Database Concepts (III)

Structured Query Language

Chaokun Wang

School of Software, Tsinghua University

chaokun@tsinghua.edu.cn

March 15, 2021

Outline

- Introduction to SQL
- Data Manipulation Language*
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- Data Definition Language*
  - Data Types
  - Schema
  - Table
  - Index
  - View
  - Transaction

Database Language

e.g.增删改查

- Create the database and relation structures;
- Perform basic data management tasks
  - insertion, modification, and deletion of data;
- Perform queries
  - Simple & complex

Structured Query Language

- SQL: Structured Query Language
  - Initially called Structured English Query Language, or SEQUEL
    - Defined by Donald D. Chamberlin and Raymond F. Boyce at IBM's Research Laboratory in 1974
    - Later named "SQL"
  - Standardized by ANSI and ISO
    - ISO 9075:1987, 1989, ***1992***, ***1999***, 2003, 2008, 2011, 2016, 2019

      SQL2 SQL3
  - Extension & *Dialect*
    - No two dialects are exactly alike
    - No dialect exactly matches the ISO standard

Structured Query Language

- Non-procedural language
  - ***What*** information you require, rather than ***how*** to get it
- Free-format
- Consists of standard English words
  - CREATE TABLE Staff (staffNo VARCHAR(5), lName VARCHAR(15), salary DECIMAL(7,2));
  - INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
  - SELECT staffNo, lName, salary FROM Staff WHERE salary > 10000

4th generation programming language

Writing SQL Commands

- SQL statement
  - Consists of *reserved* words and *user-defined* words 自带的（保留字）
  - Case **insensitive** for most of the components
  - Usually use the semicolon ';' to mark the end of each SQL statement 不加也能跑，但是加了舒服hhhh
    - Not required by ISO, but by many dialects of SQL
- Categories
  - DDL: Data Definition Language (Table 7.2)
  - DML: Data Manipulation Language (Table 7.1) 增删改查
  - *DCL: Data control language* (Table 7.3)
  - *TCL: Transaction control language* (Table 7.3)

Our Notations to Define SQL Statements

一些个语法和规则

- Upper-case letters: **reserved** words
- Lower-case letters: **user-defined** words
- A vertical bar (|): **choice** among alternatives
  - e.g. DATABASE | SCHEMA
- Curly braces: **required element**
  - e.g. CREATE {DATABASE | SCHEMA}
- Square brackets: **optional element**
  - e.g. CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
- Ellipsis (. . .): **optional repetition** of an item zero or more times

```
CREATE DATABASE name
  [ [ WITH ] [ OWNER [=] user_name ]
    [ TEMPLATE [=] template ]
    [ ENCODING [=] encoding ]
    [ LC_COLLATE [=] lc_collate ]
    [ LC_CTYPE [=] lc_ctype ]
    [ TABLESPACE [=] tablespace_name ]
    [ ALLOW_CONNECTIONS [=] allowconn ]
    [ CONNECTION LIMIT [=] connlimit ]
    [ IS_TEMPLATE [=] istemplate ] ]
```

CREATE DATABASES demodb ✗

CREATE demodb ✗

CREATE DATABASE demodb
  WITH
  OWNER = postgres
  ENCODING = 'UTF8' ✓

Outline

- Introduction to SQL
- Data Manipulation Language*
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- Data Definition Language*
  - Data Types
  - Schema
  - Table
  - Index
  - View
  - Transaction

Data Manipulation

- SELECT
  - To query data in the database;
- INSERT
  - To insert data into a table;
- UPDATE
  - To update data in a table;
- DELETE
  - To delete data from a table.

SELECT

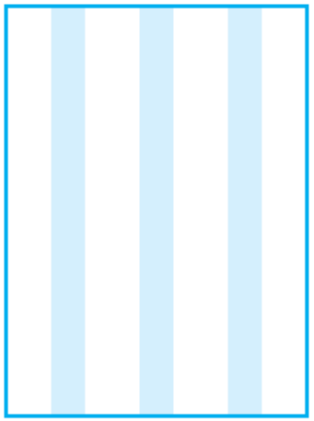
film(film_id, title, description, release_year, language_id, original_language_id, rental_duration, rental_rate, length, replacement_cost, rating, special_features, last_update)
actor(actor_id, first_name, last_name, last_update)
film_actor(actor_id, film_id, last_update)
store(store_id, manager_staff_id, address_id, last_update)
inventory(inventory_id, film_id, store_id, last_update)
rental(rental_id, rental_date, inventory_id, customer_id, return_date, staff_id)

it's mostly about knowing the schema of your data

1. List the title and rental rate for all the films
2. Show the rental rate and duration for a specified film
3. List the films that cost less than $1.00 for rental
4. Rank the actors by the number of films he/she starred
5. List the films starred by a specified actor
6. List the films starred by a specified actor that are available for rental

Relational Algebra: Projection

$\Pi_{a_1, \ldots, a_n}(R)$ The Projection operation works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.



Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

$\Pi_{\text{staffNo, fName, lName, salary}}(\text{Staff})$

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

SQL: Select

- **SELECT** staffNo, fName, lName, position, sex, DOB, salary, branchNo
  FROM Staff; 出来的结果是表格自身
- **SELECT** * 同上
  FROM Staff;
- **SELECT** staffNo, fName, lName, salary →
  FROM Staff;
- **SELECT** branchNo 选出来一列
  FROM Staff;
- **SELECT DISTINCT** branchNo 选出来不重复的一列
  FROM Staff;

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

branchNo
B005
B003
B003
B007
B003
B005

branchNo
B005
B003
B007

SQL: Select

！SELECT后面的内容是可计算的
which means你想写一个SELECT1+1都可以

- **SELECT** staffNo, fName, lName, salary/12
  FROM Staff;

找出来月薪，并给他起个新名字

- **SELECT** staffNo, fName, lName, salary/12 **AS** mSalary
  FROM Staff;

staffNo	fName	lName	mSalary
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

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

用where筛掉一些东西

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000
SG37	Ann	Beech	Assistant	12000
SG14	David	Ford	Supervisor	18000
SG5	Susan	Brand	Manager	24000

Relational Algebra: Selection/Restriction

$\sigma_{predicate}(R)$ The Selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (*predicate*).

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

$\sigma_{salary > 10000}(Staff)$

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

SQL: Select: WHERE clause

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

- Search conditions
  - Comparison
    - Compare the value of one expression to the value of another expression.
  - Range
    - Test whether the value of an expression falls within a specified range of values.
  - Set membership
    - Test whether the value of an expression equals one of a set of values.
  - Pattern match 字符串比较
    - Test whether a string matches a specified pattern
  - Null
    - Test whether a column has a null (unknown) value

SQL: Select: WHERE clause

- Comparison
  - Comparison operators

=	equals		
< >	is not equal to (ISO standard)	! =	is not equal to (allowed in some dialects)
<	is less than	< =	is less than or equal to
>	is greater than	> =	is greater than or equal to

  - Logical operators
    - AND, OR, NOT
  - Priority
    - () > Comparison operators > NOT > AND > OR
    - Left > Right
    - Parentheses is always recommended

SQL: Select: WHERE clause

AN EXAMPLE

- **SELECT** staffNo, fName, lName, salary **FROM** Staff **WHERE** salary > = 20000 **AND** salary < = 30000;
- Range (BETWEEN/NOT BETWEEN)
  - **SELECT** staffNo, fName, lName, salary **FROM** Staff **WHERE** salary **BETWEEN** 20000 **AND** 30000;

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

staffNo	fName	lName	salary
SL21	John	White	30000
SG5	Susan	Brand	24000

SQL: Select: WHERE clause

- **SELECT** staffNo, fName, lName, position **FROM** Staff **WHERE** position = ‘Manager’ **OR** position = ‘Supervisor’;
- Set membership (IN/NOT IN)
  - **SELECT** staffNo, fName, lName, position **FROM** Staff **WHERE** position **IN** (‘Manager’, ‘Supervisor’);

