雨课堂
Rain Classroom

本次课程是

线上+线下

融合式教学

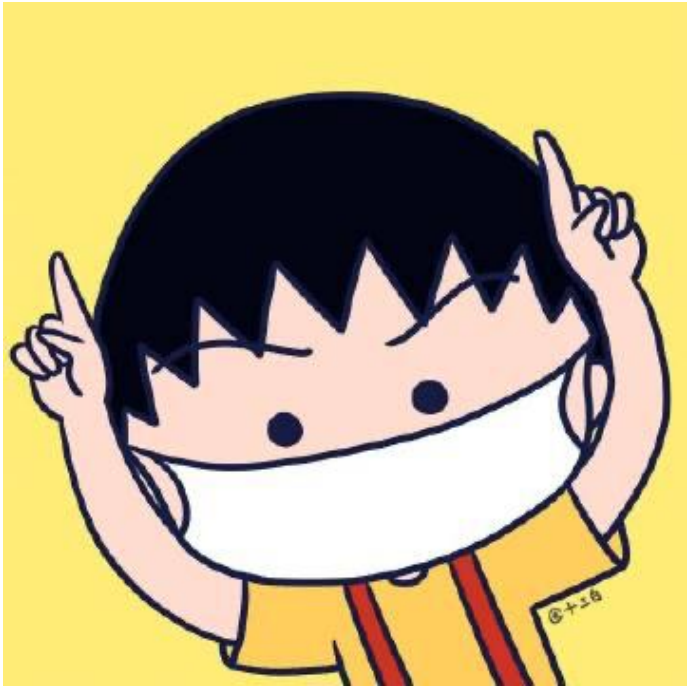
请**现场**的同学们：

1. 打开雨课堂，点击页面右下角喇叭按钮调至静音状态

请**远程上课**的同学们：

1. 打开雨课堂，点击页面右下角喇叭按钮调至静音状态
2. 打开“瞩目”（会议室：182 943 865；密码：见学堂公告），进入会议室，并关闭麦克风

请在教室内佩戴口罩



Database Concepts (III)

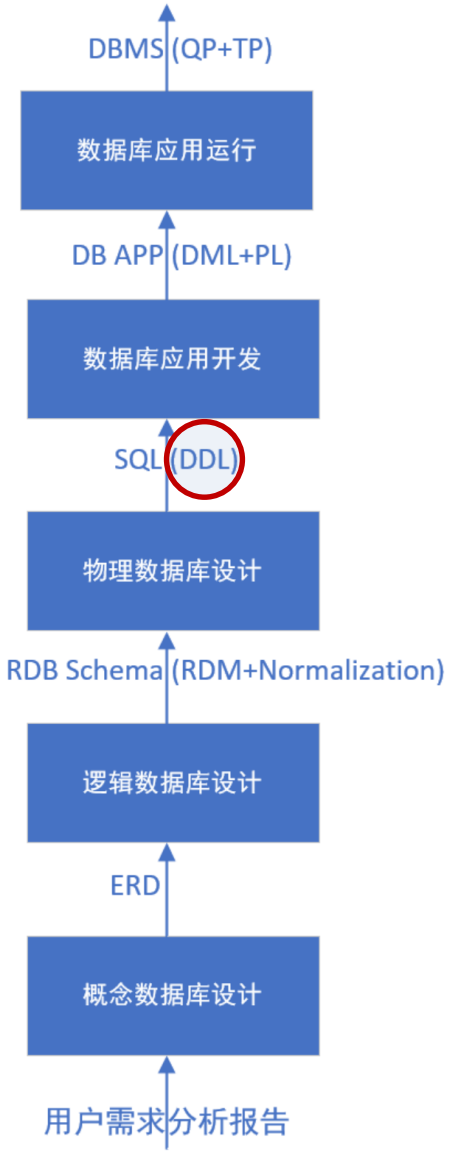
Structured Query Language

Chaokun Wang

School of Software, Tsinghua University

chaokun@tsinghua.edu.cn

April 19, 2022



Outline

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

SQL: Alter Table

相当于对表格进行修补

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

SQL: Alter Table

emp_ssn	emp_name	emp_addr	emp_sal	emp_id
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20
abcd	abcd	abcd	10	20

```
ALTER TABLE emp_test ALTER emp_sal TYPE char(1);
```

