

Basic SQL

Dr Paolo Guagliardo



THE UNIVERSITY *of* EDINBURGH
informatics

v26.0

Changelog

v26.0 Initial version

The data model of SQL

Data is organised in **tables** (also called **relations**)
which are collections of **tuples** (also called **rows** or **records**)
which are all of the same length

Schema

- ▶ Set of **table names**
- ▶ List of typed distinct **column names**
(also called **attributes**) for each table
- ▶ **Constraints** within a table or between tables ⇐ not for now

Instance

- ▶ Actual data (that is, the rows of the table)
- ▶ Must satisfy typing and constraints

SQL

- ▶ **Structured Query Language**
- ▶ **Declarative** language for relational databases
- ▶ Implemented in all major (free and commercial) RDBMSs
- ▶ International Standard since 1987 (latest rev. June 2023)
- ▶ Consists of two sublanguages:
 - DDL** (Data Definition Language)
operations on the schema
 - DML** (Data Manipulation Language)
operations on the instance

Creating tables

Basic syntax

```
CREATE TABLE table_name (  
    column_name1 column_type1,  
    column_name2 column_type2,  
    ⋮  
    column_namen column_typen  
) ;
```

Example

```
CREATE TABLE Customer (  
    custid varchar(10),  
    name varchar(20),  
    city varchar(30),  
    address varchar(30)  
) ;
```

Most common SQL data types

Strings

- ▶ **varchar** (*n*) – variable length, at most *n* characters

Numbers

- ▶ **smallint**
- ▶ **integer** or **int**
- ▶ **bigint**
- ▶ **numeric** (*p*, *s*) – arbitrary precision number
At most *p* total digits with *s* digits in the fractional part

Date & Time

- ▶ **date** – e.g., '2016-10-03'
- ▶ **time** – time of the day: e.g., '21:09'
- ▶ **timestamp**

Default values

Syntax

```
CREATE TABLE table_name (  
    column_name1 column_type1 [DEFAULT value1],  
    column_name2 column_type2 [DEFAULT value2],  
    ⋮  
    column_namen column_typen [DEFAULT valuen]  
);
```

Example

```
CREATE TABLE Account (  
    accnum varchar(12),  
    branch varchar(30),  
    custid varchar(10),  
    balance numeric(14,2) DEFAULT 0  
);
```

Populating tables

General syntax

```
INSERT INTO table_name VALUES (...), ..., (...);
```

Examples

```
INSERT INTO Account VALUES  
    ('243576', 'Edinburgh', 'cust1', -120);
```

```
INSERT INTO Customer VALUES  
    ('cust1', 'Renton', 'Edinburgh', '2 Wellington Pl'),  
    ('cust2', 'Watson', 'London', '221B Baker St'),  
    ('cust3', 'Holmes', 'London', '221B Baker St');
```

Populating tables with default values

Two possibilities:

1. Use the keyword **DEFAULT** in **INSERT**

Example

```
INSERT INTO Account VALUES  
('250018', 'London', 'cust3', DEFAULT);
```

2. List attributes explicitly (omitted ones will get the default)

Example

```
INSERT INTO Account (accnum, branch, custid) VALUES  
('250018', 'London', 'cust3');
```

Attributes without **DEFAULT** in **CREATE TABLE**
have default value **NULL** $\Leftarrow$ more on this later

Changing the definition of a table

```
ALTER TABLE name  
    RENAME TO new_name ;  
    RENAME column TO new_column ;  
    ADD column type ;  
    DROP column ;  
    ALTER column  
        TYPE type ;  
        SET DEFAULT value ;  
        DROP DEFAULT;
```

Destroying tables

TRUNCATE TABLE name;	delete all rows from the table
DROP TABLE name;	completely remove table from schema

Many other changes are possible ...

Basic queries in SQL

Follow the basic pattern:

```
SELECT  <comma-separated list of value expressions>  
FROM    <comma-separated list of tables>  
WHERE   <condition> ;
```

Idea

1. Loop over all rows of the tables listed in **FROM**
2. Take those that satisfy the **WHERE** condition
3. Compute and output the values listed in **SELECT**

An extremely simple example

Customer			
ID	Name	City	Address
cust1	Renton	Edinburgh	2 Wellington Pl
cust2	Watson	London	221B Baker St
cust3	Holmes	London	221B Baker St

List all customers

```
SELECT  *  
FROM    Customer ;
```

* means “all attributes”

What is the output to this query?

