**数据库性能**

**度量指标:**

- 事务吞吐量(Transaction throughput): 单位时间中能够处理的交易(事务)数量.

- 响应时间(Response time): 完成单个交易所

**提高数据库性能的策略**

1挖掘和利用：

2 数据特性，

3 硬件特性，

4 以及数据访问特性

**提高数据库性能的方法**

方法1: 排序;

• 方法2: 索引（哈希索引）；

• 方法3: 连续的磁盘存储; (把经常要做联接运算的表中相应记录存储在一起)

• 方法4: 分类、聚簇;

• 方法5: 减少联接;

• 方法6: 内存缓冲;

• 方法7: 并发执行;

• 方法8: 查询优化;

• 方法9: 日志和数据分盘存储;(单独、专用的日志磁盘)

其中与数据库设计有关的方法 1 2 3 4 5 6 9

**Approach 2.create Index guideline**

CREATE INDEX studentIndex ON student(dno, name);

(1) 不要对数据量少的表创建索引;

(2) Add secondary index to a FK if it is frequently

accessed.

(3) Add secondary index on attributes that are involved

in: heavily selection or join criteria; ORDER BY; GROUP BY;

and other operations involving sorting (such as UNION or

DISTINCT).

(4) Avoid indexing attributes that consist of long

character strings.

**Approach 4: Consecutive Disk Space**

CREATE TABLESPACE basicdata DATAFILE

‘c:\oracle\data\basicdata.dbf’ SIZE 100M;

CREATE TABLE employee (

eno CHAR(5) NOT NULL,

ename VARCHAR(30),

PRIMARY KEY (eno),

FOREIGN KEY (dno) REFERENCES Dept(dno),

TABLESPACE basicdata,

STORAGE (INITIAL 10M NEXT 10M)

)

**事务处理故障恢复**

**事务概念:**

数据库事务(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列操作。 事务处理可以确保除非事务性单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的单元，可以简化错误恢复并使[应用程序](http://baike.baidu.com/view/330120.htm)更加可靠。一个逻辑工作单元要成为事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。事务通常是以BEGIN TRANSACTION开始，以COMMIT或ROLLBACK结束

Spl写法：

**TRANSACTION BEGIN**

UPDATE account SET balance = balance +

10000 WHERE accountNo =‘2008043101’;

UPDATE account SET balance = balance -

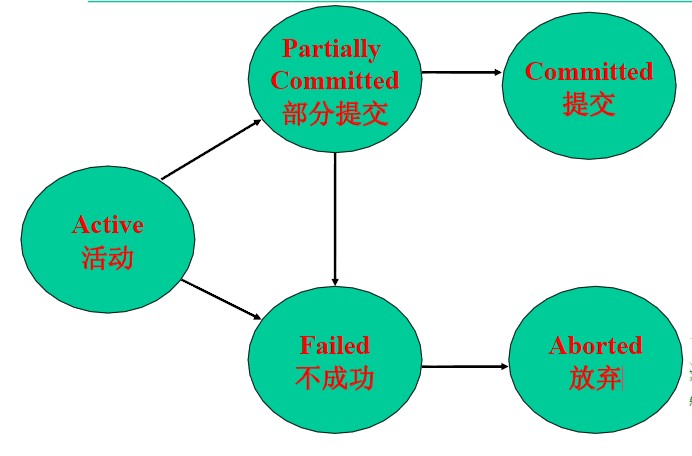
10000 WHERE accountNo =‘2008043214’;

**COMMIT**

**事务特征：**事务对数据库的一组操作，这些操作涉及对多个数据项进行更新/修改

**要求**：中间状态对外部不可见；

**事务的五个状态**



**事务的ACID属性**

**原子性（Atomicity）：** 一个事务中的操作要求要么全部执行，要么全部不执行.

**一致性（Consistency）：** 在外部看来，数据库中的数据总是正确的.

**隔离性（Isolation）：** 尽管多个事务在并发执行，但从外部看来，具有多个事务串行执行的效果；

**持久性（Durability）**：一个事务一旦提交了，即使随后发生故障，其结果在数据库中不会丢失；

**系统故障**

**• 事务故障(Transaction failure) :**

* **逻辑错误(Logical errors)**：事务由于某些内部条件无法继续正常执行。例如: divided by 0.余额不允许为负；
* **系统错误：**系统进入一种不良的状态（如死锁），结果事务无法继续正常执行。但该事务可以再以后的某个时间重新执行、

• **系统崩溃故障(System crash):**硬件故障，或者是数据库软件或操作系统的漏洞，导致易失性存储器内容丢失，并使得事务可以再以后的某个时间重新执行。

– 停电、硬件故障；

• **磁盘故障(Disk failure**)在数据传送操作过程中由于磁头损坏或故障造成磁盘快上内容丢失。

**• 通信故障(Communication failures)**

**故障恢复方法**

• 在有故障可能的情况下，如何保证事务的四

个属性；

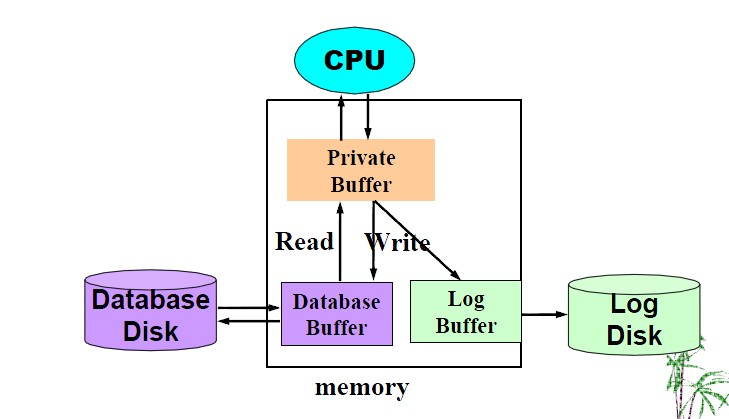
• 它包括2个部分:

1.正常执行时的防备措施，为故障恢复做准备；

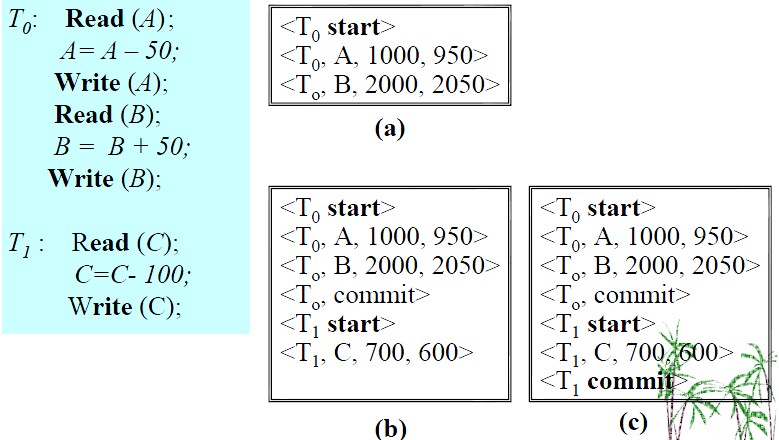
2. 在故障发生后的故障恢复措施，保证事务的四个属性；

**第一种方法：分页方法（没实用）**

**第二种方法：日志方法(被广泛采用）**日志的特点：单调递增；



**日志方法举例：**



**事务故障的恢复**

• 特征：事务不能执行下去，未完成；

• 恢复方法：执行Rollback：读取日志文件，找到该事物的<Ti start> 记录然后顺着扫描日志记录，对Ti 的记录执行undo(Ti) 操作，使用旧值恢复数据项的原有值，即撤销事务所做的数据操作；

**系统崩溃故障的恢复**

• 重启数据库管理系统；

• 从日志磁盘读取日志文件；

• 反向扫描日志，也就是从日志文件的结束位置开始后向扫描。

当发现一个事务在日志记录中没有<Ti commit> 记录时，执

行undo( Ti) 操作，使用旧值恢复数据项:

• 然后从日志文件的开始位置顺向扫描。对日志记录中含有<Ticommit>的事务，执行redo( Ti) 操作，使用新值恢复数据项；

**有检查点时的故障恢复**

• 从日志文件的结尾处逆向扫描，直到<checkpoint> 记录；

• 继续逆向扫描直到找到一条<Ti start> 记录，

• 对逆向扫描中只有<Ti start> 对没有<Ti commit>的事务执行undo(Ti)操作；

• 前向扫描对对有<Ti commit>的事务执行redo(Ti)操作；

• 检查点只是加快了故障恢复过程，缩短了恢复时间；

**Dump操作－应对数据库磁盘故障**

• 周期性地执行dump 操作，对磁盘数据库进行磁盘备份；

• 做dump操作时，要：

不再接收事务请求，让当前的所有活动事务执行完毕；

– 输出日志缓冲区中的日志记录到日志磁盘中;

– 输出数据库缓冲区中的缓冲数据到数据库磁盘中;

– 把磁盘中的数据库文件拷贝到另一个磁盘上；

* 往日志磁盘中写入一条<dump> 日志记录；
* **数据库磁盘故障时的恢复过程**
* 把dump的备份数据库文件拷贝到新的数据库磁盘上；
* 从日志文件的末尾逆向扫描直至<dump>记录，
* 再顺向扫描日志记录，对有<Ti commit>日志记录的事务做redo(Ti)操作；.

**日志磁盘故障时的恢复过程**

• 不再接收事务请求，让当前的所有活动事务执行完毕；

• 输出数据库缓冲区中的缓冲数据到数据库磁盘中

(即checkpoint）;

• 执行Dump操作，把磁盘中的数据库文件拷贝到另一个磁盘上；

• 更换日志磁盘；

执行Dump操作时，可以恢复正常事务处理

**数据安全**

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。

**完整性，隐藏，不可抵赖性**

**完整性：用户，权限，对象**

数据库的**完整性**是指保护数据库数据的正确性和一致性。它反映了现实中实体的本来面貌。数据库系统要提供保护数据完整性的功能。系统用一定的机制检查数据库中的数据是否满足完整性约束条件。