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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

SQL: Select: WHERE clause

- **SELECT** ownerNo, address **FROM** PrivateOwner **WHERE** address **LIKE** '%Glasgow%';
- Pattern-matching symbols
  - %: any sequence of zero or more characters (wildcard)
  - _: any single character
  - Examples
    - SELECT 'David!' LIKE 'David_';
      - t true
    - SELECT 'David' LIKE 'David_';
      - f false
    - SELECT 'David!' LIKE 'David|_' ESCAPE '|';
      - f 这一行打错了,不对不对好像又没错？？
    - SELECT 'David_' LIKE 'David|_' ESCAPE '|';
      - t

那万一我给他起名就叫xx_呢？

PrivateOwner

ownerNo	fName	lName	address	telNo
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
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

ownerNo	address
CO87	6 Achray St, Glasgow G32 9DX
CO40	63 Well St, Glasgow G42
CO93	12 Park Pl, Glasgow G4 0QR

SQL: Select: WHERE clause

- *List the details of all viewings on property PG4 where a comment has not been supplied*
  - **SELECT** * **FROM** Viewing
    WHERE propertyNo = 'PG4' **AND** comment **IS NULL**;

如果我想要一个“无评论”的？

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

clientNo	propertyNo	viewDate	comment
CR56	PG4	26-May-04	

SQL: Select: ORDER BY clause

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

- Sorting Results
  - Ascending (ASC) or Descending (DESC)
  - **SELECT** StaffNo, position, salary **FROM** Staff **ORDER BY** salary **DESC**;
  - **SELECT** branchNo, StaffNo, salary **FROM** Staff **ORDER BY** branchNo, salary **DESC**; 也可以写ORDER BY 1,3 (不建议)

写在越前面，优先级越靠前

staffNo	position	salary
SL21	Manager	30000
SG5	Manager	24000
SG14	Supervisor	18000
SG37	Assistant	12000
SA9	Assistant	9000
SL41	Assistant	9000

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

branchNo	staffNo	salary
B003	SG5	24000
B003	SG14	18000
B003	SG37	12000
B005	SL21	30000
B005	SL41	9000
B007	SA9	9000

Relational Algebra: Aggregation

$\Im_{AL}(R)$ Applies the aggregate function list, AL, to the relation R to define a relation over the aggregate list. AL contains one or more (<aggregate_function>, <attribute>) pairs.

这是个I

- Main aggregate functions 聚集函数
  - COUNT: returns the number of values in the associated attribute.
  - SUM: returns the sum of the values in the associated attribute.
  - AVG: returns the average of the values in the associated attribute.
  - MIN: returns the smallest value in the associated attribute.
  - MAX: returns the largest value in the associated attribute.

$\rho_R(\text{myMin, myMax, myAverage})\ \Im_{\text{MIN salary, MAX salary, AVERAGE salary}}\ (\text{Staff})$

这里的参数指，返回字段的名字（更名操作）

myMin	myMax	myAverage
9000	30000	17000

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

SQL: Select: Aggregate Functions

- ISO standard defines five aggregate functions
  - COUNT, SUM, AVG, MIN, MAX
- *How many different properties were viewed in May 2004?* TYPICAL CASE
  - **SELECT COUNT**(**DISTINCT** propertyNo) **AS** myCount
    FROM Viewing
    WHERE viewDate **BETWEEN** '1-May-04' **AND** '31-May-04';

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

myCount
2

Relational Algebra: Grouping

$_{GA}\mathfrak{I}_{AL}(R)$ Groups the tuples of relation R by the grouping attributes, GA, and then applies the aggregate function list AL to define a new relation. AL contains one or more (<aggregate_function>, <attribute>) pairs. The resulting relation contains the grouping attributes, GA, along with the results of each of the aggregate functions.

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

$\rho_R(\text{branchNo, myCount, mySum})\ {}_{\text{branchNo}}\mathfrak{I}_{\text{COUNT staffNo, SUM salary}}(\text{Staff})$

branchNo	myCount	mySum
B003	3	54000
B005	2	39000
B007	1	9000

SQL: Select: GROUP BY clause

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

- SELECT clause may contain
  - Column names;
    - Must in GROUP BY clause
  - Aggregate functions;
  - Constants;
  - Combinations of the above.
- GROUP BY clause may contain
  - Column names

一些个人理解：group以后就像是生成了以[条件]为主键的若干条记录

SQL: Select: GROUP BY clause

- *Find the number of staff working in each branch and the sum of their salaries*
  - **SELECT** branchNo, **COUNT**(staffNo) **AS** myCount, **SUM**(salary) **AS** mySum
    FROM Staff
    GROUP BY branchNo **ORDER BY** branchNo;

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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

SQL: Select: GROUP BY clause

- Restricting groupings (HAVING clause)
- *For each branch office with more than one member of staff, find the number of staff working in each branch and the sum of their salaries.*
  - **SELECT** branchNo, **COUNT**(staffNo) **AS** myCount, **SUM**(salary) **AS** mySum **FROM** Staff
  - **GROUP BY** branchNo
  - **HAVING COUNT**(staffNo) > 1 相当于说人数只有1的就不看了
  - **ORDER BY** branchNo;

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Example

- *List the names of all clients who have viewed a property with their comments*

· 是两张表的信息

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

Client

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	425
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

Relational Algebra: Cartesian Product

R × S The Cartesian product operation defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

得到一系列的二元组

P	Q		P × Q	
a	1	=	a	1
b	2		a	2
	3		a	3
			b	1
			b	2
			b	3

Client

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	425
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

clientno是外键的一部分，name是我们需要的信息

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

Relational Algebra: Cartesian Product (Cont.)

4 5

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$ =20条记录

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR56	PA14	too small
CR76	John	Kay	CR76	PG4	too remote
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	no dining room
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	no dining room
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	too small
CR74	Mike	Ritchie	CR76	PG4	too remote
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	no dining room
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	too small
CR62	Mary	Tregear	CR76	PG4	too remote
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	no dining room
CR62	Mary	Tregear	CR56	PG36	

条件：两个表里面的clientno对得上

$$\sigma_{\text{Client.clientNo} = \text{Viewing.clientNo}}(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing})))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

Relational Algebra: θ-join

R ⋈$_F$ S The Theta join operation defines a relation that contains tuples satisfying the predicate *F* from the Cartesian product of R and S. The predicate *F* is of the form $R.a_i\ \theta\ S.b_i$ where θ may be one of the comparison operators (<, ≤, >, ≥, =, ≠).

$$R \bowtie_F S = \sigma_F(R \times S)$$

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie_{\text{Client.clientNo = Viewing.clientNo}} (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

SQL: Select: Multi-Table Queries

- *List the names of all clients who have viewed a property with their comments*
  - **SELECT** Client.clientNo, fName, lName, propertyNo, comment
    FROM Client, Viewing
    WHERE Client.clientNo = Viewing.clientNo;

    FROM Client JOIN Viewing ON Client.clientNo=View

    这里列出来的是老式的写法，join的条件不是非常显然
  - **SELECT** c.clientNo, fName, lName, propertyNo, comment
    FROM Client c, Viewing v
    WHERE c.clientNo = v.clientNo;

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

Client

clientNo	fName	lName	telNo
CR76	John	Kay	0207-774-5632
CR56	Aline	Stewart	0141-848-1825
CR74	Mike	Ritchie	01475-392178
CR62	Mary	Tregear	01224-196720

SQL: Select: Multi-Table Queries

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent)
PrivateOwner	(ownerNo, fName, lName, address, telNo)
Viewing	(clientNo, propertyNo, viewDate, comment)

*核心的写法：
1. 从给定的关系模式出发，找到需要的元素+对应关系
2.可以把操作本身用关系代数表达式表达出来 ->语句是关系代数表达式的更加具体的呈现
3.画出ER图，在图的基础上写语句

- *Find the number of properties handled by each staff member and sort the list by branch and staff.*
  - **SELECT** s.branchNo, s.staffNo, **COUNT**(*) **AS** myCount
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo
    GROUP BY s.branchNo, s.staffNo
    ORDER BY s.branchNo, s.staffNo;

branchNo	staffNo	myCount
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

返回的时候要排序

Relational Algebra: Natural Join

R ⋈ S The Natural join is an Equijoin of the two relations R and S over all common attributes *x*. One occurrence of each common attribute is eliminated from the result.