错误: 对于字符类型来说这个值太长了(1)

```
ALTER TABLE emp_test ALTER emp_sal TYPE char(2);
```

OK

```
ALTER TABLE emp_test ALTER emp_sal TYPE int4;
```

错误: 字段 "emp_sal" 不能自动转换成类型 integer
HINT: 您可能需要指定"USING emp_sal::integer"。

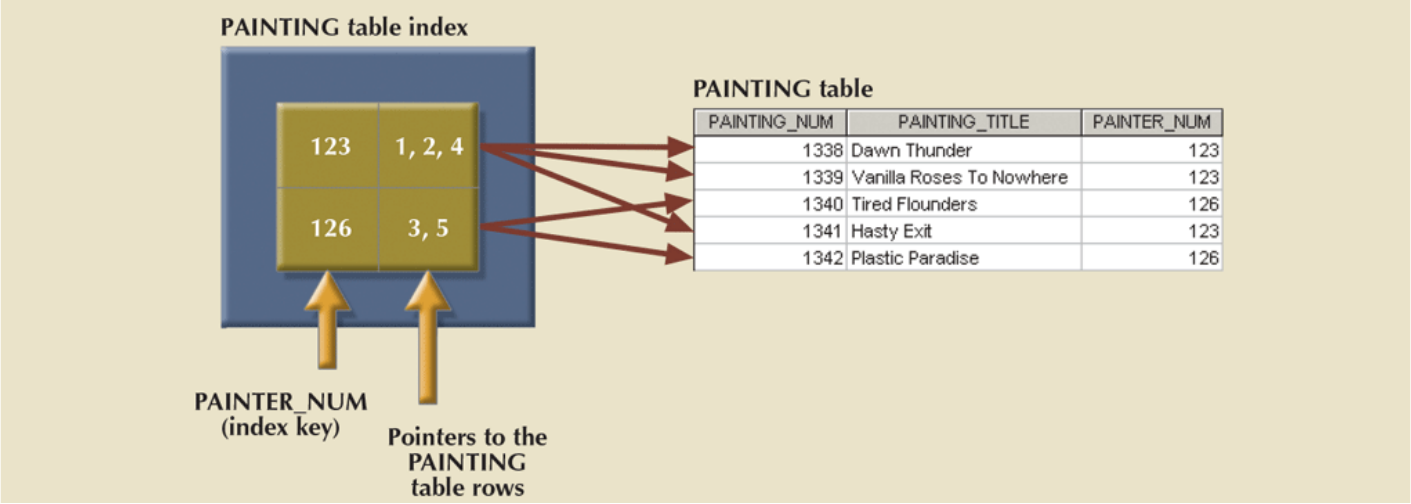
```
ALTER TABLE emp_test ALTER emp_sal TYPE int4 USING emp_sal :: int4;
```

OK

【注意】：
· 该类型是要注意原有数据的size能不能fit进新的类型的规定里。
· 改不好的时候pg的系统有可能会给你一个hint。

Indexes

- Orderly arrangement to logically access rows in a table 建立index以后找对应项就快得多，但是这个是一个类似于“存储空间换时间”的操作
  - Index key: index's reference point that leads to data location identified by the key 对于提高性能非常关键<-在表结构合理的前提下



  - 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

Indexes

```
CREATE TABLE t_test ( ID int4 );
INSERT INTO t_test
    SELECT *  FROM generate_series ( 1, 10000000 );


SELECT  *  FROM   t_test WHERE  ID = 423425;
```

运行时间: 0.512s

这个EXPLAIN就相当于，让系统告诉你 他是打算怎么找你要的信息的

```
EXPLAIN ANALYZE SELECT  *
    FROM  t_test WHERE  ID = 423425;
```

QUERY PLAN
Gather (cost=1000.00..97331.31 rows=1 width=4) (actual time=35.285..477.936 rows=1 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on t_test (cost=0.00..96331.21 rows=1 width=4) (actual time=173.865..312.740 rows=0 loops=3)
Filter: (id = 423425)
Rows Removed by Filter: 3333333
Planning Time: 0.079 ms
Execution Time: 477.973 ms