应用程序必须对用户进行进行身份验证，并确保用户只允许完成授权任务。在应用级别，授权用户访问特定接口，并可能进一步被限制只能查看或更新某些数据项。

**特权**

**特权**是执行一种特殊类型的SQL语句或存取另一用户的对象的权力。有系统特权和对象特权两类特权。

**角色**

角色是相关特权的命名组。数据库系统利用角色可更容易地进行特权管理。

**隐藏性：**数据加密

对称密匙，加密密匙也用于解密数据。

非对称密匙：一个共匙，一个私匙。

**不可抵赖性：**

**审计追踪**：是关于应用程序数据的所有更改（插入/删除/更新）的日志，以及一些信息。如哪个用户执行了更改和什么时候执行的更改。如果应用程序安全被破坏，或者即使安全性没有被破坏但是是一些更新错误执行，一个审计追踪能够（a）

帮助找出发生了什么，以及可能是由谁执行的操作，并（b）帮助修复安全漏洞或错误更新造成的损害。

**数字签名：**私匙用来对数据“签名”，即加密，且签名后的数据可以公开。所有人都可以通过用共匙解密数据来验证签名，但没有私匙的人无法生成签名的数据。，这样就可以认证该数据；也就是我们可以验证数据确实是由声称创建这些数据的人所创建。

另外，数字签名也可以用来确保认可。也就是，一个人创建数据过后声称她没有创建它得情况下，我们可以证明那个人肯定创建该数据。

**数字证书：**

一个验证机构签署证书以证明公钥的真实性。

**实现数据库安全性控制的常用方法和技术有：**

**安全性**问题主要是指保护数据库以防止不合法的使用造成数据泄露、更改或破坏。

数据库安全可分为二类 ：（1）系统安全性：指在系统级控制数据库的存取和使用的机制。（2）数据安全性：指在对象级控制数据库的存取和使用的机制。

**1用户标识和鉴定**：通过核对用户的名字或身份(ID)，决定该用户对系统的使用权。数据库系统不允许一个未经授权的用户对数据库进行操作。

**2.身份验证：**应用程序必须对用户进行进行身份验证，并确保用户只允许完成授权任务。在应用级别，授权用户访问特定接口，并可能进一步被限制只能查看或更新某些数据项。

**3.VPN** :VPN通过建立一个隧道，利用加密技术对传输数据进行加密，以保证数据的私有性和安全性,可以为不同要求用户提供不同等级的服务质量保证。

**4.数字签名**：私匙用来对数据“签名”，即加密，且签名后的数据可以公开。所有人都可以通过用共匙解密数据来验证签名，但没有私匙的人无法生成签名的数据。，这样就可以认证该数据；也就是我们可以验证数据确实是由声称创建这些数据的人所创建。另外，数字签名也可以用来确保认可。也就是，一个人创建数据过后声称她没有创建它得情况下，我们可以证明那个人肯定创建该数据。

**5.访问控制**：按用户[身份](http://baike.baidu.com/view/1034208.htm)及其所归属的某项定义组来限制用户对某些信息项的访问，或限制对某些控制功能的使用。访问控制通常用于[系统管理员](http://baike.baidu.com/view/83228.htm)控制用户对服务器、目录、文件等[网络资源](http://baike.baidu.com/view/21050.htm)的访问。 　　

**6.存取控制模型**（1）访问控制矩阵模型（2）扩展的访问控制矩阵模型（3）多级安全模型（4）信息流安全模型

**7.视图** ：有了视图机制，就可以在设计数据库应用系统时，对不同的用户定义不同的视图，使机密数据不出现在不应看到这些数据的用户视图上。即通过定义不同的视图及有选择地授予视图上的权限，可以将用户、组或角色限制在不同的数据子集内。

**8.加密存储：**（1）数据库密码系统的选择:可采用对称加密体制。 （2）数据库加密的范围，数据库中不能加密的部分包括：索引字段、关系运算的比较字段 、表间的连接码字段 。 （3）数据库加密对数据库管理系统原有功能的影响

**9.防火墙**

**10.IDS/IPS:**IPS位于防火墙和网络的设备之间。这样，如果检测到攻击，IPS会在这种攻击扩散到网络的其它地方之前阻止这个恶意的通信。IDS存在于网络之外起到报警的作用

**数据库的安全主要受到以下因素的威胁：**

1．数据错误2．软硬件故障与灾害破坏3．缺乏保护机制4．管理漏洞

5．黑客攻击6．不掌握数据库核心技术7．敏感数据的泄漏问题

**数据库安全的重要性：**1．保护敏感信息和数据资产2．计算机网络系统安全的关键环节

**数据库的安全目标：**1完整性：2可信性：3系统灵活性：4用户方便性：5篡改检测：6降低管理成本：

**数据库的安全策略：**是指导信息安全的高级准则，即组织、管理、保护和处理敏感信息的法律、规章及方法的集合。它包括安全管理策略、访问控制策略和信息控制策略。

**完整性，隐藏，不可抵赖性**

**完整性：用户，权限，对象**

数据库的完整性是指保护数据库数据的正确性和一致性。它反映了现实中实体的本来面貌。数据库系统要提供保护数据完整性的功能。系统用一定的机制检查数据库中的数据是否满足完整性约束条件。

**特权**是执行一种特殊类型的SQL语句或存取另一用户的对象的权力。有系统特权和对象特权两类特权。

**角色** 角色是相关特权的命名组。数据库系统利用角色可更容易地进行特权管理。

**隐藏性：**数据加密

对称密匙，加密密匙也用于解密数据。非对称密匙：一个共匙，一个私匙。

**可审计性：**为了能够跟踪对数据库的访问，及时发现对数据库的非法访问和修改，需要对访问数据库的一些重要事件进行记录，利用这些记录可以协助维护数据库的完整性，还可以帮助事后发现是哪一个用户在什么时间影响过哪些值。

**分布式数据库系统中的故障恢复**

提交协议（协同）:两阶段提交协议（**2PC**）：线性2PC，分布式2PC

协调者，参与者

1 : 协调者收集参与者执行子事务的结果；Phase 2 : 全体成员对事务结果作出一致的处理；

**全局提交规则**：全票通过则提交，一票否决则放弃

**终止协议**（非阻塞——理想状态）：协调者，参与者

参与者在Ready状态，没有收到协调者的最终结果消息，超时，有2种情况：

单故障：只有协调者故障；多故障：协调者和参与者同时故障，须进一步判别最终结果；

**恢复协议**（独立恢复——理想状态）系统重启，检查日志文件。有五种可能的情况，其中四种情况，可独立恢复，有一种情况需要咨询协同者来判定最终结果消息。

**3PC协议**

复制：提高性能，提高可用性，最直观的做法是ROWA（read one,write all）

复制中的团组协议（Quorum Protocol）

每个副本*C*i拥有票数 *V*i ，所有副本: Σ *V*i= *V*

对一个数据项执行读操作时，需要获得一个读团组( *read quorum) V*r ;

