL tutorial will teach you how to use SQL to access and manipulate data in: MySQL, SQL Server, Access, Oracle, Sybase, DB2, and other database systems.

Examples in Each Chapter

With our online SQL editor, you can edit the SQL statements, and click on a button to view the result.

Assignment #2

Assignment #2

Assignment 2 Instructions & Deliverable

Assignment 2.pdf - Adobe Acrobat Re

File Edit View Window Help

Home Tools Document 1 / 1 Sign In

PREDICT 420

Individual Assignment 2:
Accessing and Manipulating Relational Data

The business analysts are working on a model for customer lifetime value (CLV) analysis. They are looking at total revenues and costs associated with each customer. This initial request is for data for a prototype CLV model. The data we need are on the PostgreSQL database server, and you can access them by using your NetID and password. Hook up via the VPN and SSCC to maintain security.

Analysts need data from the **mail**, and **customer** tables under the **pilot** schema for the **xyz** database. To work on the prototype for the CLV model, analysts would like a full set of records for a sample of 100 customers—you can take the first 100 customers in the **customer** table. Any joins can be accomplished using the **acctno** columns of the tables. There should be one record per account number in the **customer** table. (There can be multiple records for each customer in the **item** table, which represents sales transactions, but we do not need to work with the item table for the CLV prototype.) What the analysts need is a single record for each customer that has the following columns/fields from the **customer** table: **acctno**, **ltd_sales**, **ltd_transactions**, **ytd_sales_2009**, and **ytd_transactions_2009**. That single customer record should also have all the fields from the **mail** table (one field from each of the sixteen mailing dates). But if there is no record in the **mail** table, we still need to get the customer record—that would be a customer who placed orders without getting any promotional mailings in 2009. You can use SQL syntax under psql to join the database tables. After you have the merged data, put them in a comma-delimited text file, something our business analysts can read into their CLV programs in Python, R, and Excel.

After you get the data on the SSCC, you can FTP the comma-delimited text file down to your personal computer before sending the work off to me. We do not care how you do the file transfer from the server to your personal computer—direct sftp may be the easiest, but some like Filezilla.

Deliverable: Submit the CSV file on canvas.

Assignment #2

This file has the set of steps that you could use to copy & paste the commands

Step-by-step

SQL Commands - Notepad

File Edit Format View Help

```
NOTES:

- In the examples of SQL statments below I used my netid ajb254, and you need to change that to your NETID

- If you want to RERUN the create view commands you will get the error message that view is already created. Here is an example fo the error message:

xyz=> CREATE TEMP VIEW ajb254rightview AS SELECT * FROM pilot.mail;
ERROR:  relation "ajb254rightview" already exists


- So you need to clean up before you create the view again, and you clean up by dropping the temp view

xyz=> drop view ajb254leftview;
DROP VIEW
xyz=> drop view ajb254rightview;
DROP VIEW
xyz=> drop view ajb254join;
```

```
Step 1: Connect dornick


Step 2: Connect to postgree database server:

psql -h 129.105.208.226 -U ajb254 -d postgres


Step 3: Connect databse instance
```

Assignment #2- Deliverable

- **After you execute all SQL commands and ftp the CSV file from dornick server**
- **Submit the CSV file on canvas.**

Exercise #4

Exercise #4– Ipython Notebook Script

Run Ipython Notebook Script

Editor - Canopy

File Edit View Search Run Tools Window Help

File Browser

Filter: All Supported Files

- Atef
- Recent Files

Exercise4.ipynb

Edit Insert Cell Kernel Help

Heading 1 Cell Toolbar: None

Week #4 - Python Practice

Where we left off last session was with having read from and written to a simple SQLite3 RDB. This DB, 'xyz.db,' started out with having one table of XYZ customer purchase transaction records in it, 'xyztrans.' You wrote a second table to this DB with customer information called 'xyzcust.'

It turns out that XYZ has purchased some "enhancement data" on its customers from Experian. Enhancement data are often used by retailers, direct marketers, and others to better understand their customers and for marketing campaign targeting. In what follows we're going to read XYZ's enhancement data and merge it with the XYZ customer data you've previously worked with.