T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

T ⋈ *U*

A	B	C
a	1	x
a	1	y

是theta join的子集，
主要的区别是，自然连接用的列名字是相同的

$$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \bowtie (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$$

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	too remote
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	no dining room

SQL: Select: Join

- *List the names of all clients who have viewed a property with their comments*
  - **SELECT** Client.clientNo, fName, lName, propertyNo, comment **FROM** Client **JOIN** Viewing **ON** Client.clientNo = Viewing.clientNo

  * **FROM** Client **JOIN** Viewing **USING** (clientNo) 这儿的括号表示“参与到自然连接中”

  * **FROM** Client **NATURAL JOIN** Viewing 这个写法不太好，你不知道他用的是哪个字段连接

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	too remote
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	no dining room

SQL: Select: Join

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent)
PrivateOwner	(ownerNo, fName, lName, address, telNo)
Viewing	(clientNo, propertyNo, viewDate, comment)

- *For each branch, list the numbers and names of staff who manage properties, including the city in which the branch is located and the properties that the staff manage.*
  - **SELECT** b.branchNo, b.city, s.staffNo, fName, lName, propertyNo

前面是表名，
后面的是在自己的
这段语句中起的名字

  - **FROM** Branch b, Staff s, PropertyForRent p
    WHERE b.branchNo = s.branchNo **AND** s.staffNo = p.staffNo
    ORDER BY b.branchNo, s.staffNo, propertyNo;
  - **FROM** Branch b **JOIN** Staff s **USING** (branchNo)
    JOIN PropertyForRent p **USING** (staffNo)

或者是 ON s.branchNo=b.branchNo
USING指的是可以形成自然连接的
[but in some charts their names are different]

这里JOIN是可以连续用若干次的！

Relational Algebra: Outer Join

R ⟕ S The (left) Outer join is a join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation. Missing values in the second relation are set to null.

T

A	B
a	1
b	2

U

B	C
1	x
1	y
3	z

T ⟕ *U*

A	B	C
a	1	x
a	1	y
b	2	

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-04	too small
CR76	PG4	20-Apr-04	too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	no dining room
CR56	PG36	28-Apr-04	

$(\Pi_{\text{propertyNo, street, city}}(\text{PropertyForRent})) \text{ ⟕ } \text{Viewing}$

PropertyForRent

propertyNo	street	city	postcode	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	B007
PL94	6 Argyll St	London	NW2	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	B003
PG36	2 Manor Rd	Glasgow	G32 4QX	B003
PG21	18 Dale Rd	Glasgow	G12	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	B003

propertyNo	street	city	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-May-04	too small
PA14	16 Holhead	Aberdeen	CR62	14-May-04	no dining room
PL94	6 Argyll St	London	**null**	**null**	**null**
PG4	6 Lawrence St	Glasgow	CR76	20-Apr-04	too remote
PG4	6 Lawrence St	Glasgow	CR56	26-May-04	
PG36	2 Manor Rd	Glasgow	CR56	28-Apr-04	
PG21	18 Dale Rd	Glasgow	**null**	**null**	**null**
PG16	5 Novar Dr	Glasgow	**null**	**null**	**null**

SQL: Select: Inner/Outer Join

实际应用中，什么表格放在左边什么表放在右边是有讲究的，但是其实对于日常作业来说没有那么重要

- *List the branches and properties which are in the same city.*
  - **SELECT** b.*, p.* **FROM** Branch1 b
    [INNER] JOIN PropertyForRent1 p **ON** b.bCity = p.pCity;
- *List <u>all</u> branch offices and <u>any</u> properties that are in the same city.*
  - **SELECT** b.*, p.* **FROM** Branch1 b 展示出来有的中介机构没活儿干
    LEFT [OUTER] JOIN PropertyForRent1 p **ON** b.bCity = p.pCity; 【左外连接】FROM后面的全显示
- *List <u>all</u> properties and <u>any</u> branch offices that are in the same city.*
  - **SELECT** b.*, p.* **FROM** Branch1 b 展示出来有的房产没人管
    RIGHT [OUTER] JOIN PropertyForRent1 p **ON** b.bCity = p.pCity; 【右外连接】JOIN后面的全显示
- *List the branch offices and properties that are in the same city along with any unmatched branches or properties.*
  - **SELECT** b.*, p.* **FROM** Branch1 b 都列全了
    FULL [OUTER] JOIN PropertyForRent1 p **ON** b.bCity = p.pCity; 【全连接】

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

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

SQL: Select: Sub Queries

子查询：从实现的角度来说，子查询会生成一个临时表，而这个临时表不见得有索引，没有办法有效的使用SQL的数据结构

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)

- *List the staff who work in the branch at '163 Main St'.*
  - **SELECT** staffNo, fName, lName, position **FROM** Staff **WHERE** branchNo = (**SELECT** branchNo **FROM** Branch **WHERE** street = '163 Main St');

括号里相当于说是一个前提，叫关联子查询，在这个case是放在WHERE语句里的，其实放在哪个语句里好像都可以，只要选出来是个表就可以

这里变为WHERE

- *List all staff whose salary is greater than the average salary, and show by how much their salary is greater than the average.*
  - **SELECT** staffNo, fName, lName, position, salary - (**SELECT AVG**(salary) **FROM** Staff) **AS** salDiff **FROM** Staff **WHERE** salary > (**SELECT AVG**(salary) **FROM** Staff);

起个名儿

这个case下返回的是一个值，所以又叫做标量查询

SQL: Select: IN/SOME(ANY)/ALL

这个可以叫做非标量查询

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)

- *List the properties that are handled by staff who work in the branch at '163 Main St'.* IN枚举了一些东西
  - **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'));

    挑出了staff在我们选出的店的所有staff里的
- *Find all staff whose salary is larger than the salary of at least one member of staff at branch B003.* 只要比这里任意一个薪水高就行
  - **SELECT** staffNo, fName, lName, position, salary **FROM** Staff **WHERE** salary > **ANY** (**SELECT** salary **FROM** Staff **WHERE** branchNo = 'B003');
- *Find all staff whose salary is larger than the salary of every member of staff at branch B003.* 要比所有的薪水都高
  - **SELECT** staffNo, fName, lName, position, salary **FROM** Staff **WHERE** salary > **ALL** (**SELECT** salary **FROM** Staff **WHERE** branchNo = 'B003');

这些东西返回的应该是一张表，而不是一个值

SQL: Select: EXISTS/NOT EXISTS

表示出一阶逻辑中的一些关系

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)

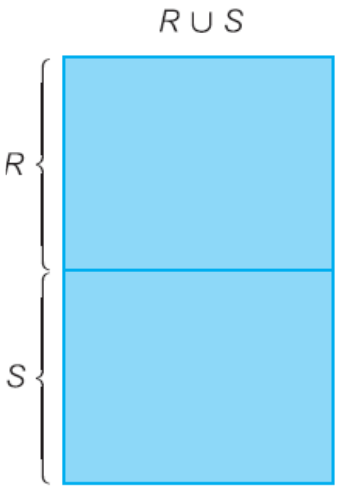
- *Find all staff who work in a London branch office.*
  - **SELECT** staffNo, fName, lName, position
    FROM Staff s, Branch b
    WHERE s.branchNo = b.branchNo **AND** city = 'London';
  - **SELECT** staffNo, fName, lName, position
    FROM Staff s
    WHERE EXISTS (**SELECT** *
    FROM Branch b
    WHERE s.branchNo = b.branchNo **AND** city = 'London');