对一个数据项执行写操作时，需要获得一个读团组(*write quorum) V*w ;

规则：

*V*r + *V*w > V :不能由两个事务同时对一个数据项执行读操作和写操作；

*V*w > V/2 :不能由两个事务同时对一个数据项执行写操作；

**数据正确性问题**

数据正确性的威胁来自两个方面：

1.正常运行时可能遇到的数据丢失、数据冗余、数据不正确、数据不一致等异常情况；

2.故障（包括软件和硬件），或者发生停电之类的外部异常事件时，导致的数据丢失、数据不一致；

**如何来解决？**

1.正常运行时的数据正确性问题要通过数据库设计来解决。数据库设计理论、方法是本课程的核心内容。

2.企业中，数据丢失是不允许的，同时系统故障又是难以避免的。事务处理（Transaction processing)和故障恢复。

**数据操作简单性问题**

1.数据操作的简单性可通过视图（View）和存储过程来提升。

2.视图概念将数据操作的复杂性从普通用户转移动至数据库专业人员，为普通用户屏蔽了表与表的联接运算以及复杂查询条件的表达；

3.而存储过程则将数据操作语句转化为函数调用形式，进一步为数据库普通用户屏蔽了数据操作语言的语法规则

视图：

**CREATE VIEW *SoftwareStudent* AS SELECT**

存储过程**CREATE PROCEDURE AddStudent(@studentNo IN student.studentNo%TYPE, @name IN VARCHAR）AS**

**BEGIN INSERT INTO student(studentNo, name) VALUES(@ studentNo, @name); END;**

**数据库设计人员（database designer)；**

数据库系统管理人员DBA (database Administrator)；负责数据库系统的日常维护，确保数据库的安全、可靠、可用具体职责包括:

数据库服务器的建立和配置;用户管理;安全管理;数据库访问审计;

系统扩展与升级;数据备份，故障恢复;性能优化。

**数据库应用程序开发人员(Application Developer)**：主要工作是满足数据库用户的业务需求，为他们设计和开发出数据库应用程序

数据库应用程序设计与开发中，首要的**宗旨**有四条：

在**界面方面**，要求数据组织结构清晰，层次分明，表达简洁，符合习惯，操作简单方便；

在**程序方面**，要模块化，层次化；

在**可靠性**和**稳定性方面**，凡是跨进程的调用都要检查是否成功，不成功时要通告用户，并作恰当处理；

在**通用性方面**，尽量采用标准技术，通用架构，符合标准规范，少用专有的东西。

**数据库设计要回答的三个问题：**

1) 覆盖性问题：确定一个单位有哪些数据项（字段）？

2） 划分问题： 确定哪些字段构成一个表？ 3） 关系问题： 确定表之间有什么关系？

**特征：**业务表单和数据库中表的不一致；

业务表单中通常包含的是综合信息；而数据库中的一个表只能存储单一类的信息；

如果直接把业务表单中的数据项组成一个表，会带来一系列的数据正确性问题；

**分布式数据库的定义 ：**

物理上分散而逻辑上集中的系统，它使用计算机网络将地理位置分散而管理和控制又需要不同程度集中的多个逻辑单位（通常是集中式数据库系统）连接起来，共同组成一个统一的数据库系统。分布式数据库系统可以看成是计算机网络和数据库系统的有机结合。

**简述分布式数据库系统的透明性：**1分片透明性（完全分布透明性）：2位置透明性（中级分布透明性）：3局部数据模型透明性（低级分布透明性）：4无分布透明性（异构数据）：

**优点：** (1) 具有灵活的体系结构 。 (2)适应分布式的管理和控制机构。

　　 (3)经济性能优越 。 　　 (4)系统的可靠性高、可用性好 。

　　 (5)局部应用的响应速度快。 (6)可扩展性好，易于集成现有系统。

**缺点：** (1)[系统开销](http://baike.baidu.com/view/860875.htm)大，主要花在通信部分。

　　 (2)复杂的存取结构，原来在[集中式系统](http://baike.baidu.com/view/3495751.htm)中有效存取数据的技术，在分布式系统中都不再适用。 　　 (3)数据的安全性和保密性较难处理。

**分布式系统**：共赏资源，性能，容错

**分布式计算：**沟通语言：XML沟通方式：HTTP

**Web service**SOAP：简单对象访问协议，简单对象访问协议（SOAP）是一种轻量的、简单的、基于 XML 的协议，它被设计成在 WEB 上交换结构化的和固化的信息。

*UDDI*，英文为 "Universal Description, Discovery and Integration"，可译为“通用描述、发现与集成服务

Web Services Description Language的缩写，是一个用来描述Web服务和说明如何与Web服务通信的XML语言。

DDB分布式数据库

**数据库的完整性**

数据库的完整性是指数据的正确性和相容性。

数据的完整性和安全性是两个不同的概念。前者是为了防止数据库中存在不符合语义的数据，防止错误信息的输入和输出，而后者是保护数据库防止恶意的破坏和非法的存取。

1．设置触发器 。触发器可以完成以下功能： “条件限制”

（1）检查取值类型与范围：检查每个字段输入数据的类型与该字段的类型是否一致 ；（2）依据状态限制：指为保证整个数据库的完整性而设置的一些限制，数据库的值在任何时候都不应该违反这些限制；（3）依据业务限制：指为了使数据库的修改满足数据库存储内容的业务要求，而作出相应的限制。

2．两阶段提交 ：为了保证数据更新结果的正确性，必须防止在数据更新过程中发生处理程序中断或出现错误。解决这个问题的办法是在DBMS中采用两阶段提交(更新)技术。第一阶段称为准备阶段。第二阶段的工作是对需要更新的字段进行真正地修改，这种修改是永久性的。