So, let's launch a Canopy IPython session, and import into it pandas (as pd), numpy (as np), and SQLAlchemy. Import from pandas DataFrame and Series. Finally, read into your session the customer information data that's in the xyzcust table in the SQLite DB xyz.db. (You did save it, right?) Put it into a DataFrame called xyzcust.

XYZ's Experian data are in a json (javascript object notation) file called zdata.json. This file is available on Canvas. Put it it your default working directory to make your life easier.

Here's what the first customer record looks like in this zdata.json file:

'{"0":
{"ACCTNO":"PHSWPGWPQ","ZCREDIT":"Y","ZCRAFTS":"U","ZGOURMET":"U","ZCOMPUTR":"Y","ZHITECH":"Y","ZONLINE":"Y","ZSPENDER":"U","ZGOLFE
RS":"U","ZGOLFERP":5.0,"ZDONORS":"U","ZDONORSP":2.0,"ZPETS":"U","ZPETSP":8.0,"ZARTS":"U","ZARTSP":6.0,"ZMOB":"Y","ZMOBP":0.0,"ZFIT
NESS":"U","ZFITNESP":5.0,"ZOUTDOOR":"U","ZOUTDOOP":5.0,"ZTRAVANY":"U","ZTRAVANP":7.0,"ZINVESTR":"U","ZINVESTP":5.0,"ZAUTOOWN":"U",
"ZAUTOOWP":6.0,"ZGARDEN":"U","ZGARDENP":3.0,"ZCOLLECT":"U","ZCOLLECP":7.0,"ZCRUISE":"U","ZCRUISEP":7.0,"ZSPORTS":"Y","ZSPORTSP":0.
0,"ZSWEEPS":"U","ZSWEEPSP":5.0,"ZPOLITIC":"U","ZPOLITIP":2.0,"ZMUSIC":"U","ZMUSICP":8.0,"ZREAD":"U","ZREADP":4.0,"ZCHLDPRD":"Y","Z
CHLDPRP":0.0,"ZDIY":"U","ZDIYP":5.0,"ZSELFIMP":"U","ZSELFIPP":8.0,"ZRELIGON":"U","ZRELIGOP":4.0,"ZGRANDPR":"U","ZGRANDPP":4.0,"ZCL
OTHNG":"Y","ZCLOTHNP":0.0,"ZDONENVR":"U","ZDONENVP":6.0,"ZMUTUAL":"U","ZMUTUALP":6.0,"ZWGHTCON":"U","ZWGHTCOP":7.0,"ZPRCHPHN":"U",
"ZPRCHPHP":8.0,"ZPRCHTV":"U","ZPRCHTVP":8.0,"ZMOBMULT":"Y","ZMOBMULP":0.0,"ZCREDPLP":6.0,"ZCREDPLT":"U","ZDOGS":"U","ZDOGSP":7.0,"
ZCATS":"U","ZCATSP":7.0,"ZHEALTH":"U","ZHEALTHP":3.0,"ZAUTOINT":"U","ZAUTOINP":8.0,"ZSKIING":"U","ZSKIINGP":8.0,"ZASTRLGY":"U","ZA
STRLGP":9.0,"ZBOATS":"U","ZBOATSP":7.0,"ZCELL":"U","ZCELLP":8.0,"ZCOMMCON":"U","ZCOMMCOP":8.0,"ZHMDECOR":"Y","ZHMDECOP":0.0,"ZHOME
ENT":"U","ZHOMEENP":8.0,"ZKITCHEN":"U","ZKITCHEP":1.0,"ZMOBAV":"U","ZMOBAVP":9.0,"ZMOBBOOK":"Y","ZMOBBOOP":0.0,"ZMOBCLTH":"Y","ZMO
BCLTP":0.0,"ZMOBFIN":"U","ZMOBFINP":3.0,"ZMOBGIFT":"U","ZMOBGIFP":6.0,"ZMOBGRDN":"U","ZMOBGRDP":2.0,"ZMOBJWL":"U","ZMOBJWLP":8.0,"
ZMUSCLAS":"U","ZMUSCLAP":6.0,"ZMUSCNTR":"U","ZMUSCNTP":5.0,"ZMUSCRST":"U","ZMUSCRSP":6.0,"ZMUSOLDI":"U","ZMUSOLDP":7.0,"ZMUSROCK":
"U","ZMUSROCP":9.0,"ZPBCARE":"U","ZPBCAREP":5.0,"ZPHOTO":"U","ZPHOTOP":8.0,"ZPRCHONL":"Y","ZPRCHONP":0.0,"ZTENNIS":"U","ZTENNISP":
9.0,"ZTRAVDOM":"U","ZTRAVDOP":6.0,"ZTRAVFOR":"U","ZTRAVFOP":6.0,"ZVOLUNTR":"U","ZVOLUNTP":6.0}}',"ZMUSCLAS":"U","ZMUSCLAP":6.0,"ZM
USCNTR":"U","ZMUSCNTP":5.0,"ZMUSCRST":"U","ZMUSCRSP":6.0,"ZMUSOLDI":"U","ZMUSOLDP":7.0,"ZMUSROCK":"U","ZMUSROCP":9.0,"ZPBCARE":"U"
,"ZPBCAREP":5.0,"ZPHOTO":"U","ZPHOTOP":8.0,"ZPRCHONL":"Y","ZPRCHONP":0.0,"ZTENNIS":"U","ZTENNISP":9.0,"ZTRAVDOM":"U","ZTRAVDOP":6.
0,"ZTRAVFOR":"U","ZTRAVFOP":6.0,"ZVOLUNTR":"U","ZVOLUNTP":6.0}}'