A very simple example

Customer			
ID	Name	City	Address
cust1	Renton	Edinburgh	2 Wellington Pl
cust2	Watson	London	221B Baker St
cust3	Holmes	London	221B Baker St

List name and address of all customers

```
SELECT Name, Address
FROM Customer ;
```

Output:

Name	Address
Renton	2 Wellington Pl
Watson	221B Baker St
Holmes	221B Baker St

A simple example

Customer			
ID	Name	City	Address
cust1	Renton	Edinburgh	2 Wellington Pl
cust2	Watson	London	221B Baker St
cust3	Holmes	London	221B Baker St

List name and address of customers living in Edinburgh

```
SELECT Name, Address
FROM Customer
WHERE City = 'Edinburgh' ;
```

Output:

Name	Address
Renton	2 Wellington Pl

More than one table in FROM

Table1	
A	B
1	2
3	4
1	3

Table2		
C	D	E
2	1	0
3	2	1

```
SELECT B, C
FROM Table1, Table2 ;
```

1. Each row of Table1 is **concatenated** with every row of Table2

A	B	C	D	E
1	2	2	1	0
1	2	3	2	1
3	4	2	1	0
3	4	3	2	1
1	3	2	1	0
1	3	3	2	1

More than one table in FROM

Table1	
A	B
1	2
3	4
1	3

Table2		
C	D	E
2	1	0
3	2	1

```
SELECT B, C
FROM Table1, Table2 ;
```

2. For each resulting row, the values of B and C are returned

B	C
2	2
4	2
3	2
2	3
4	3
3	3

Joining tables

Customer			Account		
ID	Name	City	AccNum	CustID	Balance
cust1	Renton	Edinburgh	123321	cust3	1330.00
cust2	Watson	London	243576	cust1	-120.00
cust3	Holmes	London			

List customers' names and their accounts' numbers

```
SELECT Name, AccNum
FROM Customer, Account
WHERE ID = CustID ;
```

Semantics: nested loop over the tables listed in FROM

Output:	Name	AccNum
	Renton	243576
	Holmes	123321

The basic WHERE clause

term :=

- | attribute
- | value

comparison :=

- | term op term, with $op \in \{=, <>, <, >, <=, >=\}$
- | term **IS NULL**
- | term **IS NOT NULL**

condition :=

- | comparison
- | condition **AND** condition
- | condition **OR** condition
- | **NOT** condition

Database modification: Deletion

General form

```
DELETE FROM table_name  
WHERE <condition>;
```

All rows in *table_name* satisfying *<condition>* are deleted

Example

Remove accounts with zero balance and unknown owner

```
DELETE FROM Account  
WHERE Balance = 0 AND CustID IS NULL ;
```

Database modification: Replacement

General form

```
UPDATE table_name  
SET      <assignments>  
WHERE    <condition> ;
```

Replace the values of some attributes (using *<assignments>*)
in each row of *table_name* that satisfies *<condition>*

Examples

Set a new balance on account 745622

```
UPDATE Account  
SET    balance = 1503.82  
WHERE   accnum = '745622' ;
```

Accounts in London with positive balance get a 0.2% bonus

```
UPDATE Account  
SET    balance = balance + 0.002 * balance  
WHERE   branch = 'London' AND balance > 0 ;
```

WHERE conditions in queries

- **filter** data within a table

```
SELECT Name, Address
FROM Customer
WHERE City = 'Edinburgh' ;
```

- **join** data from different tables

```
SELECT Name, AccNum
FROM Customer, Account
WHERE ID = CustID ;
```

Filtering and join together

```
SELECT Name, Address, AccNum
FROM Customer, Account
WHERE ID = CustID AND City = 'Edinburgh' ;
```

Explicit join syntax

```
table1 JOIN table2 ON <condition>1
      :
      JOIN tablen ON <condition>n-1
```

Logically separate join conditions from filters

```
SELECT Name, Balance
FROM Customer, Account
WHERE ID = CustID AND Balance < 0 ;
```

```
SELECT Name, Balance
FROM Customer JOIN Account ON ID=CustID
WHERE Balance < 0 ;
```

Qualification of attributes

Customer			Account		
CustID	Name	City	AccNum	CustID	Balance
cust1	Renton	Edinburgh	123321	cust3	1330.00
cust2	Watson	London	243576	cust1	-120.00
cust3	Holmes	London			

List the name of customers whose account is overdrawn

```
SELECT Customer.Name
FROM Customer, Account
WHERE Account.CustID = Customer.CustID
      AND Account.Balance < 0 ;
```

We need to specify the relations attributes are coming from

What is the output of this query?

Range variables

Assign new names to tables in FROM

```
SELECT Customer.Name, Account.Balance
FROM Customer, Account
WHERE Account.CustID = Customer.CustID
      AND Account.Balance < 0 ;
```

```
SELECT C.Name, A.Balance
FROM Customer C, Account AS A
WHERE A.CustID = C.CustID
      AND A.Balance < 0 ;
```

```
SELECT C.Name, A.Balance
FROM Customer C JOIN Account A ON C.CustID=A.CustID
WHERE A.Balance < 0 ;
```

Renaming attributes

```
SELECT C.Name CustName, A.Balance AS AccBal
FROM    Customer C, Account A
WHERE   A.CustID = C.CustID
          AND A.Balance < 0 ;
```

This does not work:

```
SELECT C.Name CustName, A.Balance AS AccBal
FROM    Customer C, Account A
WHERE   A.CustID = C.CustID
          AND AccBal < 0 ;
```

Concluding remarks

- ▶ SQL is case-insensitive (for keywords and table/column names)
but **string constants are case-sensitive**: 'abc' $\neq$ 'aBc'
- ▶ (SQL) queries are read-only
they do **not modify** the **schema**
nor the **instance** of the database
- ▶ Always use range variables (aliases for tables)
and fully qualify references to attributes
 - ⇒ improves **readability** of queries
 - ⇒ more **robust** against schema changes