存在xxx可以理解为“表S满足这个性质”

NOTICE：从exists外面的条件往里看，逻辑才好理解

Relational Algebra: Union

R ∪ S The union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated. R and S must be union-compatible. 不能直接并集，但是投影到city上就可以作并集了



Branch

branchNo	street	city
B005	22 Deer Rd	London
B007	16 Argyll St	Aberdeen
B003	163 Main St	Glasgow
B004	32 Manse Rd	Bristol
B002	56 Clover Dr	London

PropertyForRent

propertyNo	street	city	postco
PA14	16 Holhead	Aberdeen	AB7 5S
PL94	6 Argyll St	London	NW2
PG4	6 Lawrence St	Glasgow	G11 9Q
PG36	2 Manor Rd	Glasgow	G32 4Q
PG21	18 Dale Rd	Glasgow	G12
PG16	5 Novar Dr	Glasgow	G12 9A

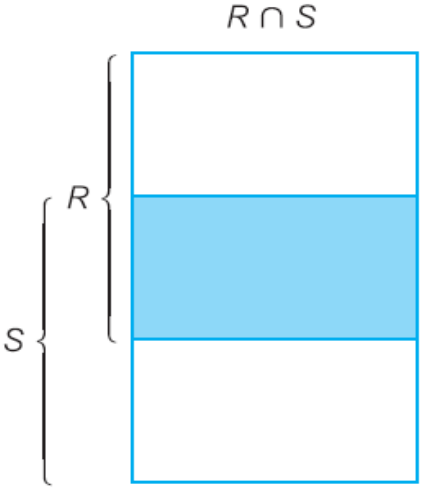
$\Pi_{city}(\text{Branch}) \cup \Pi_{city}(\text{PropertyForRent})$

city
London
Aberdeen
Glasgow
Bristol

```
(SELECT city
 FROM Branch
 WHERE city IS NOT NULL)
 UNION 模式一样才能用UNION
(SELECT city
 FROM PropertyForRent
 WHERE city IS NOT NULL);
```

Relational Algebra: Intersection

R ∩ S The Intersection operation defines a relation consisting of the set of all tuples that are in both R and S. R and S must be union-compatible.



Branch

branchNo	street	city
B005	22 Deer Rd	London
B007	16 Argyll St	Aberdeen
B003	163 Main St	Glasgow
B004	32 Manse Rd	Bristol
B002	56 Clover Dr	London

PropertyForRent

propertyNo	street	city	postco
PA14	16 Holhead	Aberdeen	AB7 5S
PL94	6 Argyll St	London	NW2
PG4	6 Lawrence St	Glasgow	G11 9Q
PG36	2 Manor Rd	Glasgow	G32 4Q
PG21	18 Dale Rd	Glasgow	G12
PG16	5 Novar Dr	Glasgow	G12 9A

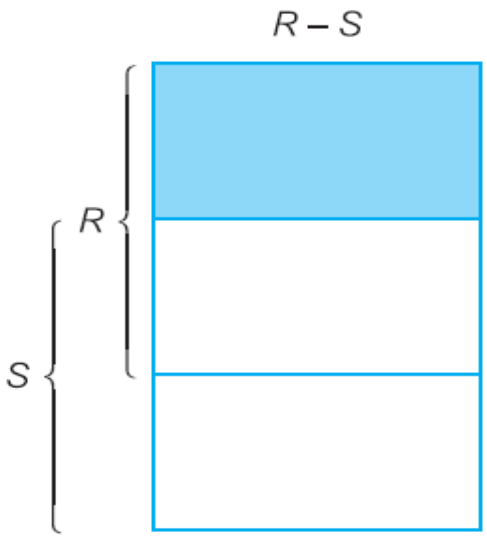
$$\Pi_{city}(\text{Branch}) \cap \Pi_{city}(\text{PropertyForRent})$$

city
Aberdeen
London
Glasgow

(**SELECT city**
FROM Branch)
INTERSECT 模式一样才能用INTERSECT
(**SELECT city**
FROM PropertyForRent);

Relational Algebra: Set Difference

R – S The Set difference operation defines a relation consisting of the tuples that are in relation R, but not in S. R and S must be union-compatible.



Branch

branchNo	street	city
B005	22 Deer Rd	London
B007	16 Argyll St	Aberdeen
B003	163 Main St	Glasgow
B004	32 Manse Rd	Bristol
B002	56 Clover Dr	London

PropertyForRent

propertyNo	street	city	postcc
PA14	16 Holhead	Aberdeen	AB7 5S
PL94	6 Argyll St	London	NW2
PG4	6 Lawrence St	Glasgow	G11 9Q
PG36	2 Manor Rd	Glasgow	G32 4Q
PG21	18 Dale Rd	Glasgow	G12
PG16	5 Novar Dr	Glasgow	G12 9A

$$\Pi_{city}(\text{Branch}) - \Pi_{city}(\text{PropertyForRent})$$

city
Bristol

```
(SELECT city
 FROM Branch)
 EXCEPT
(SELECT city
 FROM PropertyForRent);
```

我把这俩字段都投影出来，使其具有union兼容性

SQL: INSERT INTO

INSERT INTO TableName [(columnList)]
VALUES (dataValueList)

Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)

- *Insert a new row into the* Staff *table supplying data for all columns.*
  - **INSERT INTO** Staff
    VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M', **DATE** '1957-05-25', 8300,'B003'); 顺序和STAFF定义的含义要对得上
- *Insert a new row into the* Staff *table supplying data for all mandatory columns:* staffNo, fName, lName, position, salary, *and* branchNo.
  - **INSERT INTO** Staff (staffNo, fName, lName, position, salary, branchNo)
    VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');
  - **INSERT INTO** Staff
    VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL, NULL, 8100, 'B003');

实际应用中insert会很慢，因为 可能会有index之类的东西限制住

SQL: INSERT INTO

INSERT INTO TableName [(columnList)]
SELECT . . .

等等等等这个还需要再理解

Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)

- *Populate the* StaffPropCount *table using details from the* Staff *and* PropertyForRent *tables.*
  - StaffPropCount(staffNo, fName, lName, propCount)
    - **INSERT INTO** StaffPropCount
      SELECT s.staffNo, fName, lName, **COUNT**(*) **FROM** Staff s, PropertyForRent p
      WHERE s.staffNo = p.staffNo **GROUP BY** s.staffNo, fName, lName
      UNION SELECT staffNo, fName, lName, 0 **FROM** Staff s
      WHERE NOT EXISTS (**SELECT** * **FROM** PropertyForRent p **WHERE** p.staffNo = s.staffNo);
    - **CREATE TABLE** StaffPropCount1 AS
      SELECT s.staffNo, fName, lName, **COUNT**(*) **FROM** Staff s, PropertyForRent p
      WHERE s.staffNo = p.staffNo **GROUP BY** s.staffNo, fName, lName
      UNION SELECT staffNo, fName, lName, 0 **FROM** Staff s
      WHERE NOT EXISTS (**SELECT** * **FROM** PropertyForRent p **WHERE** p.staffNo = s.staffNo);

SQL: UPDATE

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

Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)