3．纠错与恢复：许多DBMS提供数据库数据的纠错功能，主要方法是采用冗余的办法，下面介绍几种冗余纠错的技术 ：（1）附加校验纠错码（2）使用镜像技术（3）恢复

**数据库的并发控制**

数据库系统通常支持多用户同时访问数据库，为了有效地利用数据库资源，可能多个程序或一个程序的多个进程并行地运行，这就是数据库的并发操作。当多个用户同时读写同一个字段的时候，会存取不正确的数据，或破坏数据库数据的一致性。据不一致总是由两个因素造成：一是对数据的修改，二是并发操作的发生。并发操作带来的数据不一致性包括三类： 1．丢失修改2．不可重复读3．读“脏”数据

**SQl语句**

**删表**： drop table stu\_2012;

**创建表：**create table STU\_2012

(

sno char(12),

name varchar2(12) not null,

degree number,

约束的名字

score number,

constraint primary\_key primary key(sno ),

constraint c\_degree check(degree in ('1','2'))

);

**改变表**：alter table STU\_2012

add constraint C\_score check(score between 0 and 100);

alter table STU\_2012

add birthday date;

alter table stu\_2012

add(a number,n clob);

alter table stu\_2012

add(c blob);

alter table stu\_2012

drop(A,N,B,C);

**插入数据到表：**insert into STU\_2012 (sno, name, degree, birthday, score)

values ('S12103002', '张亚如', 1, to\_date('24-09-1997 17:14:00', 'dd-mm-yyyy hh24:mi:ss'), 71.1);

**复制表:**create table tmp\_stu\_2012

as

select \*

from stu\_2012;

**只复制结构：**create table tmp2\_stu\_2012

as

select \*

from stu\_2012

where 1=2;

**数表：**elect 3+5 as 结果

from dual

**删出数据**：delete from stu\_2012

where sno='S1224W167';

**更新表：**update stu\_2012

set score=(select round(avg(score)+5,1) from stu\_2012)

where sno='S1224W145';

**更新表同时更新两个数据：**

➀：update stu\_2012

set degree=1,

score=(select round(avg(score)+5,1) from stu\_2012 )

where sno='S1224W145';

➁：update stu\_2012

set (degree,score)=(select 2,round(avg(score)+5,1) from stu\_2012 )

where sno='S1224W145';

**查询：**select degree as 专业方向， avg(score) as 平均分数-----聚合函数还有max,min,sum,count

from stu\_2012

group by degree;

只能出现聚合函数和按排名的列

（分组）

select degree as 专业方向， avg(score) as 平均分数

from stu\_2012

group by degree

having avg(score）>=75;-----having的运用

**With：**with xyz as(

select degree as 专业方向， avg(score) as 平均分数

from stu\_2012

group by degree

)

select 专业方向， 平均分数

from xyz

where 平均分数>=75；

select \* from

(select degree 专业, avg(score) 平均分数

from stu\_2012

group by degree

) **--子查询**

where 平均分数>=75;

select sno as 学号，name as 姓名，degree as 专业，

score-（select avg（score） from stu\_2012 where degree=s.degree) 与专业平均分之差 ,

score-（select avg（score） from stu\_2012 ) 与年纪平均分之差

from stu\_2012 as s;

**排名：**select \* from

（select sno 学号，name 姓名，score 成绩，

rank() over(order by score desc) 名次

from stu\_2012）

where 名次<=10;

select sno 学号，name 姓名，score 成绩，degree 专业，

rank() over(order by score desc) 名次,

rank() over(partition by degree order by score desc) 年纪排名-----分区排名

from stu\_2012;

Like: 模糊查询

%:包含

\_:包含一个字符

[]: 指定范围（如 [ a-z]）

[^]: 不属于指定的范围（如[^a-z]）;

Natural join 自然连接两个表和 where s.id=t.id功能类似

**随机找五个人**

select \*

from stu\_2012

order by dbms\_random.value();

select \* from

(select \*

from stu\_2012

order by dbms\_random.value()

)

where rownum<=5;

**创建视图**

create view v\_5people

as

select \* from

(select \*

from stu\_2012

order by dbms\_random.value()

)

where rownum**<=5;**

表示列的数量

*/\*case when then*

*when then*

*else end case \*/*

create view cvv1\_5people(学号,姓名,成绩)

as

select sno, name,

case when score>=90 then '优'

when score<90 and score>=80 then '良'

**使用视图：**

select \*

from cvv1\_5people

when score<80 and score>=70 then '中'

else '一般' end case from

（ select \*

from stu\_2012

order by dbms\_random.value()

）

where rownum<=5;

**函数**

select round(123.56,1),round(123.56,-1),round(125.56,-1)

from dual;

*-------------四舍五入*

select greatest

(1,2,3,44,12,31,53,66,-3,-21,668) 最大

from dual;

select least

(1,2,3,44,12,31,53,66,-3,-21,668) 最小

from dual;

select sign(-12),sign(25),sign(0)

from dual;*---------取符号*

*---------字符串*

select length('human'||'university') 字符串长度

from dual;

select substr('human'||'university',3,5) 子串

from dual;

*/\*substr('This is a test', 6, 2) would return 'is'*

*substr('This is a test', 6) would return 'is a test'*

*substr('TechOnTheNet', -3, 3) would return 'Net'*

*substr('TechOnTheNet', -6, 3) would return 'The'\*/*

select rownum 记录号,a.\*

from stu\_2012 a;

select lpad(to\_char(rownum),3,'0') 记录号,a.\*

from stu\_2012 a;

*---lpad(字段名,填充长度,填充的字符)*

select replace('hunam university','nan','bei')

from dual;

varchar(8) char(8)

select sysdate from dual

where trim('yyyyy')=ltrim((rtrim ('yyyyy ')));