Indexes

```
CREATE TABLE t_test ( ID int4 );
INSERT INTO t_test
    SELECT *  FROM generate_series ( 1, 10000000 );



CREATE INDEX idx_id ON t_test ( ID );

SELECT  *  FROM   t_test WHERE  ID = 423425;
```

运行时间: 0.004s

```
EXPLAIN ANALYZE SELECT  *
    FROM  t_test WHERE  ID = 423425;
```

QUERY PLAN
Index Only Scan using idx_id on t_test (cost=0.43..4.45 rows=1 width=4) (actual time=0.024..0.025 rows=1 loops=1)
Index Cond: (id = 423425)
Heap Fetches: 0
Planning Time: 0.096 ms
Execution Time: 0.044 ms

Indexes

- It makes sense to inspect the size of the table as well as the size of the indexes

```
SELECT
  pg_size_pretty(pg_relation_size('t_test')) AS size_table_t_test,
  pg_size_pretty(pg_relation_size('idx_id')) AS size_index_idx_id;
```

size_table_t_test	size_index_idx_id
346 MB	214 MB

- Despite the index, PostgreSQL will still use a sequential scan
  - When reading the index and the entire table is a lot more expensive than just reading the table

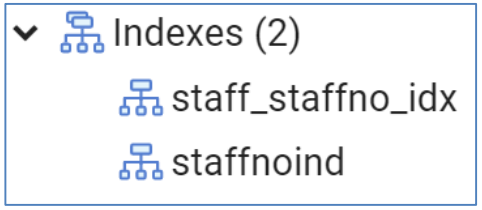
```
CREATE TABLE t_test2 ( id int, x text );
INSERT INTO t_test2
    SELECT x, 'campus'  FROM generate_series ( 1, 10000000 ) AS x;
CREATE INDEX idx_x ON t_test2 ( id );
EXPLAIN ANALYZE SELECT  *
    FROM  t_test2 WHERE x = 'campus';
```

```
QUERY PLAN
Gather  (cost=1000.00..112138.33 rows=50000 width=36) (actual time=0.510..1883.504 rows=10000000 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on t_test2  (cost=0.00..106138.33 rows=20833 width=36) (actual time=0.251..469.045 rows=3333333 loops=3)
        Filter: (x = 'campus'::text)
```

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;

Update Table

```
CREATE TABLE test_staff AS TABLE staff;
```

看一个view的语句也是 SELECT * FROM view_name

staffno	fname	lname	position	sex	dob	salary	branchno
SL21	John	White	Manager	M	1965-10-01	30000	B005
SL41	Julie	Lee	Assistant	F	1985-06-13	9000	B005
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003
SA9	Mary	Howe	Assistant	F	1990-02-19	9000	B007

```
CREATE VIEW manager3staff
   AS SELECT * FROM test_staff WHERE branchNo = 'B003';
```

staffno	fname	lname	position	sex	dob	salary	branchno
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003

```
UPDATE test_staff SET branchno = 'B003' where staffno = 'SL21';
SELECT * from manager3staff;
```

staffno	fname	lname	position	sex	dob	salary	branchno
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003
SL21	John	White	Manager	M	1965-10-01	30000	B003

改变基表的信息的时候，view里的信息会改变

Update View

```
CREATE TABLE test_staff AS TABLE staff;
```

同时改变view的时候，基表里的信息也会改变

staffno	fname	lname	position	sex	dob	salary	branchno
SL21	John	White	Manager	M	1965-10-01	30000	B005
SL41	Julie	Lee	Assistant	F	1985-06-13	9000	B005
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003
SA9	Mary	Howe	Assistant	F	1990-02-19	9000	B007

```
CREATE VIEW manager3staff
   AS SELECT * FROM test_staff WHERE branchNo = 'B003';
```

staffno	fname	lname	position	sex	dob	salary	branchno
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003

```
UPDATE manager3staff SET salary = 16000 WHERE staffno = 'SG37';
DELETE FROM manager3staff WHERE staffno = 'SG14';
SELECT * from test_staff;
```

staffno	fname	lname	position	sex	dob	salary	branchno
SL21	John	White	Manager	M	1965-10-01	30000	B005
SL41	Julie	Lee	Assistant	F	1985-06-13	9000	B005
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003
SA9	Mary	Howe	Assistant	F	1990-02-19	9000	B007
SG37	Ann	Beech	Assistant	F	1980-11-10	16000	B003