- *Give all staff a 3% pay increase.*
  - **UPDATE** Staff **SET** salary = salary*1.03;
- *Give all Managers a 5% pay increase.*
  - **UPDATE** Staff **SET** salary = salary*1.05
    WHERE position = 'Manager';
- *Promote David Ford (*staffNo = *'SG14') to Manager and change his salary to £18,000.*
  - **UPDATE** Staff **SET** position = 'Manager', salary = 18000
    WHERE staffNo = 'SG14'; 加了一个过滤条件

SQL: DELETE

DELETE FROM TableName
[**WHERE** searchCondition]

- *Delete all viewings that relate to property PG4.*
  - **DELETE FROM** Viewing
    WHERE propertyNo = 'PG4';
- *Delete all rows from the* Viewing *table.*
  - **DELETE FROM** Viewing;

NOTICE:这个操作是不可逆的

Outline

- Introduction to SQL
- Data Manipulation Language*
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- Data Definition Language*
  - Data Types
  - Schema
  - Table
  - Index
  - View
  - Transaction
- Procedural SQL

Data Definition

	Create	Change	Destroy
Schema	CREATE SCHEMA		DROP SCHEMA
Domain	CREATE DOMAIN	ALTER DOMAIN	DROP DOMAIN
Table	CREATE TABLE	ALTER TABLE	DROP TABLE
View	CREATE VIEW		DROP VIEW
Index	CREATE INDEX		DROP INDEX

SQL Identifiers

- Used to identify objects in the database, such as table names, view names, and columns.
- Characters must appear in a **character set**
  - ISO default character set consists of
    - The upper-case letters A . . . Z
    - The lower-case letters a . . . z
    - The digits 0 . . . 9, and
    - The underscore (_) character.
  - Restrictions
    - An identifier can be no longer than 128 characters (most dialects have a much lower limit than this);
    - An identifier must start with a letter;
    - An identifier cannot contain spaces.

Create/Destroy a Schema

- Create a schema
  - **CREATE SCHEMA** Name [**AUTHORIZATION** CreatorIdentifier]
    - **CREATE SCHEMA** SqlTests **AUTHORIZATION** Smith;
- Destroy a schema
  - **DROP SCHEMA** Name [**RESTRICT** | **CASCADE**]
    - RESTRICT: the schema must be empty or the operation fails
    - CASCADE: the operation cascades to drop all objects associated with the schema

```
CREATE SCHEMA schema_name [AUTHORIZATION role_specification]
       [schema_element[ ... ]]
CREATE SCHEMA AUTHORIZATION role_specification [schema_element
       [...]]
CREATE SCHEMA IF NOT EXISTS schema_name [AUTHORIZATION
       role_specification]
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION role_specification
```

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

Create a Table

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Branch (branchNo, street, city, postcode)
Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Data Types

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit†	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

† BIT and BIT VARYING have been removed from the SQL:2003 standard.

Data Types: Character

Data Type	SQL Server	Oracle	MySQL
Character	CHAR(n)	CHAR(n)	CHAR(n)
	8000 Bytes	2000 Bytes	255 Characters
National Character	NCHAR(n)	NCHAR(n)	
	4000 Characters	4000 Bytes	
Character Varying	VARCHAR(n)	VARCHAR2(n)	VARCHAR(n)
	8000 Bytes	2000 Bytes	65535 Bytes
National Char Varying	NVARCHAR(n)	NVARCHAR2(n)	
	4000 Characters	4000 Bytes	
Text	TEXT	CLOB	[MEDIUM|LONG]TEXT
	2^31 – 1 (2GB)	128 TB	64 KB/16 MB/4 GB
National Text	NTEXT	NCLOB	
	2^30 – 1 Characters	128 TB	

Data Types: Character

In PostgreSQL:

Name	Description
character varying(n), varchar(n)	variable-length with limit
character(n), char(n)	fixed-length, blank padded
text	variable unlimited length

Data Types: Numeric Data

Data Type	Bytes	Minimum (S)	Maximum (S)	SQL Server	MySQL
TINYINT	1	-128	127	Unsigned	Signed, Unsigned
SMALLINT	2	-32768	32767	Signed	Signed, Unsigned
MEDIUMINT	3	-8388608	8388607	Signed	Signed, Unsigned
INT	4	-2147483648	2147483647	Signed	Signed, Unsigned
BIGINT	8	-9.2233E+18	9.2233E+18	Signed	Signed, Unsigned

Data Type	Bytes	SQL Server	Oracle	MySQL
REAL	4	REAL	BINARY_FLOAT	FLOAT
DOUBLE PRECISION	8	FLOAT	BINARY_DOUBLE	DOUBLE
FLOAT [precision]		FLOAT(p)	FLOAT(p)	

Data Type	SQL Server	Oracle	MySQL
DECIMAL(precision[,scale]) NUMERIC(precision[,scale])	38 (17 Bytes)	38	65 (29 Bytes)

Data Types: Numeric Data

In PostgreSQL:

Name	Storage Size	Range
smallint	2 bytes	-32768 to +32767
integer	4 bytes	-2147483648 to +2147483647
bigint	8 bytes	-9223372036854775808 to +9223372036854775807
decimal	variable	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	6 decimal digits precision
double precision	8 bytes	15 decimal digits precision
smallserial	2 bytes	1 to 32767
serial	4 bytes	1 to 2147483647
bigserial	8 bytes	1 to 9223372036854775807

Data Types: Date & Time

Data Type	Fields	Example
DATE	YEAR, MONTH, DAY	'2014-11-04'
TIME [timePrecision]	HOUR, MINUTE, SECOND	'10:12:09.019473'
TIMESTAMP [timePrecision]	YEAR, MONTH, DAY, HOUR, MINUTE, SECOND	'2014-11-04 10:12:09.019473'
TIME [p] WITH TIME ZONE	…, TIMEZONE_HOUR, TIMEZONE_MINUTE	'10:12:09.019473 +08:00'
TIMESTAMP [p] WITH TIME ZONE	…, TIMEZONE_HOUR, TIMEZONE_MINUTE	'2014-11-04 10:12:09.019473 +08:00'

In PostgreSQL:

Name	Low Value	High Value	Resolution
timestamp [(p)] [without time zone]	4713 BC	294276 AD	1 microsecond
timestamp [(p)] with time zone	4713 BC	294276 AD	1 microsecond
date	4713 BC	5874897 AD	1 day
time [(p)] [without time zone]	00:00:00	24:00:00	1 microsecond
time [(p)] with time zone	00:00:00+1459	24:00:00-1459	1 microsecond
interval [fields] [(p)]	-178000000 years	178000000 years	1 microsecond

Integrity

- Required Data
  - position **VARCHAR**(10) **NOT NULL**
- Domain Constraints
  - sex **CHAR NOT NULL CHECK** (sex **IN** ('M', 'F'))
- General Constraints
  - **CHECK** (**NOT EXISTS**
    (**SELECT** staffNo
    FROM PropertyForRent
    GROUP BY staffNo
    HAVING COUNT(*) > 100))

Integrity

- Entity Integrity
  - **PRIMARY KEY**(clientNo, propertyNo)
- Referential Integrity
  - **FOREIGN KEY**(branchNo) **REFERENCES** Branch
  - **FOREIGN KEY** (staffNo) **REFERENCES** Staff **ON DELETE SET NULL**
  - **FOREIGN KEY** (ownerNo) **REFERENCES** PrivateOwner **ON UPDATE CASCADE**
  - ON DELETE/UPDATE options
    - **CASCADE**: delete/update the matching rows in the child table
    - **SET NULL**: set the foreign key value(s) in the child table to NULL
    - **SET DEFAULT**: set the foreign key value(s) in the child table to the specified default value
    - **NO ACTION**: Reject the delete/update operation from the parent table