*/\*. LTRIM(string,trim\_set)*

*从左边删除字符，此处“string”是数据库的列，或者是字面字符串，而“trim\_set” 是我们要去掉的字符的集合。*

*示例 SELECT LTRIM('abcdab','a') FROM DUAL;\*/*

SELECT LTRIM('abcdab','a') FROM DUAL;

*/\* RTRIM(string,trim\_set)*

*从右侧删除字符，此处“string”是数据库的列，或者是字面字符串，而“trim\_set” 是我们要去掉的字符的集合。*

*示例 SELECT RTRIM('abcdef', 'f') FROM DUAL;\*/*

SELECT RTRIM('abcdef', 'f') FROM DUAL;

*/\*Trim()*

*功能删除字符串首部和尾部的空格。*

*语法Trim ( string )*

*参数string：string类型，指定要删除首部和尾部空格的字符串返回值String。函数执行成功时返回删除了string字符串首部和尾部空格的字符串，发生错误时返回空字符串（""）。如果任何参数的值为NULL，Trim()函数返回NULL。*

*========================================================================*

*SQL 中的 TRIM 函数是用来移除掉一个字串中的字头或字尾。最常见的用途是移除字首或字尾的空白。这个函数在不同的资料库中有不同的名称：*

*MySQL: TRIM(), RTRIM(), LTRIM()*

*Oracle: RTRIM(), LTRIM()*

*SQL Server: RTRIM(), LTRIM()*

*各种 trim 函数的语法如下：*

*TRIM([[位置] [要移除的字串] FROM ] 字串): [位置] 的可能值为 LEADING (起头), TRAILING (结尾), or BOTH (起头及结尾)。 这个函数将把 [要移除的字串] 从字串的起头、结尾，或是起头及结尾移除。如果我们没有列出 [要移除的字串] 是什么的话，那空白就会被移除。*

*LTRIM(字串): 将所有字串起头的空白移除。*

*RTRIM(字串): 将所有字串结尾的空白移除。 \*/*

SELECT TRIM(' abcdefa ') , LTRIM(' abcdefa '), RTRIM(' abcdefa ')

FROM DUAL;

*--date*

system*--系统时间*

select sysdate from dual;

select to\_date('1998.02', 'yyyy.mm') from dual;

select last\_day(sysdate) from dual;

*/\*last\_day是个函数，返回某个时间值的当月最后一天*

*函数： 返回特定日期所在月份的最后一天*

*返回特定日期所在月份的最后一天*

*例： 　　SQL>SELECT last\_day(sysdate) FROM dual; 　　LAST\_DAY(SYSDATE)*

*例如：找出每个月倒数第三天入职的员工*

*SELECT \* FROM EMP WHERE last\_day(hiredate)-2=hiredate;*

*\*/*

select to\_date('1997.02','yyyy.mm'),

last\_day(to\_date('1997.02','yyyy.mm')),

sysdate,sysdate-1

from dual;

select add\_months(sysdate,2), to\_char(sysdate, 'yyyy/mm/dd hh24:mi:ss d ddd')

from dual;

*/\*d 一周的第几天*

*ddd 年初到现在多少天\*/*

*/\**

*The following are number examples for the to\_char function.*

*to\_char (1210.73, '9999.9') would return '1210.7'*

*to\_char (1210.73, '9,999.99') would return '1,210.73'*

*to\_char (1210.73, '$9,999.00') would return '$1,210.73'*

*to\_char (21, '000099') would return '000021'*

*\*/*

create table EMP

(

empno NUMBER(4) not null,

ename VARCHAR2(10),

job VARCHAR2(9),

mgr NUMBER(4),

hiredate DATE,

sal NUMBER(7,2),

comm NUMBER(7,2),

deptno NUMBER(2)

)

;

alter table EMP

add constraint PK\_EMP primary key (EMPNO);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7369, 'SMITH', 'CLERK', 7902, to\_date('17-12-1980', 'dd-mm-yyyy'), 800, null, 20);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7499, 'ALLEN', 'SALESMAN', 7698, to\_date('20-02-1981', 'dd-mm-yyyy'), 1600, 300, 30);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7521, 'WARD', 'SALESMAN', 7698, to\_date('22-02-1981', 'dd-mm-yyyy'), 1250, 500, 30);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7566, 'JONES', 'MANAGER', 7839, to\_date('02-04-1981', 'dd-mm-yyyy'), 2975, null, 20);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7654, 'MARTIN', 'SALESMAN', 7698, to\_date('28-09-1981', 'dd-mm-yyyy'), 1250, 1400, 30);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7698, 'BLAKE', 'MANAGER', 7839, to\_date('01-05-1981', 'dd-mm-yyyy'), 2850, null, 30);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7782, 'CLARK', 'MANAGER', 7839, to\_date('09-06-1981', 'dd-mm-yyyy'), 2450, null, 10);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7788, 'SCOTT', 'ANALYST', 7566, to\_date('19-04-1987', 'dd-mm-yyyy'), 3000, null, 20);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7839, 'KING', 'PRESIDENT', null, to\_date('17-11-1981', 'dd-mm-yyyy'), 5000, null, 10);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7844, 'TURNER', 'SALESMAN', 7698, to\_date('08-09-1981', 'dd-mm-yyyy'), 1500, 0, 30);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7876, 'ADAMS', 'CLERK', 7788, to\_date('23-05-1987', 'dd-mm-yyyy'), 1100, null, 20);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7900, 'JAMES', 'CLERK', 7698, to\_date('03-12-1981', 'dd-mm-yyyy'), 950, null, 30);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7902, 'FORD', 'ANALYST', 7566, to\_date('03-12-1981', 'dd-mm-yyyy'), 3000, null, 20);

insert into EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

values (7934, 'MILLER', 'CLERK', 7782, to\_date('23-01-1982', 'dd-mm-yyyy'), 1300, null, 10);

select \*

from EMP

select level

from dual

connect by level <=100;