Materialized View

```
TRUNCATE test_staff;
INSERT INTO test_staff SELECT * FROM staff;
```

staffno	fname	lname	position	sex	dob	salary	branchno
SL21	John	White	Manager	M	1965-10-01	30000	B005
SL41	Julie	Lee	Assistant	F	1985-06-13	9000	B005
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003
SA9	Mary	Howe	Assistant	F	1990-02-19	9000	B007

```
CREATE MATERIALIZED VIEW manager3staff
   AS SELECT * FROM test_staff WHERE branchNo = 'B003';
```

staffno	fname	lname	position	sex	dob	salary	branchno
SG37	Ann	Beech	Assistant	F	1980-11-10	12000	B003
SG14	David	Ford	Supervisor	M	1978-03-24	18000	B003
SG5	Susan	Brand	Manager	F	1960-06-03	24000	B003

```
UPDATE manager3staff SET salary = 16000 WHERE staffno = 'SG37';
```

错误: 不能改变物化视图 "manager3staff"

如果我不想让view能改变我们基表里面的东西，可以把它设置成物化视图。这时候再改view里面的东西就会报错

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 va 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 | N
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

Transactions

- Transactions 可以说是防止双方同时对数据进行读写
  - 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;
```

Transactions

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

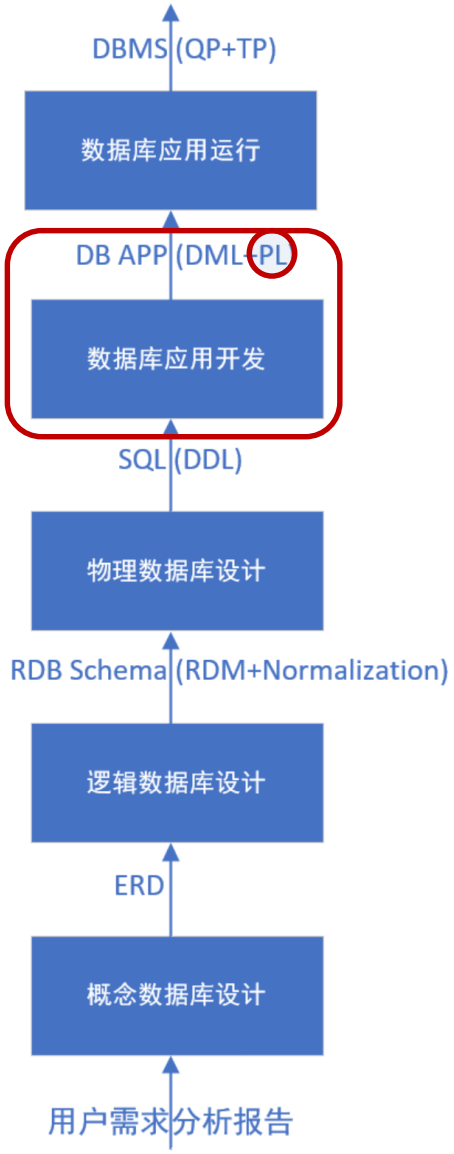
提交的意思是，transaction里面的东西会在数据库中真正实现

```
START TRANSACTION
> OK
> 查询时间: 0s

DROP TABLE IF EXISTS avg_salary
> 注意:  表 "avg_salary" 不存在
> OK
> 查询时间: 0s

CREATE TABLE avg_salary (salary INTEGER)
> OK
> 查询时间: 0.001s

INSERT INTO avg_salary (SELECT AVG(salary) FROM staff)
> Affected rows: 1
> 查询时间: 0s

COMMIT
> OK
> 查询时间: 0.008s

SELECT * FROM avg_salary
> OK
> 查询时间: 0.001s
```

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

ROLLBACK相当于撤销，里面说的都不算

```
START TRANSACTION
> OK
> 查询时间: 0s

DROP TABLE IF EXISTS avg_salary
> 注意:  表 "avg_salary" 不存在
> OK
> 查询时间: 0s

CREATE TABLE avg_salary (salary INTEGER)
> OK
> 查询时间: 0.001s

INSERT INTO avg_salary (SELECT AVG(salary) FROM staff)
> Affected rows: 1
> 查询时间: 0.008s