SQL: Create/Destroy a Table

```
CREATE TABLE tbl_name (create_definition,...)

create_definition:
    col_name column_definition |
    PRIMARY KEY (col_name,...) |
    UNIQUE [index_name] (col_name,...) |
    INDEX [index_name] (col_name,...) |
    FOREIGN KEY [index_name] (col_name,...) reference_definition |
    CHECK (expr)

column_definition:
    data_type [NOT NULL] [DEFAULT default_value] [UNIQUE] [CHECK (expr)]

reference_definition:
    REFERENCES tbl_name (col_name,...) [MATCH {FULL|PARTIAL}]
    [ON DELETE reference_option] [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

```
DROP TABLE [IF EXISTS] tbl_name [, ...] [RESTRICT | CASCADE]
```

SQL: Create Table

```
CREATE TABLE PropertyForRent(
    propertyNo VARCHAR(5) NOT NULL,
    street VARCHAR(25) NOT NULL,
    city VARCHAR(15) NOT NULL,
    postcode VARCHAR(8),
    type CHAR(1) NOT NULL DEFAULT 'F' CHECK(type IN ('B', 'C', 'D', 'E', 'F', 'M', 'S')),
    rooms SMALLINT NOT NULL DEFAULT 4 CHECK(rooms BETWEEN 1 AND 15),
    rent DECIMAL(6,2) NOT NULL DEFAULT 600 CHECK(rent BETWEEN 0 AND 9999.99),
    ownerNo VARCHAR(5) NOT NULL,
    staffNo VARCHAR(5),
    branchNo CHAR(4) NOT NULL,
    PRIMARY KEY (propertyNo),
    FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (ownerNo) REFERENCES PrivateOwner ON UPDATE CASCADE,
    FOREIGN KEY (branchNo) REFERENCES Branch ON UPDATE CASCADE
);
```

SQL: Alter Table

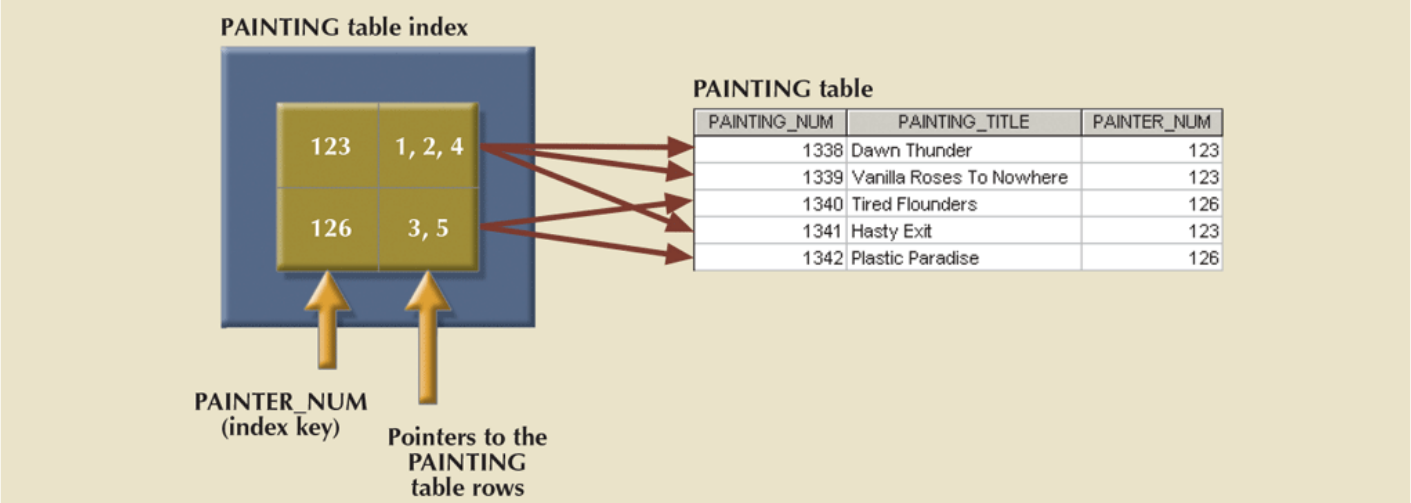
```
ALTER TABLE [IF EXISTS] [ONLY] name [ * ] action [, ... ]

action :
ADD [COLUMN] [IF NOT EXISTS] column_name data_type [COLLATE collation]
    [column_constraint [ ... ]]
DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]
ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING
    expression]
ALTER [COLUMN] column_name SET DEFAULT expression
ALTER [COLUMN] column_name DROP DEFAULT
ALTER [COLUMN] column_name {SET | DROP} NOT NULL
ALTER [COLUMN] column_name ADD GENERATED {ALWAYS | BY DEFAULT} AS IDENTITY
    [(sequence_options)]
ALTER [COLUMN] column_name {SET GENERATED {ALWAYS | BY DEFAULT} | SET
    sequence_option | RESTART [[WITH] restart]} [...]
ALTER [COLUMN] column_name DROP IDENTITY [IF EXISTS]
...
```

- **ALTER TABLE** Staff **ALTER** position **DROP DEFAULT**;
- **ALTER TABLE** Staff **ALTER** sex **SET DEFAULT** 'F';
- **ALTER TABLE** Staff **ALTER** sex **TYPE** char(1) ;
- **ALTER TABLE** Client **ADD** prefNoRooms SMALLINT;

Indexes

- Orderly arrangement to logically access rows in a table
  - Index key: index's reference point that leads to data location identified by the key

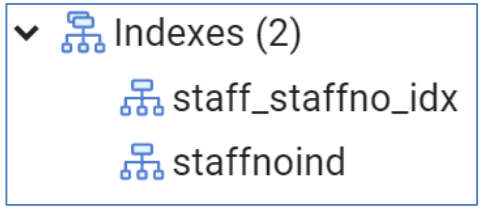


  - Unique index: index key can have only one pointer value associated with it
- Each index is associated with only one table
  - The index key can have multiple attributes

SQL: Create/Destroy an Index

```
CREATE [UNIQUE] INDEX [CONCURRENTLY] [[IF NOT EXISTS] name] ON
      table_name [USING method]
    ({column_name | (expression)} [COLLATE collation] [opclass]
      [ASC | DESC] [NULLS {FIRST | LAST}] [, ...])
    [WITH (storage_parameter = value [, ... ])]
    [TABLESPACE tablespace_name]
    [WHERE predicate]
```

```
DROP INDEX [CONCURRENTLY] [IF EXISTS] name [, ...] [CASCADE |
      RESTRICT]
```



- **CREATE UNIQUE INDEX ON** Staff (staffNo);
- **CREATE UNIQUE INDEX** StaffNoInd **ON** Staff (staffNo);
- **CREATE UNIQUE INDEX** PropertyNoInd **ON** PropertyForRent (propertyNo);
- **CREATE INDEX** RentInd **ON** PropertyForRent (city, rent);

Virtual Tables: Creating a View

View: virtual table based on a SELECT query

- Base tables: tables on which the view is based

Used to update attributes

- Batch update routine: pools multiple transactions into a single batch to update a master table field in a single operation

Updatable view restrictions

- GROUP BY expressions or aggregate functions cannot be used
- Set operators cannot be used
- Most restrictions are based on the use of JOINs or group operators in views

SQL: Create/Destroy a View

```
CREATE VIEW view_name [(column_list)]
    AS select_statement
```

CREATE VIEW statement: data definition command that stores the subquery specification in the data dictionary

```
DROP View [IF EXISTS] view_name [RESTRICT | CASCADE]
```

- **CREATE VIEW** Manager3Staff
  AS SELECT * **FROM** Staff **WHERE** branchNo = 'B003';
- **CREATE VIEW** Staff3
  AS SELECT staffNo, fName, lName, position, sex
  FROM Staff **WHERE** branchNo = 'B003';
- **CREATE VIEW** Staff3
  AS SELECT staffNo, fName, lName, position, sex **FROM** Manager3Staff;
- **CREATE VIEW** StaffPropCnt (branchNo, staffNo, cnt)
  AS SELECT s.branchNo, s.staffNo, **COUNT**(*) **FROM** Staff s, PropertyForRent p
  WHERE s.staffNo = p.staffNo **GROUP BY** s.branchNo, s.staffNo;

Sequences

- Many similarities in the use of sequences across these DBMSs
  - Independent object in the database
  - Have a name and can be used anywhere a value expected
  - Not tied to a table or column
  - Generate a numeric value that can be assigned to any column in any table
  - Table attribute with an assigned value can be edited and modified