Python C:\Users\Atef

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]:
```

C:\bader\nu\420\Sync Sessions\Sync Session 4\ExercisePractice4\Exercise4.ipynb

Exercise #4– Ipython Notebook Script

Write your python code snippet to answer this question

File Edit View Search Run Tools Window Help

Exercise4.ipynb

Edit Insert Cell Kernel Help

Heading 1 | Cell Toolbar: None

10938	SGGLWWDHQ	U	1	8	U	7	U	U	4	U	...	0	Y
10989	LDSSWDQAS	U	2	8	U	6	U	U	6	U	...	1	U
11020	AQHWWGYWL	U	1	9	U	8	U	U	3	U	...	0	Y
11071	GHDDAHQWA	U	2	9	U	8	U	U	3	U	...	2	U
11089	ADWYQAQPD	U	3	3	U	0	Y	U	8	U	...	7	U
11112	GGSHAHPWQ	U	2	9	U	8	U	U	3	U	...	0	Y
11158	GQGLHAGD	U	5	8	U	7	U	U	4	U	...	2	U
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8649	SLGAAQLGG	U	4	8	U	7	U	U	3	U	...	2	U
8705	GOSHPSHY	U	3	8	U	7	U	U	5	U	...	1	U

zGhostCust should be a DataFrame with as many rows as the set zNotC has items. Save custZData and zGhostCust in the format(s) of your choice for possible use in a later Session. You might want to poke custZData into your RDB xyz.db. Note that it can be useful to have made copies of your RDB and other data files from time to time as backups. SQLite3 DB's are usually small enough for this to be convenient. Pickles can be little or big.

Here's a last question. Suppose we wanted to extract from zdata all the records that do have a match on ACCTNO in xyzcust. We already have them, of course, but is there a complement to the Boolean row selection above that would get them directly out of zdata? (Hint: Remember what the twiddle, "~", can be used for in Pandas methods calls?)

Deliverable :

What is the total number of the Ghost Customers in the data? Write and execute Python code snippet to get the total number of the Ghost Customers

In [ ]:

~\AppData\Local\Temp\Temp2_Sync Session 4.zip\ExercisePractice4\Exercise4.ipynb

Exercise #4 - Deliverable

- Submit your Ipython Notebook Script
- Show the Python code snippet you wrote
- Show the total number of Ghost Customers after you ran you script