*/\*oracle中 connect by prior 递归算法*

*Oracle中start with...connect by prior子句用法 connect by 是结构化查询中用到的，其基本语法是：*

*select ... from tablename start with 条件1*

*connect by 条件2*

*where 条件3;*

*例：*

*select \* from table*

*start with org\_id = 'HBHqfWGWPy'*

*connect by prior org\_id = parent\_id;*

*简单说来是将一个树状结构存储在一张表里，比如一个表中存在两个字段:*

*org\_id,parent\_id那么通过表示每一条记录的parent是谁，就可以形成一个树状结构。*

*用上述语法的查询可以取得这棵树的所有记录。*

*其中：*

*条件1 是根结点的限定语句，当然可以放宽限定条件，以取得多个根结点，实际就是多棵树。*

*条件2 是连接条件，其中用PRIOR表示上一条记录，比如 CONNECT BY PRIOR org\_id = parent\_id就是说上一条记录的org\_id 是本条记录的parent\_id，即本记录的父亲是上一条记录。*

*条件3 是过滤条件，用于对返回的所有记录进行过滤。*

*简单介绍如下：*

*早扫描树结构表时，需要依此访问树结构的每个节点，一个节点只能访问一次，其访问的步骤如下：*

*第一步：从根节点开始；*

*第二步：访问该节点；*

*第三步：判断该节点有无未被访问的子节点，若有，则转向它最左侧的未被访问的子节，并执行第二步，否则执行第四步；*

*第四步：若该节点为根节点，则访问完毕，否则执行第五步；*

*第五步：返回到该节点的父节点，并执行第三步骤。*

*总之：扫描整个树结构的过程也即是中序遍历树的过程。*

*1． 树结构的描述*

*树结构的数据存放在表中，数据之间的层次关系即父子关系，通过表中的列与列间的关系来描述，如EMP表中的EMPNO和MGR。EMPNO表示该雇员的编号，MGR表示领导该雇员的人的编号，即子节点的MGR值等于父节点的EMPNO值。在表的每一行中都有一个表示父节点的MGR（除根节点外），通过每个节点的父节点，就可以确定整个树结构。*

*在SELECT命令中使用CONNECT BY 和蔼START WITH 子句可以查询表中的树型结构关系。其命令格式如下：*

*SELECT 。。。*

*CONNECT BY {PRIOR 列名1=列名2|列名1=PRIOR 裂名2}*

*[START WITH]；*

*其中：CONNECT BY子句说明每行数据将是按层次顺序检索，并规定将表中的数据连入树型结构的关系中。PRIORY运算符必须放置在连接关系的两列中某一个的前面。对于节点间的父子关系，PRIOR运算符在一侧表示父节点，在另一侧表示子节点，从而确定查找树结构是的顺序是自顶向下还是自底向上。在连接关系中，除了可以使用列名外，还允许使用列表达式。START WITH 子句为可选项，用来标识哪个节点作为查找树型结构的根节点。若该子句被省略，则表示所有满足查询条件的行作为根节点。*

*START WITH： 不但可以指定一个根节点，还可以指定多个根节点。*

*2． 关于PRIOR*

*运算符PRIOR被放置于等号前后的位置，决定着查询时的检索顺序。*

*PRIOR被置于CONNECT BY子句中等号的前面时，则强制从根节点到叶节点的顺序检索，即由父节点向子节点方向通过树结构，我们称之为自顶向下的方式。如：*

*CONNECT BY PRIOR EMPNO=MGR*

*PIROR运算符被置于CONNECT BY 子句中等号的后面时，则强制从叶节点到根节点的顺序检索，即由子节点向父节点方向通过树结构，我们称之为自底向上的方式。例如：*

*CONNECT BY EMPNO=PRIOR MGR*

*在这种方式中也应指定一个开始的节点。*

*3． 定义查找起始节点*

*在自顶向下查询树结构时，不但可以从根节点开始，还可以定义任何节点为起始节点，以此开始向下查找。这样查找的结果就是以该节点为开始的结构树的一枝。*

*4．使用LEVEL*

*在具有树结构的表中，每一行数据都是树结构中的一个节点，由于节点所处的层次位置不同，所以每行记录都可以有一个层号。层号根据节点与根节点的距离确定。不论从哪个节点开始，该起始根节点的层号始终为1，根节点的子节点为2， 依此类推。*

*5．节点和分支的裁剪*

*在对树结构进行查询时，可以去掉表中的某些行，也可以剪掉树中的一个分支，使用WHERE子句来限定树型结构中的单个节点，以去掉树中的单个节点，但它却不影响其后代节点（自顶向下检索时）或前辈节点（自底向顶检索时）。*

*6．排序显示*

*象在其它查询中一样，在树结构查询中也可以使用ORDER BY 子句，改变查询结果的显示顺序，而不必按照遍历树结构的顺序*

*\*/*

select \* from emp where mgr=7839;

select level,emp.\*

from emp

start with mgr=7839

connect by prior empno=mgr;

customer(custid, name, sex, mobile, email)

item(itemid, name, category, price)

itemSale(transid, itemid, quantity)

transaction(transid, custid, date)

creat or replace view v\_sale\_by\_ as

select i.category,to\_char(a.date,'yyyymm') 月份,

sum(b.quantity\*i.price)

from transaction a , itemSale b, item i

where a.transid=b.transid and b.itemid=i.itemid

group by i.category,to\_char(a.date,'yyyymm') ;

create table tras( tranid number, custid char(8),

tra\_date date);

drop table tras;

create table tras( tranid number, custid char(8),

"date" date);

drop table tras;

create table tras(rq date , qty number, price number,

amount number generated always as (round(qty\*price,2)));