```
postgres=# create sequence mySequence start with 20010;
CREATE SEQUENCE
postgres=# select nextval('mySequence');
 nextval
---------
   20010
(1 行记录)

postgres=# select nextval('mySequence');
 nextval
---------
   20011
(1 行记录)

postgres=# select currval('mySequence');
 currval
---------
   20011
(1 行记录)

postgres=# select nextval('mySequence');
 nextval
---------
   20012
(1 行记录)
```

CREATE SEQUENCE — define a new sequence generator

Synopsis

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

Transactions

- Transactions
  - Changes made by a transaction are not visible to other concurrently executing transactions until the transaction completes
    - COMMIT: ends the transaction successfully, making the database changes permanent
    - ROLLBACK: aborts the transaction, backing out any changes made by the transaction statement.

```
START TRANSACTION;
DROP TABLE IF EXISTS avg_salary;
CREATE TABLE avg_salary (salary INTEGER);
INSERT INTO avg_salary (SELECT AVG(salary) FROM staff);
COMMIT;
```

Procedural SQL

- Performs a conditional or looping operation by isolating critical code and making all application programs call the shared code
  - Better maintenance and logic control
- Persistent stored module (PSM): block of code
  - Contains standard SQL statements and procedural extensions that is stored and executed at the DBMS server
- Procedural Language SQL (PL/SQL in Oracle)
  - Called PL/pgSQL in PostgreSQL
  - Use and storage of procedural code and SQL statements within the database
  - Merging of SQL and traditional programming constructs
- Procedural code is executed as a unit by DBMS when invoked by end user
  - Anonymous PL/SQL blocks
  - Stored procedures
  - Triggers
  - PL/SQL functions

Sample (In PG)

DO — execute an anonymous code block

Synopsis

```
DO [ LANGUAGE lang_name ] code
```

Examples

Grant all privileges on all views in schema `public` to role `webuser`:

```
DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT table_schema, table_name FROM information_schema.tables
             WHERE table_type = 'VIEW' AND table_schema = 'public'
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO webuser';
    END LOOP;
END$$;
```

Compatibility

There is no `DO` statement in the SQL standard.

Stored Procedures

- Named collection of procedural and SQL statements
  - Stored in the database
  - Can be used to encapsulate and represent business transactions
- Advantages
  - Reduce network traffic and increase performance
  - Decrease code duplication by means of code isolation and code sharing

Sample (In PG)

```
1  CREATE OR REPLACE PROCEDURE addStaff(inputNum int) AS $$
2  DECLARE myNum smallint = 0;
3  BEGIN
4  WHILE myNum < inputNum LOOP
5  INSERT INTO Staff VALUES ('SG16'||to_char(myNum,'9999'), 'Alan',
6      'Brown', 'Assistant', 'M', DATE '1957-05-25', 8300+myNum,'B003');
7  myNum := myNum + 1;
8  END LOOP;
9  END
10 $$ LANGUAGE plpgsql;
```

Query Editor

```
1  call addStaff(10);
```

Query Editor

```
1  select * from staff;
```

Data Output | Explain | Messages | Notifications | Query History

	staffno [PK] character varying (255)	fname character varying (255)	lname character varying (255)	position character varying (255)	sex character varying (255)	dob character varying (255)	salary numeric (7,2)	branchno character vary
1	SG16 0	Alan	Brown	Assistant	M	1957-05-25	8300.00	B003
2	SG16 1	Alan	Brown	Assistant	M	1957-05-25	8301.00	B003
3	SG16 2	Alan	Brown	Assistant	M	1957-05-25	8302.00	B003
4	SG16 3	Alan	Brown	Assistant	M	1957-05-25	8303.00	B003
5	SG16 4	Alan	Brown	Assistant	M	1957-05-25	8304.00	B003
6	SG16 5	Alan	Brown	Assistant	M	1957-05-25	8305.00	B003
7	SG16 6	Alan	Brown	Assistant	M	1957-05-25	8306.00	B003
8	SG16 7	Alan	Brown	Assistant	M	1957-05-25	8307.00	B003
9	SG16 8	Alan	Brown	Assistant	M	1957-05-25	8308.00	B003
10	SG16 9	Alan	Brown	Assistant	M	1957-05-25	8309.00	B003

PL/SQL Processing with Cursors

- Cursor: special construct used to hold data rows returned by a SQL query
  - Implicit cursor: automatically created when SQL statement returns only one value
  - Explicit cursor: holds the output of a SQL statement that may return two or more rows
  - Syntax:

    CURSOR *cursor_name* IS *select-query*;
- Cursor-style processing involves retrieving data from the cursor one row at a time
  - Current row is copied to PL/SQL variables

Sample (In Oracle)

FIGURE 8.32 A SIMPLE PRC_CURSOR EXAMPLE

SQL Plus

```
SQL> CREATE OR REPLACE PROCEDURE PRC_CURSOR_EXAMPLE IS
  2  W_P_CODE    PRODUCT.P_CODE%TYPE;
  3  W_P_DESCRIPT        PRODUCT.P_DESCRIPT%TYPE;
  4  W_TOT               NUMBER(3);
  5  CURSOR PROD_CURSOR IS
  6     SELECT P_CODE, P_DESCRIPT
  7        FROM PRODUCT
  8        WHERE P_QOH > (SELECT AVG(P_QOH) FROM PRODUCT);
  9  BEGIN
 10  DBMS_OUTPUT.PUT_LINE('PRODUCTS WITH P_QOH > AVG(P_QOH)');
 11  DBMS_OUTPUT.PUT_LINE('=====================================');
 12  OPEN PROD_CURSOR;
 13  LOOP
 14     FETCH PROD_CURSOR INTO W_P_CODE, W_P_DESCRIPT;
 15     EXIT WHEN PROD_CURSOR%NOTFOUND;
 16     DBMS_OUTPUT.PUT_LINE(W_P_CODE ||' -> ' || W_P_DESCRIPT );
 17  END LOOP;
 18  DBMS_OUTPUT.PUT_LINE('=====================================');
 19  DBMS_OUTPUT.PUT_LINE('TOTAL PRODUCT PROCESSED ' || PROD_CURSOR%ROWCOUNT);
 20  DBMS_OUTPUT.PUT_LINE('--- END OF REPORT ----');
 21  CLOSE PROD_CURSOR;
 22  END;
 23  /

Procedure created.

SQL> EXEC PRC_CURSOR_EXAMPLE;
PRODUCTS WITH P_QOH > AVG(P_QOH)
=====================================
PVC23DRT -> PVC pipe, 3.5-in., 8-ft
SM-18277 -> 1.25-in. metal screw, 25
SW-23116 -> 2.5-in. wd. screw, 50
=====================================
TOTAL PRODUCT PROCESSED 3
--- END OF REPORT ----

PL/SQL procedure successfully completed.