ROLLBACK
> OK
> 查询时间: 0.001s

SELECT * FROM avg_salary
> 错误:  关系 "avg_salary" 不存在
LINE 1: SELECT * FROM avg_salary
                      ^

> 查询时间: 0s
```



Outline

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

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, 持久存储模块): block of code
  - Contains standard SQL statements and procedural extensions that is stored and executed at the DBMS server
- Procedural Language (过程化语言) 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

Anonymous PL/SQL blocks: Sample (In PG)

DO — execute an anonymous code block

Synopsis

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

Examples

具体的case可以看这节课里的notes
这一部分感觉在大作业里非常有用

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.

Project

- Nowadays, almost everyone has something they don't need. Instead of leaving such unused items undisposed, selling them to those in need is a better choice. In order to build a bridge between the sellers and the buyers, a second-hand goods trading platform is developed to allow the sellers to upload goods and allow the buyers to browse and purchase goods. However, due to some database issues, the platform can not work well. As an outstanding student taking the Database Concepts course, you are invited to design a database for the platform to support its normal operations.

Notes & Tasks

Notes

- For some commercial considerations, the payment function and delivery function of the platform have been simplified.
- Each of the goods can only be bought by one customer.

Tasks

- Design a database for the platform that allows a person to
  - Join the platform with username and password. (Note that a username can only be registered for one person.)
  - Manage the goods
    - Upload goods specified by name, description, image, price and postage;
    - Browse, search, filter and sort goods;
    - Modify goods information; think：怎么更新啊
    - Delete goods uploaded by him/her.

Task

- Manage the orders
  - (Buyers) Apply to buy goods;
  - (Sellers) Approve the applications;
  - (Buyers) Provide delivery address for an order and pay for the goods;
  - (Sellers) Deliver the goods, and provide the name of the courier services company as well as the courier number;
  - (Buyers) Confirm receiving the goods and finish the order.
- Manage the comments
  - Comment on goods;
  - Browse comments;
  - Delete comments posted by him/her.
- Manage the delivery address
  - Add a new delivery address;
  - Browse his/her delivery addresses;
  - Delete his/her delivery addresses.

Task

- Given a semi-finished platform and a physical model of its database,
  - Give the DDL statements for the physical model;
  - Create the database in a PostgreSQL server;
  - Implement the 25 interfaces of the platform to make it work;
  - Optimize the platform in any way (e.g., pagination).

Example

- Here is an example for one of the database interfaces.

```
def create_user(username, password):
    """
    功能：在数据库中创建用户，用于实现注册功能。
    分值：2分
    :param username: 用户名称
    :param password: 用户明文密码，建议存储时使用md5加密
    :return: 如果创建成功，返回True；否则返回False。
    """
    # TODO：请实现该函数的功能。
    print(username, password)
    return True
```

Deliverables

- ERD and relational schemas (15')
- Source code (75')
  - DDL statements (5')
  - Database interface implementation for
    - User module: 4 interfaces (2 + 2 + 2 + 3 = 9')
    - Goods module: 6 interfaces (2 + 2 + 5 + 3 + 2 + 3 = 17')
    - Order module: 9 interfaces (2 + 3 + 3 + 2 + 2 + 2 + 2 + 4 + 5 = 25')
    - Comment module: 3 interfaces (2 + 3 + 2 = 7')
    - Address module: 3 interfaces (2 + 3 + 2 = 7')
  - Any optimization (5')

Deliverables

- Documentation (10')
  - System flowchart
  - Details on the database interface implementation and any optimization
  - Screenshots on
    - Operations on the platform
    - Schemas in PostgreSQL
  - Summary
  - Suggestions w.r.t. the course work and the class

Important Dates

- ERD and relational schemas due
  - **April 29, 2022** (Friday)
- Source code and documentation due
  - **June 12, 2022** (Sunday)

Conclusions

- Alter Table
- Indexes
- Views
- Sequences
- Transactions
- Procedural SQL
- Anonymous PL/SQL blocks
- Project
  - Note
  - Task
  - Deliverables
  - Important Dates

Homework

- Read Chapter 8 of DS1
  - § 8.5 (pp 387 - 391)
  - § 8.6 (pp 391 - 396)
  - § 8.7 (pp 396 - 400)
- Project

Thank you!