*--virtual field*

select \* from tras for update;

with x

as (

select \*

from (

select to\_char(trunc(sysdate,'mm')+level-1,'iw') wk,

to\_char(trunc(sysdate,'mm')+level-1,'dd') dm,

to\_number(to\_char(trunc(sysdate,'mm')+level-1,'d')) dw,

to\_char(trunc(sysdate,'mm')+level-1,'mm') curr\_mth,

to\_char(sysdate,'mm') mth

from dual

connect by level <= 31

)

where curr\_mth = mth

)

select max(case dw when 2 then dm end) 星期一,

max(case dw when 3 then dm end) 星期二,

max(case dw when 4 then dm end) 星期三,

max(case dw when 5 then dm end) 星期四,

max(case dw when 6 then dm end) 星期五,

max(case dw when 7 then dm end) 星期六,

max(case dw when 1 then dm end) 星期天

from x

group by wk

order by wk

**---index**

**---创建索引**

create index idx\_sno on stu\_2012 upper(name);

create unique index idx\_stu\_name on stu\_2012(name);

表示索引是唯一的

**删除索引**

drop index idx\_sno;

**---重建索引**

alter index idx\_sno rebuild;

**---geberate sql by sql**

**---数字字典**

**---dict**

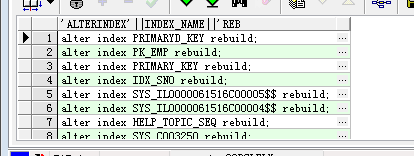
select \* from dict;

select \* from USER\_INDEXES;

select 'alter index ' ||index\_name|| ' rebuild;'

from user\_indexes;

运行之后得：



直接复制粘贴得：

alter index PRIMARYD\_KEY rebuild;

alter index PK\_EMP rebuild;

alter index PRIMARY\_KEY rebuild;

alter index IDX\_SNO rebuild;

alter index SYS\_IL0000061516C00005$$ rebuild;

alter index SYS\_IL0000061516C00004$$ rebuild;

alter index HELP\_TOPIC\_SEQ rebuild;

alter index SYS\_C003250 rebuild;

select count(\*) from stu\_2012;

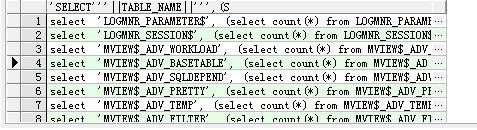
select \* from user\_tables;

select 'select ''' ||table\_name|| ''', (select count(\*) from '||table\_name|| ') from dual union all'

from user\_tables;

*-----用两个单引号来表示单引号里引用的一个单引号*

*运行之后得：*

**

直接复制粘贴得：

select 'LOGMNR\_PARAMETER$', (select count(\*) from LOGMNR\_PARAMETER$) from dual union all

select 'LOGMNR\_SESSION$', (select count(\*) from LOGMNR\_SESSION$) from dual union all

select 'MVIEW$\_ADV\_WORKLOAD', (select count(\*) from MVIEW$\_ADV\_WORKLOAD) from dual union all

select 'MVIEW$\_ADV\_BASETABLE', (select count(\*) from MVIEW$\_ADV\_BASETABLE) from dual union all

select 'MVIEW$\_ADV\_SQLDEPEND', (select count(\*) from MVIEW$\_ADV\_SQLDEPEND) from dual union all

select 'MVIEW$\_ADV\_PRETTY', (select count(\*) from MVIEW$\_ADV\_PRETTY) from dual union all

select 'MVIEW$\_ADV\_TEMP', (select count(\*) from MVIEW$\_ADV\_TEMP) from dual union all

select 'MVIEW$\_ADV\_FILTER', (select count(\*) from MVIEW$\_ADV\_FILTER) from dual

这里先简单的给定这四个集合运算定义，并以一段代码实例，例子今后有时间再附上：

select \*

from emp

where sal < 4000

union

select \*

from emp

where sal > 2000

Union，对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；

Union All，对两个结果集进行并集操作，包括重复行，不进行排序；

Intersect，对两个结果集进行交集操作，不包括重复行，同时进行默认规则的排序；

Minus/except，对两个结果集进行差操作，不包括重复行，同时进行默认规则的排序。

**---序列号**

create sequence seq\_sno;

create table test(id number, name varchar(30) );

insert into test(id , name)

values(seq\_sno.nextval, 'abc');

insert into test(id , name)

values(seq\_sno.nextval, 'yun');

select \*

from test;

commit

**---fuchtion**

**---procedure**

**---y=f(x)**

select \* from stu\_2012;

create or replace function fun\_avg\_score( v\_degree in number)

return number as

v\_avg\_score number;

begin

select round(avg(score),1) into v\_avg\_score from stu\_2012

where degree=v\_degree;

return(v\_avg\_score);

end fun\_avg\_score;

select fun\_avg\_score(1), fun\_avg\_score(2)

from dual;

create or replace procedure p\_avg\_score(

v\_degree in number, v\_avg\_score out number)

is

v\_test number;

begin

select round(avg(score),1) into v\_test from stu\_2012

where degree=v\_degree;

v\_avg\_score:=v\_test;

dbms\_output.put\_line(v\_avg\_score);

end p\_avg\_score;

declare

x number;

begin

p\_avg\_score(1,x);

exception when others then null;

end

**sql触发器**

update of 表名 on （列名）

delete on

create trigger 触发器名 after\before insert on 表名

referencing new row as nrow-----可表示要操作的列(如要插入的列)

referencing old row as orow-----可表示旧列（如表示删除的列）

for each row

when

条件

And

begin atomic

要触发的语句

End