SQL> _
```

Sample (In PG)

Query Editor

```
1  CREATE OR REPLACE PROCEDURE PRC_CURSOR_EXAMPLE() AS $$
2  DECLARE
3     v_staffno STAFF.STAFFNO%TYPE;
4     v_salary STAFF.SALARY%TYPE;
5     PROD_CURSOR CURSOR FOR
6        SELECT STAFFNO, SALARY
7        FROM STAFF
8        WHERE FNAME = 'Alan';
9
10 BEGIN
11 RAISE INFO '=====================================';
12 OPEN PROD_CURSOR;
13 LOOP
14    FETCH PROD_CURSOR INTO v_staffno, v_salary;
15    EXIT WHEN NOT FOUND;
16    RAISE INFO '% -> %', v_staffno, v_salary;
17 END LOOP;
18 RAISE INFO '=====================================';
19 RAISE INFO '--- END OF REPORT ----';
20 CLOSE PROD_CURSOR;
21 END;
22 $$ LANGUAGE plpgsql;
23
```

Data Output | Explain | Messages | Notifications | Query History

```
CREATE PROCEDURE
```

Query Editor

```
1  CALL PRC_CURSOR_EXAMPLE();
```

Data Output | Explain | Messages | Notifications | Que

```
信息:  =====================================
信息:  SG16    0 -> 8300.00
信息:  SG16    1 -> 8301.00
信息:  SG16    2 -> 8302.00
信息:  SG16    3 -> 8303.00
信息:  SG16    4 -> 8304.00
信息:  SG16    5 -> 8305.00
信息:  SG16    6 -> 8306.00
信息:  SG16    7 -> 8307.00
信息:  SG16    8 -> 8308.00
信息:  SG16    9 -> 8309.00
信息:  =====================================
信息:  --- END OF REPORT ----
CALL
```

Outline

- Introduction to SQL
- Data Manipulation Language*
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- Data Definition Language*
  - Data Types
  - Schema
  - Table
  - Index
  - View
  - Transaction
- Procedural SQL

Triggers

- Procedural SQL code automatically invoked by RDBMS when given data manipulation event occurs
- Parts of a trigger definition
  - Triggering timing: indicates when trigger's PL/SQL code executes
  - Triggering event: statement that causes the trigger to execute
    - Triggering level: statement- and row-level
    - Triggering action: PL/SQL code enclosed between the BEGIN and END keywords
- DROP TRIGGER trigger_name command
  - Deletes a trigger without deleting the table
- Trigger action based on conditional DML predicates
  - Actions depend on the type of DML statement that fires the trigger

Sample (in Oracle)

FIGURE 8.21 THE THIRD VERSION OF THE TRG_PRODUCT_REORDER TRIGGER

SQL Plus

```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
  2  BEFORE INSERT OR UPDATE OF P_QOH, P_MIN ON PRODUCT
  3  FOR EACH ROW
  4  BEGIN
  5     IF :NEW.P_QOH <= :NEW.P_MIN THEN
  6        :NEW.P_REORDER := 1;
  7     ELSE
  8        :NEW.P_REORDER := 0;
  9     END IF;
 10  END;
 11  /

Trigger created.

SQL> _
```

Sample (In Oracle)

FIGURE 8.22 EXECUTION OF THE THIRD TRIGGER VERSION

SQL Plus

```
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
  2  FROM PRODUCT;

P_CODE   P_DESCRIPT                          P_QOH P_MIN P_MIN_ORDER P_REORDER
-------- ---------------------------------- ----- ----- ----------- ---------
11QER/31 Power painter, 15 psi., 3-nozzle      29     5          25         1
13-Q2/P2 7.25-in. pwr. saw blade               32    15          50         0
14-Q1/L3 9.00-in. pwr. saw blade               18    12          50         0
1546-QQ2 Hrd. cloth, 1/4-in., 2x50             15     8          35         0
1558-QW1 Hrd. cloth, 1/2-in., 3x50             23     5          25         0
2232/QTY B&D jigsaw, 12-in. blade               8     5          15         0
2232/QWE B&D jigsaw, 8-in. blade                6     7          15         1
2238/QPD B&D cordless drill, 1/2-in.           12     5          12         0
23109-HB Claw hammer                           23    10          25         0
23114-AA Sledge hammer, 12 lb.                  8    10          12         1
54778-2T Rat-tail file, 1/8-in. fine           43    20          25         0
89-WRE-Q Hicut chain saw, 16 in.               11     5          10         0
PVC23DRT PVC pipe, 3.5-in., 8-ft              188    75          50         0
SM-18277 1.25-in. metal screw, 25             172    75          50         0
SW-23116 2.5-in. wd. screw, 50                237   100         100         0
WR3/TT3  Steel matting, 4'x8'x1/6", .5" mesh   18     5          10         0

16 rows selected.

SQL> UPDATE PRODUCT SET P_QOH = P_QOH;

16 rows updated.

SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_MIN_ORDER, P_REORDER
  2  FROM PRODUCT
  3  WHERE P_CODE = '11QER/31';

P_CODE   P_DESCRIPT                          P_QOH P_MIN P_MIN_ORDER P_REORDER
-------- ---------------------------------- ----- ----- ----------- ---------
11QER/31 Power painter, 15 psi., 3-nozzle      29     5          25         0

SQL> _
```

Sample (In PG)

y Editor

```
CREATE OR REPLACE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;

        -- Who works for us when they must pay for it?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
        END IF;

        -- Remember who changed the payroll when
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW EXECUTE FUNCTION emp_stamp();
```

Output Explain Messages Notifications Query History

```
TE TRIGGER
```

Query Editor

```
1 CREATE TABLE emp (
2     empname text,
3     salary integer,
4     last_date timestamp,
5     last_user text
6 );
```

Query Editor

```
1 insert into emp values ('test', 1000);
2 select * from emp;
3
```

Data Output Explain Messages Notifications Query History

	empname text	salary integer	last_date timestamp without time zone	last_user text
1	test	1000	2020-03-23 23:45:29.516687	postgres

Query Editor

```
1 insert into emp values ('test', -1000);
2
```

Data Output Explain Messages Notifications Query Hi

```
ERROR:  错误:  test cannot have a negative salary
CONTEXT:  在RAISE的第13行的PL/pgSQL函数emp_stamp()


SQL state: P0001
```

PL/SQL Stored Functions

- Stored function: named group of procedural and SQL statements that returns a value
  - Indicated by a RETURN statement in its program code
- Can be invoked only from within stored procedures or triggers
  - Cannot be invoked from SQL statements unless the function follows some very specific compliance rules

Sample (In PG)

Query Editor

```
CREATE OR REPLACE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
BEGIN
    IF (subtotal < 1000) THEN
        RETURN subtotal * 0.06;
    ELSE
        RETURN subtotal * 0.10;
    END IF;
END
$$ LANGUAGE plpgsql;
```

Data Output | Explain | Messages | Notifications | Query History

```
CREATE FUNCTION
```

Query Editor

```
select sales_tax(500);
```

Data Output | Explain | Messages | Noti

	sales_tax real
1	30

Query Editor

```
select sales_tax(5000);
```

Data Output | Explain | Messages | N

	sales_tax real
1	500

Embedded SQL

- SQL statements contained within an application programming language
  - Host language: any language that contains embedded SQL statements
- Differences between SQL and procedural languages
  - Run-time mismatch
    - SQL is executed one instruction at a time
    - Host language runs at client side in its own memory space
  - Processing mismatch
    - Conventional programming languages process one data element at a time
    - Newer programming environments manipulate data sets in a cohesive manner
  - Data type mismatch
    - Data types provided by SQL might not match data types used in different host languages

Embedded SQL (cont')

- Embedded SQL framework defines:
  - Standard syntax to identify embedded SQL code within the host language
  - Standard syntax to identify host variables
  - Communication area used to exchange status and error information between SQL and host language
- Static SQL: programmer uses predefined SQL statements and parameters
  - SQL statements will not change while application is running
- Dynamic SQL: SQL statement is generated at run time
  - Attribute list and condition are not known until end user specifies them
  - Slower than static SQL
  - Requires more computer resources
  - Inconsistent levels of support and incompatibilities among DBMS vendors

```
EXEC SQL
        SQL statement;
END-EXEC.
```